To the Graduate Council:

I am submitting herewith a thesis by Michael A. Goldston entitled "Image Processing: A Digital Signal Processor Application." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

R. E. Bodenheimer, Major Professor

We have read this thesis and recommend its acceptance:

Donald W. Bouldin

R. C. Gonzalez

Accepted for the Council:

Vice Provost and
Dean of the Graduate School

IMAGE PROCESSING:
A DIGITAL SIGNAL PROCESSOR APPLICATION

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Michael A. Goldston

May 1989

ABSTRACT

Digital signal processors are a new type of microprocessors that are specially suited for high-speed and high-performance systems. A brief application, both hardware and software, is presented of a specific DSP, the TMS320C25. This DSP application is used in conjunction with a commercially available image processor. The idea presented increased performance of the image processor by enabling mathematically complicated algorithms to be performed on video images using random accessing.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Image processing is a fast moving discipline of electrical engineering that currently is undergoing swift changes in algorithms, methods, and designs. New algorithms are becoming more complex and mathematically complicated. As these algorithms become more complex, the CPU time required to perform them lengthens exponentionally. Many times these algorithms require large amounts of memory in order to make their computations which can be as simple as arithmetic operations, or as complex as a Fast Fourier Transfrom (FFT) butterflies [1]. Image processing is one area that emerging and expanding new technologies in microprocessor and other specialized areas of electronic components are enabling the engineer to do more, better, and faster.

A major problem facing image processing engineers is that increased algorithm complexity increasingly burdens the CPU. To help combat this problem, hardware designers, based in image processing, have seen many innovations in chip design surface in the last few years. Such chips as video RAMs (VRAMs) [2] that allow for greater storage of image memory and RAMDACs [3] that provide intricate high-resolution displays have been introduced in the last two years alone.

Lastly, but most importantly, the development of microprocessors that are designed specifically to be fast at arithmetic and algorithmic calculations produces an alternative to slow CPUs. These microprocessors are often called digital signal processors (DSPs). This thesis deals with one attempt to increase the throughput of an image processor by incorporating a digital signal processor of some kind into the system.

The primary use of this DSP was to enhance imaging operations on the Perceptics IP9200, so speed was of the utmost importance. An attempt was made to allow the DSP to run as fast as possible. Most decisions of preferences of one type of design over another were based upon speed. The second, but not an unimportant requirement, was that the design not take up too much space. This design is eventually going to be included on a 1024x1024 memory board in an image processor system. With that much memory and accompanying control circuitry on a board, space is at a premium.

Chapter II deals with introduction of the IP9200 and explanations of some of its key features. A brief discussion is also included on the need for some type of additional hardware to increase performance. The hardware required is a DSP and is discussed in Chapter III. This chapter deals with different DSPs, compares them, and finally discusses the TI TMS320C25. Chapter IV deals with the hardware design, and Chapter V presents the software design. Both hardware and software considerations are crucial in a DSP design if optimal speeds are to be achieved. Lastly, Chapter VI deals with results of the design and enhancements that are suggested to make the DSP even faster and more efficient in future designs.

CHAPTER II

ELEMENTS OF IMAGE PROCESSING SYSTEMS

An image processing engineer needs computer resources that combine to form a workstation for development of algorithms. A typical system consists of a host computer, image processor (often called an IP), camera, display monitor, and a keyboard/terminal as shown in Figure 1.

The host computer is used mainly as a mass storage device to save images that are to be used in the image processor. Image processor control is also typically provided by the host computer. Cameras are used as a means of recording images and monitors are used to display them. As algorithms are developed, a monitor shows the progress of these algorithms, so that the developer can see where improvements might be made.

IMAGE PROCESSORS

An image processor is the main component of an imaging workstation and represents the fundamental tool of an image processing engineer and is shown in Figure 2. Video images input from cameras are 'grabbed' into an IP's memory. These images are digital representations of photographs or video images and are stored in a manner that computer algorithms of various types can be executed on them. Algorithms perform image analysis on the video images to examine detail that may at times be impossible for humans to decipher. A good example of this type of application is that of an X-ray. X-rays are typically examined on IPs to enhance regions to make diagnoses of bone structures easier for doctors to make. IPs are also used to examine the fuel in solid-fuel rocket boosters in order to enhance flaws in the structure normally hidden from view. Algorithms can be

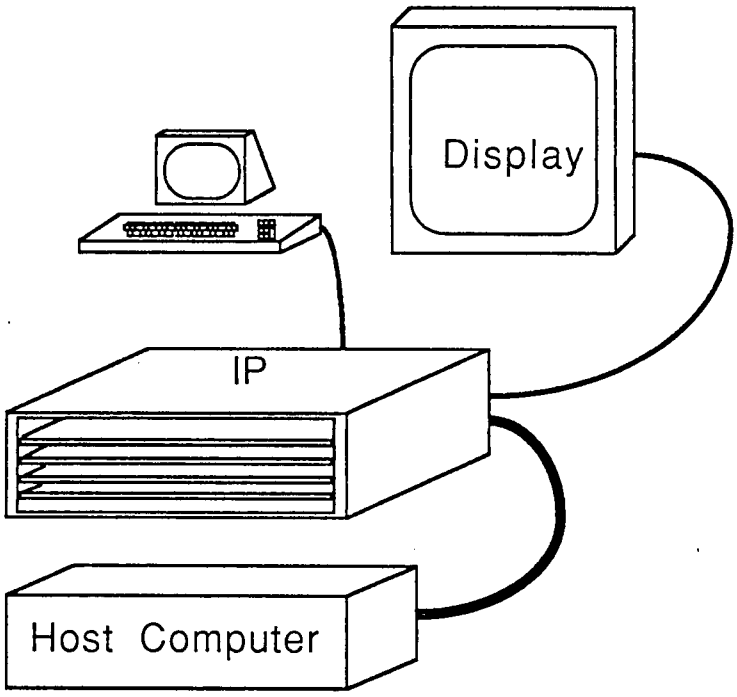


Figure 1. Image Processing Workstation

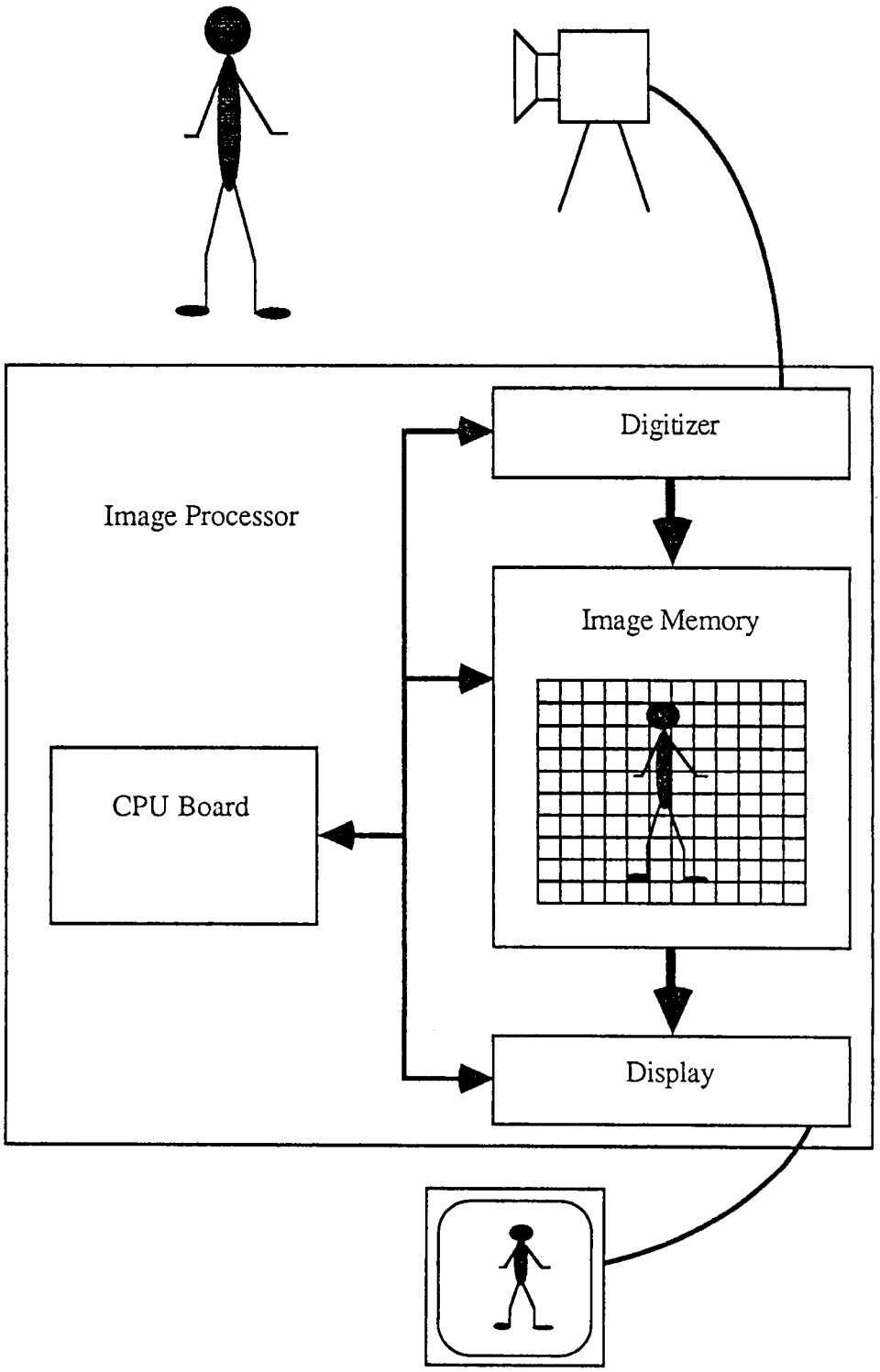


Figure 2. Image Processor

used to develop an automatic process that would be too mundane for a human to do repetitively [4]. A prime example of an automation system is the reading license plates on the back of moving vehicles. While a human can certainly read plates with greater accuracy, a computer can do this without boredom or fatigue.

Image processors are typically high-end computers that use the latest in device technology and are usually large enclosures with the capacity to hold several boards of varying degrees of complexity and functionality. Several boards are basic to all IPs and form a required foundation. These essential boards are a display, digitizer, and memory board as depicted in Figure 2. Digitizer boards receive an analog input from a camera and convert this information into a digital form that is to be stored in the IP. This data is saved in an image memory board of some kind. Display boards take this same digital data and reconvert it into an analog signal that is used to drive a monitor. All of this data manipulation is passed via a backplane where there is a series of video buses. Control and timing of these video buses is varied from IP to IP and can be very complex.

These three types of boards (display, digitizer, and memory) form the basic building block of an image processor. They do not, however, provide a means to execute sequential code or make algorithmic or arithmetic operations. IPs differ on the design philosophy of how to perform various functions on these images. Very low end IPs use the host computer to perform all operations on the memory. This provides for a relatively cheap system, but processing time becomes a casualty. Due to slow host processors and lag in communication timing, operations take as much as 100 times longer for a host computer than if a dedicated CPU were included into the IP.

This introduces the need for a controller board for an IP. Because of its proximity to the accompanying devices, memory access times become shorter than those for the host. Also, system upkeep for the IP is a much easier task than for the host, so that a dedicated

microprocessor becomes inherently faster. A controller board of an IP typically has a high-performance CPU such as a Motorola 68000, Intel 82386, or National Semiconductor 32000. This microprocessor is usually given tasks such as communication protocols, management of system memory, and loading and saving of images. While these tasks require less time than disk drive control, memory paging, and multi-processing tasks of a host CPU, they do consume valuable CPU time.

Many times algorithms to be performed on images are done in a sequential order. That is, the same operation is done on adjacent pixels in image memory. A pixel is defined to be the smallest unit of a picture element that makes up a video image. Pixel processors present a viable solution to fast execution of these operations. Depending upon the operation to be performed, an entire image can be passed through the processor 30 times a second. This phenomenal increase in speed, over standard CPUs, is done through pipelining of discrete components such as multipliers, ALUs, and shifters. As stated, a requirement for this fast operation, is the continuity of image accesses. Random accesses in pixel processors are forbidden.

There is always the need to make systems perform better and image processors are used to enhance and update algorithms on video images. In order to make algorithms perform better and with greater accuracy, higher resolutions of memory are needed as well as more calculations need to be made. A simple doubling of resolution quadruples the amount of image memory needed. If the same algorithm (before and after memory expansion) is performed, then the CPU time is typically four times longer than before. But, the algorithms seldom stay the same and continue to get more complex. This introduces heavy constraints on the already burdened CPUs. As mentioned above, the pixel processors don't provide much of a solution to this problem as they only perform algorithms that are sequentially oriented.

IP9200

The Perceptics IP9200 is a general purpose IP. The IP9200 is a computer subsystem with a backplane and chassis in which up to 22 boards can be inserted as shown in Figure 3. Seven 16-bit video buses are used for transferring images back and forth between boards in the system as well as a 16-bit CPU data bus with a 25-bit address bus. Each video bus has a 15MHz bandwidth. In other words, each video bus can pass a 512x512x16-bit image across the backplane 30 times a second. This represents over 100MBytes per second of throughput.

Currently there are seven different boards that are used in the IP9200. They are the controller, display, digitizer, memory, sync, pixel processor, and FTA boards. All IP9200s require a sync board to provide clocks to other boards in the system and to generate all of the video bus timing.

The microprocessor on the controller board is the Motorola 68010. Additionally, the controller board has DMA capability, an IEEE-488 bus interface, several serial communication ports and the host interface. A host computer can take over IP9200 buses and force the 68010 into a dormant state.

The digitizer board accepts standard RS-170 input from a camera. Camera sync can be driven from the IP9200 or the digitizer can lock to the camera. There is an input look-up-table (LUT) for image transformations that helps normalize non-linearity in camera digitization. Images of up to 512x512 pixels in resolution can be grabbed 30 times a second.

Color (24-bit) or monochrome (8-bit) displays are generated on the display board. Provisions are also made for a graphic overlay and an alpha-numeric overlay. Intricate displays can also be shown with the use of a split-screen mode which simultaneously shows two images at once. A cursor generator is also provided.

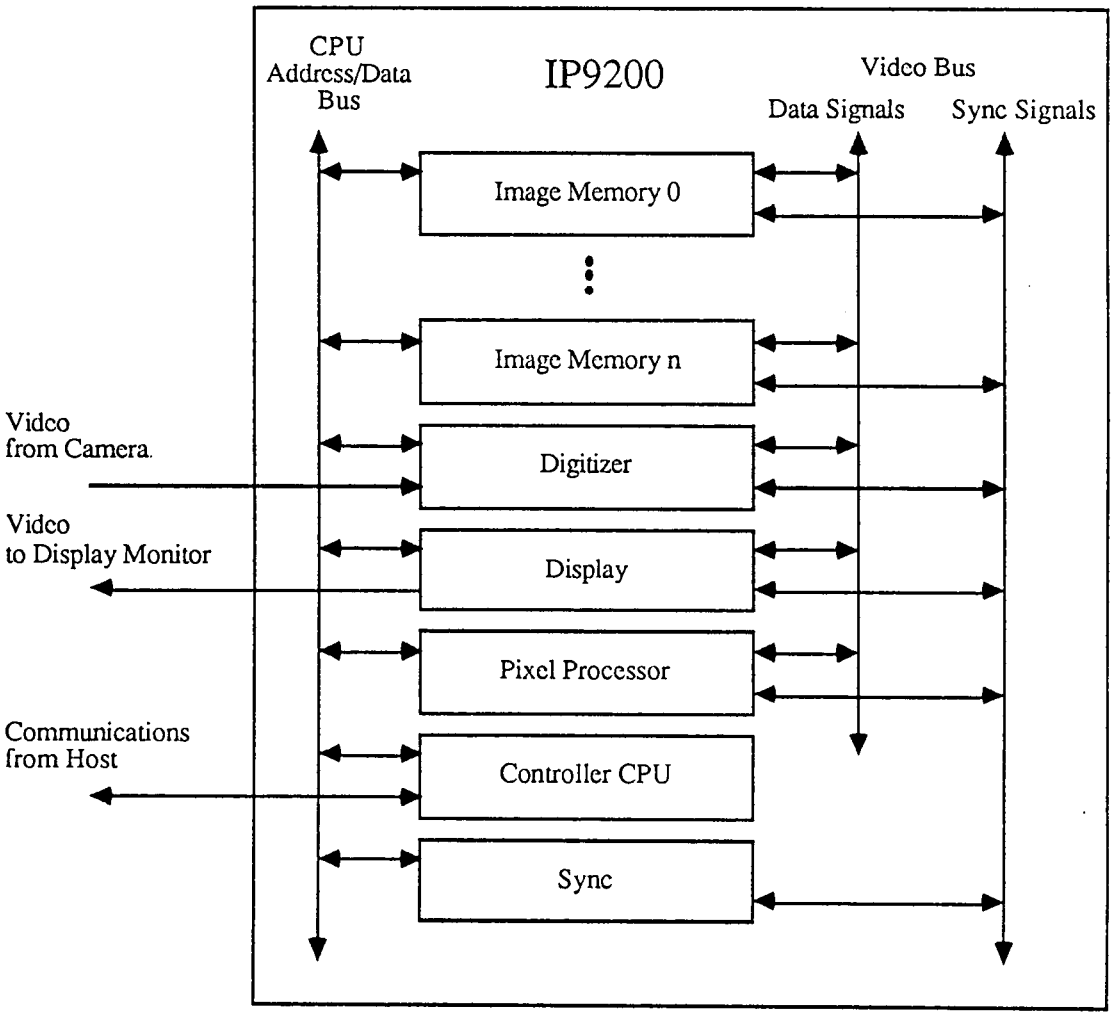


Figure 3. IP9200 System Configuration

The standard memory board holds a 512x512x16-bit image. A special high-resolution memory board holds a 1024x1024x16-bit image. Zooming and scrolling of the image is provided, the result of which can be passed either to the pixel processor or to the display. Images can be grabbed into the low or high byte of image memory. Several memory boards can be logically joined together in order to form a large image. If sixteen high-resolution boards are used, an image size of 4096x4096 can be achieved.

The pixel processor has a pipelined architecture made up of two multipliers, two independent ALUs, shifters, and other discrete devices. Many different arithmetic and logical functions can be performed such as XOR, OR, ADD, or SUBTRACT. These discrete components can be configured to work on up to three images simultaneously to produce a single desired output. Operations can be limited to a specific region of interest (ROI). Geometric averaging and subtraction can be performed on 'live' images. All of these components combine to form a pipeline that operates on a 512x512x16-bit image in 1/30 of a second.

The only specialized board at present is the FTA (Fourier Transform Accelerator) board. It performs a 24-bit floating point FFT on a 512x512 image in 24 seconds.

IP9200 DISADVANTAGES

In the IP9200, the 68010 is used for system upkeep, and runs a resident image processing system (RIPS). This makes performance of extremely fast operations (~700ns/pixel) difficult, if not impossible. Typically, it takes greater than 6 seconds to perform a 'real-time' (1/30 second) ADD operation using the pixel processor. This enormous time differential is due to system overhead. Even without the overhead, a direct memory fill (using no pixel processor) takes more than 2 seconds. More advanced

operations such as a 6-parameter warp (image rotation) takes in excess of 25 seconds. Also, there is no floating-point coprocessor (68881) in the system.

In addition to the drawbacks of the controller board, a minimum IP9200 system contains five boards (controller, display, digitizer, sync, and memory). This minimal system does not include a pixel processor used for quick operations. So a minimal practical configuration becomes six boards. This also contains only one memory board. Typically an IP system has at least three memory boards necessary for RGB (color) processing. This addition of memory increases the burden on the already weakened 68010.

These deficiencies are not specific to the IP9200. All IPs have these same addressed problems and some have more. The IP9200 is a high-performance machine that is quite complex. Some IPs require as many as 15 boards in order to meet the IP9200 in functionality.

One solution to improving performance is to include a processor of some kind on each memory board. This helps in several regards. First, it provides for a dedicated processor located next to the memory for the fastest possible access timing. Secondly, as memory boards are added to the system, the CPU power of the system actually increases instead of decreasing as previously explained. The solution presented in this thesis is the inclusion of a digital signal processor into an IP9200 to enhance the performance of the system.

CHAPTER III

DIGITAL SIGNAL PROCESSORS

As image processing routines become more advanced and complex, the need for specialized hardware increases. The demands for hardware are for faster, smaller, and cheaper systems. A specialized set of imaging products involve digital signal processors (DSPs). These DSPs vary widely in capability and speed. On the lowest level, DSPs can be discrete components doing specific operations at optimum speeds. This group comprises ALUs, multipliers, shifters, and similar chips or chip-sets. Usually, these DSPs are costly in board space, and are hard to re-configure for different algorithms, but these chips can be extremely fast (~40ns/pixel) [5].

Another group of DSPs are those in the CPU category. They are designed to run sequential code just as a microprocessor would. The only difference between these DSPs and microprocessors is that these chips are designed to be optimal at arithmetic-type operations. CPU-type DSP chips tend to allow smaller address and data spaces than microprocessors; however, they operate at much higher clock rates.

DSPs IN IMAGE PROCESSING

Several integrated circuit companies have designed DSP chips that are of the CPU-type. A short comparison is shown in Figure 4. There are currently four major DSPs of interest available in at least sample quantities. Each one has distinct advantages and disadvantages. One aspect common to most DSPs is that they have a Harvard type architecture. This means that there are separate address and data buses for program and data space for a total of four buses as demonstrated in Figure 5.

	AT&T	TI	Analog	Motorola
On-chip RAM - bytes	4K	544	0	768
On-chip ROM - bytes	2K	8K	0	6.7K
Internal Registers	16	22	34	34
External Space - bytes	64K	256K	80K	576K
FFT Addressing	No	Yes	Yes	Yes
Clock Rate	16MHz	40MHz	8MHz	20MHz
MIPs	4	10	8	10.25
ALU Width - bits	40	32	16	56
Availability	3Q88	Now	3Q88	1Q89
Cost	$200	$30	$150	$200
Serial Capability	1	1	No	2
Interrupts	0	2	4	2
Halt Capability	Difficult	HALT	BR/HALT	BR
Instruction Set	117	133	?	62
Programming	Cryptic	Assembly	Cryptic	Assembly

AT&T	WE-DSP32
TI	TMS320C25
Analog	ADSP-2100
Motorola	DSP56000

Figure 4. Comparison of DSPs

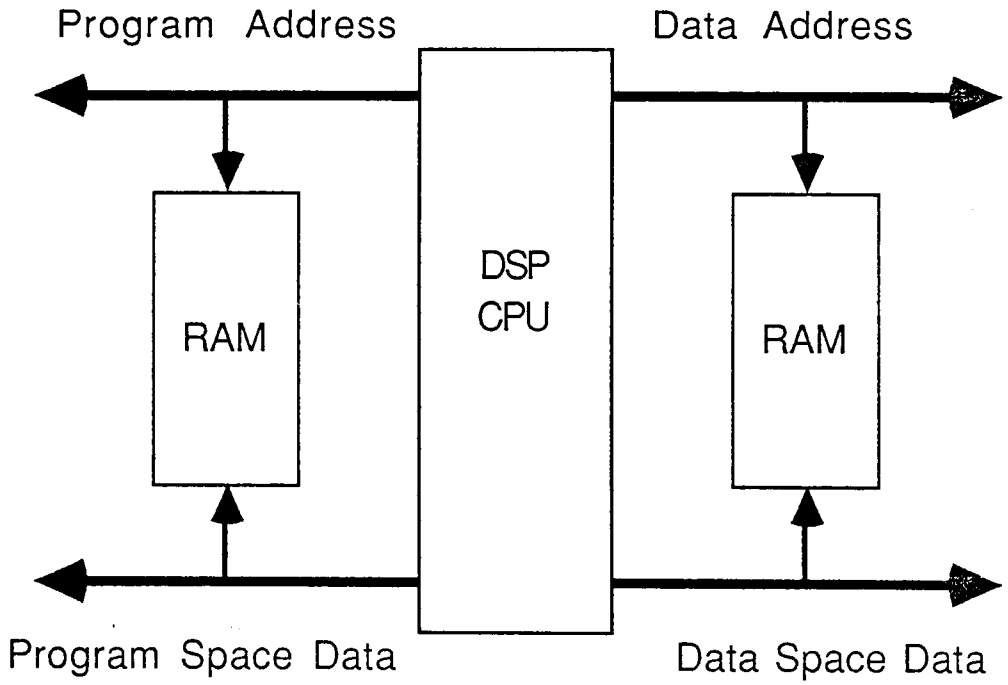


Figure 5. Harvard Architecture

AT&T makes the WE-DSP32 [6] which can perform floating-point operations establishing an advantage over the others. Another advantage is that the WE-DSP32 has four 40-bit accumulators and a rich instruction set. A disadvantage of the AT&T DSP is the cryptic language that is necessary to learn in order to program the DSP. Additionally, disadvantages include small addressing capabilities, high price, and late availability. Finally, the operating system presented would have to be drastically altered because halting of the DSP is difficult.

Similarly, Analog Devices ADSP-2100 [7] has a very cryptic programming language. In both the Analog Device and the AT&T DSP, programming is similar to bit-banging. This causes software development to be laboring and difficult. In addition to this problem, the ADSP-2100 has a relatively small (80KB) of external address space, making access to image memory more difficult. The Analog DSP does, however, have a means of accessing FFT points (bit reversed addressing) and has a high MIP rate. It also has 34 internal registers and halting of the DSP is easy.

The most promising DSP is the Motorola DSP56000 [8] because it has the highest MIP rate, largest address space, and largest ALU width. Its major disadvantage is the availability of production units. Although it is readily available in sample quantities, it will not get under full production until the first quarter of 1989.

The final DSP is a compromise of all of the others. The TI TMS320C25 [9] has a high MIP rate, large external address space, rich instruction set, and is available now. Production of the TMS320C25 has been long enough that it has established itself as the standard, and in a PLCC costs about $30. Although this DSP will soon be outclassed by the others mentioned, it was chosen because of availability and ease of use. The next generation of TMS320C25 is scheduled to be released soon and will establish a new

standard that future generation DSPs will try to equal. Compatibility needs for future generation DSPs was also considered in the decision to use the TI DSP.

TMS320C25 DSP

The Texas Instruments TMS320C25 is a CPU type DSP and has several features typical of DSPs. Program and data spaces are completely separate maintaining a Harvard architecture and both are 64KW in depth. Internally, the program and data address and data buses are completely separate to maintain maximum speed. Externally, both program and data space data buses are multiplexed, as well as both address buses. This is done to keep the pin-count of the device at a minimum. This restriction does not hinder the speed as much as one might first think. The reason is that the TMS320C25 has 544 words of RAM internal to the device that can be configured to best suit an application. Many programs can be written and run internal to the device and keep external accesses to a minimum. Internal is a 32-bit ALU and accumulator preceded by a 16x16-bit multiplier. The architecture allows the TMS320C25 to do a multiply-accumulate (MAC) in 100ns. A 40MHz clock is used and is immediately reduced to a four-phase 10MHz clock for single cycles. There is also 4KW of masked ROM located on chip to allow for single chip operation, but this thesis does not cover that feature in this application.

The TMS320C25 has a complete range of mathematical functions in its instruction set and has 8 addressing modes including a bit-reversed mode for accessing pixels in FFT algorithms. Most functions in the instruction set make use of the wide range of addressing capability.

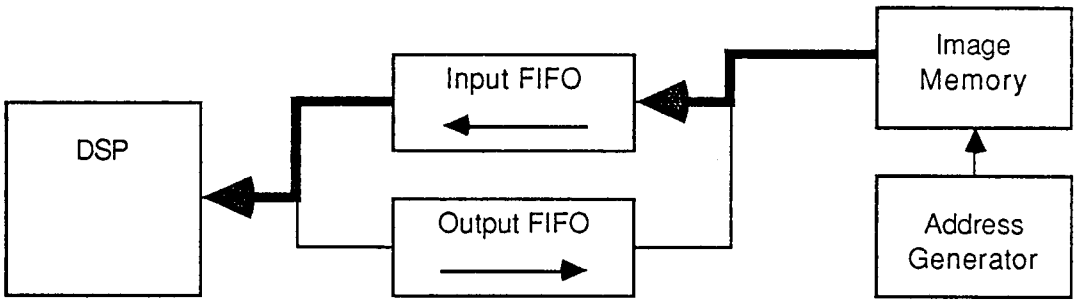
MEMORY CONSTRAINTS

The first consideration for fast operation is the access to local memory making an access to memory in one clock cycle with no wait-states. Internal memory is rated as no wait-states at the maximum specified clock (40MHz). For optimal execution speeds, external memory needed to be no wait-state also. Upon evaluating cycle times for the TMS320C25, external RAMs needed to have access speeds on the order of 20ns. These are readily available in small size RAMs, but not in large sizes.

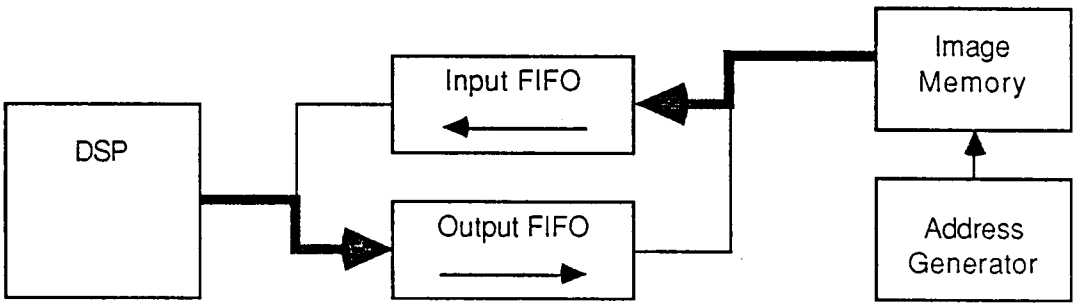
An EPROM is needed to store the number of image processing subroutines. The above cycle times are far to fast for EPROMs, so a design was chosen that used these EPROMs as little as possible.

Image memory, on the other hand, is another timing concern. Present image memory boards have a fixed access time, for the most part, of around 700ns and cannot be decreased. Several designs, that tried to get around this restraint, were conceived.

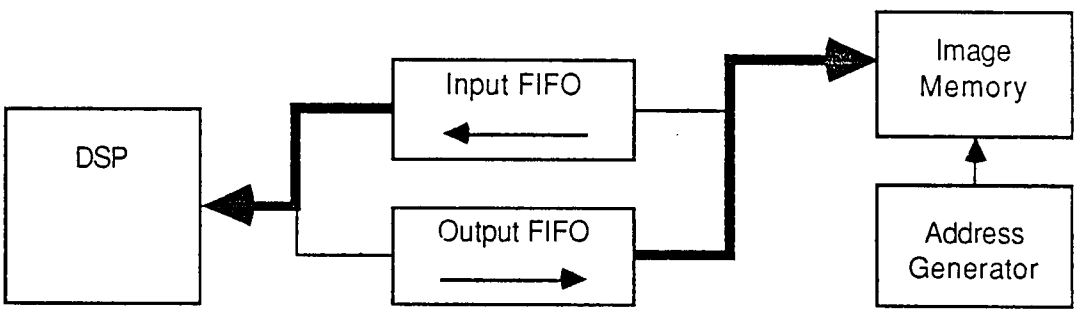
One of the design ideas to reduce image memory access time was to use a set of FIFOs to transport data to and from the image memory. The basic scheme is shown in Figure 6. An initialization step is done to set up the pipeline of the FIFOs. Figure 6.a shows that the input FIFO is loaded with a row, column or region of interest (ROI) of the image. When the first section is loaded into the FIFO, the DSP then quickly unloads the FIFO into its own fast RAM. While the operation is being performed in the DSP, another ROI is being loaded into the input FIFO. As this occurs, the DSP is putting the results in the output FIFO. This process is shown in Figure 6.b. When both the input and output FIFOs are full, the process reverses. The image is filled with the resultant data, contained in the output FIFO, while the DSP reads in the next ROI, contained in the input FIFO. This is shown in Figure 6.c. When this step is complete, the process repeats, toggling



a. Set-up



b. Step 1



c. Step 2

Figure 6. FIFO Memory Operation

between the two states represented in Figures 6.b & 6.c until all of the desired image has been processed.

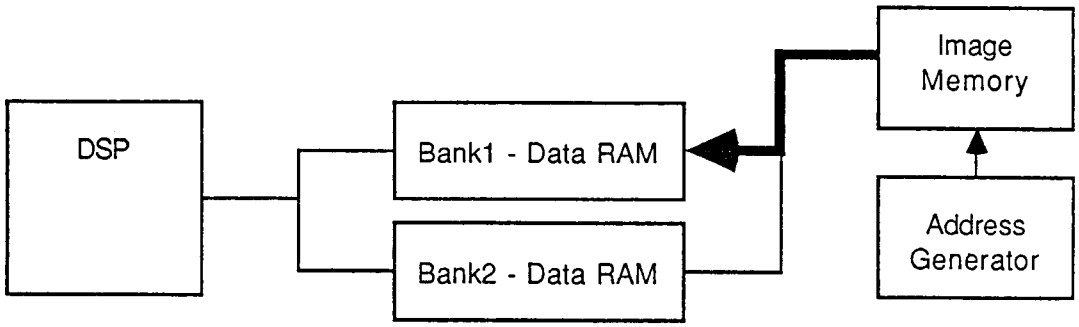
Computationally complicated algorithms (ones that the DSP spends as much time calculating the results as the image memory is being extracted into the FIFO) make this idea useful, but on most simple operations, this procedure actually causes a bottle neck waiting on the FIFOs to empty.

Another idea very similar is the process of ping-ponging banks of memory. The process is shown in Figure 7. First, the initialization step similar to the previous FIFO set-up step is done. The start-up step, shown in Figure 7.a is the loading of the first ROI into bank1 of fast DSP RAM. After that is accomplished, the ping-ponging can start. As shown in Figures 7.b & 7.c, the DSP is working on one set of RAM while the image memory is being loaded into another bank. This design proves to be a little more efficient than the FIFO approach.

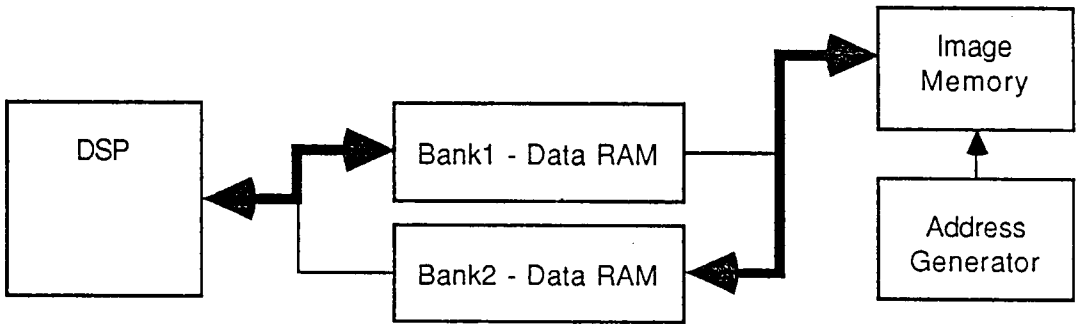
The problem with both of these applications is that random access to the memory is still a limiting factor. Also, the address generator used in both schemes to control the loading of the RAMs and the FIFOs can become quite complex. In interest of simplicity and size requirements, a simple page-mode type direct access to the image memory was opted in this thesis. This cost time in the access to image memory, but simple suggestions are made in the enhancements section that could increase this access.

IP9200 DESIGN CONSIDERATIONS

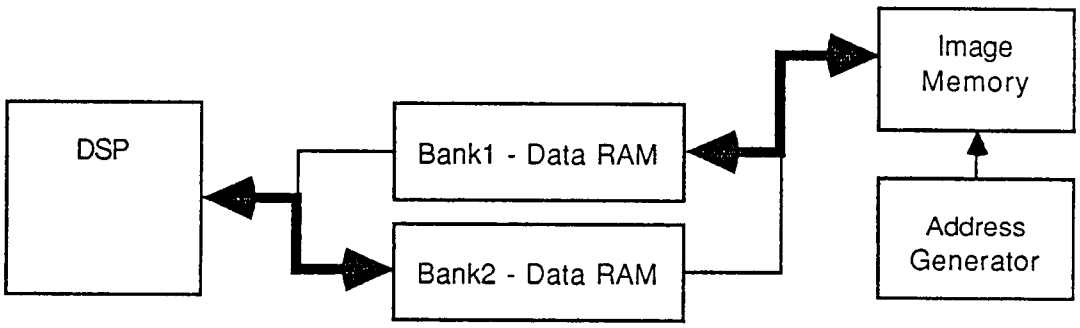
The design consisted of a wire-wrap board that fits into a Perceptics IP9200 chassis, requiring a standard 15x15 inch card with the IP9200 bus connections on fingers at the rear of the board. This board replaces the standard interface card to the IP9200, so cable connections must also be made to the host.



a. Set-up



b. Step 1



c. Step 2

Figure 7. Ping Pong Memory Operation

A host computer is connected to the IP9200 backplane through an interface and two 50-pin ribbon cables. All signals need to be doubly terminated through pull-up and pull-down resistors to prevent reflections on the cable.

The DSP board has two basic requirements in conjunction with the interfacing of the host and the IP9200. First, the design should behave like any other board in the system. Secondly, the design should be under complete control of the host at all times. When the DSP is not running, the host has direct access of the IP9200 backplane.

The DSP has direct access of the IP9200 backplane at times, so all control signals need to be generated. Most CPU-type DSPs have only a 16-bit address bus while the IP9200 has a 25-bit wide address bus, so some type of paging mechanism is required. Since the DSP and the host both have control of the backplane, control signals have to be bi-directional and care needs to be taken whenever these signals change directions.

In order to gain speed advantages with the DSP, dual access of the IP9200 from both the DSP and the host simultaneously is desirable. This means that a separate internal bus is needed for the DSP operating internal code at the same time that the host is making other accesses to image memory on other boards.

ADDRESSING MODES

Another consideration is the type of accesses desired to and from the DSP. An access mode from the IP9200 is required. Since the address range of the IP9200 is much larger than the DSP's, no paging mode is required. However, additional signals (IS-, DS-, and PS-) is also needed to be mapped into the address map in order to get the correct DSP resource to be accessed.

The DSP instruction set contains many useful functions and eight addressing modes. In order to make the DSP perform as smoothly as possible, the design should take

advantage of these addressing modes. At least two additional addressing modes to the image memory are needed. Some kind of direct mapping into the DSP space is needed as well as a page-mode type addressing. These are discussed at the end of the hardware description.

OPERATING SYSTEM

The first kind of software that a microprocessor needs to run is an operating system (OS) of some kind. Letting the DSP act as a slave-type device to the host computer was deemed as the best type of operation. A bare minimum of an OS is sufficient, demonstrating that a complete operating system is not required. The DSP is designed to run image processing algorithms on an image plane. So, an operating system should allow the DSP to run subroutines quickly with little overhead.

Several options were considered for an operating system. The first was a mail-box type system, in that the DSP is constantly running and watching for a status word or interrupt signal from the host CPU. This type operation has the advantage that the DSP could conceivably be running several tasks simultaneously. Status of the progress of the DSP would also be kept in another mail-box that the host computer can monitor. Again, the need for simplicity and speed made this operating system in the present state undesirable. The operating system decided upon was similar to a mailbox OS except that the DSP is dormant most of the time remaining idle until the host has fully loaded all the parameters and signals the DSP to start whatever function was requested.

This choice selection raises another question of how to pass parameters to the DSP. An obvious choice would be similar to the C language in that parameters needed would be placed upon a stack. The intent of the DSP, however, was to possibly run several routines back to back on the same region requiring the host to place the needed parameters in the

DSP RAM between each call. Also, each routine written for the DSP would have to decide about where, in relation to the stack, are certain parameters. A simple solution was to assign common parameters a fixed location in DSP data RAM, so that all needed parameters would be available and at predictable locations for all routines. This prevented the host from having to re-load parameters after each routine. As an added bonus, this makes debugging of the software much easier.

EPROM access should be kept to a minimum to increased speed, but the subroutines are to be stored in EPROM. The solution to this problem was to have all subroutines written and compiled to run out of internal DSP RAM. The first action that the DSP takes on a subroutine call is to copy the desired routine into fast memory and then the second action is to jump to the entry point of that routine. This cuts EPROM access to the absolute bare minimum.

CHAPTER IV

HARDWARE DESCRIPTION

The design fits into a IP9200 backplane and communicates with a host computer, and the basic system configuration is shown in Figure 8. Most of the design fits into five distinct categories of operation. This logical separation include the DSP, memory, memory control, host interface, and IP9200 interface and are shown in a block diagram in Figure 9. System bus structure is shown in Figure A.1 and interconnecting signals in Figure A.2. The DSP section includes the TMS320C25 and support chips. Data and program RAM as well as the program EPROM are included in the memory section. The memory control section consists of timing chips, chip decoders, and devices that control the direction of data flow. Devices that allow the wire-wrap board to meet the timing and bus specifications of the host interface are described in the host interface section. Finally, the IP9200 interface contains the circuitry that generates the bus signals that enable the DSP to communicate with image memory.

DSP

The DSP section is the smallest in chip count, simplest in control, but the most difficult to understand [10]. This part of the design includes, of course, the TMS320C25 and is shown in Figures 10 and A.3. The TI DSP has been designed to run at 40MHz; however, a 30MHz oscillator was used instead. All timing restrictions in the design were considered and made to run the DSP at full speed (40MHz) and have access to external RAM in no wait-states. This caused tight timing requirements on the RAM chips. A RAM chip was found (CY7C169-25) [11] that met all of the desired timing requirements, but

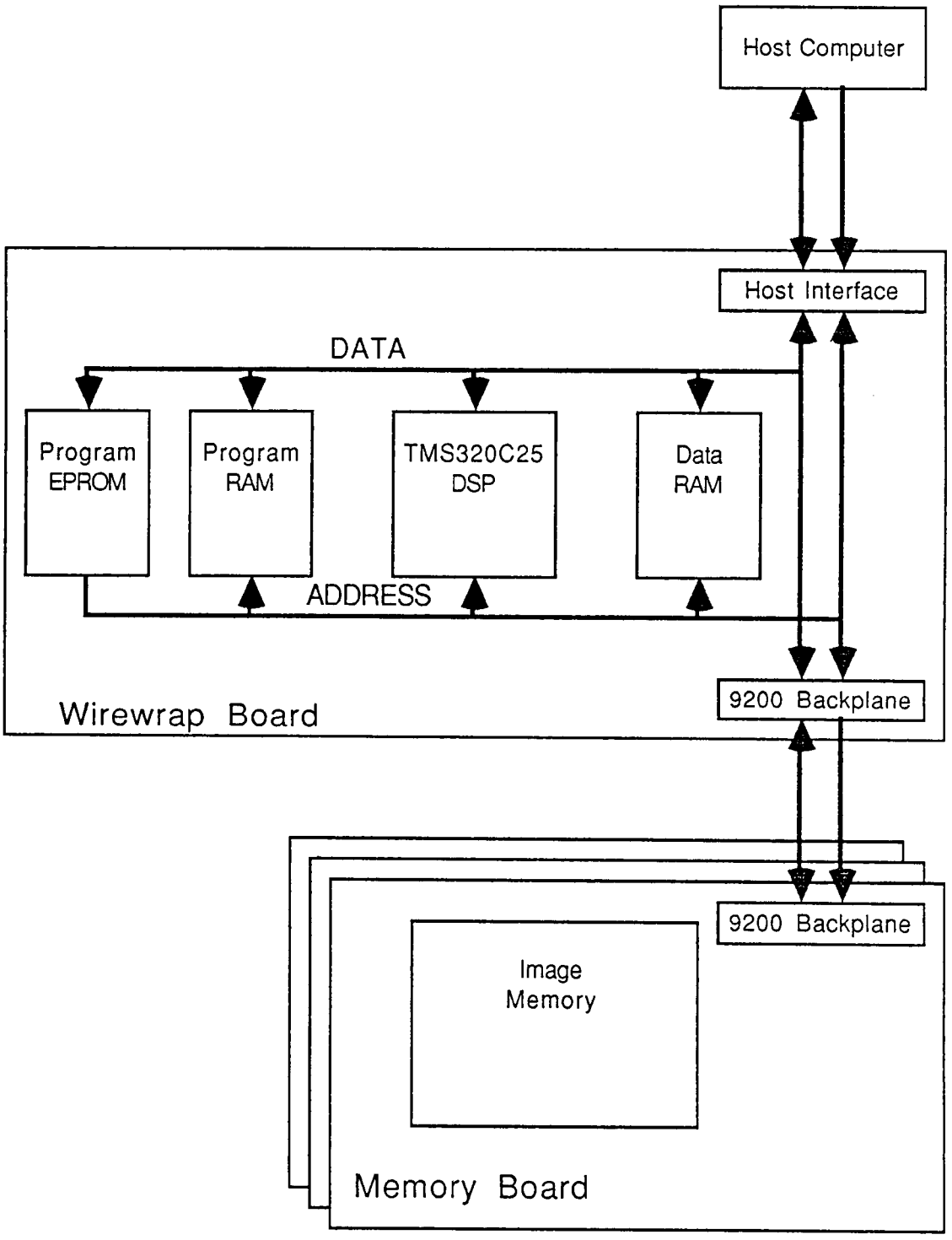


Figure 8. System Configuration

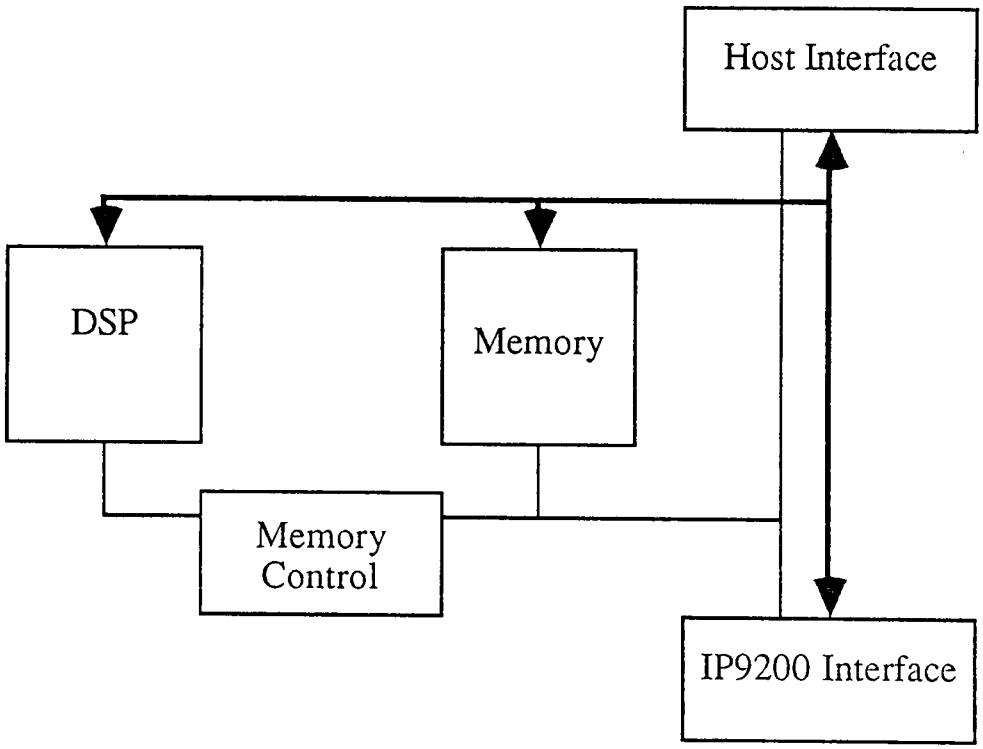


Figure 9. Block Diagram

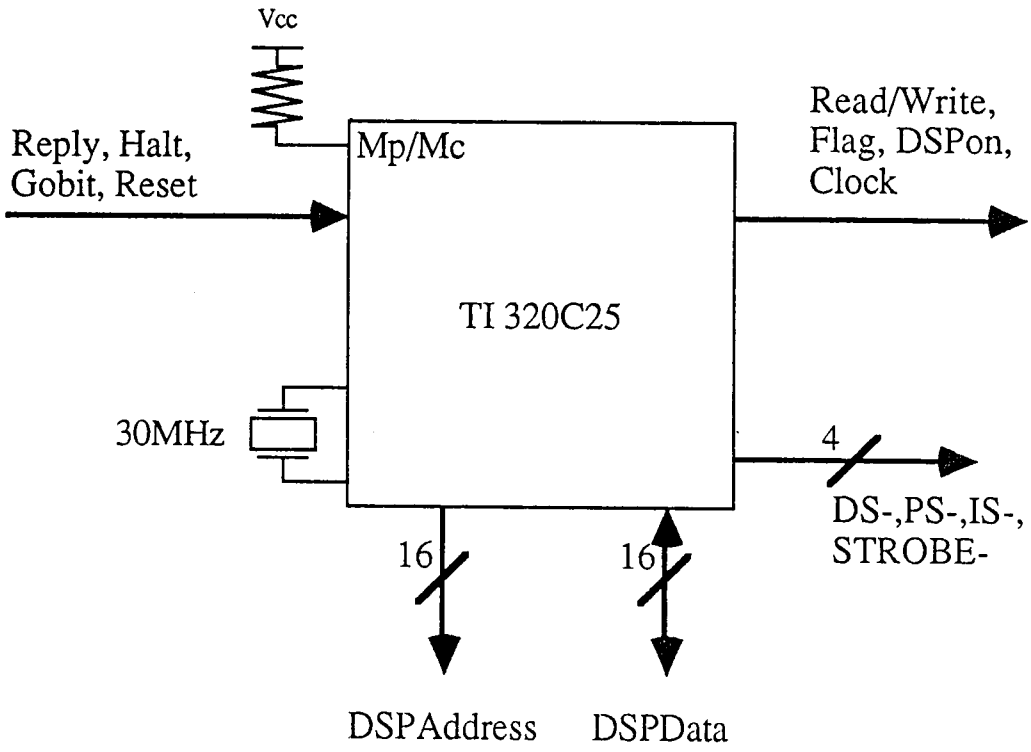


Figure 10. DSP Design Section

these special chips could not be obtained. A very close equivalent was found, but a slightly slower clock was needed to meet all timing restrictions.

All of the DSP address and data lines were used as well as all of the strobes. The TI DSP has three special strobes: Program Strobe (PS-), Data Strobe (DS-), and I/O Strobe (IS-). Combined together, these strobes separate address spaces and behave as address lines in set-up timing. Separate address spaces are provided for program, data, and I/O devices and gives the DSP over 128KW of address space instead of the usual 64KW that 16 address lines permit. Once these strobes (as well as the address lines) are stable, a universal or global timing strobe (STROBE-) becomes active. This signal is the one that initiates all of the accesses to external resources.

The RD/WR- line, as well as the CLK2 signal, are used to generate accesses to external memory. RD/WR- controls direction of data flow with respect to the DSP. The CLK2 output is used to generate a delayed clock from the DSP external access cycle that gives all of the DSP address, data, and strobes lines time for set-up.

The DSPON signal is connected to the DSP's HOLDA- output and specifies whether or not the DSP is running and has control of the external data bus connected to the backplane. When in control of the backplane, the DSP drives all address and data lines, all four strobes, and the RD/WR- outputs. If the host wanted to make an access that required control of the DSP bus, a request to halt the DSP is required. Upon halting, the DSP would release all of the mentioned control signals, go into a low-power state, and signal a halted condition by asserting HOLDA-. This signal is routed back to the host as DSPON.

The last output is the XF pin and is programmable under software control. When the design was completed, this pin was not used; so for debugging purposes, a test point was added. The sole purpose of this output is used to verify that the DSP is running without having to have a host present. This facilitated debugging of the DSP and its

accesses to EPROM, RAM and registers. Also when testing DSP RAM and an error occurred, the flag would toggle providing a trigger for the logic analyzer probe so that the access where the error occurred could be captured.

The remaining signals are all inputs to the DSP. The MP/MC control pin tells the DSP whether to be in the microprocessor mode or in the micro-controller mode. The main difference between these two options is where the DSP goes to get start-up information. In the micro-controller mode, the DSP maps the 4KW of internally masked ROM at 0x0000. This is used for standalone DSP applications. Since the boot device is the program EPROM, this pin is pulled high.

DSPRPL is connected to the READY input pin. If all accesses to external devices are no wait-states, this pin would be pulled high, signalling that all of the accesses are ready as soon as the DSP requests them. However, because of the slow EPROM and image memory access, this input must be conditioned upon what type of access is made. This is generated in the memory control section.

The RESET- does exactly as expected and resets the DSP to a known state (the DSP has no power-up reset capability). Upon resetting, the DSP restarts running at 0x0000, and resets all counters, pointers, and interrupts. This design included a special software power-up init that pre-loaded some of the parameters as they are usually set to aid in debugging of the software. A reset can be generated from several sources in the host control section and must be 1ms in duration.

DSPHALT- is connected to the HOLD- input pin. As discussed earlier, this is used when the host requires access to the DSP bus.

Lastly, the GOBIT is used to signal the DSP to start execution and is contained in the host control register. Connected to the BIOZ input of the DSP, the GOBIT signal is used to do conditional branching under software control by the DSP. This is done to make

polling of the control register as simple and as fast as possible. In this manner, the software can make conditional branches based upon the state of this input.

MEMORY CONTROLLER

The memory controller contains five PALs and some bus drivers. These PALs are the DSPIO, DSPENAB, DSPWAIT, DSPRDWR, and DSPADDR. These PALs control the enabling of the RAMs, direction of the buffers, and the timing of the replies.

The memory controller is shown in Figures 11 and A.4. This section is the heart of the design and required several iterations in order to meet all timing requirements. Basically two PALs control the selection of external devices (DSPIO and DSPENAB). These PALs generate control signals and clocks to the RAMs, EPROM, and registers used in the design. The DSPWAIT PAL determines the number of wait-states required to access EPROM and image memory and decides which mode of access is to be used as well as which set of address lines to enable. Generation of control signals needed to access image memory are made by the DSPRDWR PAL. All of these PALs interact with one another in order to gain access to external devices. Because of the fast timing of the DSP, most of these PALs were required to be B-type (fast) PALs [12].

The DSPIO PAL controls the resources in the I/O map of the DSP. This PAL generates four clocks as well as two control signals. Clocks generated are those to the DSP control, Y-address, Extra Address, and Address Registers. The DSP control register tells what type of access to make to the image memory by specifying which byte (or both) that is to be written and also which address mode is to be used. Other registers are used to hold various address values and page-mode values and are enabled during different addressing modes to the image memory and are discussed later.

The two control signals generated by the DSPIO PAL are a special set of control

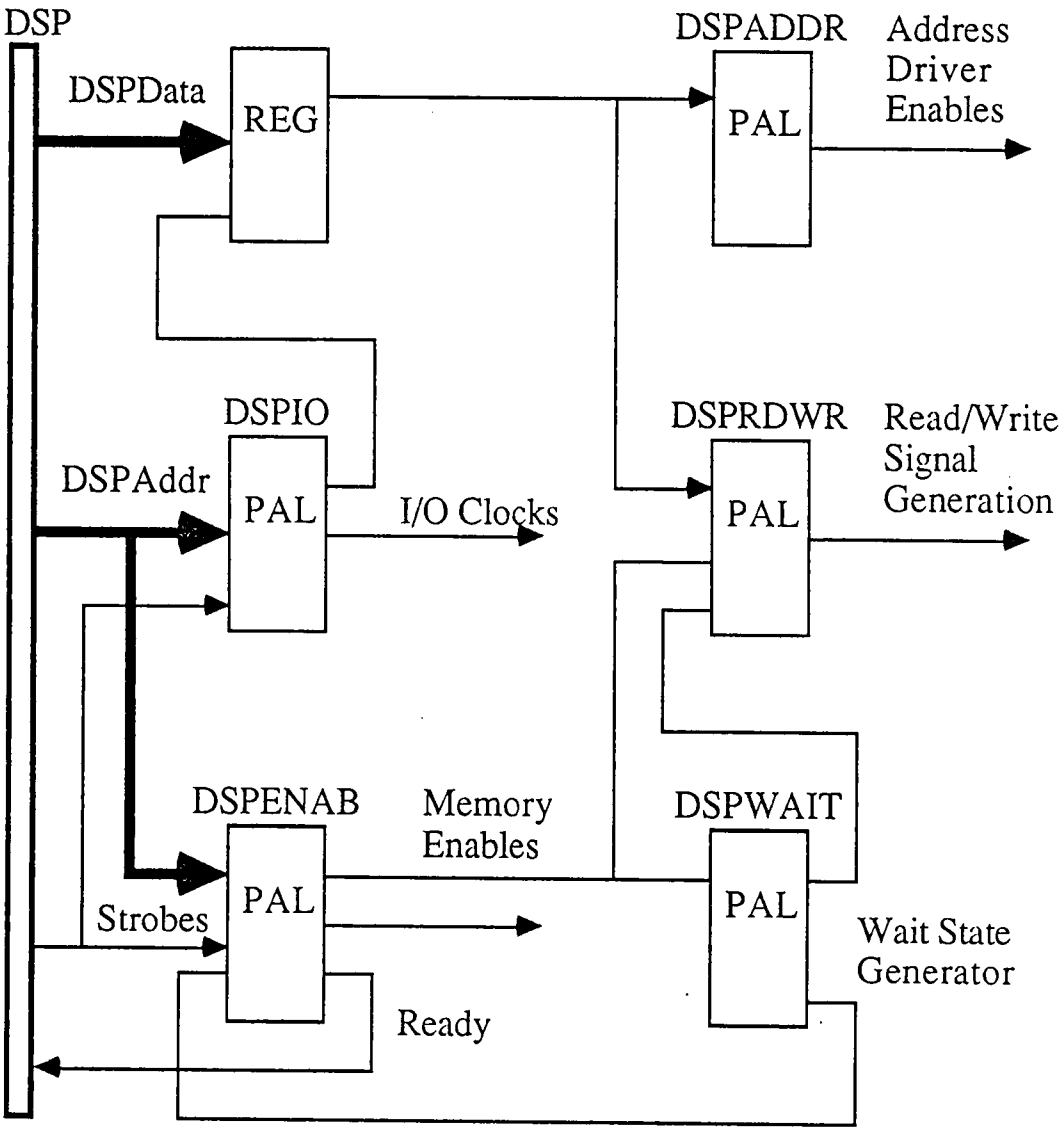


Figure 11. Memory Controller

lines that are generally used only after a DSP subroutine has been run. One signal clears the GOBIT in the host control register preventing the DSP from continuously running the same subroutine. The host sets the GOBIT to tell the DSP to start execution and the DSP clears the signal to indicate that program execution has completed. Another control line is used to reset and force the DSP into the halted state and is provided for two specific reasons. First is to provide for a software reset to all of the DSP functions and, secondly, to act as another way of signalling the host that the DSP is done. After each subroutine, the DSP will halt itself providing another way of signalling the host that execution of the desired subroutine has terminated. This is the preferred method of communication between the host and the DSP as only a single line needs to be monitored. All signals of this PAL are initiated on the active edge of STROBE-. All addresses are properly decoded, as well as IS- in time to ensure no glitches on any of the clocks.

The DSPENAB PAL decodes accesses to program and data space with the address map is shown in Figure 12. Chip enables to the RAMs are generated depending upon which bank is selected. Two enables for other slower devices are also generated. These are for EPROM and image memory. Both signals are passed to the DSPWAIT PAL for wait-state generation.

The DSPENAB PAL also generates the READY signal or reply back to the DSP. When no wait-state RAMs or any I/O locations are accessed, the READY signal immediately becomes active. PROMRDY- becomes active to terminate an EPROM cycle, and /92REPLY signals the end of a IP9200 access. The DSPENAB PAL uses these signals to help generate READY for long wait-state devices.

One interesting problem encountered with the DSP occurred with this PAL. The problem was that image memory replies tend to be quite long (>100ns). While this is not exceedingly long for normal microprocessors, for a processor running at 10MHz, this

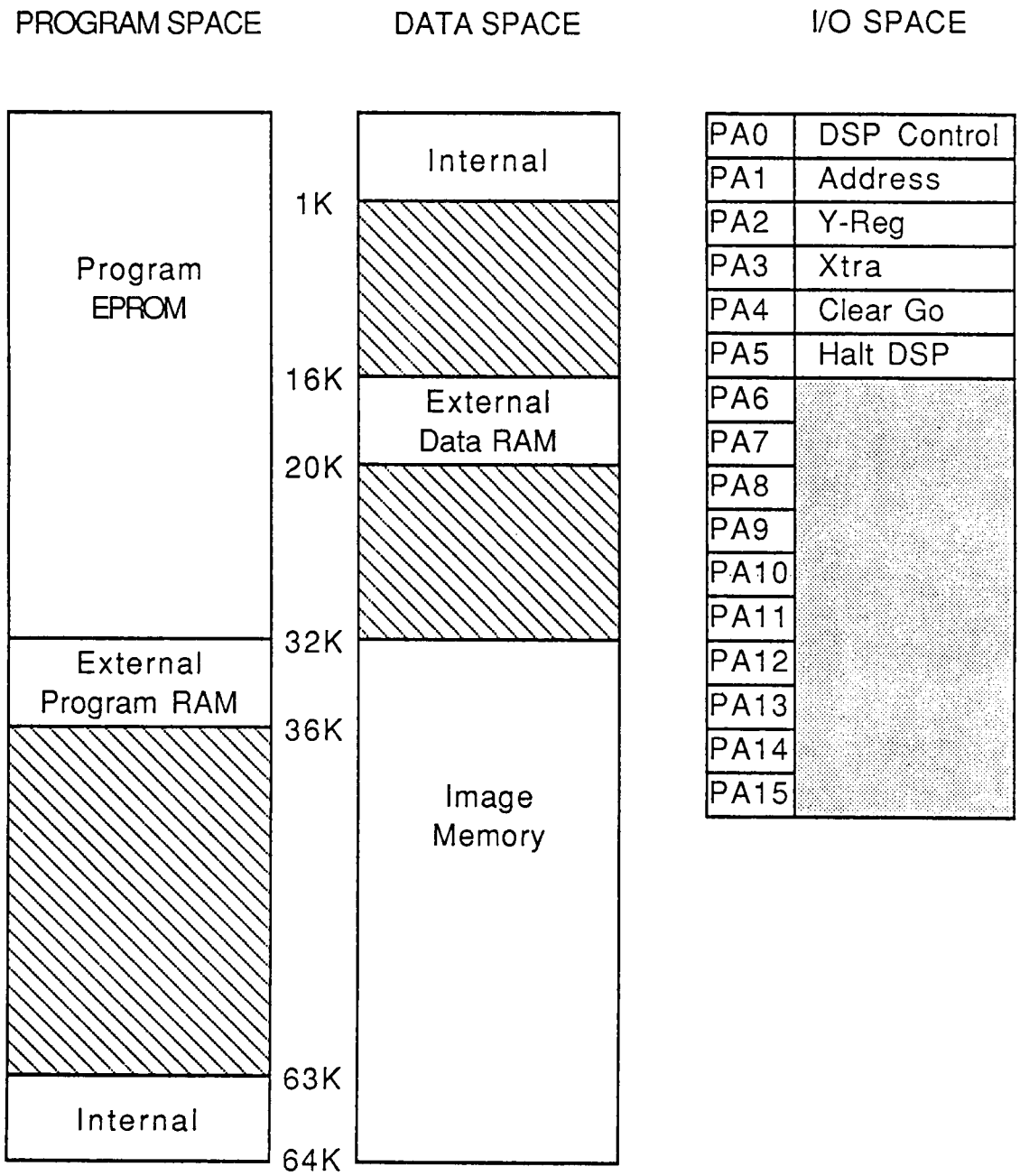


Figure 12. Memory Map

reply takes one full clock cycle. The problem surfaced when the DSP was making rapid and successive accesses to image memory. Replies from one access would still be active at the beginning of the cycle for the next access. One solution would be to delay the reply back to the DSP, but this would degrade system performance. A final solution was to insert a one-shot after each access to image memory. This one-shot would allow the reply to be generated as fast as possible, but would inhibit successive accesses from occurring too quickly together. Accesses to DSP RAM are allowed to take place while the image memory is still replying.

This leads to the DSPWAIT PAL with the primary function of generating the necessary wait-states for access to EPROM. Additionally, the DSPWAIT PAL generates the trigger for a one-shot and a required control signal needed for image memory access. Image memory on an IP9200 bus requires a longer set-up time of the address than the DSP resources. So, the image enable strobe (generated by from DSPENAB) is delayed a half-cycle, giving the image memory plenty of set-up time.

While the DSP is controlling the IP9200 bus, the DSPRDWR PAL generates the signals required for access to the backplane and image memory. If the host requires access to the DSP resources, the DSPRDWR PAL changes the direction of the control signals and generates the required timing signals necessary to access resources of the DSP. This device provides for the DSP to behave as any other board in the system and to have no special accesses required.

When the DSP is in control, backplane signals generated to the backplane are READ, WRITELO, and WRITEHI. The control register tells which byte, or both, is to be written and the READ signal controls direction of access on the backplane. These signals are activated by STROBE- and are held active until a cycle has completed.

Under host control, the inputs are READ, WRITELO, and WRITEHI. All three control direction of data flow to and from the DSP resources which are controlled by RDWR-. These three controls also generate the STROBE- that is used in the DSP decoder structure.

The DSPADDR PAL controls all of the address and data buffers to and from the DSP generating enables necessary to create the different address modes described later. All of these signals are used to connect the DSP data and address busses to intermediate busses (D0-D15 & A0-A15). Another set of drivers are used to drive the signals to the backplane (controlled in the host interface section). This was done to maintain compatibility of the DSP with other boards in the system as well as to allow the DSP to run code and crunch on data in fast RAM while at the same time, the host is performing operations on the IP9200 backplane. A type, though primitive, of a parallel architecture is now conceived. Another reason this was done to facilitate the actual movement of the DSP section onto a 1024x1024 memory board without changing the design to a great extent. (The DSP, memory, memory controller and IP9200 interface sections would remain almost untouched).

MEMORY

The memory section is shown in Figures 13 and A.5-A.7. Memory consists of 4KW of data RAM, 4KW of program RAM and 32KW of program EPROM. All memory is 16-bits wide. Both program and data RAMs are identical in operation and control and were selected to provide no wait states. All of the address lines, as well as the write enable, is set-up in the first part of the access cycle. When all of these lines are stable, the chip enable becomes active. This requires RAMs that performed write cycles on either chip select or write enable strobes to be used. Also the DSP reads in the data on the next phase of the 40MHz cycle. Calculating timing requirements meant that the RAMs needed to have

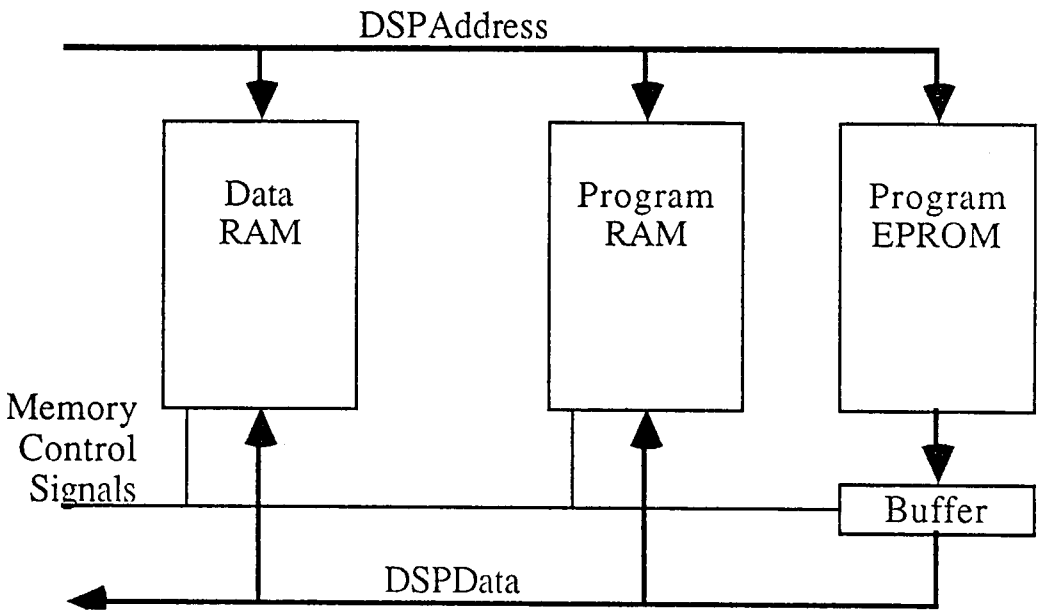


Figure 13. Memory Organization

less than 15ns access times from chip select. This is the specification that was not met by the substitute RAMs. The substitute RAMs had 25ns delays from chip select which required a reduction in clock speed. An oscillator of 35MHz should work, but one of 30MHz was readily available and so was used.

The EPROMs also presented some design considerations. While they are slow in access, they are also slow in turning off. They are not particularly slow in regular designs, but this design required that they not be driving the bus as little as 15ns after their access and presented a problem. The solution was to have the EPROMs always enabled. As the addresses settle to the desired location, the EPROMs drive a set of bus drivers. These drivers are turned on and off depending upon whether or not the desired access is to an EPROM or not. These bus drivers can easily release the data bus in 15ns.

IP9200 BUS INTERFACE

The IP9200 bus interface is shown in Figures 14 and A.9. Address and data drivers to and from the backplane are shown. These are controlled by the DSPADDR PAL in their memory control section. Each of these drivers connect to their respective intermediate address and data buses. These intermediate busses are then in turn connected to the DSP address and data busses. In order to initiate page-mode or X-Y addressing, two additional registers are included that also drive the intermediate data bus. Also, the IP9200 backplane has 25 address lines that need driving. These are driven by an Extra address register.

All of theses address bus drivers need to be bi-directional because the host makes access to the DSP board through these bus drivers. This makes the DSP appear as any other board in the system, while also giving the DSP the capability to drive the address busses.

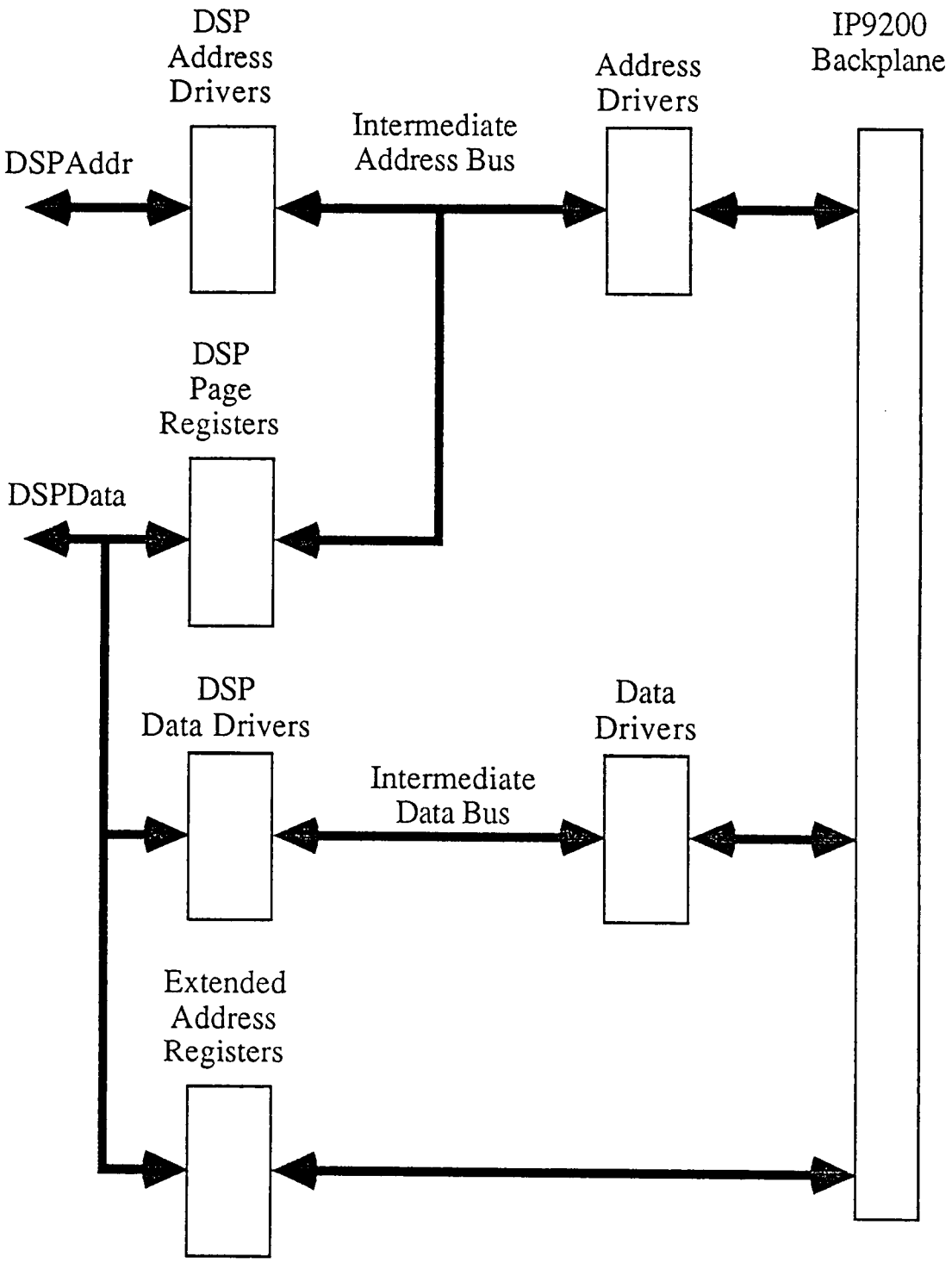


Figure 14. IP9200 Bus Interface

HOST BUS INTERFACE

As shown in Figures 15 and A.10-A.13, the main function of the host bus interface is to connect signals coming from the host through 50 pin ribbon cables to the IP9200 backplane. This section includes the host control register for control of the DSP, and three PALs used for control of the interface and board decoding of the DSP.

The host control register contains only three bits where the LSB is used to cause a reset of the DSP from the host. Halting of the DSP is accomplished by the second bit. The last bit signals the DSP to start execution and is called the GOBIT.

All three of these bits are fed into the DSPHALT PAL where they are conditioned before being passed to the DSP. RHALT- (halt from the register) is combined with a backplane halt. This enables the host to have two different methods of halting the DSP if desired. RRES- (reset form the register) is combined with backplane reset for exactly the same reason, and is also combined with RESETHALT- from the DSPIO PAL to give the DSP a way of resetting itself. This PAL also provides for read-back of the host control register to the host. (All other registers of the DSP are write-only).

The DSPSEL PAL decodes backplane addresses to determine if the host is requesting access to the DSP resources and generates the read and write signals to the host control register as well as general purpose board read and write control signals needed for another PAL. Additionally, this PAL generates a register reply that eventually gets combined with DSPRPL. This reply is used to drive the backplane for requests from the host.

The last PAL is the DSPMAC. This PAL's sole purpose is to determine direction of the address and data drivers of the host interface as well as the IP9200 interface.

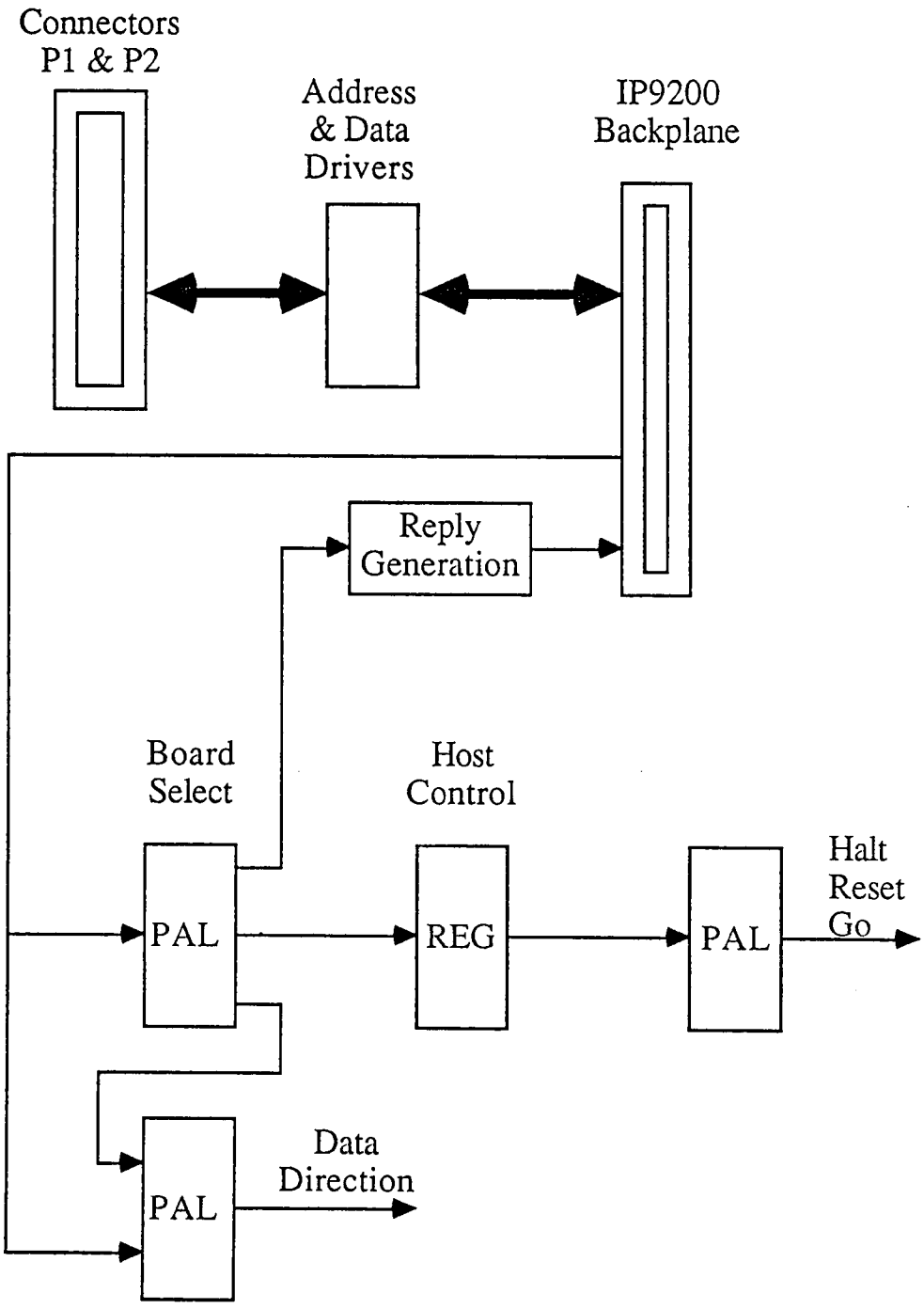


Figure 15. Host Bus Interface

ADDRESSING MODES

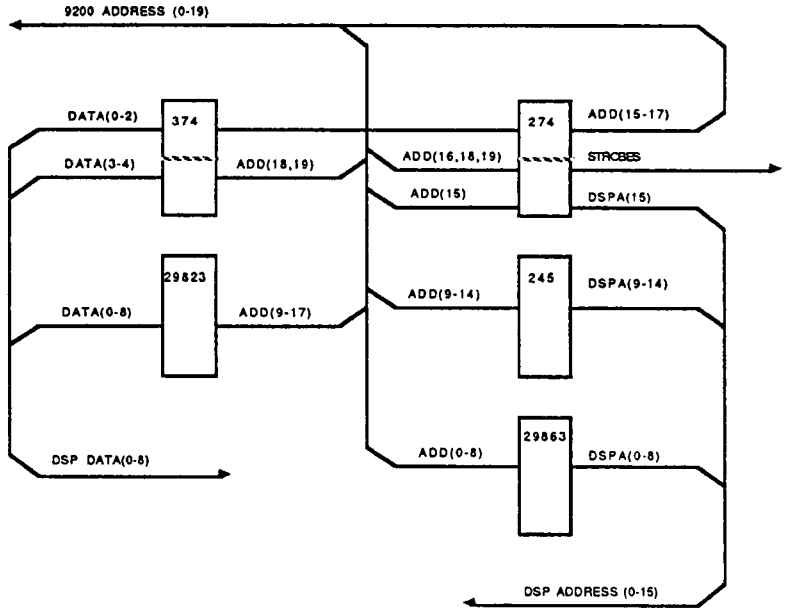
Essentially there are three modes of addressing for which allowances need to be made. The three modes include host addressing, Y-register addressing, and page-mode addressing. Host addressing is the action of the host taking over the DSP address bus in order to access DSP resources. Y-register addressing maps a 512x512 image memory into an X-Y column orientation. Page-mode addressing maps as much of image memory (32KW) as possible into one continuous DSP address space. All three addressing modes are shown in Figures 16 and A.8.

Figure 16.a shows the buffers in a non-functioning state. Figure 16.b shows the mechanism for the host driving the DSP address bus.

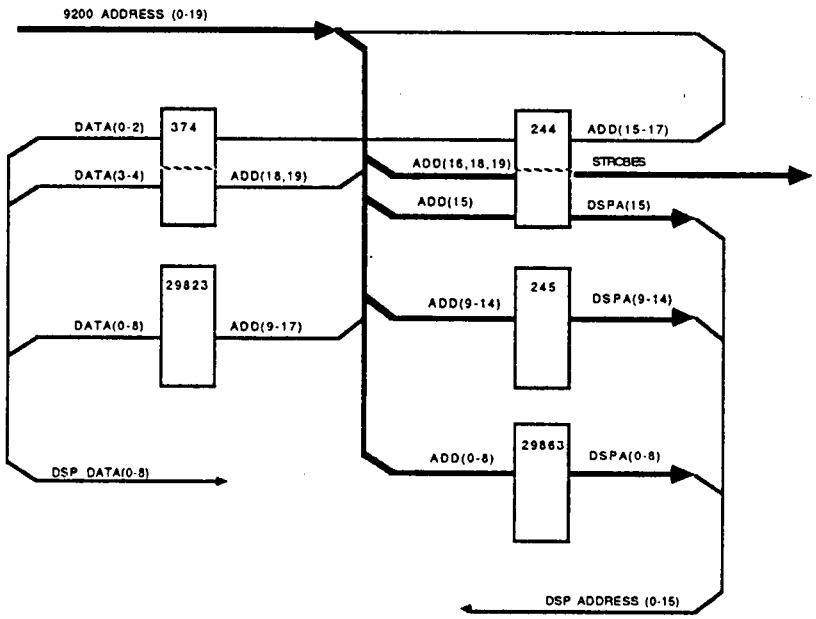
When using Y-register addressing, the DSP specifies which row (Y-address) to access in the image plane. An access to the desired X-location in the array is then made. This method is the one used most frequently of image processing subroutines because of the highly repetitive row/column operations. The address enables for Y-register addressing is shown in Figure 16.c.

Lastly is 32KW page-mode addressing. In this mode, each image memory is broken down into eight 32KW regions. A region requested for mapping is selected by writing the page-mode value into the Address register (lower bits) and is shown in Figure 16.d. The lower 15 address lines of the DSP then get passed on to the IP9200 bus for access to image memory.

In all of these accesses two other registers must be set-up. The upper bits of the address register contains which tile or memory plane that is to be accessed. This can be thought to be an extension of page-mode mapping. These two bits are used later to describe which 512x512 tile is requested when accessing a 1024x1024 image memory.

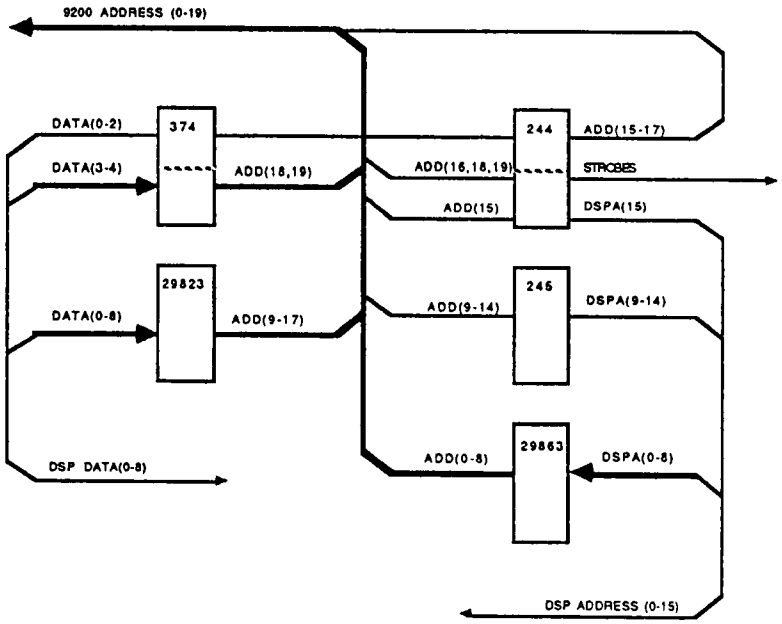


a. Address Drivers

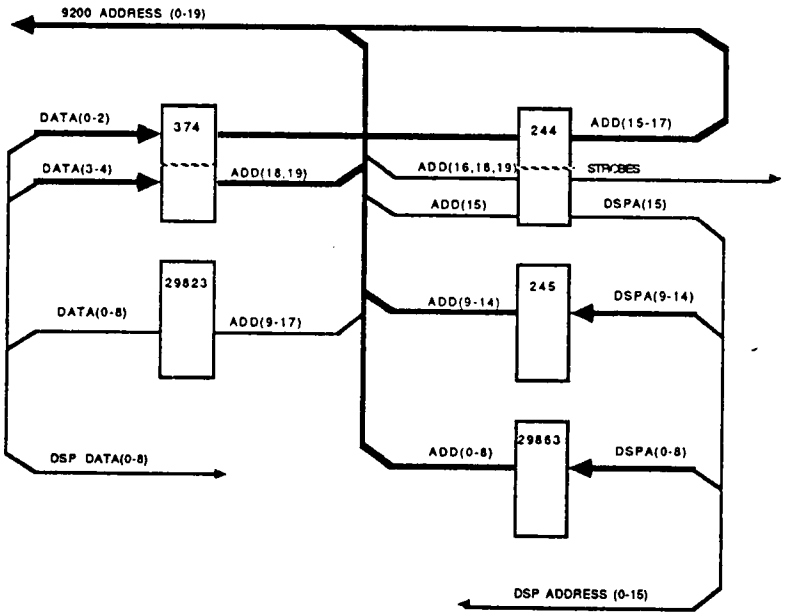


b. Host Direct Access

Figure 16. Addressing Modes



c. Y-Register Addressing



d. 32K Mapping

Figure 16. (continued)

The last register is needed only on the wire-wrap board and is used to drive the remaining addressing lines beyond the DSP's address space. These extended address bits are not needed when the DSP is put on a 1024x1024 image memory board.

CHAPTER V

SOFTWARE STRUCTURE

The interface between the DSP and the host computer is made as simple as possible so that a programmer from the host needs to remember as little as possible in order to get the DSP to operate. Figure 17 is a flowchart showing a representation of the inter-reaction protocol of communication between the DSP and the host computer.

Statements in squares on the left hand side of the flowchart represent the commands or steps that the host must do in order to run the DSP. The host halts the DSP and writes the desired function code in the proper place in DSP Data RAM as well as all needed parameters. Release of the DSP is accomplished by setting the GOBIT in the host control register. The host then can sit in a loop and wait for the DSP to finish processing. After the DSP is done, the host is free to repeat the procedure. At any time during the waiting process, the host may perform other functions needed (as long as no access to the IP9200 is attempted). Also, the host may halt the DSP at any time in order to gain access to the IP9200. This causes just a temporary shut-down of the DSP and is different from a reset or an interrupt (no service routine is performed). Once the halt condition is released, the DSP resumes execution of the code at the point that the halt occurred. If, while the DSP is running, a different routine would like to be called or the current one restarted, it is necessary to halt and reset the DSP and then proceed with the subroutine call as normal.

Statements on the right hand side in ovals represent the tasks that the DSP performs to execute the desired function. The DSP is in a idle-loop most of the time waiting for the GOBIT to be set in the host control register. A request by the host for a subroutine is signified by the setting of the GOBIT. The host has written the subroutine number as

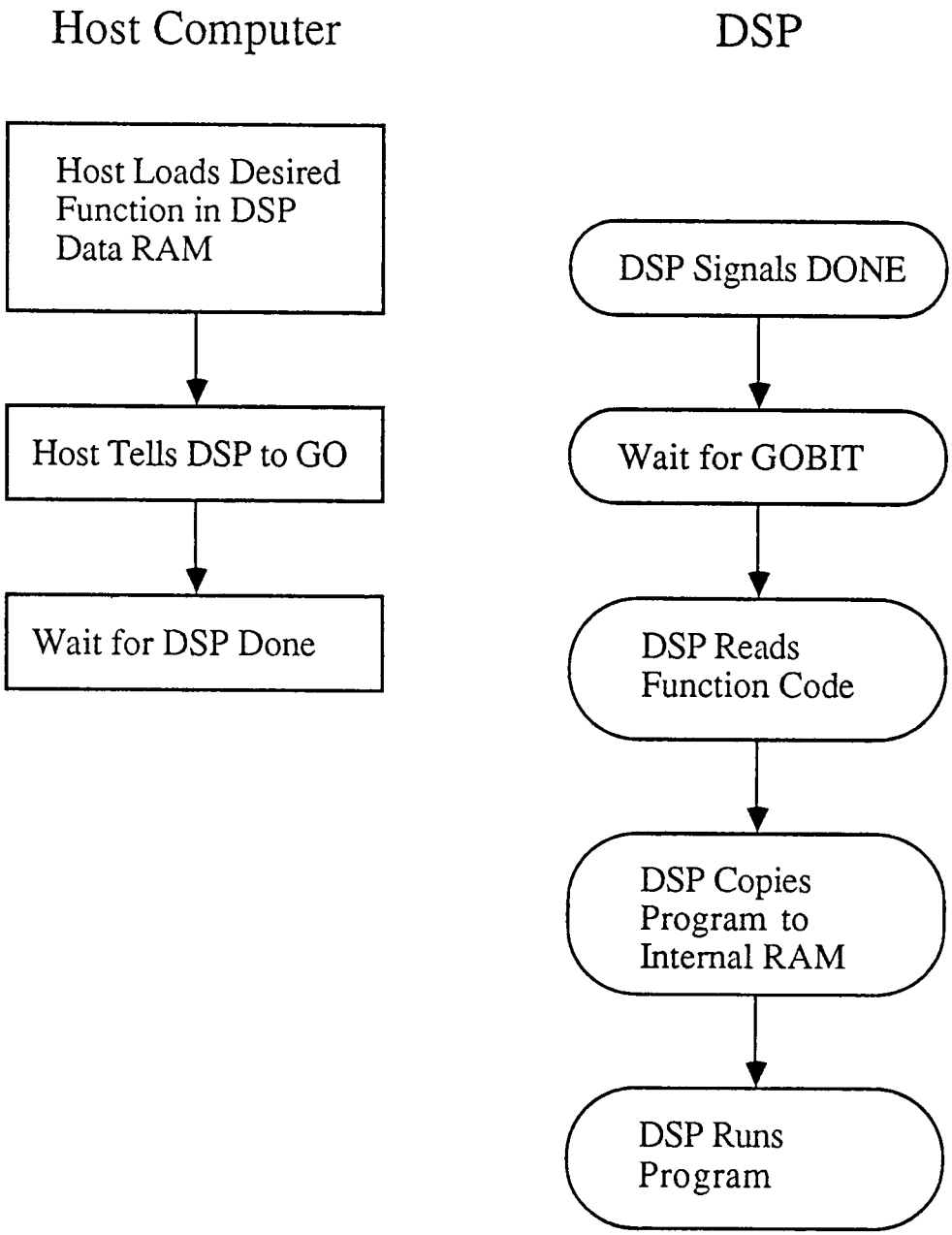


Figure 17. DSP / Host Communication Protocol

well as pertinent parameters into DSP Data RAM. The DSP takes this function code, looks up the address in the EPROM and copies that program from EPROM into internal RAM (for much more efficient operation). Next, the specified block of code is run by the DSP and signals the host when finished, becoming dormant until the next successive GOBIT.

SHELL OPERATING SYSTEM

The operating system of the DSP is a type of a shell program that is self-contained except for the program code as requested by the host. Figure 18 shows the basic flow of the SHELL Operating System. There are two locations in Data RAM that most of the communication between the DSP and host take place. The first location is the function code and basically tells the DSP the function that is to be performed and the other location is the address code and is the address of the routine to be run.

Most of the time, the DSP is in a dormant state. When released, the DSP reads the function code that has multiple meanings depending upon the value. If the function code is equal to zero, this signifies that a test routine is desired. The DSP EPROM comes with a complete set of test routines that test both the DSP and the image memory. If a test is to be run then the address code represents the test number to be run. Tests include all accesses to external DSP Ram and all different type of addressing to image memory.

If a test routine is not selected, then a decision is made as to whether or not to run from external RAM. If the function code is equal to one, this signifies a run from RAM. The address function code tells what location in RAM is the starting location of the routine to be run.

A subroutine function is the last option. If the function code is greater then 512, then this represents a bad function code and the DSP immediately returns. The DSP takes the function code and finds the address of the routine in a look-up table at the start of

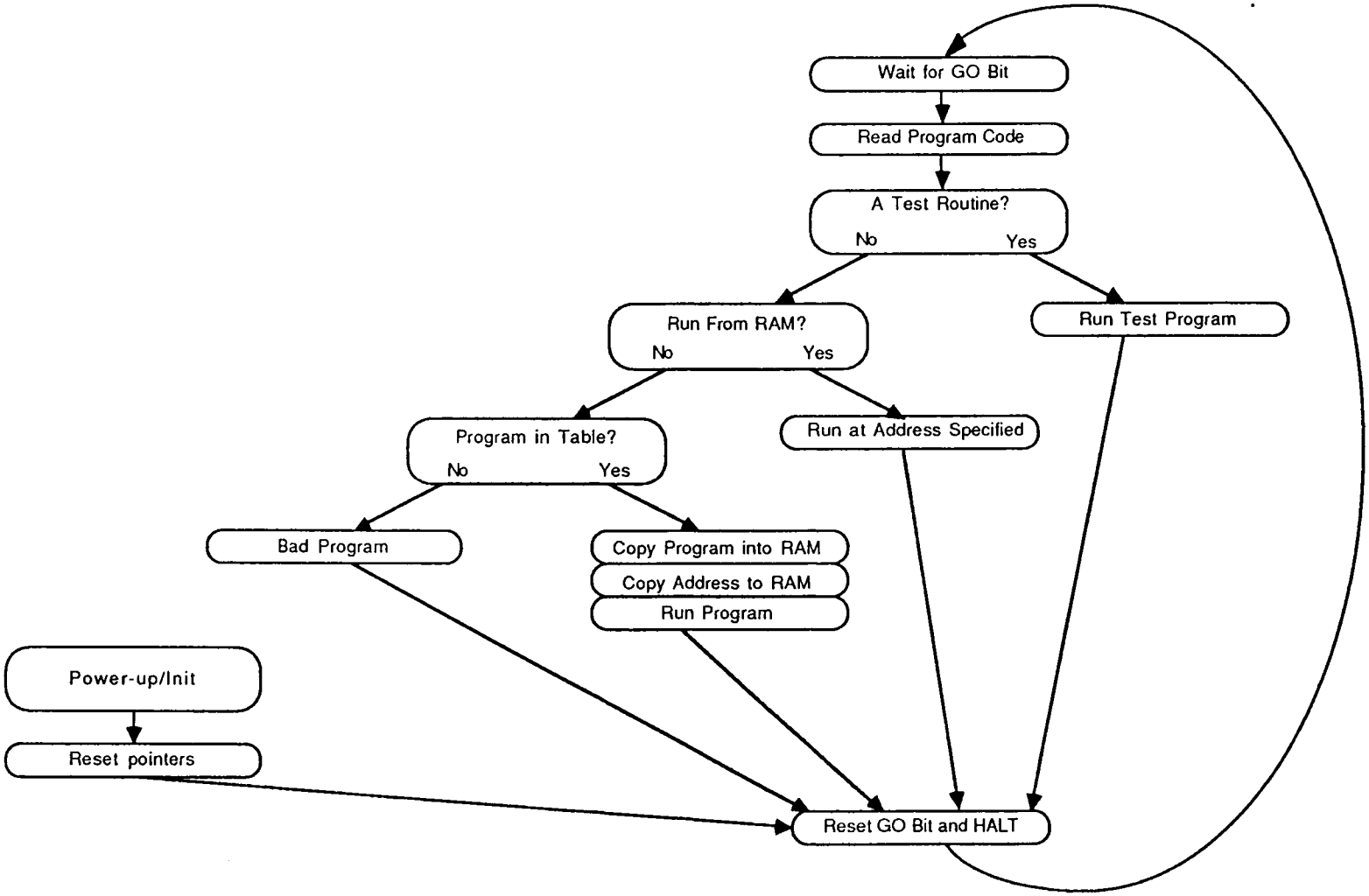


Figure 18. Shell Operating System

DSP EPROM. This look-up table allows for DSP EPROM routines to change in size and expandibility and all existing programs on the host never be changed. Software updates in the DSP code are completely transparent to the host. This location in the look-up table points to where in EPROM the desired subroutine is located. All subroutines have been linked to run in internal DSP RAM for greater speed, so the routine must first be copied into the DSP. The DSP then inserts the address of internal RAM into the address function. This allows the user to skip the 'copy' portion of the DSP for even faster execution (by writing a one into the function code - run from RAM - and making another DSP access request).

The desired program, whether it is a test routine, run from RAM, or a subroutine call, is executed. When the routine returns back to the SHELL, the DSP resets the GOBIT. This signifies to the host that the DSP is done. The DSP then waits for the GOBIT to become set again, signifying another request by the host. Located at the start of EPROM, the SHELL routine is automatically started on power-up or on a reset.

SAMPLE ROUTINES

The SHELL routine is the program executed on power-up, on reset, and in between image subroutines. During an initialization state, the OS sets up several default parameters to be used by subsequent routines. After the init, a jump is made back to the beginning of the program where the DSP is put into a tight loop. Inside the loop, the DSP is monitoring the GOBIT. As soon as the DSP is told to start, the address of the desired routine is loaded and execution is started.

All routines have several things in common. First, the image tile pointers of interest for each routine are stored in an internal location of the DSP. Before an access or a block of accesses to a particular image tile is desired, one is required only to load these two pointers

into PA0 and PA1 (DSP control and Address registers). These registers completely define the type access and location of the image (up to 16 boards). This prevents the DSP from recalculating addresses and pointers every time an access to a different tile is needed. Secondly, most subroutines access the image only once and don't operate on the image directly. Instead, blocks of the image are brought in and the DSP operates on external fast RAM. After the desired function is complete, the DSP then writes a block to the destination image as fast as possible.

The ADD routine, whose flowchart is shown in Figure 19, is a typical routine that accesses three image planes simultaneously. All three image plane pointers (source1, source2, and destination) are kept in the parameters section of internal DSP RAM. The routine also shows the standard 3-statement loop used to pull image memory into DSP RAM before execution begins (LAC, SAC, BANZ).

The 3CONV routine, shown in Figure 20, demonstrates an innovative way of reducing image access to a minimum. Normally in a 3x3 convolution, each pixel is accessed 9 times. The DSP accesses each pixel only once. As a start-up state the DSP loads the first two rows into fast RAM and records the location of these addressing of fast RAM into pointers. The final init state is the loading of the third row. An operation is performed on these three rows. Rows 1, 2, and 3 are stored in fast RAM. When processing the next row, row 1 is no longer needed. So, this vacated space in fast RAM (recorded earlier) is stuffed with the fourth row's data. Now rows 4, 2, and 3 are stored in memory. This is repeated using a recycling buffer until the entire image has been processed.

A convolution-type operation also poses on another problem. The ALU in the DSP is an integer-only ALU (non-floating point). However, coefficients used in convolutions are almost always fractions. A solution to this problem involved 'assuming' a decimal

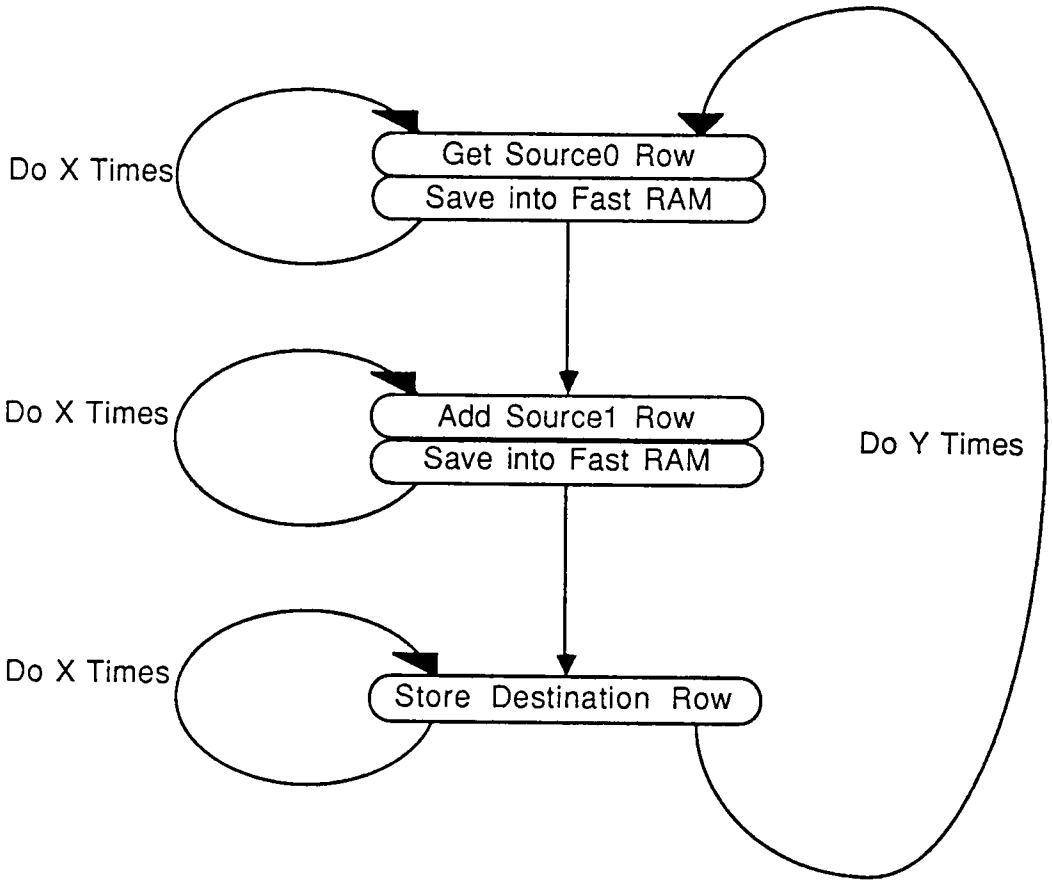


Figure 19. ADD Subroutine Flowchart

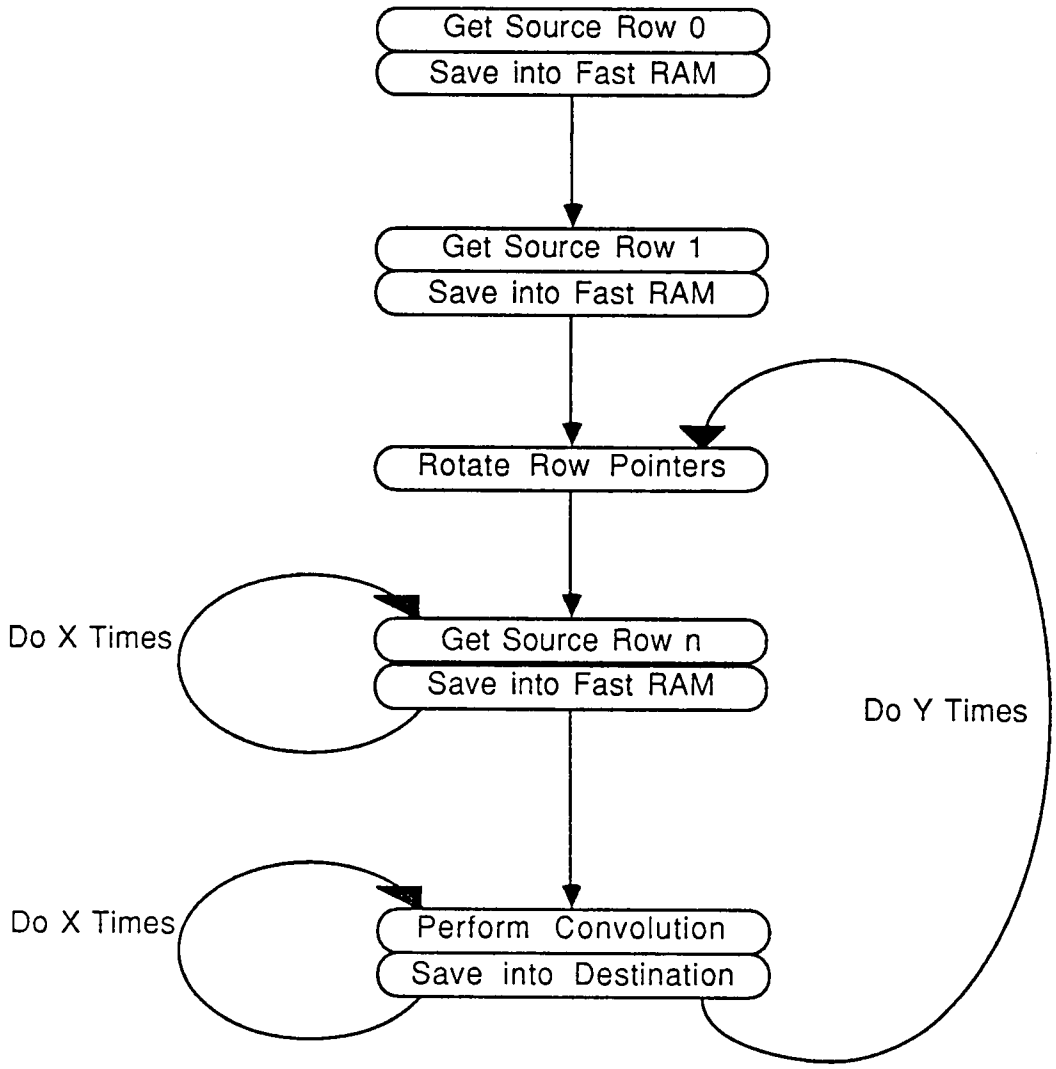


Figure 20. 3CONV Subroutine Flowchart

point to be at the start of each 8 or 16-bit number. Therefore, a gray-scale image instead of having a scale factor from 0 to 255 now has a scale from 0 to .999. If this system is maintained, then all calculations will result in a number less than one (shifting is sometimes required).

Another complex operation routine is warping. Warping is the process of mapping one image into another represented in Equations 1 and 2.

EQN 1: $x' = Ax^2 + By^2 + Cxy + Dx + Ey + F$

EQN 2: $y' = Gx^2 + Hy^2 + Ixy + Jx + Ky + L$

If these two equations were calculated for every pixel in a 512x512, then the time required would be >10 seconds. After evaluating these equations, some simple simplifications fall out. The X-equation can be reduced to Equation 3.

EQN 3: $x' = (Ax^2 + Dx) + (Cxy) + (By^2 + Ey + F)$

This rearranging makes several short-cuts apparent. First, the portion dependent only upon Y can be calculated once at the start of each row. Another application is that all results from every legal value of X is stored in a table. Later, when determining the value of X', only a simple look-up into this table is required instead of 6 multiplications and 3 additions.

Warps fall into two categories. The one described above is a nearest-neighbor warp, in that the nearest location to the derived equations is used. There is also an interpolative warp. This interpolates the value of the pixel based upon the the neighboring pixels. In the nearest-neighbor warp, one access to image memory is required to get the pixel value. In an interpolative warp, however, four accesses are required.

Any way of reducing this number of accesses to image memory would probably speed up the algorithms. This is done in 6WARPI, shown in Figure 21, which is a simple 6-parameter warp given by Equations 4 and 5.

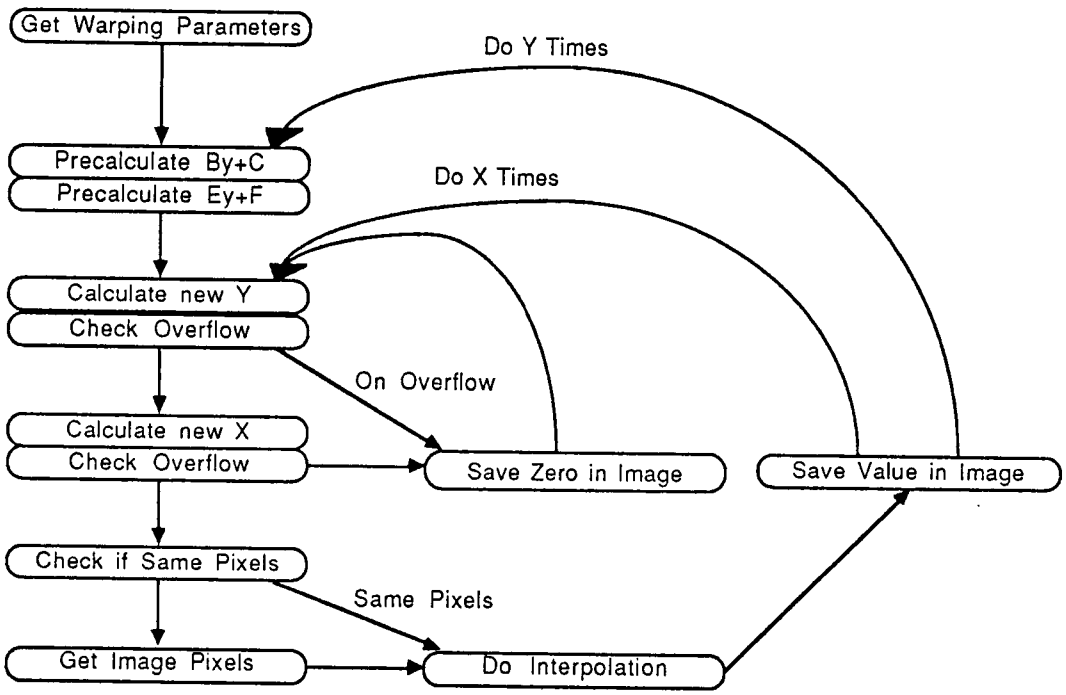


Figure 21. 6WARPI Subroutine Flowchart

EQN 4: $x' = Ax + By + C$

EQN 5: $y' = Dx + Ey + F$

The resultant pixel location is calculated and compared against the previous value. In warps, the same pixel may be accessed several times by subsequent equations (especially in zooms). If the pixel location is the same as the previous one, then no access to image memory is required at all. All of these examples are simple ways of using the DSP and short-cuts in trying to optimize code and accesses to image memory.

TEST ROUTINES

There are two types of testing routines. One to test hardware and the other to test software. The hardware routines are embedded into the DSP EPROM and test access to DSP peripherals and image memory. There is also a test routine that test operation of the DSP that merely toggles the external flag. This flag-bit also helped in the debugging of the DSP by providing a trigger. During a test routine, if an access was determined to be bad, the flag was toggled giving a trigger to examine the previous access on a logic analyzer.

The other tests are for software. When the DSP arrived, there was no emulator of any kind for debugging DSP code. Therefore, a breakpoint routine was written that enables a software programmer to test code and determine internal status of the DSP at any point during the running of his code.

CHAPTER VI

SUMMARY

All functions of the design worked as expected; however, due to an inability to get required parts, the DSP was only able to be operated at 30MHz. This decrease in speed, from the desired 40MHz, was because the RAMs that were used to implement the system were not completely compatible with those specified in the design. These substitute RAMs had a slower address set-up time than with those specified in the design. The system ran as expected to 30MHz and no attempt was made to run faster.

Most DSP routines turned out to run slower than anticipated. This is due to complications in the addressing scheme used to access image memory. Most of the idle time is to due to mapping the DSP back and forth between local RAM and image RAM. The addressing scheme was not the main culprit. TI designed the TMS320C25 to run fast with external RAM, and not with large banks of memory. Currently, the TMS320C25 has 16 address lines. If this were to increase to 20 address lines or more, this complication would disappear. (TIs next generation DSP - the TMS320C30 - has 40 address lines).

Even though the DSP was not as fast as desired, many speed improvements in the IP9200 were achieved. A CPU memory fill that used to take 2 seconds, now only takes 0.35 seconds. Rotation of images (6-parameter warp) used to take greater than 25 seconds, and now take less than 2 seconds. This increase in speed is particularly noticeable when operations are to be performed on RGB (color) images. With a DSP dedicated to each memory board, the speed increases threefold, because these images are processed in parallel.

Another advantage gained is that in future systems such as the NuVision system, board count will be reduced. With the addition of the new video interface board, a minimal image processor is now only two boards, which reduces system cost (from 5 boards) considerably.

ENHANCEMENTS

Even though the DSP performed well, there are several additions that would improve operation through increase in speed. One of the major disappointments with the system was its slow access to image memory. An access takes on the average 10 clock cycles to access image memory. While this does not degrade the system by a large percentage, the time delay is noticeable. One type design might be to integrate the DSP more closely with the memory than at present. Now the DSP is located on a different board from the memory. When an access to image memory is made, the cycles are similar to a microprocessor. A great deal of speed would be gained if the DSP were incorporated with the address generation logic of the memory board. In fact, if the DSP were coupled directly to the memory, random accesses would be achieved in half the current time, and sequential accesses could be speed up to an average of 2 clock cycles (which is faster than EPROM).

One enhancement to decrease access time to image memory could be conceived in some type of VRAM controller [13]. This controller would latch the data so as to immediately reply to the DSP. While the DSP is performing other functions, this controller is completing the access. Also, if image memory were arranged so that several pixels were read at the same time, this would also increase DSP speed and performance.

Another desired function would be an X auto-increment register. This register would increment a pointer in the X direction every time a certain access is made to the

memory. This would enable the DSP to forget about troublesome addressing upkeep and memory mapping. However, it would help only on sequential type operations such as logical operations and convolutions. This would not help operations such as warping very much at all.

Similar to the X auto-increment, a Y auto-increment mode would enable the Y-register to increment every time another access was made. This type register would only help in operations that needed column type sequential accesses. However, an FFT algorithm could use a y auto-increment register quite nicely. This way the transpose of the image would be done exactly the same way as the row transformation.

CONCLUSIONS

The TMS320C25 inclusion into the IP9200 has added speed improvements as well as increased processing power. Algorithms that were previously difficult to compute now are resident in the DSP EPROM and can be run at the same time that the controller CPU is performing system tasks. Parallel processing is performed providing algorithmic operations on RGB images simultaneously. With the future inclusion of the DSP on each memory board, processing power will be achieved that is not available in other image processors.

BIBLIOGRAPHY

[1] Tiefenthaler, C., (1987), "DSP chip runs Fourier transforms on nonstop input", *Electronic Products*, June 1, pp. 26-32.

[2] Mitsubishi Electric, (1987), IC Memories, Tokyo, pp.2.159 - 2.183

[3] Brooktree Corp., (1988), Brooktree Product Databook, San Diego

[4] Gonzalez, R. C. and P. Wintz, (1987), Digital Image Processing, Reading, MA, Addison-Wesley.

[5] Bursky, D., (1988), "Tackle real-time DSP tasks with CMOS chip set", *Electronic Design* , March 31, pp. 45-49.

[6] AT&T, (1986), WE DSP32 Digital Signal Processor Information Manual, Allentown, PA.

[7] Analog Devices, (1987), DSP Products Databook, Norwood, MA

[8] Motorola, Inc., (1986), DSP5600 Digital Signal Processor User's Manual

[9] Texas Instruments, (1986), TMS320C25 User's Guide, Dallas, TX

[10] Troullinos, George and Jon Bradley, (1987), Hardware Interfacing to the TMS320C25 Product Application, Dallas, TX, Texas Instruments INC.

[11] Cypress Semiconductor, (1988), CMOS Data Book, San Jose, CA

[12] Advanced Micro Devices, (1988), PAL Device Data Book, Sunnyvale, CA

[13] National Semiconductor, (1988), Advanced Peripherals Graphics Handbook, Santa Clara, CA

APPENDICES

APPENDIX I

SCHEMATICS

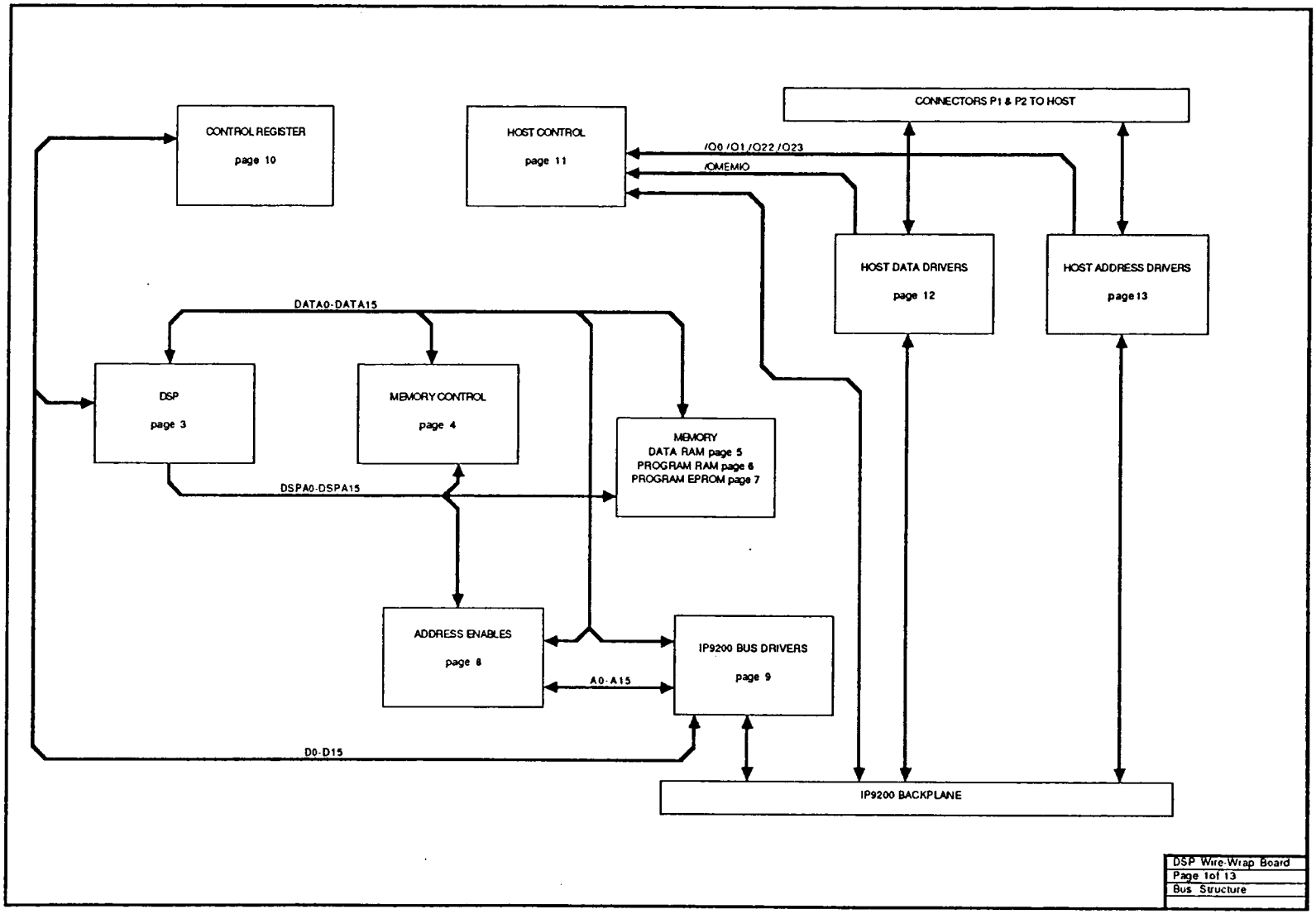


Figure A.1 DSP Bus Structure

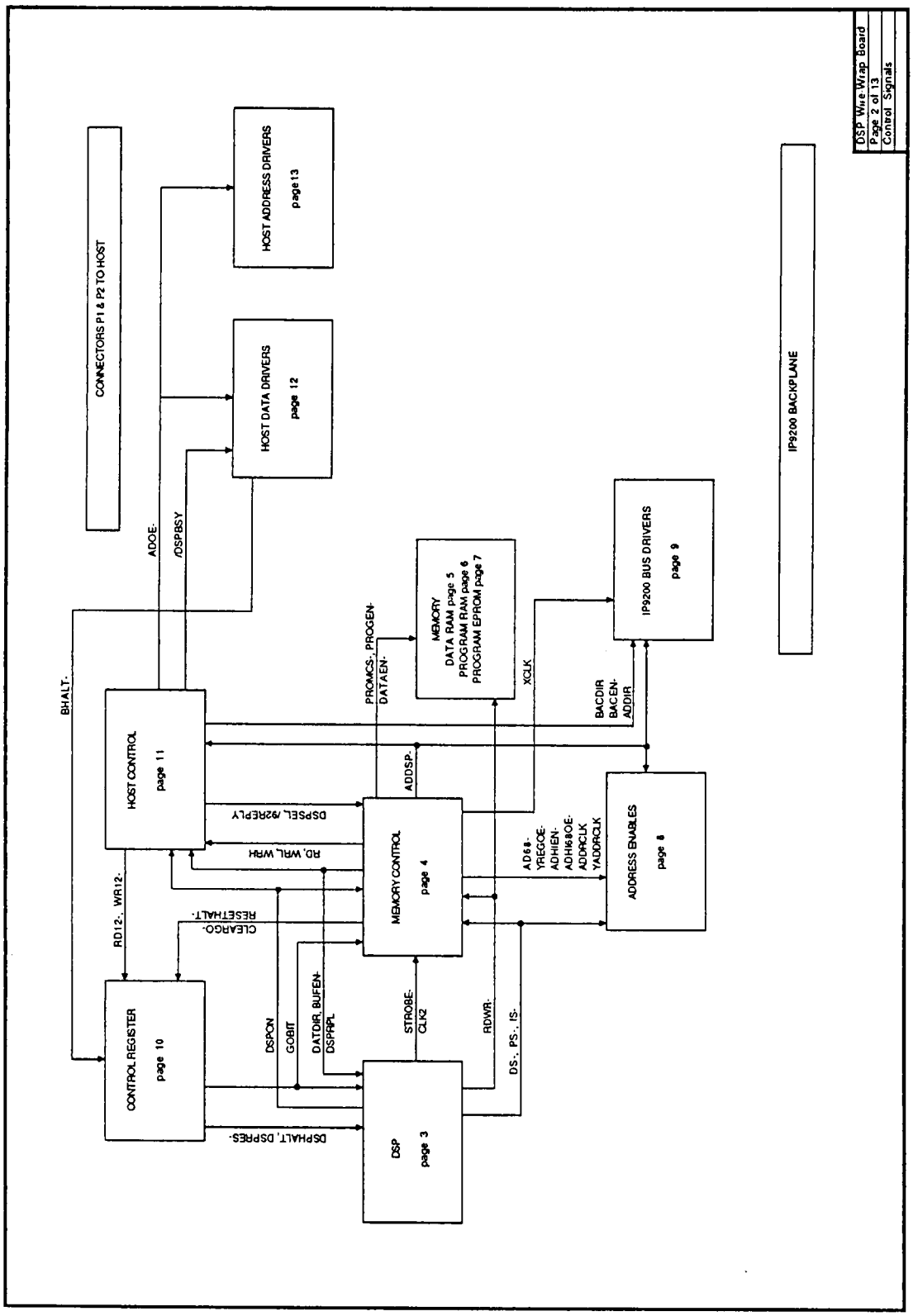


Figure A.2 DSP Control Signals

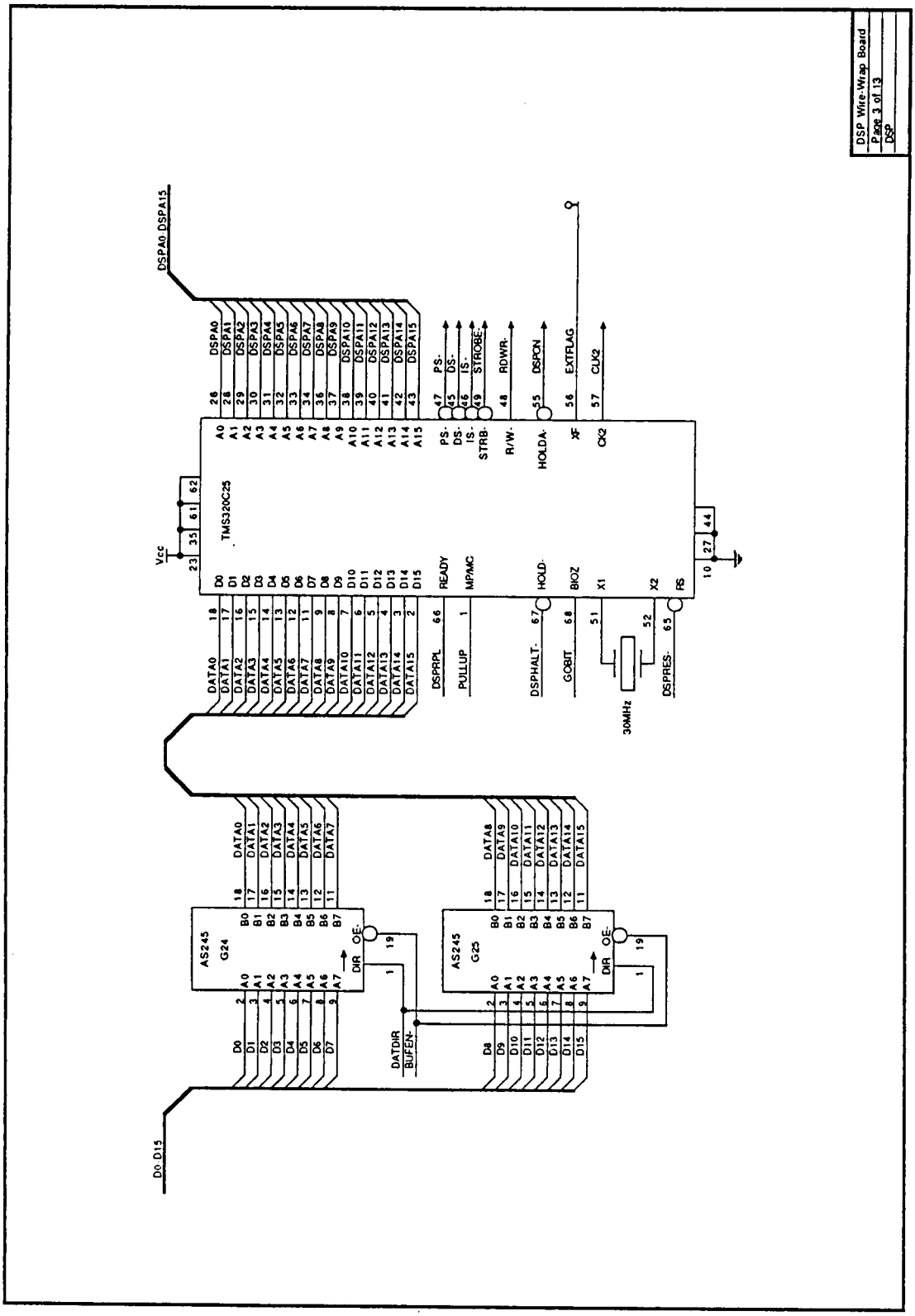


Figure A.3 DSP

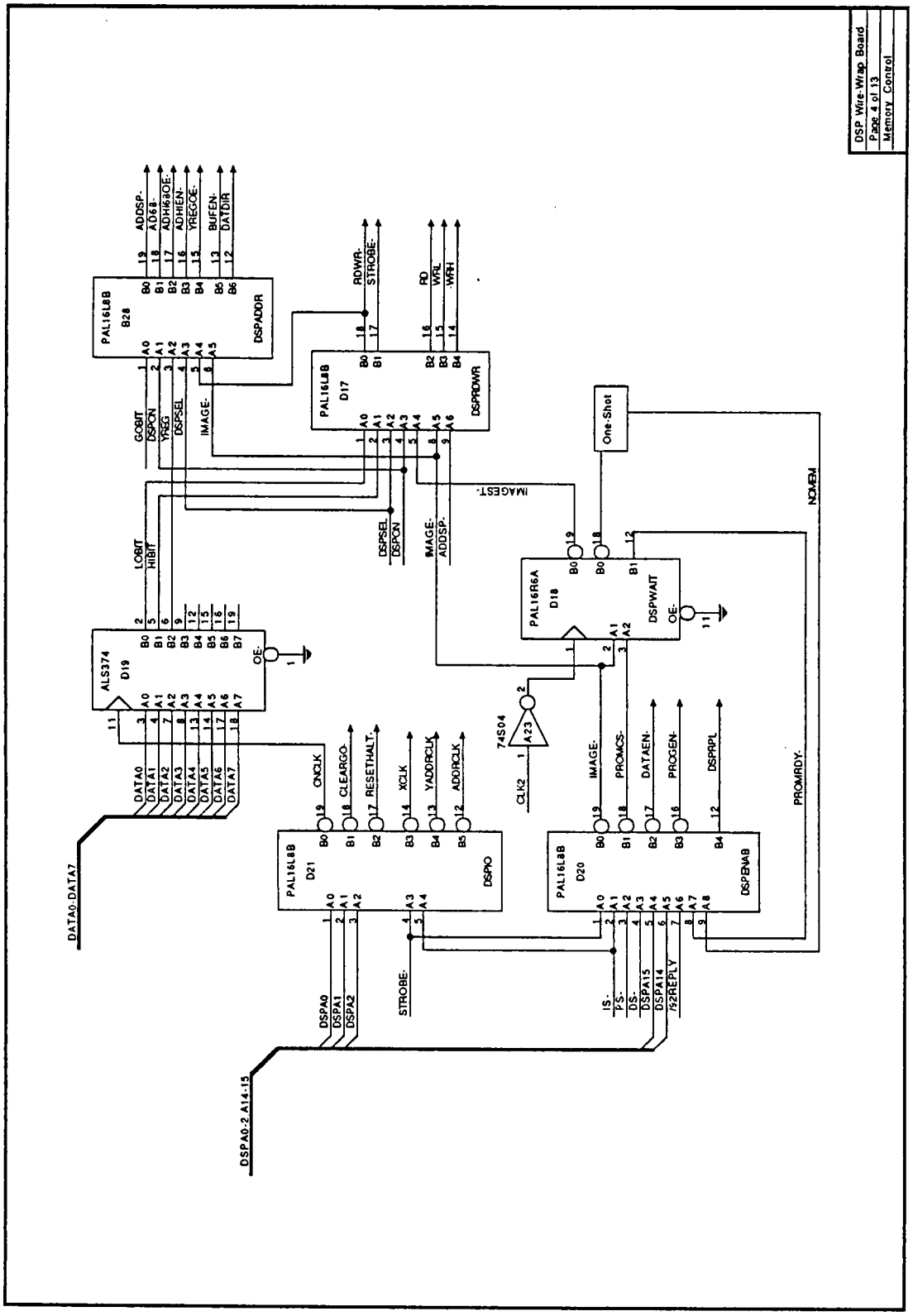


Figure A.4 Memory Control

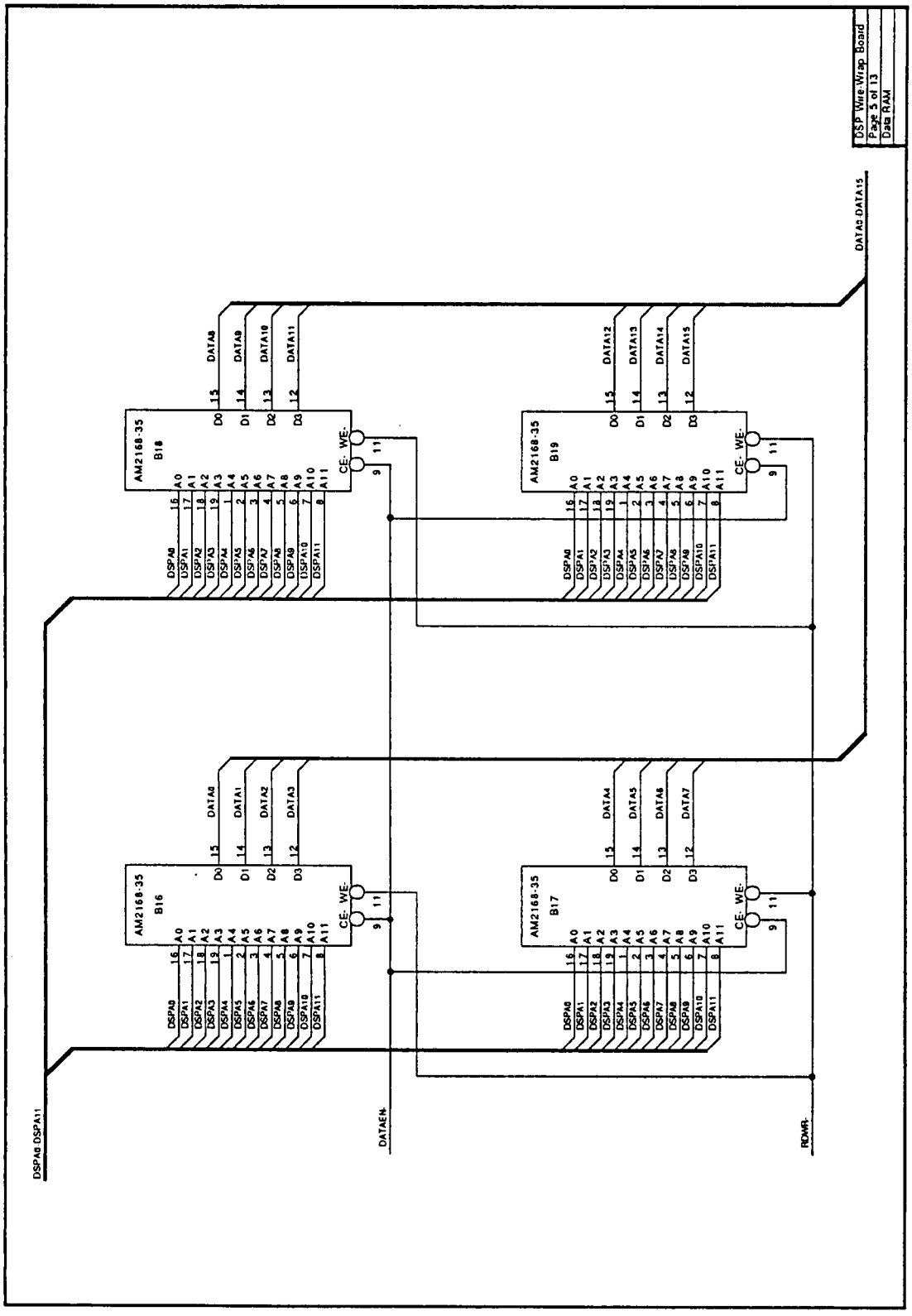


Figure A.5 DSP Data RAM

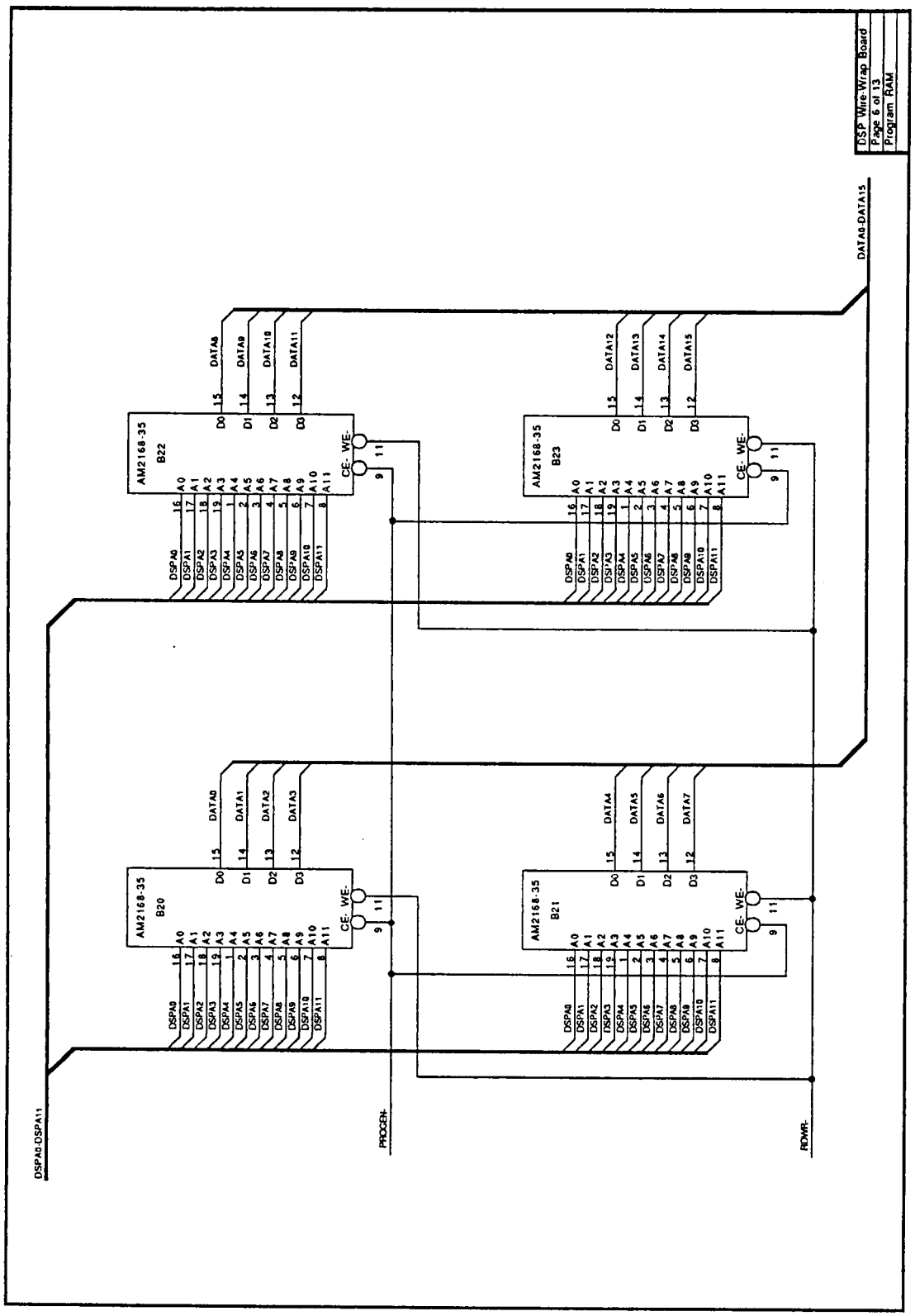


Figure A.6 DSP Program RAM

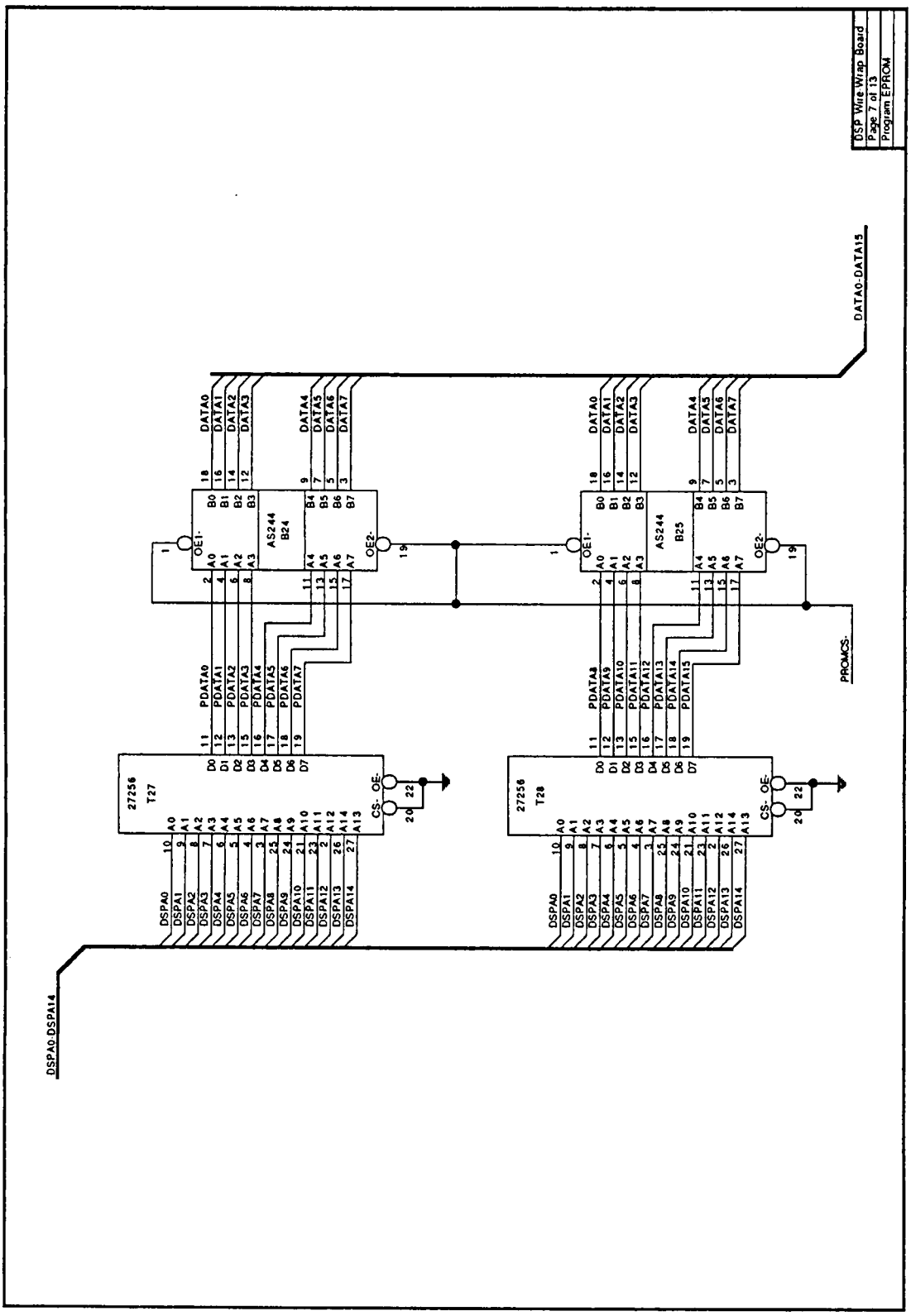


Figure A.7 DSP Program EPROM

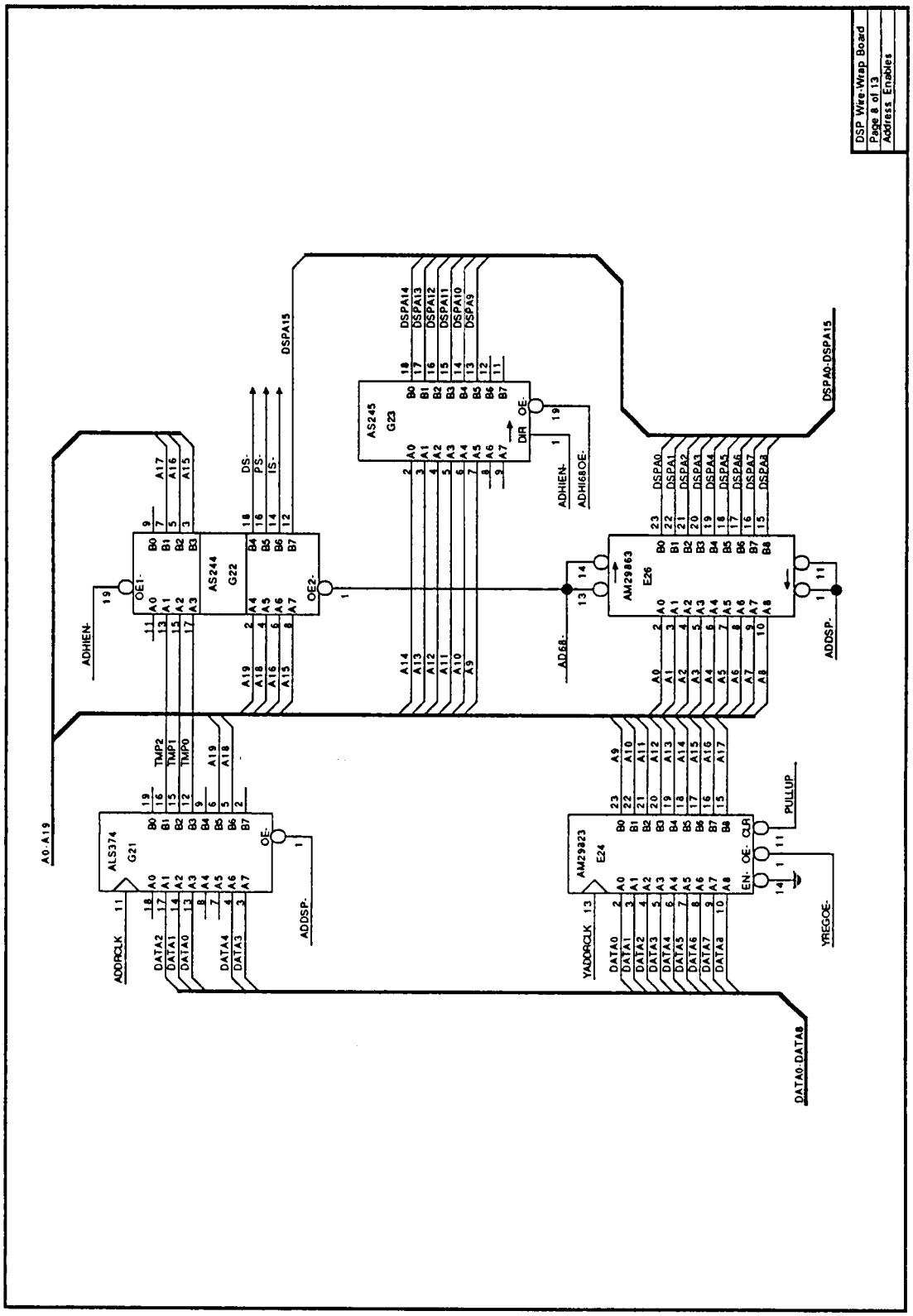


Figure A.8 Addressing Mux

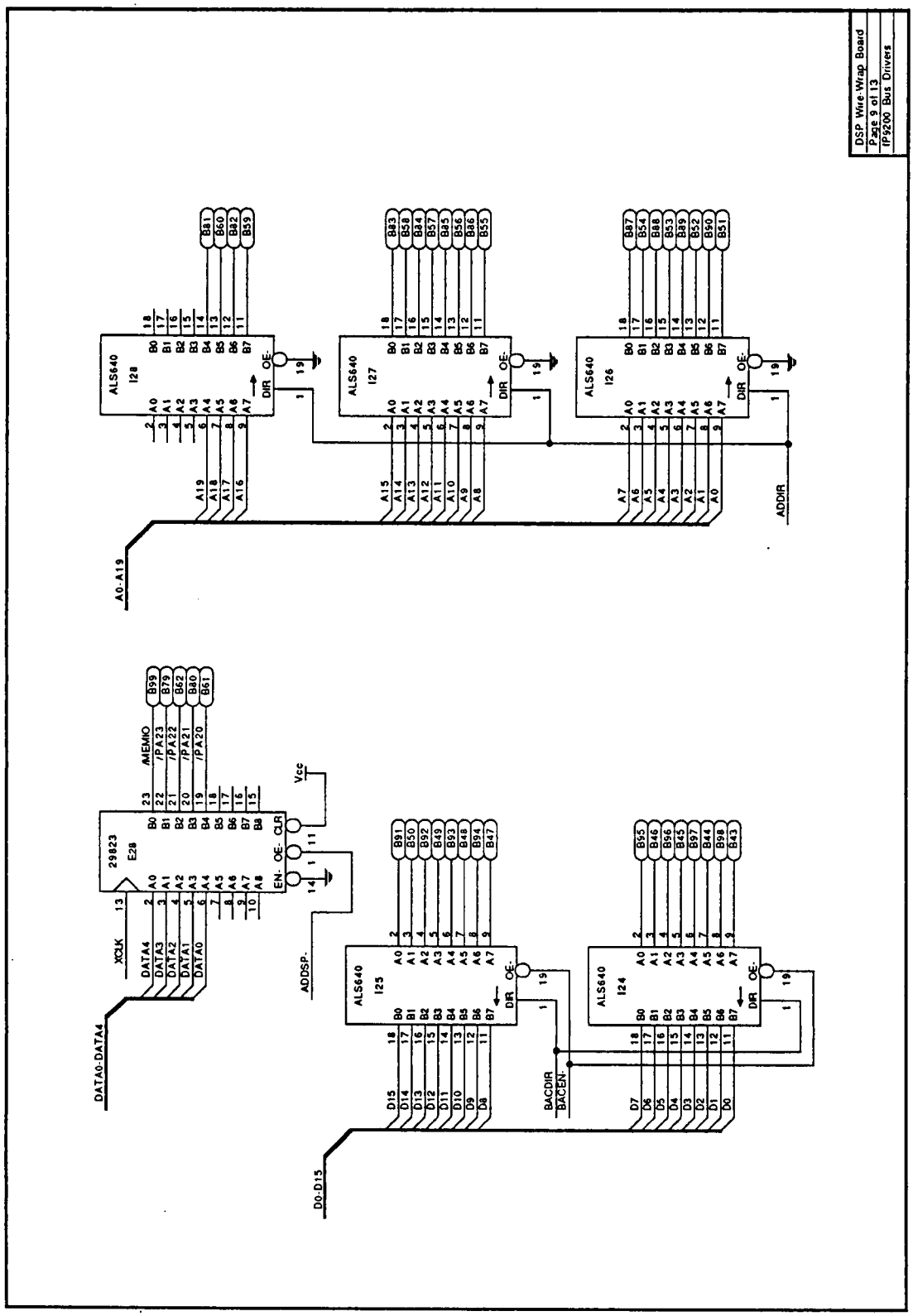


Figure A.9 IP9200 Bus Drivers

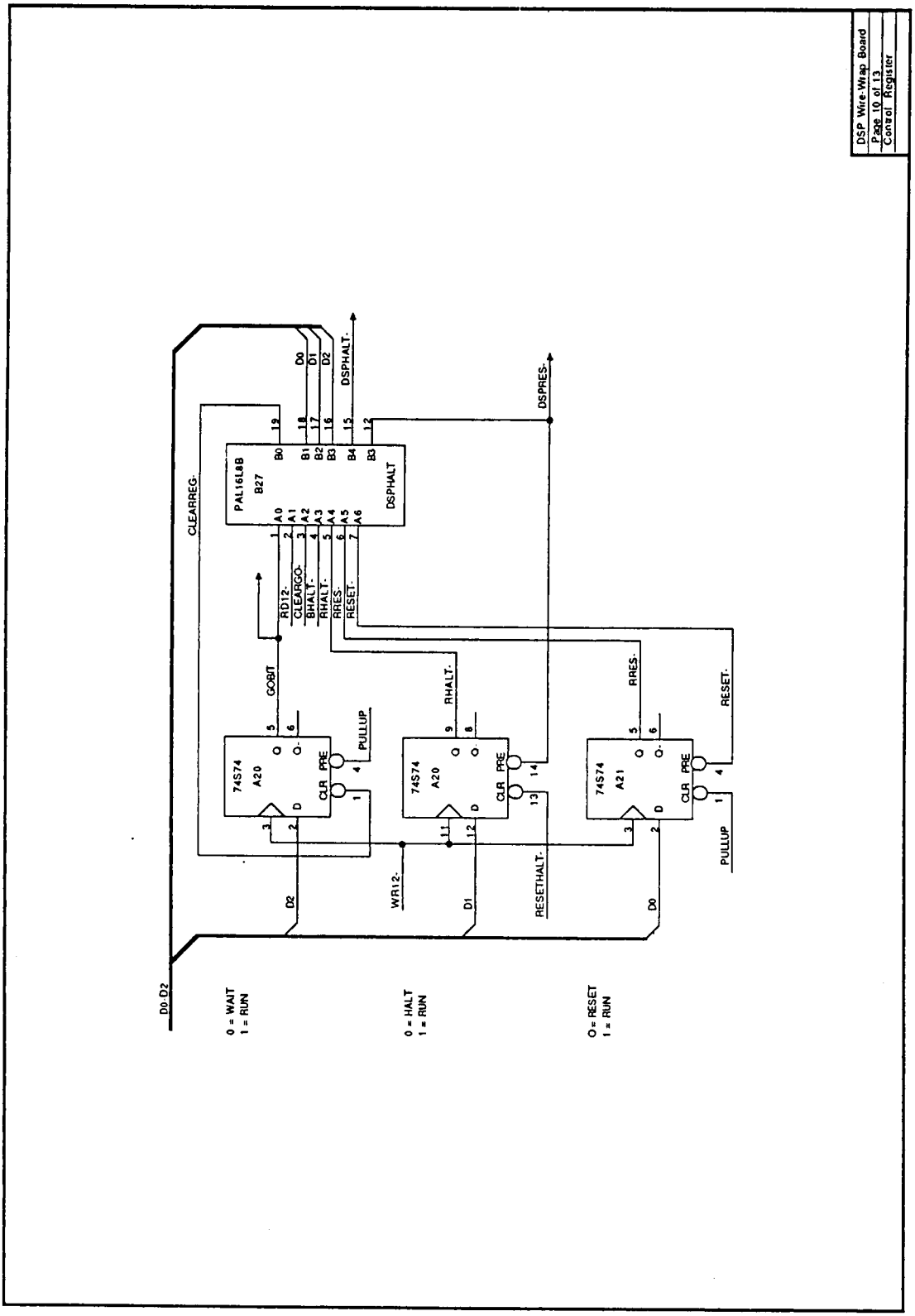


Figure A.10 Control Register

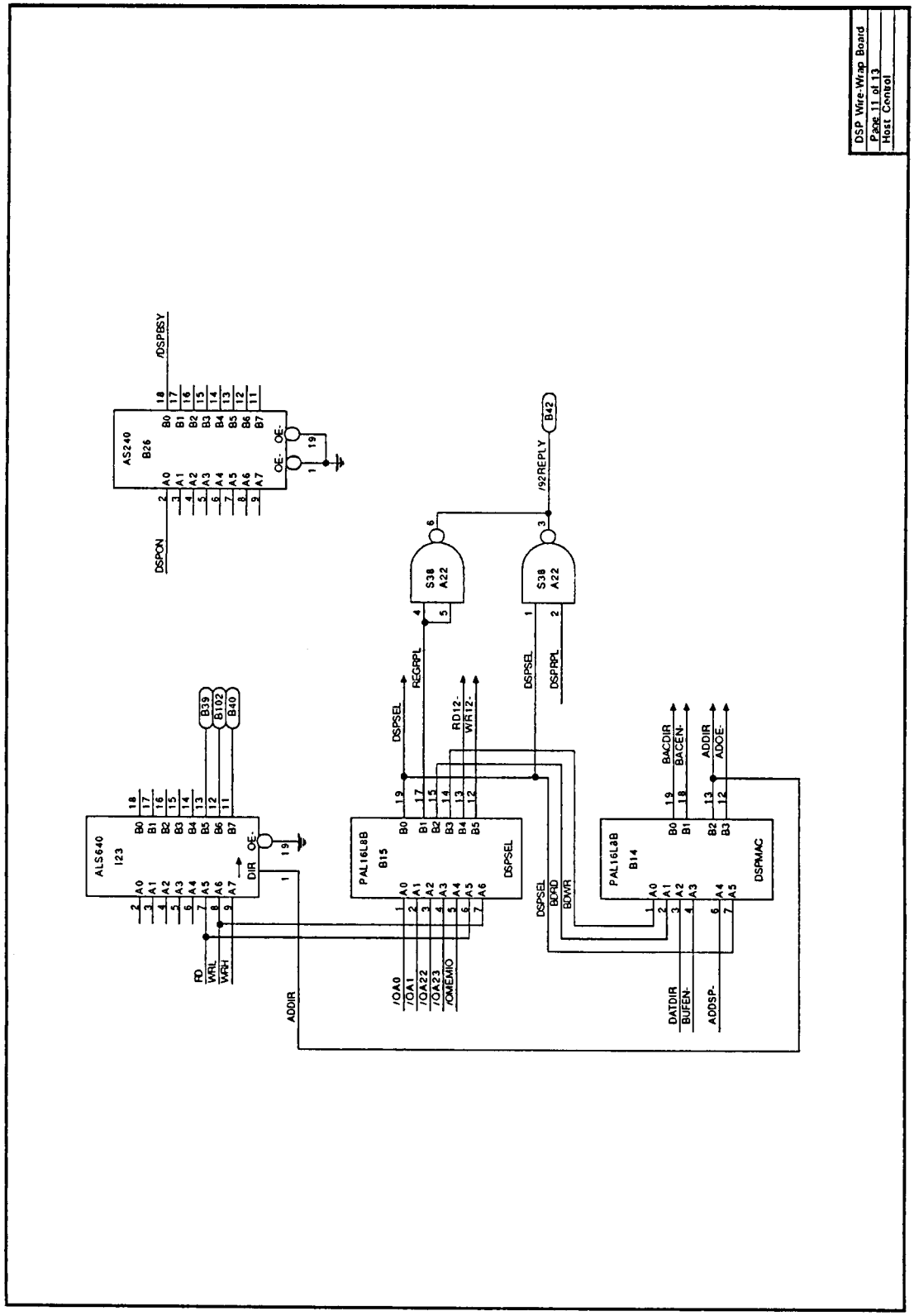


Figure A.11 Host Control

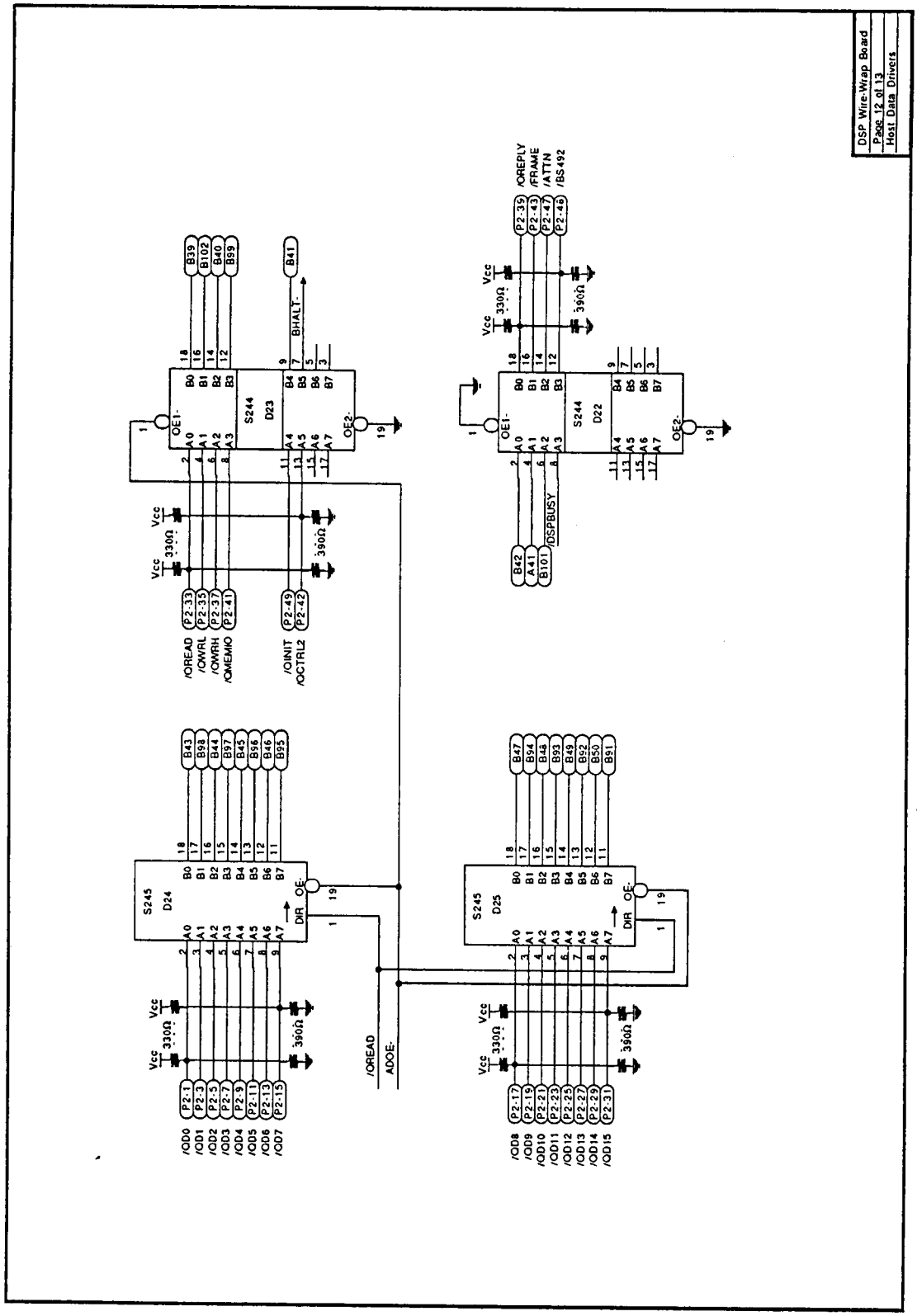


Figure A.12 Host Data Drivers

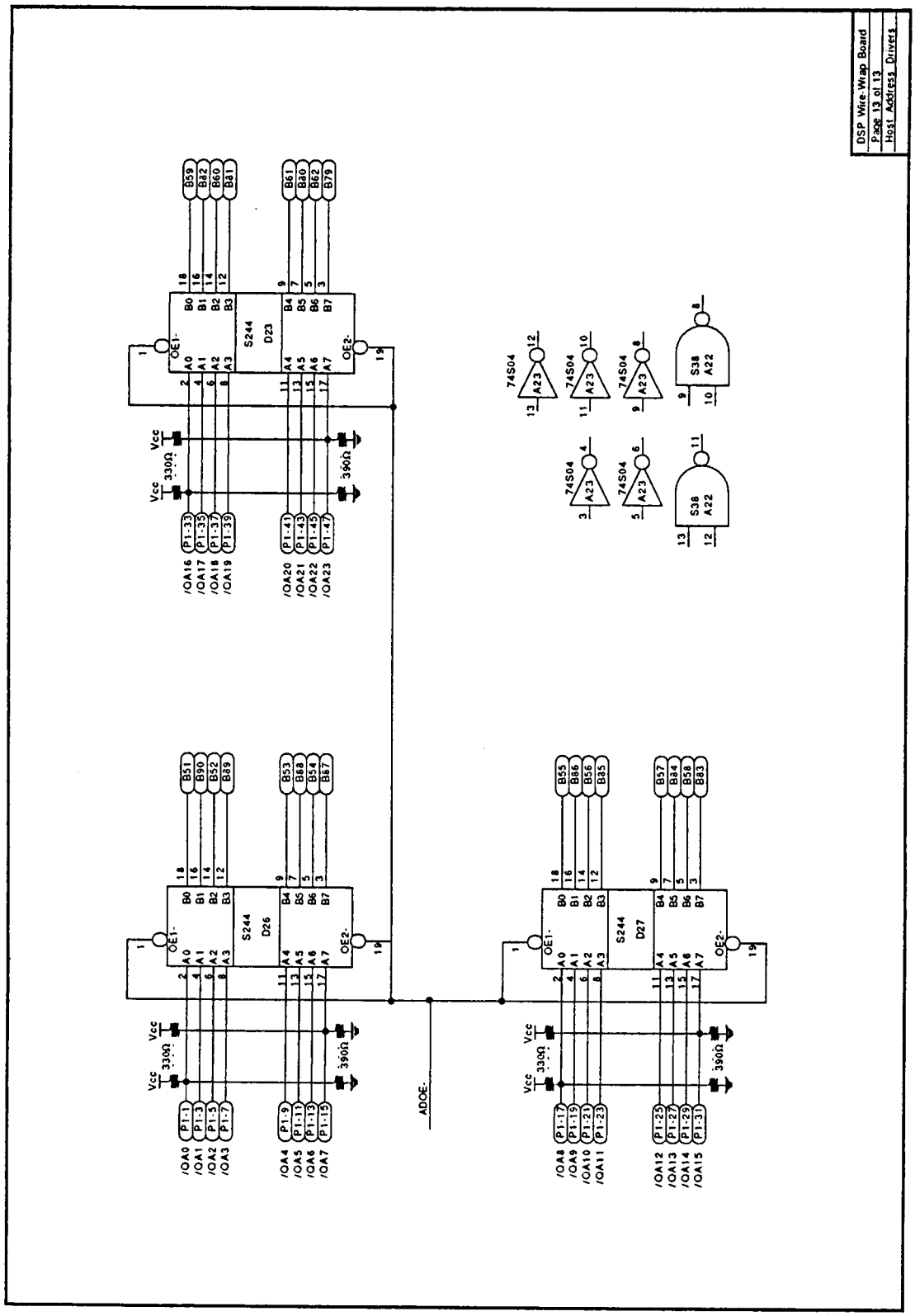


Figure A.13 Host Address Drivers

APPENDIX II

PARTS LIST AND PLACEMENT

T1	TMS320C25
T2	AM27256
T3	AM27256
A20-A21	74S74
A22-A23	74S38
B14	PAL16L8B DSPMAC
B15	PAL16L8B DSPSEL
B16-B23	AM2168-35
B24-B25	74AS244
B26	74AS240
B27	PAL16L8B DSPHALT
B28	PAL16L8B DSPADDR
D17	PAL16L8B DSPRDWR
D18	PAL16R6A DSPWAIT
D19	74ALS374
D20	PAL16L8B DSPENAB
D21	PAL16L8B DSPIO
D22-D23	74S244
D24-D25	74F245
D26-D28	74S244
E24	AM29823
E26	AM29863
E28	AM29823
G21	74ALS374
G22	74AS244
G23-G25	74AS245
I23-I28	74ALS640

APPENDIX III

PAL EQUATIONS

DSPADDR

PAL16L8

ADDRESS BUS ENABLES BETWEEN DSP AND 68020 (MAC OR HOST)

GOBIT	DSPON	YREG	DSPSEL	READ
/IMAGE	NC	NC	NC	GND
NC	DATDIR	/BUFEN	NC	/YREGOE
/ADHIEN	/ADHI68OE	/AD68	/ADDSP	VCC

IF(VCC) ADDSP = GOBIT * DSPON

IF(VCC) AD68 = /DSPON

IF(VCC) ADHI68OE = /DSPON + GOBIT * DSPON */YREG

IF(VCC) ADDHIEN = GOBIT * DSPON * /YREG

IF(VCC) YREGOE = GOBIT * DSPON * YREG

IF(VCC) /DATDIR = /DSPON * DSPSEL * READ
+GOBIT *DSPON * IMAGE * /READ

IF(VCC) BUFEN = /DSPON * DSPSEL
+ GOBIT * DSPON * IMAGE

DSPENAB

PAL16L8

RAM, PROM, IMAGE-MEM ENABLES FORM DSP

/STROBE	/IS	/PS	/DS	DSPA15
DSPA14	/DSPACK	/PROMRDY	NOMEM	GND
NC	/HOLD	NC	NC	NC
/PROGEN	/DATAEN	/PROMCS	/IMAGE	VCC

IF(VCC) IMAGE = DSPA15 * DS * /PS * /IS * STROBE

IF(VCC) DATAEN = /DSPA15 * DSPA14 * DS * /PS * /IS * STROBE

IF(VCC) PROGEN = DSPA15 * /DSPA14 * /DS * PS * /IS * STROBE

IF(VCC) PROMCS = /DSPA15 * /DS * PS * /IS * STROBE

IF(VCC) HOLD = DS * /DSPACK * DSPA15
+ PS * /PROMRDY * /DSPA15
+ DS * DSPA15 * NOMEM
+ /DS * /PS * /IS
+ /STROBE

DSPHALT

PAL16L8

DSP HALT PAL AND REBACK TO 68020

GOBIT	/RD12	/CLEARGO	/BHALT	/RHALT
/RRES	/RESET	NC	NC	GND
NC	/DSPRES	NC	NC	/DSPHALT
D2	D1	D0	/CLEARREG	VCC

IF(RD12) /D0 = RRES

IF(RD12) /D1 = RHALT

IF(RD12) /D2 = /GOBIT

IF(VCC) DSPRES = RESET + RRES

IF(VCC) CLEARREG = CLEARGO + RESET + RRES

IF(VCC) DSPHALT = BHALT + RHALT

DSPIO

PAL16L8

DSP I/O DECODE ENABLES

DSPA0	DSPA1	DSPA2	/STROBE	/IS
NC	NC	NC	NC	GND
NC	ADDRCLK	YADDRCLK	XCLK	NC
NC	/RESETHALT	/CLEARGO	CNCLK	VCC

IF(VCC) /CNCLK = /DSPA2 * /DSPA1 * /DSPA0 * IS * STROBE

IF(VCC) /ADDRCLK = /DSPA2 * /DSPA1 * DSPA0 * IS * STROBE

IF(VCC) /YADDRCLK = /DSPA2 * DSPA1 * /DSPA0 * IS * STROBE

IF(VCC) /XCLK = /DSPA2 * DSPA1 * DSPA0 * IS * STROBE

IF(VCC) CLEARGO = DSPA2 * /DSPA1 * /DSPA0 * IS * STROBE

IF(VCC) RESETHALT = DSPA2 * /DSPA1 * DSPA0 * IS * STROBE

DSPMAC

PAL16L8

MAC INTERFACE PALS

BDWR	BDRD	DATDIR	/BUFEN	NC
/ADDSP	NC	NC	NC	GND
NC	/ADOE	ADDIR	NC	NC
NC	NC	/BACEN	BACDIR	VCC

IF(VCC) ADOE = /ADDSP

IF(VCC) /ADDIR = /ADDSP

IF(VCC) /BACDIR = /DATDIR * BUFEN * ADDSP
+ /ADDSP * BDRD * /BDWR

IF(VCC) BACEN = ADDSP * BUFEN
+ /ADDSP * BDRD * /BDWR
+ /ADDSP * /BDRD * BDWR

DSPRDWR

PAL16L8

IMAGE RD/WR SIGNAL GENERATOR

LOBIT	HIBIT	DSPSEL	DSPON	/IMAGEST
NC	NC	/IMAGE	/ADDSP	GND
NC	NC	NC	WRH	WRL
RD	/STROBE	/RDWR	NC	VCC

IF(ADDSP) /RD = /IMAGEST + RDWR + /STROBE
+ /IMAGEST */ IMAGE

IF(ADDSP) /WRL = /IMAGEST + /LOBIT + /RDWR + /STROBE
+ /IMAGEST * /IMAGE

IF(ADDSP) /WRH = /AIMGEST + /HIBIT + /RDWR + /STROBE
+ /IMAGEST * /IMAGE

IF(DSPSEL * /DSPON)
RDWR = WRH + WRL

IF(DSPSEL * /DSPON)
STROBE = WRH + WRL + RD

DSPSEL

PAL16L8

MAC INTERFACE RD/WR/SELECT PAL

/PA0	NC	/PA22	/PA23	/MEMIO
RD	WRL	NC	NC	GND
NC	/WR12	/RD12	BDWR	BDRD
NC	REGRPL	NC	DSPSEL	VCC

IF(VCC) /DSPSEL = /PA23 + /PA22 + MEMIO + PA0

IF(VCC) RD12 = PA23 * PA22 * /MEMIO * PA0 * RD * /WRL

IF(VCC) WR12 = PA23 * PA22 * /MEMIO * PA0 * /RD * WRL

IF(VCC) /BDWR = /PA23 + /PA22 + MEMIO + RD + /WRL

IF(VCC) /BDRD = /PA23 * /PA22 + MEMIO + WRL + /RD

IF(VCC) /REGRPL = /PA23 + /PA22 + PA0 + MEMIO + /RD * /WRL

DSPWAIT

PAL16R6

WAIT STATE GENERATOR

CLK	/IMAGE	/PROMCS	NC	NC
NC	NC	NC	NC	GND
/OE	/PROMRDY	NC	/PST0	/WAIT1
/ISTO	NC	/STROBE	/IMGSTR	VCC

IMGSTR = STROBE

STROBE := ISTO * IMAGE

ISTO := IMAGE

WAIT1 := PROMCS * /PROMRDY * PST0

PST0 := PROMCS * /PROMRDY

PROMRDY = PROMCS * WAIT1 * PST0

APPENDIX IV

REGISTER DEFINITIONS

HOST CONTROL

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	-	G	H	R

G GOBIT

CLEAR to signal the DSP to wait.
SET to tell the DSP to start processing.

H HALT

CLEAR to HALT the DSP.
SET to RESTART the DSP.

R RESET

CLEAR to RESET DSP.
SET to allow the DSP to run.

Mapped at 0xE00000 to host.
DSP cannot access this register directly. If either the RESET or HALT bit are set, then the DSP is not running. The DSP can poll the remaining bit (GOBIT) by reading the BIOZ pin.

DSP CONTROL

PA0 - DSP I/O SPACE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	-	Y	Hi	Lo

Y Use Y-Register Addresses

When CLEARED, for address use:

Xtra Register for bits	A23-A20
Address Register for bits	A19-A15
DSP Address for bits	A14-A0

When SET, for address use:

Xtra Register for bits	A23-A20
Address Register for bits	A19-A18
Use Yregister for bits	A17-A9
DSP Address for bits	A8-A0

Hi Write to High Byte of Image Memory

SET to enable Write

Lo Write to Low Byte of Image Memory

SET to enable Write

The DSP writes to the Control Register when it has been told what part of image memory to write to. The decision to use Y-Register addressing depends upon the algorithm.

ADDRESS

PA1 - DSP I/O SPACE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	A19	18	17	16	15

A19-A15

Extra Address bits needed to extend the DSP address space to a full image plane.

Y-REG

PA2 - DSP I/O SPACE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	A17	16	15	14	13	12	11	10	9

A17-A9

Address bits used to access the image by a column address. In an X/Y plane, these bits specify the Y dmension.

XTRA

PA3 - DSP I/O SPACE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	M	A23	22	21	20

M MEMIO

Extended Address bit of the IP9200.

When CLEARED, space is reserved for Registers
When SET, space is reserved for Image Memory

A23-A20

Extra Address bits needed to extend the DSP address space to full addressing space of the IP9200.

CLEAR GO

PA4 - DSP I/O SPACE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Any access to this port, causes the GOBIT in the Host Control Register to be cleared.

HALT

PA5 - DSP I/O SPACE

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Any access to this port, causes the HALT bit in the Host Control to be cleared, causing the DSP to halt execution.

APPENDIX V

DSP RESOURCE MAPPING

DSP Control Register	-	E00000
DSP Data Space	-	C50000 - C5FFFF
DSP Program Space	-	C90000 - C9FFFF
DSP I/O Space	-	CE0000 - CE000F
EPROM Starts	-	C90000
EPROM Ends	-	C97FFF
Program RAM Starts	-	C98000
Program RAM Ends	-	C98FFF
Data RAM Starts	-	C54000
Data RAM Ends	-	C54FFF

VITA

Michael A. Goldston was born in Chattanooga, TN in 1961. Attending public schools in Atlanta and Chattanooga, he graduated from Central High School in 1979. He was valedictorian and received departmental awards in English, Mathematics, and American History. He then entered the University of Tennessee College of Engineering, specializing in Computer Engineering and graduated in 1984 with highest honors. He then worked for a year at the Westinghouse Defense plant in Baltimore, MD before returning to Knoxville to start his Master's program. Starting his Master's program in 1985 with an emphasis in image processing with a grant from Texas Instruments. He completed his classwork in 1986 where he started to work for Perceptics, a Knoxville based high-tech engineering firm specializing in image processing. He worked on algorithms for their license plate reader, developed all of their DSP code, and established an architecture for their DSP systems. Finally, in the fall of 1988, due to an incredibly bad football season, he managed to complete his thesis and earn his Master's Degree. Currently, he is designing the second generation of character recognition hardware to be used in an optimized license plate reader.