Digital Communication System Design Workstation

George C. Hwa

Center for Communications and Signal Processing
Electrical and Computer Engineering Department
North Carolina State University

CCSP-TR-88/13

April 1988

Abstract

HWA, GEORGE C. Digital Communication System Design Workstation. (Under the direction of Sasan Houston Ardalan) The Purpose of this project was to design and implement an advanced Multibus-based multiprocessor workstation that could be used in digital communication system research. The workstation's bus-based architecture enables it to survive technological evolutions in DSP hardware and algorithms. The architecture consists of a Multibus I system bus interface, a DB32016 single-board microcomputer, which is based on the 32-bit NS32016 microprocessor, with a memory expansion unit and two TMS320 family-based DSP units along with data acquisition devices. The Multibus I system bus interface is the main communication channel through which the bus master, DB32016, monitors and controls system operations in the workstation. Digital signal processing tasks are performed by the two DSP units, DSP32010 and DSP32020. A dedicated data channel links the two DSP units, allowing for high-speed data transfers to take place between them without interfering with the operations on the system bus interface. Results from this high performance DSP workstation are presented for real-time DLC system implementations. In order to overcome the difficulties that arise from the hostile power line environment, special techniques are carefully studied. Among these are adaptive harmonic cancellation, pulse shaping for PSK, phase locked-loops, normalized lattice IIR filters, and duobinary pulse coding. The workstation demonstrates the ease and flexibility with which communication systems can be developed and implemented in an all-digital form.

Table of Contents

Chapter 1 Introduction

Very-Large-Scale-Integration(VLSI) technology has experienced exponential growth in the past two decades. In an area less than a quarter of a square inch, a circuit designer is now able to place over 100,000 gates to realize circuits of tremendous complexity. An electronic circuit realizing a complicated system which had to be implemented with several boards ten years ago, it can now be integrated onto one small chip. In the process, the cost per gate of fabricating such a tiny chip has been reduced to a fraction of a cent. The benefits of this technological progress have been manifested in every branch of science and engineering fields. One of the areas that has received a great impact is Digital Signal Processing(DSP).

1.1 Digital Signal Processing

Digital signal processing techniques have been widely used in speech and image processing, digital communications, and advanced digital control systems[26][30][32-37][43]. The rapid advancements in both hardware(specifically microprocessors which make DSP possible) and software(algorithms which exploit DSP implementations) make digital systems practical substitutes for their analog counterparts. Digital systems have the advantage of reliability, reproducibility, compactness, programmability, and efficiency.

Reliability means that a digital system's performance is highly stable and consistent so long as there is no catastrophic component failure. On the other hand, an analog system's performance may be degraded due to component aging or temperature variations. Reproducibility implies that digital systems can be reproduced to

meet exact design specifications without labor-intensive tuning. As the level of integration continues to increase, digital systems are becoming more and more compact, enabling applications where space constraints are severe. Since the functionality of digital systems are primarily determined by the underlying software, the same hardware often can be re-configured to perform completely different functions by simply replacing the software. This programmability significantly reduces the time required to develop new systems, allowing system expansion and upgrading to take place at a much faster pace in order to keep up with the rapid development of advanced DSP algorithms. The improved efficiency is fully manifested in performance versus cost.

DSP applications have penetrated almost every area of science, engineering, and everyday life. In speech processing, DSP technology makes speech synthesis and recognition possible. Scientists in astronomy, geology, and meteorology are using DSP to interpret images to a greater extent. Engineers in the video industry are applying DSP technology to enhance the performance of video display equipment. In communications, all-digital radios are designed and built with DSP technologies. Particularly in telecommunications, where DSP probably has the largest usage, it has become an essential technology.

1.2 DSP in Power Line Communications

DSP also finds applications in power line communications, such as load management, remote meter reading, and distribution equipment monitoring and control. In power line communications, the transmission media is the copper wires that normally carry 60 Hz AC energy. These lines were originally designed only for the low frequency waveforms. Therefore, they are not particularly suitable for higher frequency

transmissions. In addition, 60 Hz harmonic noises are so strong that high fidelity transmission is impossible without special treatment. Special techniques have been developed over the years to combat the hostile power line communication environment[22]. These techniques rely heavily on the successful implementation of mathematical computation intensive algorithms. As DSP hardware improves with the improvement in VLSI technology and as new algorithms are developed, the existing communication systems quickly become obsolete. New techniques cannot be readily tested on existing hardware platforms, although computer simulations may be used for validation. In the meantime DSP system designers and researchers are seeking advanced development tools to shorten the development cycle and to keep up with the fast technological advancements.

1.3 DSP Workstation Approach

The aim of this project was to design and implement a DSP workstation with an advanced architecture, sophisticated software support, a user-friendly operating environment, and the flexibility for system expansion and upgrade. The workstation concept is illustrated in figure 1.1. The motivation for the workstation approach is dis-

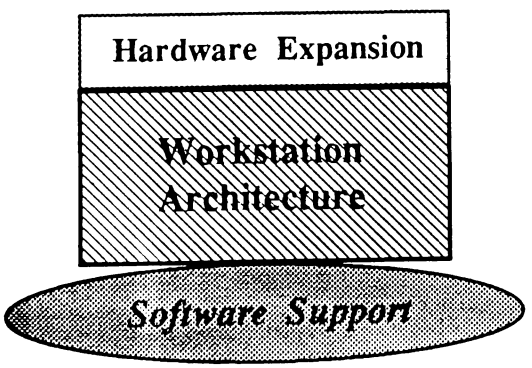


Figure 1.1 DSP Workstation Concept

cussed in chapter 2. The DSP workstation has been utilized extensively in the study of power line communication systems which overcome the difficulties associated with power line communications.

1.4 Summary of Topics

This report is organized into six chapters. The first chapter serves as a brief introduction. Starting with chapter two, the motivation behind the effort to develop the DSP workstation is elaborated in detail, along with the workstation's specifications and its application objectives for power line communications. Chapter three describes the proposed DSP workstation architecture and discusses the issues involved in the hardware implementation of the workstation. The support software and the user environment of the DSP workstation are the subject of chapter four. In chapter five, a study of the application of the workstation to power line communications is described and the results are presented. Chapter six summarizes the achievements of the project and looks at the future of the DSP workstation in terms of its capabilities in DSP research and its expansibility, both in hardware and software, to stay ahead of DSP developments. Also included in chapter six are some issues related to DSP algorithm implementations for future study.

Chapter 2 Motivation

The need for a DSP workstation, which can be used to quickly develop digital systems, is obvious. Researchers need the workstation to verify the validity of DSP algorithm in real time without the concern for hardware implementation detail. Designers may use the workstation to confirm system designs and to improve such designs prior to the actual hardware construction. Given its flexible system architecture, the DSP workstation will enable researchers and design engineers to remain at the frontier technology. For instance, in order to explore the capabilities of a newly developed DSP microprocessor, which emerges every couple of years, the new DSP microprocessors can be incorporated into the workstation without altering the basic hardware structure and software environment of the workstation. This type of architecture allows researchers and design engineers to adapt to new technology quickly with minimal effort.

2.1 Real Time DSP

DSP researchers often rely on computer simulations for verification of their research results. Computer simulation is a relatively inexpensive way to model the behaviors of physical systems. The simulation results, though not observable in real time, can be used to examine the correctness of an algorithm. Computer simulations save researchers the time and effort of building the actual hardware that implements a particular algorithm. If the initial simulation results are not satisfactory, the software that is used to simulate the physical processes of an algorithm may be easily

modified to model an improved version of the algorithm. Thus a researcher can go through the design process from a particular algorithm to the implementation(via simulation) back and forth a number of times before reaching a conclusion about the algorithm. Researchers who are not familiar with building DSP hardware can still obtain an understanding of how algorithms will behave in physical settings. The major short coming is that computer simulations can only model a physical process for a relatively short real-time frame, i.e. the simulation results can only represent a short instance of a physical process. For example, a DSP algorithm is implemented in hardware to process a speech signal at an 8K Hz sampling rate. Then 480,000 samples have to be processed by the hardware in one minute of real time. From experience, it might take more than an hour of CPU time to simulate 480,000 samples on a VAX-11/780 super mini-computer. If the simulation is performed on a multi-user system, which is normally the case, it may very well take an entire day before the results are available. Such a lengthy computer simulation is usually avoided due to the high cost of CPU time. Furthermore, data storage requirements may also be prohibitive.

Another consideration is the fact that certain behaviors of a digital system do not manifest themselves over a short period of time. For instance, research results on adaptive filters(described later in this report) clearly indicate that the filter tap drift phenomenon took place after more than a million samples representing over twenty minutes of real time. This phenomena can only be studied through real-time processing. The DSP hardware is essential to accomplish real-time observations and a DSP workstation makes the hardware components easily accessible. Yet, the workstation does not impose any constraints on the implementations of particular DSP algorithms. The freedom to implement a wide range of DSP algorithms on the workstation hardware implies that the underlying software can be altered as the user wills just as in

case of computer simulation. It reduces researchers experimental efforts and increases design engineers productivity by allowing them to go through design cycles quickly and easily. Of particular importance to the design engineer, a shortened time from the drawing board to prototype means a better product with less cost, since the designer does not solely depend on the results from computer simulations to make design decisions. The generality of the DSP workstation hardware allows a design engineer to explore his/her options in choosing necessary hardware components to implement a specific digital system and to use only those components that are needed for the system in the final product. The elimination of the prototype hardware construction phase can mean a significant savings in time and cost. Design engineers can make better design decisions without the concern for cost overruns in the design process. Therefore, the workstation offers the best of both computer simulation and hardware implementation approaches.

Another consideration for implementing DSP algorithms is the fact that numerical operations(essential for digital signal processing) in a digital system are performed with limited precision. Although floating point computation is quickly becoming a common hardware feature on DSP processors, it is still quite expensive at the moment[44]. The workstation built with actual DSP hardware components makes the implementation and observation of finite precision effects a straight forward matter. On the other hand, in computer simulation the use of floating point arithmetic can yield misleading conclusions. A stable algorithm using extended precision in computer simulations can become unstable using limited precision DSP processors due to truncation or rounding errors. In order to observe the true behavior of the limited precision arithmetic operations, one has to either implement the algorithm on a DSP processor or carry out special programming techniques to imitate limited precision arithmetic

operations in computer simulation. The later alternative is usually a rather involved process that most people avoid whenever possible. Therefore, the former alternative, testing an algorithm with a true DSP processor, leads to another supporting argument for building the DSP workstation. Also worth mentioning is the fact that many DSP algorithms have critical timing requirements. Such behaviors are better understood by studying them is the real time.

2.2 Adaptations to New Technologies

In order for DSP researchers and design engineers to keep up with the rapid advancements in VLSI technology, which is one of the driving forces behind DSP technology, a DSP workstation with hardware expansion and upgrade capabilities is desired. Instead of developing and replacing the entire DSP system every time a new DSP processor becomes available, or additional hardware features are needed to meet the more sophisticated DSP processing requirements, the enhanced DSP workstation architecture can be used to allow system expansion and upgrades. These features will give researchers and designers a leading edge in discovering advanced DSP algorithms and bringing about competitive DSP products. The workstation approach also makes the DSP research and development environment highly stable. Once an interactive and user-friendly programming environment is developed for the workstation, it will provide the user accesses to the workstation's resources in a consistent manner. This eliminates the overhead involved with the migration of research and design environments, and reduces the time and effort required to keep up with technological changes.

Therefore, the objective of this project is to design and build a DSP workstation

which efficiently employs advanced VLSI microprocessors with unique DSP capabilities, easily allows hardware expansion and upgrades, continuously accommodates a stable real-time digital signal processing environment, and genuinely provides a user-friendly programming interface. The workstation is to become a powerful research and design vehicle in DSP applications, specifically in power distribution line carrier communication research as the partial objective of this project.

Chapter 3 Proposed DSP Workstation Architecture

The motivations and requirements discussed in chapter two lead to the specifications for the proposed workstation architecture. The overall workstation architecture calls for a bus-based multiprocessing system. Major hardware components of the workstation include a general-purpose microcomputer subsystem, dedicated DSP

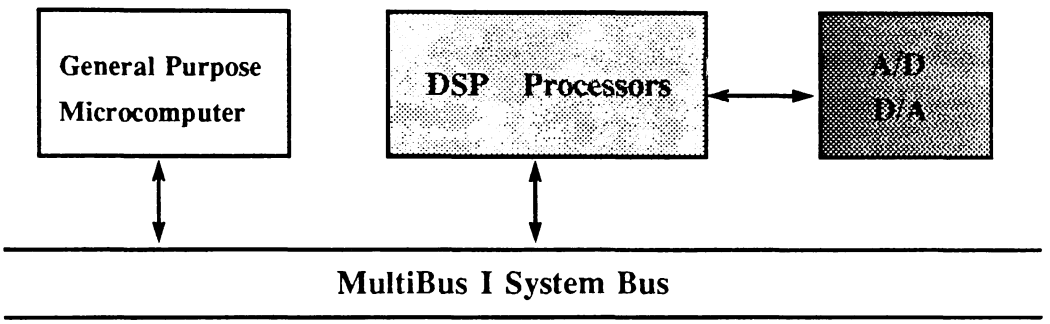


Figure 3.1 DSP Workstation Architecture

microprocessor-based subsystem, and a data acquisition subsystem. The proposed architecture is illustrated in Figure 3.1. These subsystems are described in sections 3.1 to 3.4. Section 3.5 discusses the design, construction and testing of the workstation hardware.

3.1 Bus-based system architecture and Multibus-I/IEEE796

The primary advantage of using a bus-based architecture in a workstation design is the ease with which the workstation can be expanded and updated with minimum engineering efforts. Especially when a standard system bus is adopted in the design

the workstation components may be obtained from various sources other than doing custom design. In addition to the advantage of large variety of components available by using a standard bus, there is also the benefit of maintaining a stable overall configuration and programming environment as the workstation undergoes hardware upgrade.

There are a number of industry standard system busses that are intended for microprocessor-based system applications. Some of the bus architectures have been conformed to IEEE standards. Among those is the Multibus-I system bus, also known as the IEEE/796 Standard Bus. This bus has been chosen as the system bus for the DSP workstation.

3.1.1 Multibus System Bus

Multibus-I is a well-defined advanced system bus architecture[76]. It supports both 8-bit and 16-bit data transfers at a maximum rate of ten megabytes per second. The bus interface provides independent memory and I/O address spaces. There are 16M bytes of memory address space and 64K bytes of I/O address space available in a Multibus system. The master-slave processor concept is employed in incorporating the bus devices into a Multibus system. Communication(data transfers between bus devices) and bus arbitration(resolution of conflicts when multiple requests occur on the bus) take place on the bus interface in the form of asynchronous bus cycles. A bus master is the processor(device) that initiates a bus cycle, while a slave is the one that responds during the bus cycle. In theory, any device in a Multibus system may act as a bus master at one time or another. However, the one assigned with the highest arbitration priority always gains the control of the bus first when multiple bus requests occur.

A Multibus memory access takes place during a memory cycle. A bus master places an address on the address bus lines. The addressed slave decodes the address and responds by sending an acknowledge signal back. Then the bus master issues a memory read/write command while data being transferred over the data lines.

A Multibus I/O access takes place during an I/O cycle. The process proceeds in similar fashion as in memory cycles except that the bus master issues I/O read/write commands instead of memory commands.

To facilitate system operations, i.e. bus cycles, there are special signals, including bus priority(BPRN), bus busy(BUSY), bus request(BREQ), and inhibit signals(INH1 and INH2), designated in a Multibus system to enable or disable a device to initiate bus cycles.

The Multibus-I is chosen as the candidate for the system bus for several reasons. First, it is a well-defined industry standard as well as an IEEE standard. Second, there are a large number of Multibus compatible system components available. Third, the cost of building such a system is relatively inexpensive compared to other buses, such as Multibus-II and VME bussed which are synchronous bus interface systems.

3.1.2 Workstation Hardware Components and Their Functions

To implement the Multibus-based DSP workstation, three major hardware components are essential: a Multibus compatible microcomputer, DSP units, and data acquisition units. A National Semiconductor Corporation single-board microcomputer, namely the DB32016 Development Board, is used as the bus master to control the workstation operations by providing an interface between the user and the DSP processors. Two DSP units are built around the TMS320 family of high performance DSP

single-chip microcomputers with a Multibus compatible interface to allow the bus master to carry out necessary operations. The data acquisition devices, analog-to-digital(A/D) and digital-to-analog(D/A), are not actually a part of the Multibus interface components. They are separately implemented and interfaced to one of the DSP units through two dedicated parallel ports. The analog acquisition system band-limits the incoming analog signal and samples at the Nyquist rate. The sampled data is transferred to the DSP units that perform desired DSP operations on the data. The processed data is then passed onto the D/A converter which converts the data back to an analog signal. A workstation user can control the operations by communicating to the bus master through a sophisticated user interface under software control.

3.2 General Purpose Microcomputer Subsystem

The workstation provides its powerful DSP capabilities to the users through a general purpose microcomputer subsystem along with an interactive user interface. The subsystem contains hardware elements that reside on two Multibus boards - the DB32016 single board microcomputer and the PSM512A memory board.

The DB32016 microcomputer is incorporated into the Multibus system as the bus master, using the master-slave concept. It controls the operations, such as system initialization, DSP program loading, and DSP program execution, on the workstation. The DB32016 is a complete single board microcomputer system[77]. It has an advanced 32/16 bit NS32016 microprocessor, a high performance floating point unit, a dedicated memory management unit, 128K dual port random access memory, a bootable ROM, a fully compatible Multibus interface, and I/O ports with interrupt controls. Through the two serial ports, the board can be operated in either stand-alone or

host assisted mode to execute programs. The Multibus interface allows it to commu-

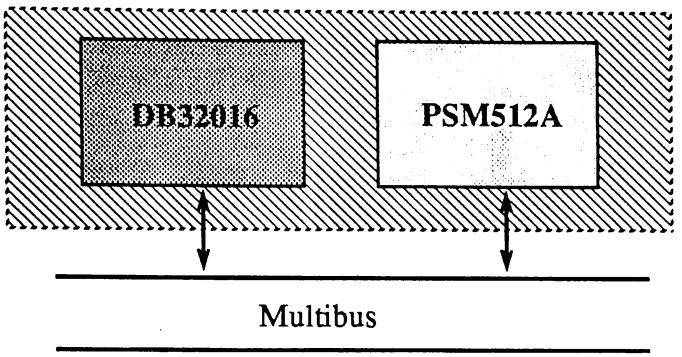


Figure 3.2 Microcomputer Subsystem

nicate with other boards on the system bus. Off-board addressing includes both memory and I/O spaces. The PSM512A is a random access memory board with error checking-correction feature[78]. It provides an additional 256K of memory for the DB32016 through the Multibus interface. The DB32016 is configured as the bus master in the workstation, having access to all the workstation hardware sources - the memory on DSP units and the I/O ports mapped onto the Multibus I/O space.

3.3 DSP Subsystem

DSP operations are performed by two independent DSP units(boards) in the DSP subsystem of the workstation. These two units are constructed around the Texas Instruments TMS320 family of DSP single-chip microprocessor chips. Dedicated memories, I/O ports and data channels are built into the units. The Multibus interface on these units give the bus master access to their local memories, enabling the bus

master to load programs and exchange data with the DSP subsystem. The two DSP

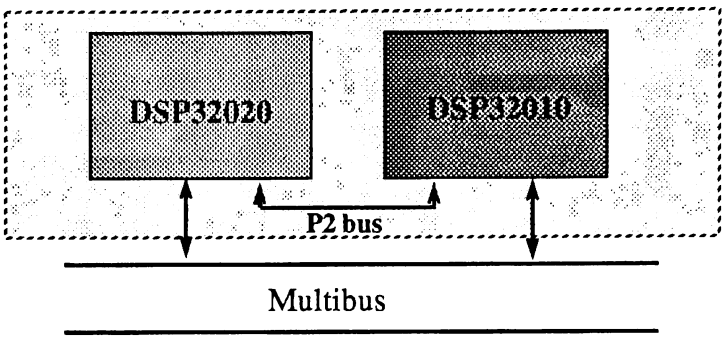


Figure 3.3 DSP Subsystem

units are loosely coupled through a dedicated communication link, allowing them to exchange information while executing independent tasks.

3.3.1 TMS32010 and DSP32010

The DSP32010 board uses a TMS32010 microprocessor chip, TI's first generation DSP microprocessor family, as the CPU. The TMS32010 is interfaced to the dual-ported memory and the I/O devices.

The TMS32010 features a 200ns instruction cycle and pipelining instruction executions[19]. A modified Harvard architecture with separate data and instruction memory spaces allowing instruction fetch and execution to overlap. It has a 16x16-bit single-instruction-cycle multiplier. The shifters combine data transfer to and from the accumulator with bit shifting for efficiency. The auto-increment and auto-decrement registers can be used for indirect addressing and loop control. There are 144x16 on-chip data memory and 4Kx16 off-chip program memory. The special single-cycle I/O instructions allow fast I/O operations that are critical to DSP implementations. There

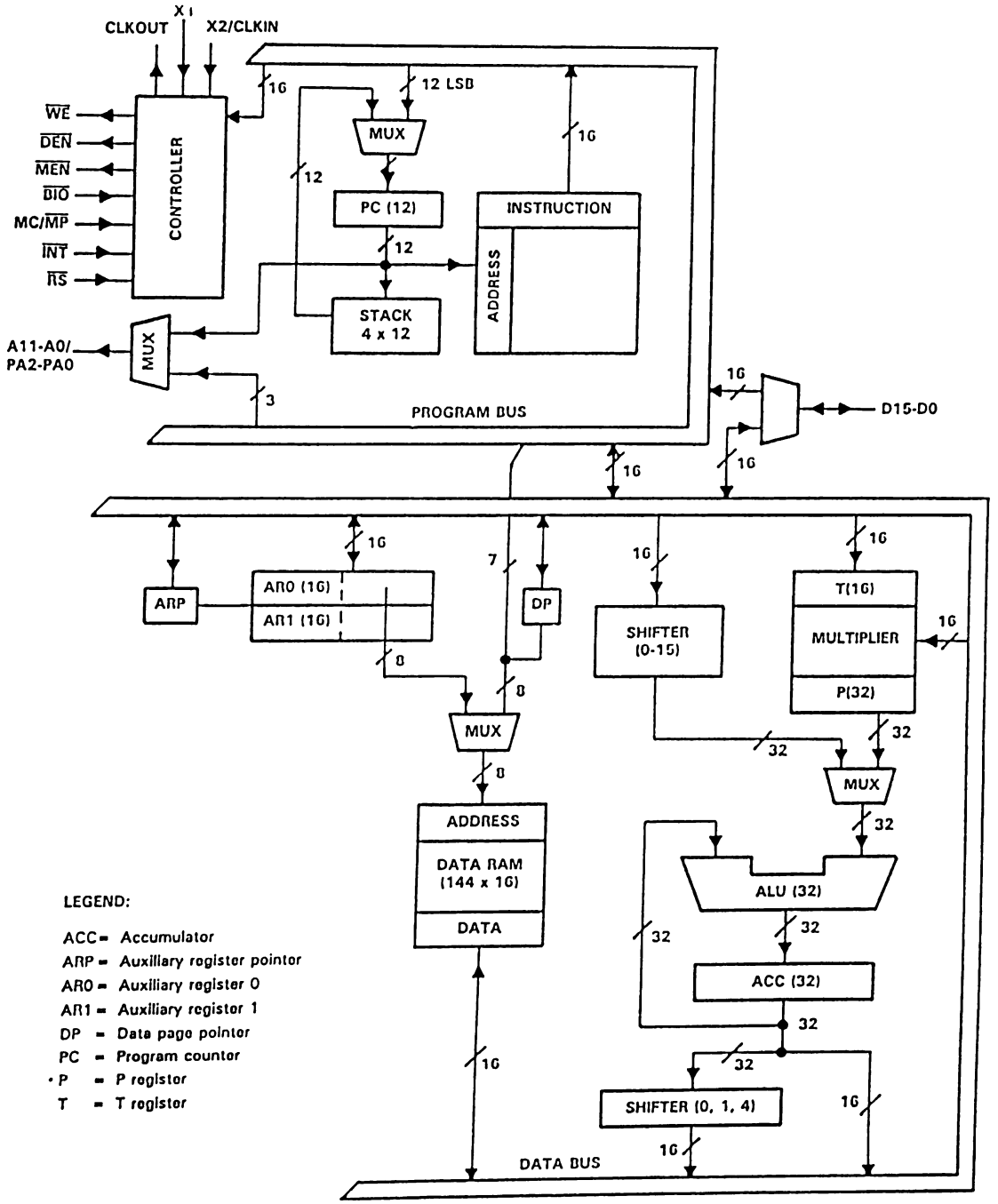


Figure 3.4 Functional Block Diagram of The TMS32010

is a four-level stack for context switching.

The DSP32010 board provides 4Kx16 static memory with 45ns access time to allow the TMS32010 to execute instructions at full speed. The memory is also mapped onto the Multibus memory space to allow direct access by the bus master(DB32016). Since there is no way to temporarily suspend the program executions on the TMS32010, the only way the DB32016 can access the dual ported memory is to halt the TMS32010 and reset it afterwards.

Six I/O ports are built into the DSP32010 board to facilitate the TMS32010's I/O operations. They are paired as the A/D input and the D/A output ports for data acqui-

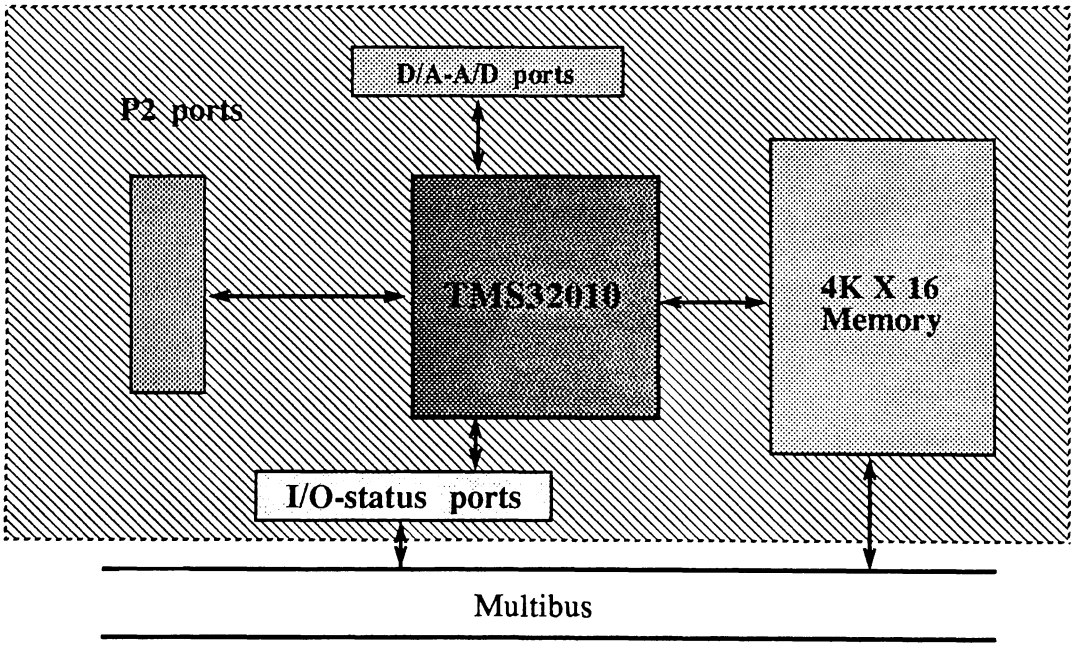


Figure 3.5 Functional Block Diagram of the DSP32010 Board

sition, the Multibus I/O ports for status control and data transfer to the DB32016, and the P2 I/O ports for high speed communications with the DSP32020.

3.3.2 TMS32020 and DSP32020

The second board, the DSP32020, implements TI's second generation DSP microprocessor chip, the TMS32020 as the CPU. The board was built with a dual-ported memory and I/O interfaces.

In addition to the features contained in the TMS32010, the TMS32020 has the increased on-chip memory capacity(from 144x16 to 544x16) and the increased program and data memory spaces(both up to 64Kx16). The new processor also increases number of I/O ports up to 16 ports. It has the ability to interface with low speed external memory chips and the ability to configure the on-chip memory for storing either data or instructions, or both. Two serial I/O ports are added on the TMS32020. It also has a multi-level interrupt capability and an enhanced instructions set.

The DSP32020 board is built with eight 2Kx8 static memory chips with 35ns access time. A memory configuration register permits the memory to be used as either data or program memory as desired. This configuration register is addressable by the TMS32020 itself as an output port or as an output port by the DB32016 from the Multibus I/O address space. This feature makes fast context switching possible. Just as with the DSP32010 board, the memory is mapped onto the Multibus memory space. The TMS32020 can be held is its wait state to interface with slower speed devices. Therefore, the bus master(DB32016) can utilize this feature to access the dual port memory on the DSP32020 during the TMS32020 's program execution without resetting it. When the DB32016 makes an attempt to access the dual ported memory, a HOLD signal is asserted to force the TMS32020 to enter the wait state as if waiting for a slower speed device to respond, temporarily suspending the TMS32020's program execution. After the DB32016 completes the access, the

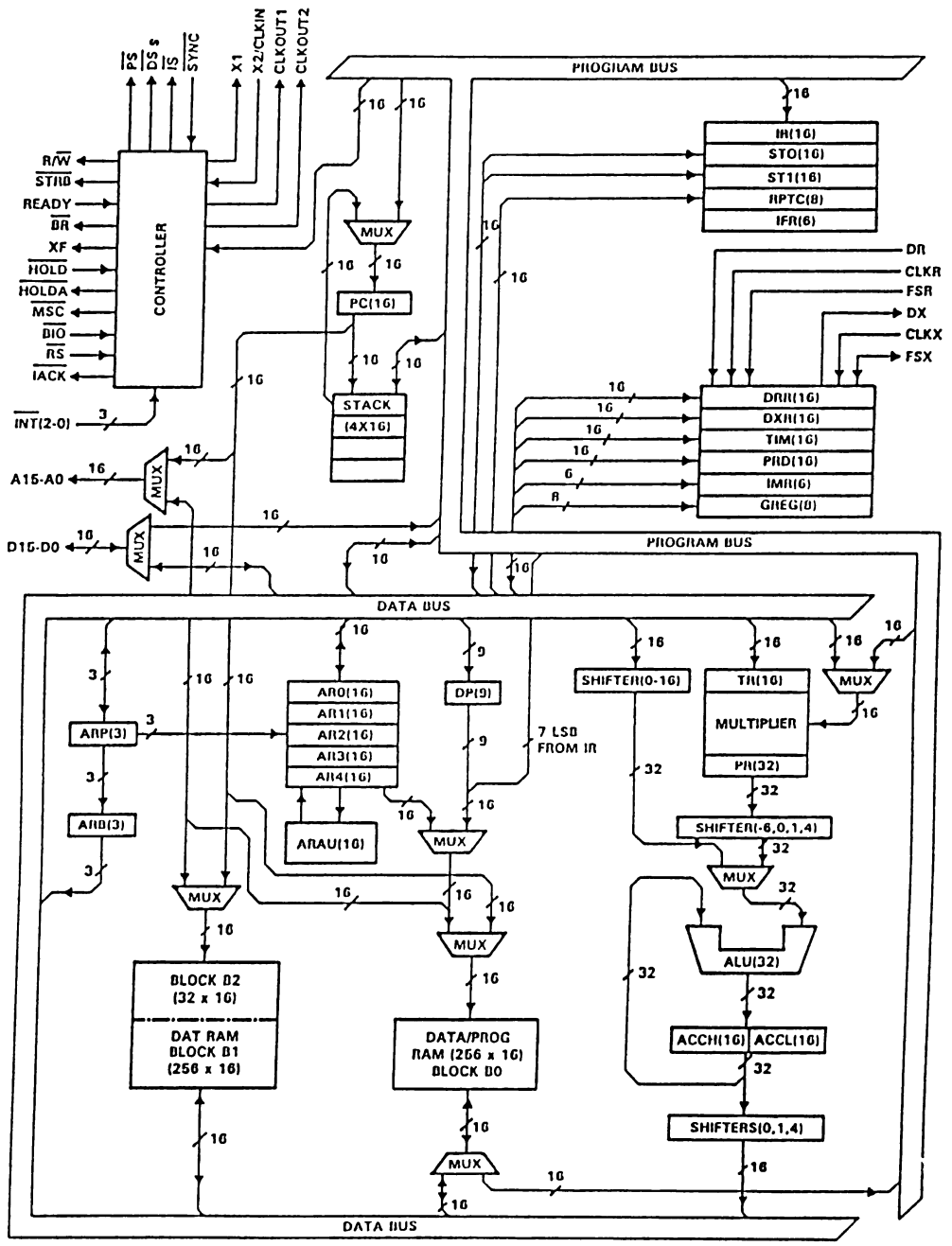


Figure 3.6 Functional Block Diagram of The TMS32020

HOLD signal is removed to allow the TMS32020 to continue its execution.

There are four I/O ports on the DSP32020, paired as the Multibus I/O ports and the P2 I/O ports. The P2 I/O ports are connected to the DSP32010, providing a private

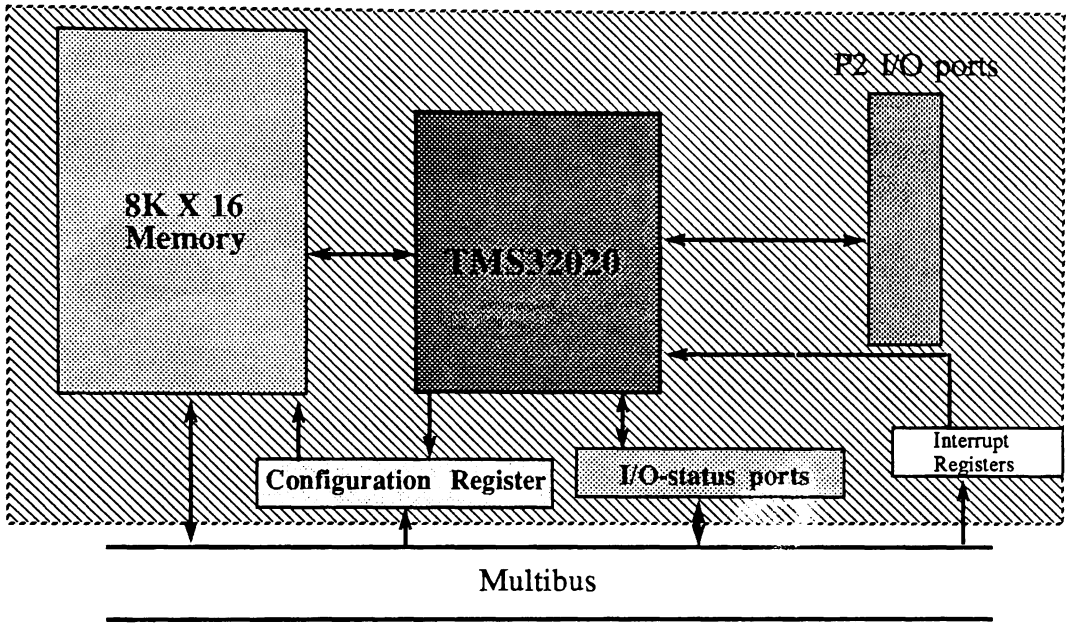


Figure 3.7 Functional Block Diagram of the DSP32020 Board

link between the two DSP units. A four level interrupt register allows the bus master to perform multi-level interrupts to the TMS32020 on the DSP32020.

3.4 Data Acquisition Subsystem

An analog to digital converter(A/D) and a digital to analog converter(D/A) comprise the data acquisition subsystem. Both converters operate at a 32KHz sampling rate with 14-bit resolution. They are linked to the DSP subsystem via the DSP32010's A/D and D/A I/O ports. This subsystem provides the interface between

the digital signals to be processed and the analog signals in the real world environment.

3.5 Hardware Design, Construction and Testing

The major hardware design, construction, and testing efforts were devoted to DSP sub-systems and were carried out in two stages. The DSP32010 board was designed and constructed first to gain familiarity with the Multibus interface. The DSP32010 board is designed with a minimum set of requirements to interface with the Multibus interface as a slave processor. These requirements are (1) Multibus address decoding, (2) Multibus memory and I/O command recognitions, and (3) command acknowledge signal generation. These features enable the board to be configured into the Multibus system as a slave processor. The design of the TMS32010's memory and I/O interfaces are relatively straight forward. The DSP32010's Multibus interface circuitry and its dual port memory are tested by the bus-master DB32016. Afterwards the bus-master loads the dual port memory with testing routines executable by the TMS32010 and flags it to perform expected functions. The DSP32010 is then connected to the data acquisition sub-system through the I/O ports PA5 and further real-time testing are conducted with test instrument observable results. A TMS32010 emulator was employed during the hardware debugging phase.

The DSP32020's Multibus interface design follows that of the DSP32010. The TMS32020's memory and I/O interfaces are slightly more complicated than the TMS32010's. While the TMS32020 allows more sophisticated memory and I/O interface controls to be performed, it does add an extra design effort of incorporating the additional control signals(HOLD, HOLDA, READY, STRB, IACK). The TMS32020

provides signals to access slower memory devices(not possible with the TMS32010). However, this feature is not implemented on the DSP32020 board to access slower speed devices. Rather the feature is used to enable the DB32016 to suspend the program execution on the TMS32020 while the DB32016 is accessing the dual port memory through the Multibus interface. During the TMS32020's program execution no wait state is necessary to access the fast static memory.

The two boards were constructed using half-height Multibus boards with the quick-connect technique for component connections. The quick-connect technique is similar to the wire-wrap technique, requiring no component-connection soldering. However, boards constructed using the quick-connect technique have lower board-heights compared with boards using the wire-wrap technique. The space constraint on a half-height Multibus board results two separate boards to house the TMS32010 and the TMS32020.

High speed(access time under 50ns) static memory chips are used on both the DSP32010 and the DSP32020 boards for simplicity of circuit design and full-speed execution of the TMS320s. Dual ported latches are used for the Multibus I/O ports and the P2-I/O ports. Therefore, transferring data on these ports must be synchronized by both input and output sides to ensure that no data is lost during the communication process.

Only the functionality tests were performed on the workstation hardware. The tests were sufficient to verify the intended design functionality of the hardware components. However, these tests are by no means to cover all the possible faults(component defects) in the system. The cost and availability of complete test generation tools prevent the more thorough test coverage.

Chapter 4 Software and User Interface

Advanced software is required to make efficient use of the workstation for DSP applications. The software tools discussed in this chapter enable the user to quickly translate DSP algorithms into programs executable on the workstation's DSP hardware. A user friendly interface makes the workstation operations transparent to the user. Program execution, control, and debugging on the workstation are as easy as on the host computer system.

4.1 DSP Program Development Tools

The variety of development tools available to the workstation enables the user to write programs in different languages and at different levels. The high level language programming makes program development fast and reliable while the lower level programming environment provides the user easy access to the various workstation's hardware resources. There are assemblers available for the TMS320s and the DB32016. And the C programming language is available for the DB32016.

To develop programs for the DB32016, the user writes most routines in the C. For hardware dependent tasks, assembler routines must be written to interface to the specific hardware components in the workstation. After the C source is compiled and the assembler codes assembled, these modules are linked together to a form executable image to be loaded into the DB32016. The DB32016 assembler and the C compiler, along with the linker, are available on a VAX host computer under the UNIX operating system.

The user codes DSP algorithms as the TMS320 assembler language routines and

uses the macro-assembler to translate the source into machine code. The macro-assembler, developed at the North Carolina State University[74], is a powerful assembler with an enhanced macro facility. The macro facility allows the user to write compact source programs and generate in-line codes for fast execution. The macro-assembler itself is written in the C and has been ported to UNIX, DOS, and VAX-VMS systems. The output format of the assembler is Intel checksum ASCII records. The records are easily transportable between systems and can be readily loaded into the workstation's DSP units.

4.2 Basic DSP Routines

To facilitate DSP program development, some basic DSP operations/algorithms are coded into subroutines/macro-definitions and can be easily incorporated into application programs. They are sinewave generator, IIR filter, Nyquist pulse shaping, random number generation, A/D-D/A interfaces, and LMS adaptive filter.

4.2.1 Sinewave Generator

The sinewave generator is implemented using a table lookup technique. Table size is chosen to be a power of two(2) for fast wrap around at the end of the table. There are two table setup methods. The first one uses a single table, consisting of sine values uniformly sampled around the unit circle. The accuracy and resolution are directly proportional to the table size. Table values are loaded into the TMS320's program memory space before the execution begins. Although the table values can be loaded into data registers for faster execution, the number of data registers available on TMS320s severely restricts the table size. On a TMS32010 there are only 144 data registers. Subtracting the several registers necessary for program overhead, the maximum table size cannot exceed 128. This give very poor frequency resolution for

the sinewave generator. Therefore, using the TMS32010 the program memory is used for table storage. These table values can be read into the registers by the TBLR instruction. In contrast, the TMS32020 has a large data memory space - up to 64K words. Therefore the table can be directly stored in data registers, eliminating the TBLR overhead.

The second method requires two tables, the first one, a coarse table, contains samples of a unit circle and the other, a fine table, contains values uniformly sampled between two samples of the first table.

Let

$$\alpha = \theta + \delta \tag{4.1}$$

using the trigonometric identity

$$sin(\alpha) = sin(\theta)\, cos(\delta) + sin(\delta)\, cos(\theta) \tag{4.2}$$

and let θ denote the coarse table angle and δ denote the fine table angle.

If δ is chosen to be sufficiently small, then

$$cos(\delta) \approx 1 \tag{4.3}$$

Therefore, (4.2) can be approximated by

$$sin(\alpha) \approx sin(\theta) + sin(\delta)\, cos(\theta) \tag{4.4}$$

Furthermore,

$$cos(\theta) = sin(\theta + \pi/4) \tag{4.5}$$

Thus $sin(\theta)$ and $cos(\theta)$ can be found in the coarse table and $sin(\delta)$ can be found in the fine table. The final result is obtained by one multiplication and one addition. The overall resolution of the implementation is

$$\frac{2\pi}{N \cdot n} \tag{4.6}$$

where N and n are coarse and fine table sizes, respectively.

The frequency resolution, f_r, can be computed by

$$f_r = \frac{F_s}{N \cdot n} \tag{4.7}$$

where F_s is the sampling frequency of the system.

The table sizes are chosen as follows. Since

$$\delta_{max} < \frac{2\pi}{N} \tag{4.8}$$

we choose N such that

$$cos(\delta_{max}) = 1 \tag{4.9}$$

in the fixed-point arithmetic. Let $N = 1024$. Then

$$\delta_{max} < 0.0061359231 \tag{4.10}$$

and
$$cos(\delta_{max}) > cos(0.0061359231) = 0.999981175274 \tag{4.11}$$

Therefore the right hand size of (4.11) is rounded to 1 using a 14-bit fixed-point arithmetic. This implies that (4.4) becomes

$$sin(\alpha) = sin(\theta) + sin(\delta)\, cos(\theta) \tag{4.12}$$

The choice of the fine table size n is then just a matter of meeting the desired frequency resolution.

By applying the trigonometric identity, much more accurate sine values has been obtained with less table values for both coarse and fine tables compared to the first method of a single table. A frequency resolution of 0.244 Hz can be achieved with 1024 samples in the coarse table and 128 samples in the fine table for a 32 KHz sampling system. This would require 131072 table values to achieve such a result with the single table approach.

The tables are generated with a program written in C. These table values are computed with floating point arithmetic for high accuracy, then they are quantized to 14-bit integers. The program can be executed on the host computer and then the table values are assembled into the TMS320 routine. The TMS320 routine is then loaded into the program memory. However, the DB32016 can also be used to generate the sine table values and directly load the table into the TMS320's memory at run-time. This means the table size can be changed dynamically to meet application's requirements.

Use of the sinewave generator is accomplished via a subroutine call by passing an angle α through the accumulator. The angle α is a 17-bit integer, which is decomposed into θ(10 bits) and δ(7 bits) by the subroutine. The corresponding table value is returned in the accumulator to the calling routine. The frequency can be computed by

$$f = \frac{F_S}{N \cdot n} \cdot \alpha \qquad (4.13)$$

4.2.2 IIR Filter with Normalized Lattice Structure

The infinite-impulse-response(IIR) filters are implemented using a normalized lattice structure. A single lattice structure is coded as a macro definition. The user designs a filter and converts the filter coefficients to second order pairs. These coefficients and the order of the filter is then given to the filter subroutine to be assembled by the macro-assembler. The macro preprocessor first expands the filter sections according to the filter order. Then the assembler assembles the expanded in-line codes into an executable program. Samples can be filtered by passing through the accumulator into the subroutine and the filtered output samples are returned from the

accumulator as well.

4.2.3 Nyquist Pulse Shaping

Nyquist pulse shaping is accomplished by using a raised cosine envelope to shape the data pulse. The raised cosine envelope is sampled in the time domain and symmetrically truncated to five periods, with two periods each at both sides of the

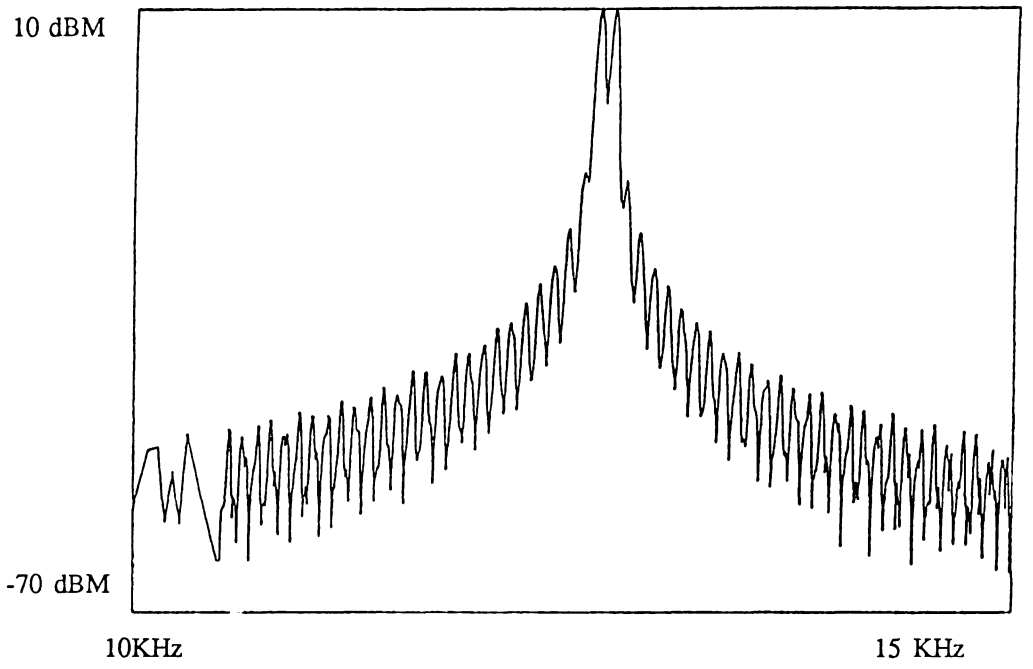


Figure 4.1 Pulse Shaping with BPSK Modulation
Spectrum of a Real-Time Signal

center peak. The samples are stored in a table in truncated 14-bit integer format. To make the five sample overlap, a delay line of size five is allocated to store and shift five data pulses. Five pointers are used to keep track of where about each pulse falls on the envelop. At each system sampling instant, the five points on the envelope are multiplied by their corresponding data pulse values(one's or minus one's) and the results summed up as the output. More overlap of samples would require a large

table to store the envelop data samples and the longer delays for data pulses.

Figure 4.1 shows the spectrum of a BPSK modulated signal with pulse shaping.

4.2.4 Random Bit Generator and A/D-D/A Interface

The random number generator is implemented to simulate a random data source. It is implemented using fixed-point arithmetic with 16-bit precision. The routine saves a number called seed and uses it to generate the next random number with a new seed. The remainder operation is done by taking only the lower-order bits from the accumulator. The routine can be used to generate binary data, rather than arbitrary integers, to be encoded for transmission tests.

Because the A/D and D/A data samples are in offset binary format and the TMS320s perform operations in two's complement, data samples must undergo the proper conversions. The A/D-D/A interface macros make the conversion transparent to the user. The data samples received from the A/D are first converted from offset binary to two's complement. After the samples are processed, they are converted back from two's complement to offset binary for the D/A to output.

4.2.5 LMS Adaptive Filter

The adaptive filter is implemented as an LMS transversal filter structure. The filter produces an output $y(T)$ from a weighted sum of delayed input samples.

$$y(T) = \mathbf{W}^T(T) \cdot \mathbf{X}(T) \tag{4.14}$$

where $\mathbf{X}(T)$ is a vector of delayed input samples and $\mathbf{W}^T(T)$ is the transpose of the

filter coefficients. The transversal filter structure is illustrated in figure 4.2.

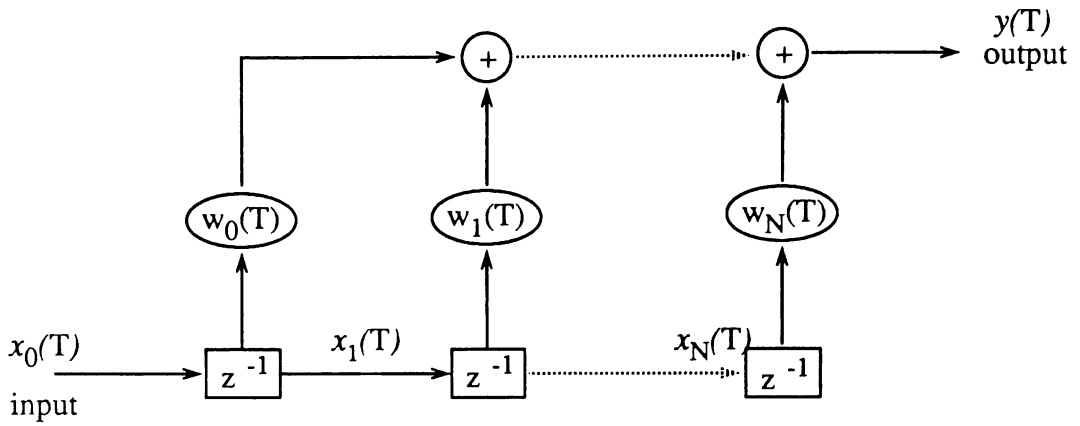


Figure 4.2 Transversal Filter Structure

Let the error

$$e(T) = d(T) - y(T) \tag{4.15}$$

be the difference between the desired response and the filter output. The goal is to update $\mathbf{W}(T)$ periodically such that the error $e(T)$ is minimized. By the LMS algorithm,

$$\mathbf{W}(T+1) = \mathbf{W}(T) + \alpha \cdot e(T) \cdot \mathbf{X}(T) \tag{4.16}$$

where α is the feedback gain constant that affects the stability and the rate of convergence of the filter coefficients. Moreover,

$$\alpha < \frac{2}{\lambda_{max}} \tag{4.17}$$

where λ_{max} is the eigenvalue of the input's autocorrelation function.

The LMS adaptive operation is depicted in figure 4.3.

Following the Barnes' implementation[23] on the TMS32010, the filter routine has to be assembled according to order and number of taps. The basic filtering operation is coded as a macro definition. The user needs only to specify the order of

the filter. The macro assembler will generate in-line codes for a particular design.

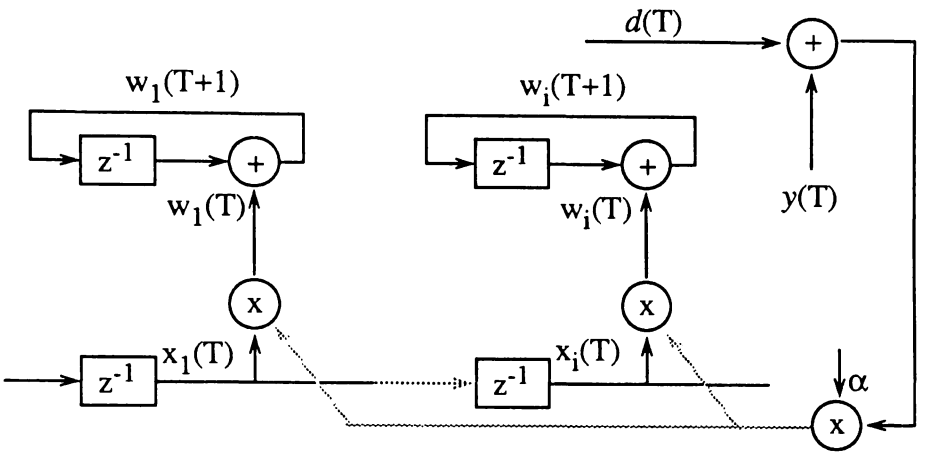


Figure 4.3 Adaptive Operations to Update Weight Coefficients

The filter building block has been modified for the TMS32020, utilizing the available hardware resources to improve the performance. The filter uses TMS32020's auxiliary registers to perform looping control, eliminating the need for in-line code execution. TMS32020 has special instructions to perform looping control without adding overhead. These instructions are highly optimized for pipelining. By taking advantage of these advanced features on the TMS32020, the filter requires less program memory than the previous implementation. By using loop control to determine the size of the filter the implementation allows the user to alter the filter order after the program is assembled. Therefore, the user can change the order of the filter at run time, making it more flexible. Appendix B lists the assembler program source code that implements the adaptive filter on the TMS32020.

4.3 Workstation User Interface

The user interface to the workstation is the debugger dbg16 running on a VAX host computer with the UNIX operating system and the ROM monitor on the DB32016 board. The user develops programs on the host system. Then the programs

are loaded into the workstation using the debugger. The execution control is accomplished through the monitor commands. The user can issue commands either to the monitor directly or to the host computer and let the debugger perform translations. Appendix A describes the detail of the workstation's operations.

4.4 TMS320 simulator

The TMS320 simulator, developed at the North Carolina State University[75], is yet another powerful simulation tool for TMS320 program debugging and execution simulation. The simulator is compatible in its object code format with the macro-assembler by using the Intel checksum records. The simulator features program tracing, break point setup/removal, single/multiple step execution, on-line assembler/dis-assembler, memory poking, hex/decimal formatted I/O, and input file wraparound facility. It can execute both the TMS32010 and TMS32020 programs. The simulator is available on Unix, DOS, and VAX-VMS systems.

Chapter 5 Application to Distribution Power Line Communications

The use of power lines as a transmission media poses some unique problems which have received considerable attention[22]. The major problem is the high noise level on the distribution lines. The noise appears mainly as 60 Hz based harmonics. An effective technique for combating this distortion has been studied and will be implemented on the workstation to demonstrate its feasibility. Other aspects of digital implementation of DLC are also discussed.

5.1 Formulation of the Problem

Applications of power line communication include load management, remote meter reading, distribution equipment monitoring and control, and possibly voice communications. These applications face the same type of threat as any other type of communication systems, primarily noise. The noise on the power lines comes from spikes induced by switching devices with various loads. These spikes are synchronous with the 60 Hz carrier, and appear as harmonics. The traditional approach is to transmit a signal with very limited bandwidth and to use fixed bandpass filters to remove the harmonics. However, with the 60 Hz carrier shifting over time and the bandwidth of the harmonics shifting, these fixed filters are often ineffective.

Adaptive filters are quite suitable for this type of application[73]. An adaptive filter can continuously adjust its response to variations in the input signal so that the output converges to a desired steady state. However, implementation of adaptive

filters in analog form is very costly if not impossible. With the dramatic reductions in DSP hardware costs, the implementation of adaptive filters has come within the reach of researchers and system designers. Integrating adaptive filters to perform harmonic noise cancellation in power line communication systems has become a practical solution to deal with the hostile environment.

Here we present how an adaptive filter implemented to perform harmonic cancellations in a digital receiver system. Typically, the incoming signal is first sampled at an appropriate sampling rate. The sampled data are bandpass filtered and demodulated to extract the baseband signal. Then the appropriate decimation is done to lower the sampling rate, followed by passing the data samples through an adaptive filter to remove the correlated 60 Hz harmonics before the final signal detection process.

The following sections discuss the building blocks of a digital communication system for DLC and the implementation issues involved.

5.2 Adaptive Harmonic Cancellation

The basic structure of an adaptive filter for harmonic cancellation is illustrated in Figure 5.1. The aim of an adaptive filter is to estimate, or predict, a signal using its past samples. An error signal is derived by taking the difference between the original signal and the predicted signal. This error signal is then used to adjust the filter's response, by altering the filter coefficients, such that the error(in a least square or least mean square sense) is minimized. The principle behind the harmonic cancellation with an adaptive filter is based on the theory of linear prediction[8]. The future time behavior of a statistically stationary signal can be predicted based on the signal's past samples. A broadband signal has a narrow auto-correlation function and is therefore more difficult to predict compared to a narrowband signal, which has a

wide auto-correlation function and is relatively easy to predict from the past samples. The delay queue de-correlates the broadband signal to allow the adaptive filter to predict only the narrowband signal.

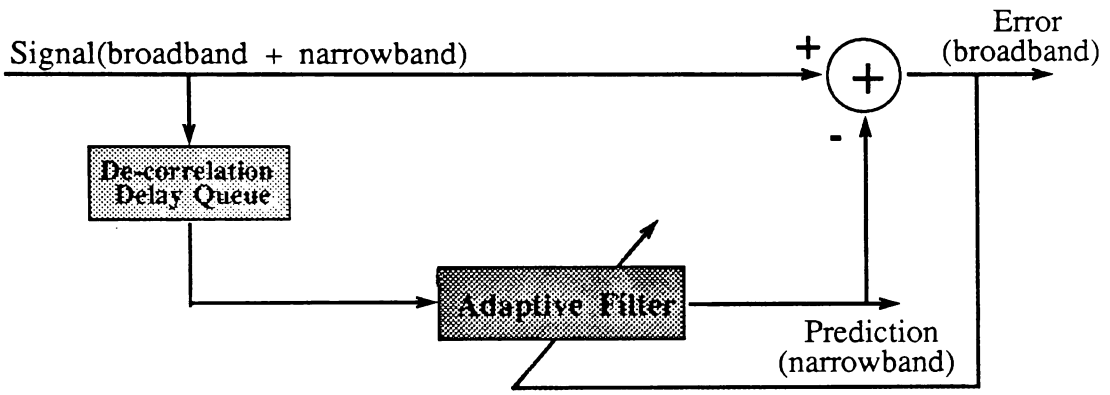


Figure 5.1 Adaptive Filter Structure

The incoming signal components consist of encoded data and 60 Hz harmonics. The encoded data component is such that it appears as a broadband signal, i.e., it is highly un-correlated. The harmonics are obviously highly correlated. Therefore, it is possible to make a prediction based on the past samples of the signal. The prediction is a close replica of the narrowband signal, harmonics of 60Hz. Subtracting this predicted signal from the incoming signal, the data signal is obtained with improved signal-to-noise ratio.

The filter is implemented on the DSP32020 and tested as a separate task. The test version uses ten(10) registers for the delay queue and fifty four(54) taps, the maximum number of taps that can be implemented at 8 KHz sampling rate. The filter coefficients and delayed samples are stored in separate pages on the TMS32020's on-chip data registers. Four auxiliary registers are used to allow the TMS32020 to

execute looping instructions automatically. Highly optimized pipelining instructions can be executed faster than in-line code type of implementations. Furthermore the pipelined looping control implementation saves a large amount of program memory. Therefore the program can be loaded into the program memory much quicker. The test results, obtained by running the filter at 8KHz sampling rate, are shown in Figure 5.2.

The adaptive filtering is to be performed after the demodulation so that the sampling rate can be reduced down to a rate covering only the baseband. At the lowered sampling rate, there is sufficient time between samples to allow the TMS32020 to execute higher order filters and perform many other tasks for the system.

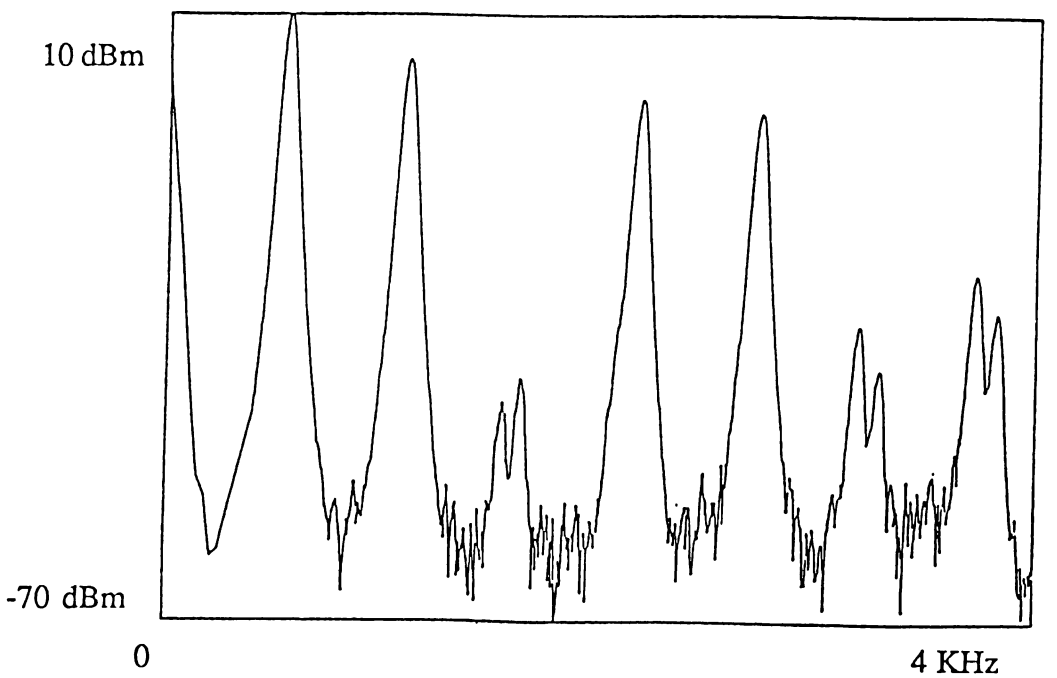
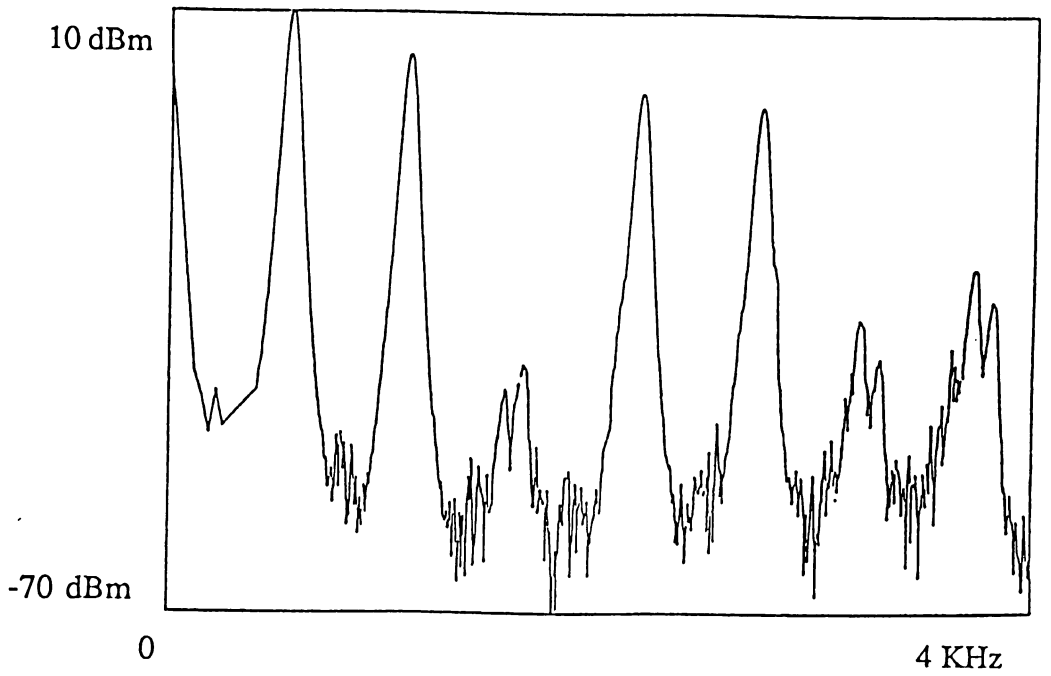
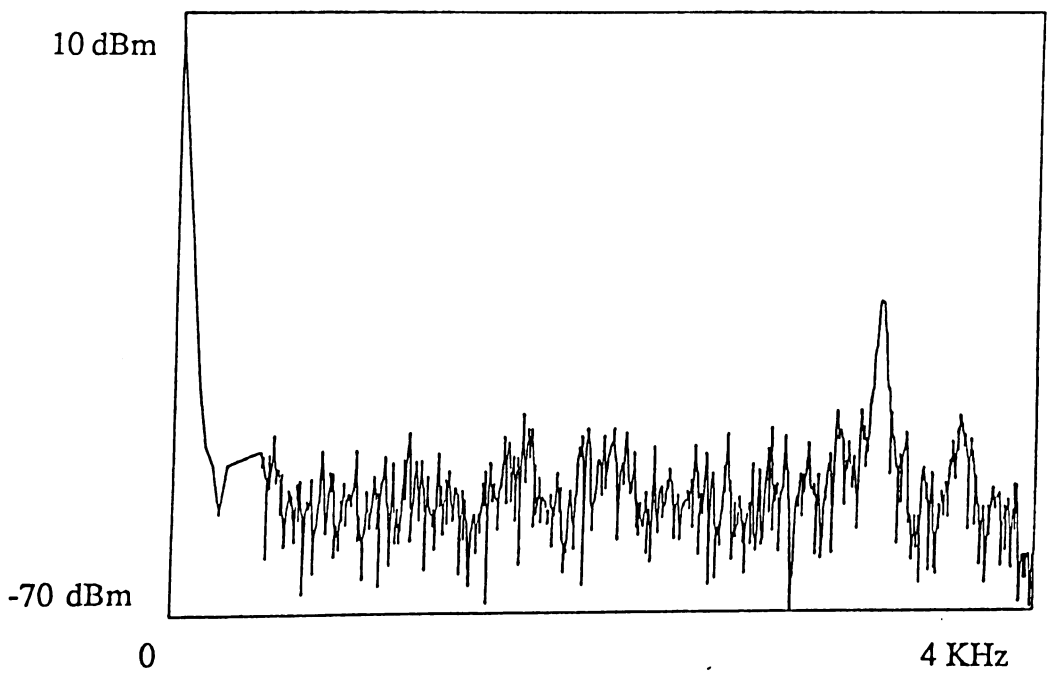


Figure 5.2 (a) Spectrum of the Incoming Signal



(b) Spectrum of the Predicted Signal



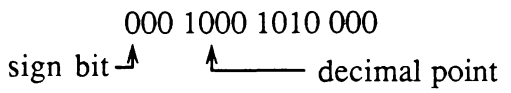
(c) Spectrum of the Error Signal

Figure 5.2 Adaptive Filter for Harmonic Cancellations

5.3 Finite Precision Algorithm

Since TMS320s have a 16-bit register lengths and only integer arithmetic is supported by the hardware, DSP operations have to be done in finite precision with fixed-point arithmetic. It is very important to choose an appropriate scaling factor to avoid overflow while keeping maximum possible accuracy.

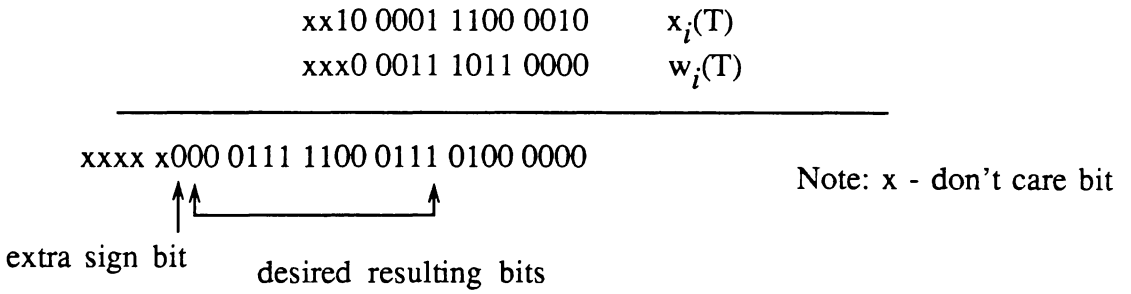
The fixed-point arithmetic is implemented by assigning i-bits for the integer part and 15-i bits for the fractional part. The most significant bit is reserved for the sign bit. The decimal point is implied. For example, the number



represents a value of 2.03125.

In the workstation the A/D and D/A converters are 14-bit devices. Therefore it is more convenient to keep data samples at 14-bit precision throughout the processing.

The adaptive filter coefficients are kept at 13-bits. The transversal operation(refer to section 4.2.5) $\mathbf{W}^T(T)\cdot\mathbf{X}(T)$ will produce N(the number of taps) 27-bit products, which are summed up to obtain $y(T)$.



Notice that there are two sign bits in the product. The desired 14 bits are in bit 12 to bit 25 in the accumulator of the TMS320. The TMS320's SACH instruction left shifts the result to store the upper half of the accumulator. In this implementation we

need to left shift 4-bits to obtain the desired results. This shifting process causes the coefficients being scaled by a factor of 2^{12}. The true values of $\mathbf{W}(T)$ can be calculated by multiplying each $w_i(T)$ by 2^{-12}.

In the LMS adaptive operation of the filter

$$\mathbf{W}(T+1) = \mathbf{W}(T) + \alpha \cdot e(T) \cdot \mathbf{X}(T) \qquad (5.1)$$

we have

$$w \cdot 2^{12} = w \cdot 2^{12} + (\alpha \cdot e) \cdot x \cdot 2^{-12} \qquad (5.2)$$

In order to the scaling consistent in (5.2) α has to be scaled by a factor of 2^{36}. In the fixed-point implementation,

$$\alpha = 256$$

which corresponds to a real value of 2^{-28}.

Therefore, the 14-bit data samples from the A/D converter are multiplied by the 13-bit filter coefficients and the products are shifted by the appropriate number of bits to obtain 14-bit results. The 13-bit coefficient implementation is based on the fact that TMS320s only support one(1) and four(4) bit left shift scaling of the accumulator. The 13-bit implementation requires only one shift operation after each multiplication to keep the results at the correct number of bits. Higher precision coefficients can be implemented with additional execution overhead to obtain the correct results after multiplication operations. Both truncation and rounding are implemented.

5.4 Phase Looked Loop and Decimation

In order to demodulate the incoming signal, a local oscillator, which generates a sinewave of the same frequency as the carrier, is needed. This can be accomplished by using the sinewave generator described earlier. To take advantage of coherent detection, the sinewave generator must be synchronized in phase with the carrier

signal, so that the local oscillator tracks the incoming signal in frequency as well as in phase. This is done by implementing Phase Locked-Loops(PLL)[41-42]. The phase difference between the local and incoming signals is detected. This phase difference is then filtered to remove high frequency noise and is used to control(or trigger) the local oscillator such that the phase difference is minimized. The implementation of PLL's in digital form has been an ongoing effort of research and development for DSP algorithms. Work done with both computer simulation and real time implementation shows that this is a promising but non trivial task.

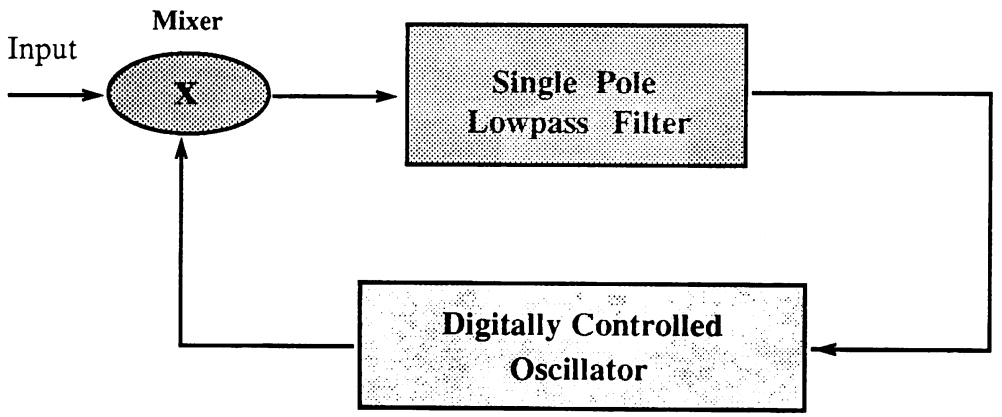


Figure 5.3 Digital Phase-Locked Loop(DPLL)

The digitally controlled oscillator(DCO) was implemented using the high resolution sinewave generator descried in section 4.2.1. The 16x16 bit multiplier is used to simulate the mixer. The 32-bit product from the mixer is directly fed into the single-pole lowpass filter which keeps 32-bit precision to maintain accuracy and the dynamic range of the DPLL. Figure 5.4 and 5.5 illustrate the performance of the DPLL

implemented on the workstation's DSP32020 board.

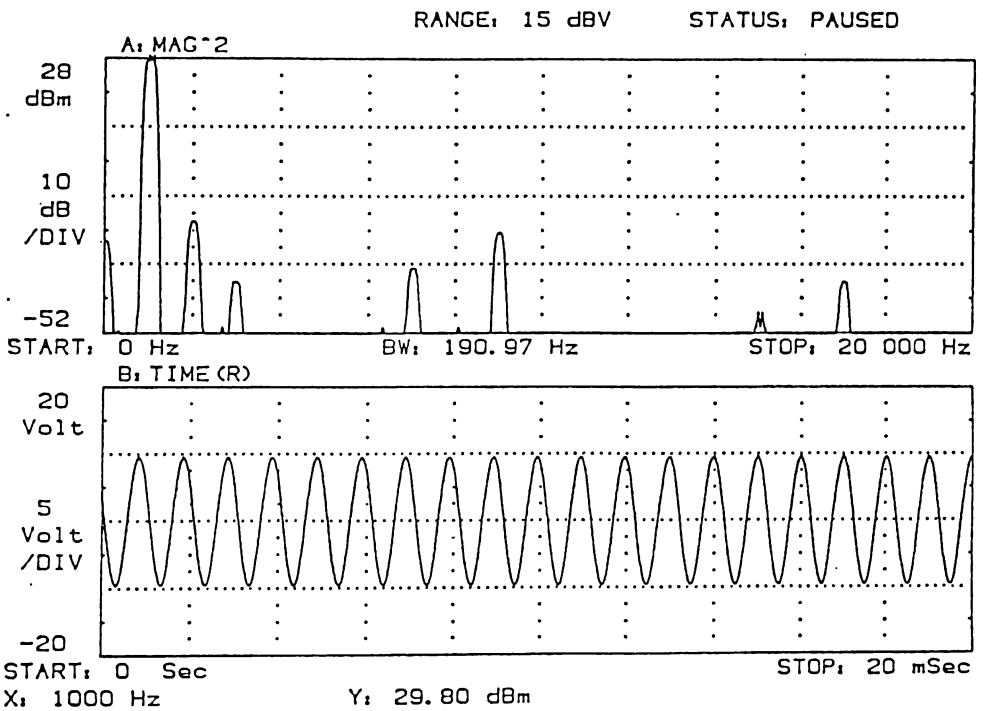


Figure 5.4 The Spectrum of DCO set at 1KHz
note the harmonic distortion is 50 dB down

After demodulation, the baseband signal can be processed at a lower sampling rate[24-25]. At this sampling rate the time interval between samples permits extensive processing of the signal. Because of the nature of the sampling process, the lower sampling rate has to be obtained through a sampling rate transformation process called decimation.

Basically, decimation is the sampling of a digital signal at a rate that is lower than the original rate. This process has to obey the laws of sampling theory just as in

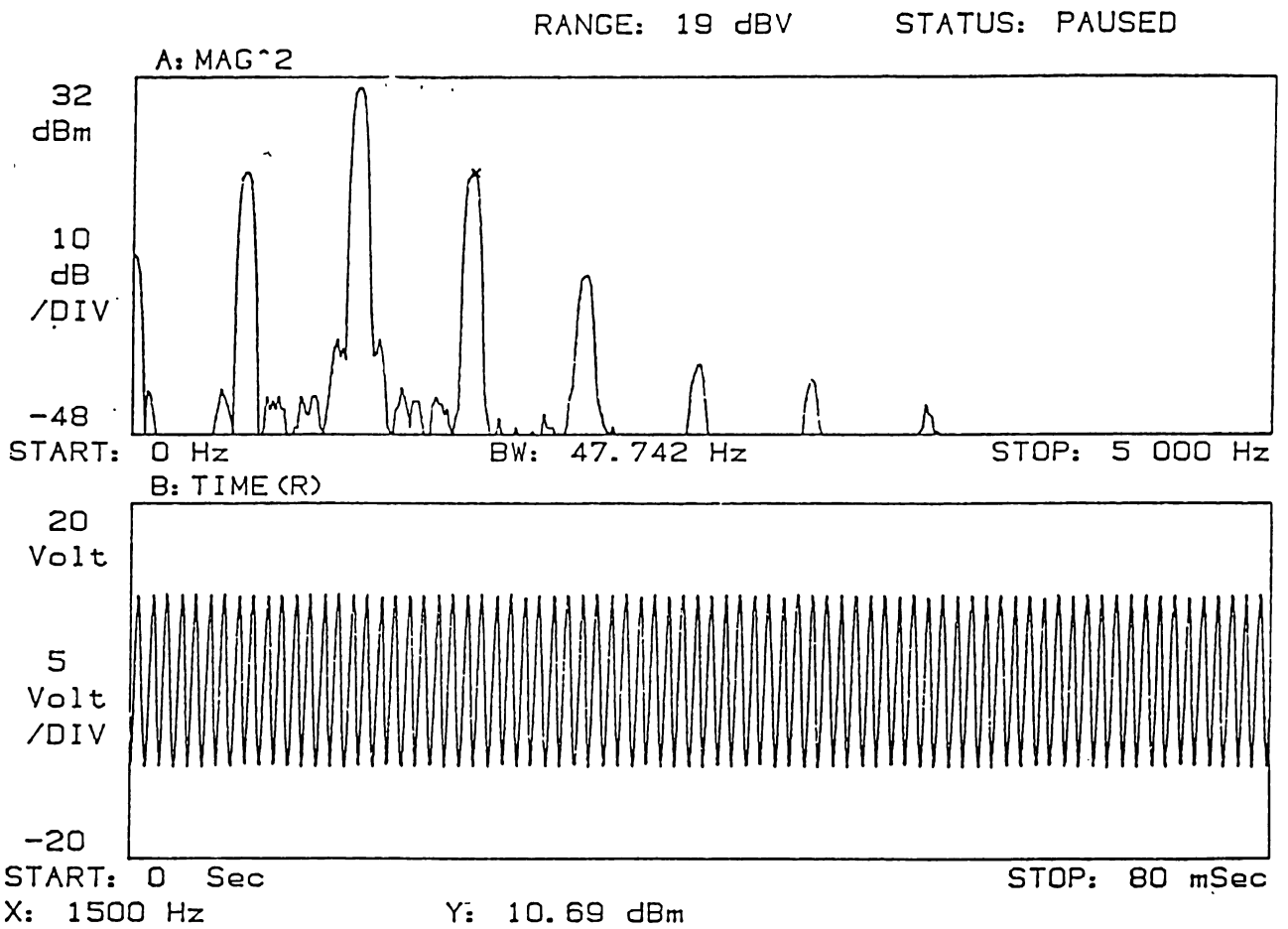


Figure 5.5 DCO Output with 500 Hz Sinusoidal Input
(FM modulation)

sampling an analog signal. Therefore, Nyquist filtering is needed before sampling the signal at the desired rate. Since the original signal is already in digital form, filtering and sampling operations can be combined to allow more efficient implementations. In deciding which implementation to use, it is necessary to consider the most suitable lower sampling rate, the bandwidth of the baseband signal, hardware performance and efficiency.

5.5 All Digital Communication System

Figure 5.6 illustrates a complete all-digital receiver system(not including the A/D converter). It consists of a bandpass filter to eliminate out of band noise, a digital phase-locked loop to generate a local carrier synchronized with the carrier signal, a perfect multiplier as the demodulator, a decimation filter to lower the sampling rate,

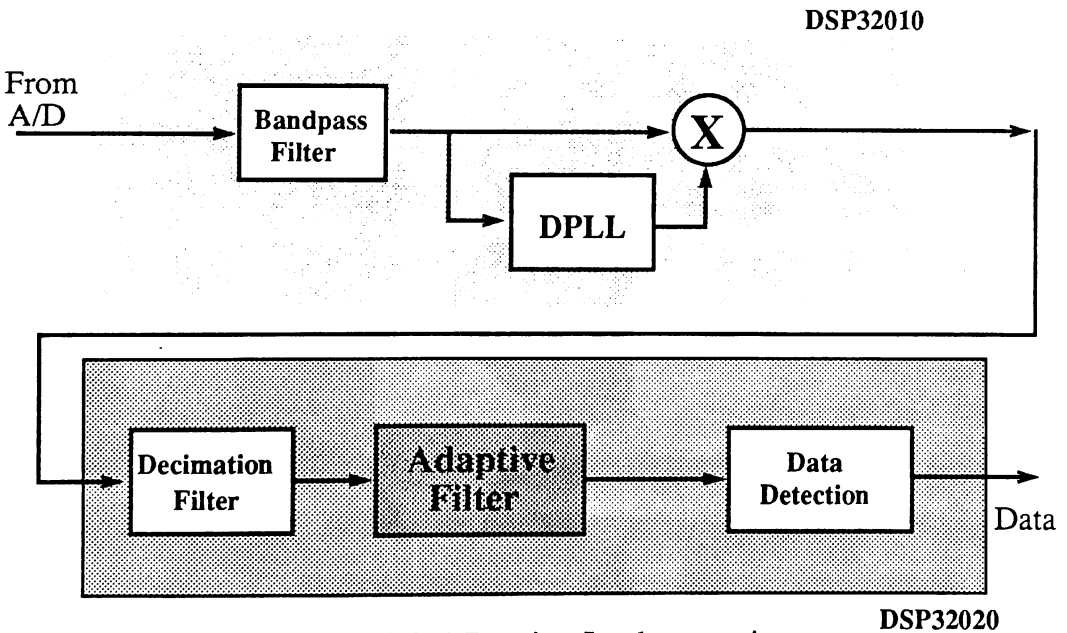


Figure 5.6 Digital Receive Implementation

an adaptive filter to separate the data signal from the harmonic noise, and a data detector. The processing tasks are divided between the DSP32010 and DSP32020, as shown in Figure 5.6.

A transmitter includes partial response coding of the data[39-40], pulse shaping, modulation, and bandpass filtering. Figure 5.7 illustrates the build blocks for a digital transmitter.

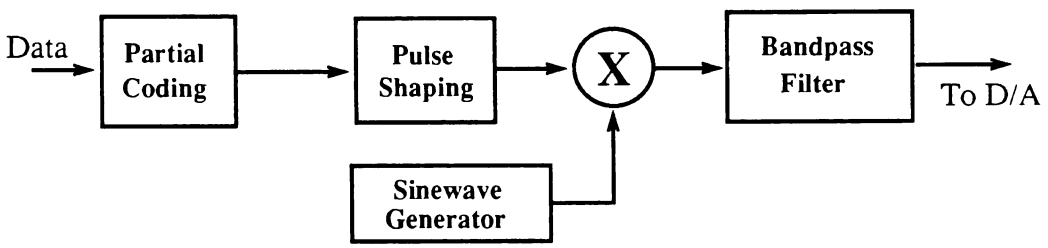


Figure 5.7 Digital Transmitter Implementation

The workstation allows the all-digital communication system to be implemented at a 32 KHz sampling rate in real-time. Since each TMS320 can execute five million instructions per second, ten MIPS(million instructions per second) performance is attainable by operating the DSP32010 and DSP32020 concurrently. The performance of the system can be directly measured in real-time(not possible with computer simulation). The workstation also offers the system designer the hardware capabilities(such as concurrent processing) that are not available from commercial DSP products.

Chapter 6 Summary of Achievements and Future Research

6.1 Summary of Achievements

A complete operational DSP workstation has been designed and implemented. The system has gone through extensive functional verifications by exercising all the hardware components using conventional hardware testing methods as well as running various programs. The workstation features concurrent processing with two loosely coupled DSP processors to achieve ten MIPS of DSP operations. High speed communications between the two DSP processors is achieved through a dedicated data channel. A general purpose microcomputer sub-system performs major workstation control functions. A floating point co-processor is available on the DB32016 to perform high precision computations at a relatively fast rate. The system offers 384K bytes of RAM for general purpose use and 24K bytes of RAM for DSP applications. The dual ported memory of the DSP boards allows the bus-master to perform direct-memory-access(DMA) on these memories. Non-DMA access can also take place using the Multibus I/O ports on the DSP boards. The memory configuration register on the DSP32020 allows the system to perform fast context switch during DSP program executions. A rich collection of software development tools and DSP routines has been developed. The development tools and library routines allow the user to quickly write DSP application specific programs.

The hardware expansibility of the workstation, discussed in the following sections, makes it an ideal candidate for future DSP research and development.

The workstation operating procedures have been established and are presented in the Workstation User Manual(see Appendix A).

6.2 Future Research

There are three major areas for future research and development. They are distribution line carrier communications, workstation expansion, and DSP algorithm implementations.

6.2.1 DLC Implementation

The implementation of DLC systems on the workstation has just began. Specifically the following areas need further development.

(1) Decimation Filter : Determine the optimal filter structure and sampling frequencies suitable for decimation in DLC applications. The data rate will determine the minimum sampling rate allowable. The available processing time will dictate how complex the filter can be.

(2) Digital Phase-Locked Loop : Define the interface between the VCO, phase detector, and lowpass filter. Given this interface, each of the functional blocks can be implemented and optimized separately for speed, performance, and efficiency. A stability analysis of the digital PLL must also be performed.

(3) System Integration : Implement and test a complete transmitter/receiver system on the workstation. Plans for field testing should also be considered.

(4) Error Rate Determination : Performed error rate testing on the workstation using the random bit generator as the data source for the transmitter. Some of the overhead tasks can be carried out by the DB32016. The test can be automatically controlled by the microcomputer subsystem and the results reported.

6.2.2 Workstation hardware Expansions

(1) Video Display SubSystem : A Multibus compatible video controller and display terminal would further enhance the workstation's functionality. Such a subsystem would allow the workstation to perform real time signal analysis, displaying signal waveforms and analysis results. For instance, the user could collect samples from the A/D and perform an FFT for spectral analysis. Then the time domain and spectral information can be sent to the display subsystem for viewing.

(2) Mass Storage Sub-System : Eventually, the workstation can be expanded to include a mass storage subsystem for the software data base, independent of any host computer. There are variety of disk/tape controllers available for Multibus to choose from. With the addition of a storage subsystem, the user could easily collect real time signals from the field and store them for later analysis.

(3) Memory Expansion : The memory of the system could be expanded to one megabyte without using the P2 bus address lines. If a large memory space is needed on the workstation, The P2 bus must be included in address decoding. This will enable up to sixteen megabytes of memory space to be addressed in the system. Certain applications may require large amounts of memory to meet speed constraints, such as in FFT operations.

(4) Next Generation DSP Processors : The new wave of DSP processors provides a hope of using floating point operations instead of fixed-point arithmetic[21][29][44-45]. These new chips, such as Texas Instruments TMS320C30 and the NEC μPD77C25, have single cycle floating point instructions implemented in hardware. These chips open up a new array of possibilities for fast DSP algorithm implementations. The new DSP chips can be integrated into the workstation the same way the existing DSP units have been implemented.

6.2.3 Additional Support Software

Additional software tools would make the workstation more powerful. These tools can be developed on the host computer.

(1) Graphic Routines : Some basic curve plotting routines are essential for the user to examine the simulation results in a graphical way. The development effort can be reduced by porting available general purpose plot routines, such as GNUPLOT.

(2) On-line Assembler/Disassembler : There are times when the user may wish to alter DSP programs after they is loaded into the DSP units' memory. To avoid re-compiling and reloading, it would be more convenient to poke the memory directly and change the code where needed. An on-line assembler/disassembler running on the workstation itself, would be a handy tool to do the job.

(3) DSP Processor Debugger : The above concept can be expanded to developing a debugger for the DSP processors, making real time DSP debugging possible.

(4) Stand-Alone Operating System : To make the workstation fully independent of any host computer assistance, a complete operating system must be developed to support the user environment. A suitable choice would be the Unix operating system.

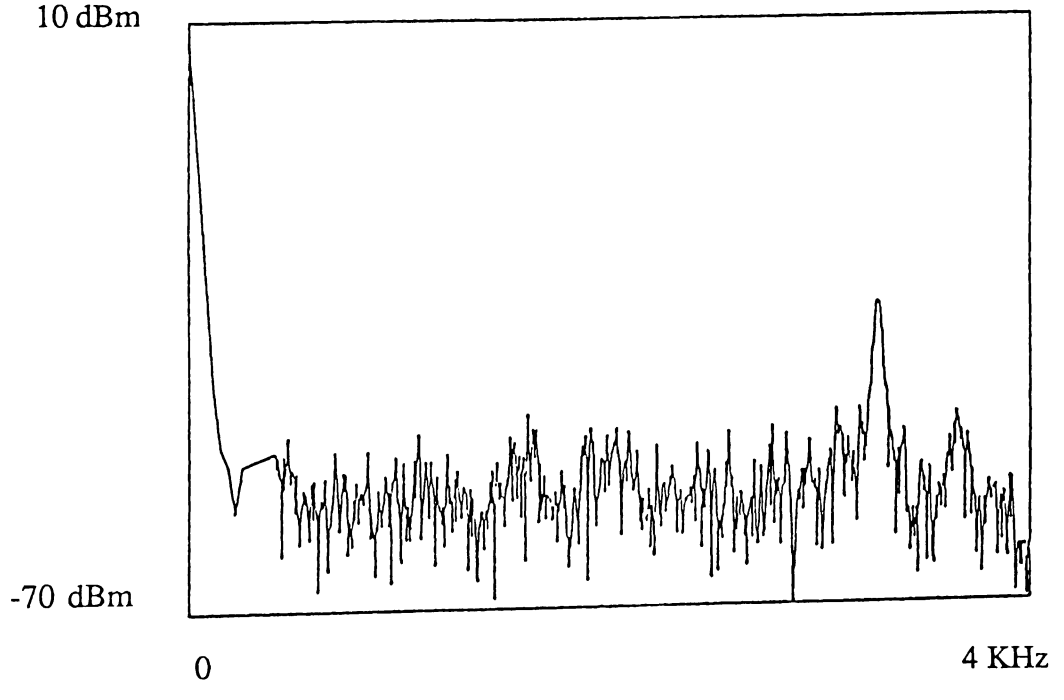
6.2.4 DSP Algorithm Implementations

There are several subjects related to DSP algorithm implementations that would be suitable for in-depth study.

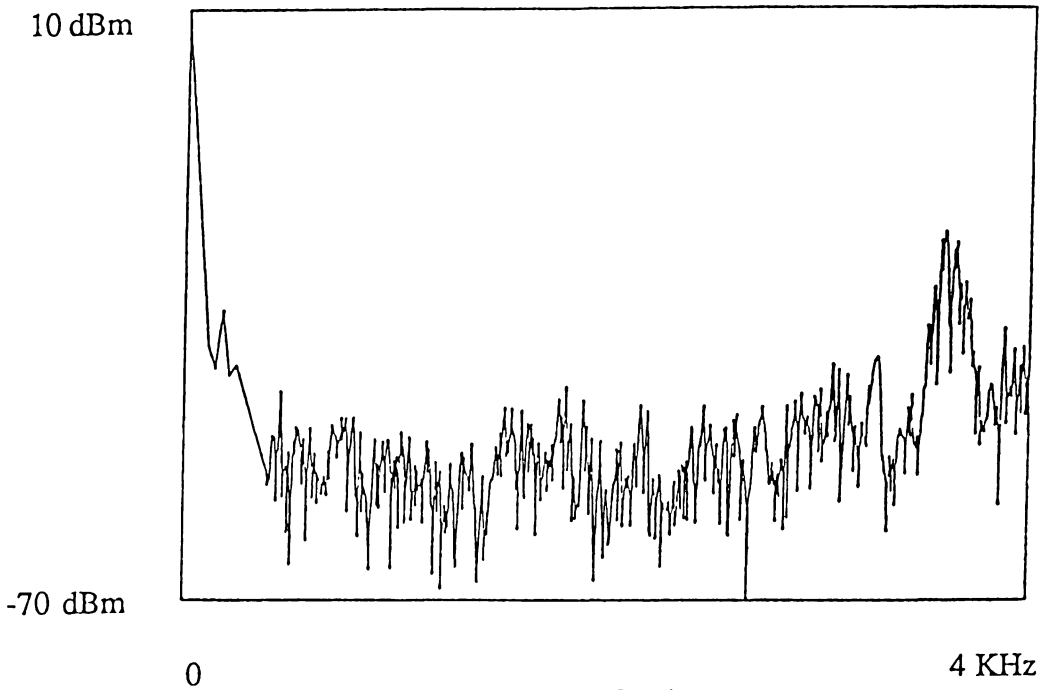
(1) Limit Cycles : This is a problem with the finite precision arithmetic operations: a filter's response oscillates even when there is no input stimuli. Although a truncation algorithm may solve this problem, truncation introduces additional noise into the system under normal conditions.

(2) Filter Tap Drift : Filter tap drift in adaptive filters, as illustrated in Figure 6.1, was observed on the workstation after the filter started normal operation for more than twenty minutes. More study needs to be conducted to gain a better understanding of the nature of the problem.

(3) Phase Locked-Loop : It remains a great challenge to implement a digital phase locked-loop entirely in DSP software. The performances of different types of DPLL implementations are yet to be studied.



(a) Initial spectrum after filter converges



(b) Filter spectrum after 30 minutes

Figure 6.1 Error Signal's Spectrums of Adaptive Filter

REFERENCES

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structures and Algorithms,* Addison-Wesley Publishing Company, Reading, MA, 1983.

[2] M. R. Baraniecki, A. Z. Baraniecki, R. Kumaresan, and M. Shridhar, "*Multiprocessor System for Speech Processing and Telecommunications,*" Proceedings of IEEE ICASSP 84, May, 1984, pp. 25A.6.1-4.

[3] J. M. Cioffi and T. Kailath, "*Fast, Recursive-Least-Squares Transversal Filters for Adaptive Filtering,*" IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-32, No. 2, April, 1984, pp. 304-337.

[4] T. A. C. M. Claasen and W. F. G. Mecklenbrauker, *"Adaptive Techniques for Signal Processing in Communications,*" IEEE Communications Magazine, Vol. 23, No. 11, Nov., 1985, pp. 8-19.

[5] L. J. Faber and T. K. Miller, *"A Multiprocessor Configuration for Transversal Adaptive Filters,*" CCSP-WP-83/18, Dept. of Electrical and Computer Engineering, NCSU, June,1983.

[6] D. D. Falconer and L. Ljung, "*Application of Fast Kalman Estimation to Adaptive Equalization,*" IEEE Transactions on Communications, Vol. COM-26, Oct., 1978, pp. 1439-1446,

[7] INMOS Limited, *Advance Information Data Sheet* - IMS T424 Transputer, UK, April, 1984.

[8] J. M. McCool and Bernard Widrow, "*Principles and Applications of Adaptive Filters: A Tutorial Review,*" IEEE Conference Paper CH1564-4/80/0000-1143, April, 1980, pp. 1143-1157.

[9] D. G. Messerschmitt, "*Echo Cancellation in Speech and Data Transmission,*" IEEE Journal on Selected Areas in Communications, Vol. SAC-2, No. 2, March, 1984, pp. 283-297.

[10] T. K. Miller and S. T. Alexander, "*An Implementation of the LMS Adaptive Filter Using an SIMD Multiprocessor Ring Architecture,*" Proceedings of IEEE ICASSP 85, Vol. 4, March, 1985, pp. 1605-1608.

[11] T. K. Miller and S. H. Ardalan, "*A Multiprocessor Configuration for the Adaptive Fast Kalman Algorithm,*" to be presented at the IEEE International Conference on Communications, June, 1986.

[12] D. P. Morgan and H. F. Silverman, "*An Investigation into the Efficiency of a Parallel TMS320 Architecture: DFT and Speech Filterbank Applications,*" Proceedings of IEEE ICASSP 85, Vol. 4, March, 1985, pp. 1601-1604.

[13] L. R. Morris, *Digital Signal Processing Software*, DSPS Inc., Ottawa, Canada, 1983.

[14] NEC Electronics, Inc., *Advanced Product Information Data Sheet* - μPD77230 Advanced Signal Processor, Feb., 1986.

[15] A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1975.

[16] H. J. Siegal, "*Interconnection Networks for SIMD Machines,*" Computer, Vol. 12, No. 6, June, 1979, pp. 57-65.

[17] L. J. Siegel, *"Highly Parallel Architectures and Algorithms for Speech Analysis,"* Proceedings of IEEE ICASSP 84, May, 1984, pp. 25A.1.1-4.

[18] Y. G. Tao, K. D. Kolwicz, C. W. K. Gritton, and D. L. Duttweiler, *"A Cascadable VLSI Echo Canceller,"* IEEE Journal on Selected Areas in Communications, Vol. SAC-2, No. 2, March, 1984, pp. 297-303.

[19] Texas Instruments, Inc., *TMS32010 User's Guide*, Dallas, TX, May, 1983.

[20] Texas Instruments, Inc., *TMS32020 User's Guide*, Dallas, TX, 1985.

[21] Steven L. Martin, *Wave of Advances Carry DSPs to New Horizons*, Computer Design, September 1987, pp. 69-83

[22] Jin-Der Wang, *Analysis of Adaptive Filter Algorithms with an Application to Harmonic Noise Cancellation in Distribution Power Line Communications,* Center for Communications and Signal Processing, Dept. of Electrical and Computer Engineering, NCSU, August 1985

[23] R.L. Barnes, *Design and Implementation of a Multiprocessor Architecture for Adaptive Digital Filters*, Center for Communications and Signal Processing, Dept. of Electrical and Computer Engineering, NCSU, July 1986

[24] Ronald E. Crochiere and Lawrence R. Rabiner, *Interpolation and Decimation of Digital Signals - A Tutorial Review,* Proceedings of the IEEE, vol. 69, no. 3, March 1981, pp. 300-331

[25] David J. Goodman and Michael J. Carey, *Nine Digital Filters for Decimation and Interpolation,* IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. ASSP-25, no. 2, April 1977, pp. 121-126

[26] James C. Candy, Bruce A. Wooley, and Oconnell J. Benjamin, *A voiceband Codec with Digital Filtering,* IEEE Transactions on Communications, vol. COM-29, no. 6, June 1981, pp. 815-830

[27] V.B. Lawrence and A.C. Salazar, *Finite Precision Design of Linear-Phase FIR Filters,* American Telephone and Telegraph Company, The Bell System Technical Journal, vol. 59, no. 9, November 1980, pp. 1575-1598

[28] T.A.C.M. Claasen and W.F.G,. Mecklenbräuker, *Adaptive Techniques for Signal Processing in Communications,* IEEE Communications Magazine, vol. 23, no 11, November 1985, pp. 8-19

[29] John Bond, *A Potpourri of DSP*, The Electronic System Design Magazine, January 1988, pp. 67-80

[30] Neal K. Riedel, David A. McAninch, Cameron Fisher, and Nahum B. Goldstein, *A Signal Processing Implementation for an IBM-PC-based Workstation*, IEEE Micro, October 1985, pp. 52-67

[31] F.D. Natali, *Accurate Digital Detection of Angle-Modulated Signals,* IEEE Electronics and Aerospace Systems Convention Record, 1968, pp. 407-413

[32] Jean-Marie Blanchard, Michel Y. Levy, and Christian L. Poinas, *New Algorithms for High Speed Data Receivers and Their Implementation by Microprocessors,* IEEE 1980, CH1506-5/80/0000-0265, pp. 57.3.1-57.3.5

[33] Kamilo Feher, Ronald P. Tetarenko, Paul R. Hartmann, and Vasant K. Prabhu, *Digital Communications by Radio*, IEEE Transactions on Communications, vol. COM-27, no. 12, December 1979, pp. 1749-1762

[34] Raymond W. Stroh, *An Experimental Microprocessor-Implemented 4800 Bit/s Limited Distance Voice Band PSK Modem*, IEEE Transactions on Communications, vol. COM-26, no. 5, May 1978, pp. 507-512

[35] Piet J. Van Gerwen, Niek A.M. Verhoeckx, Henk A. Van Essen, and Fred A. M. Snijders, *Microprocessor Implementation of High-Speed Data Modems,* IEEE Transactions on Communications, vol. COM-25, no. 2, February 1977, pp. 238-250

[36] Jon W. Bayless, Arthur A. Collins, and Robert D. Pedersen, *The Specification and Design of Bandlimited Digital Radio Systems,* IEEE Transactions on Communications, vol. COM-27, no. 12, December 1979, pp. 1763-1770

[37] Friedrich Jondral, *Digital Signal Processing in a Commercial Short Wave Receiver*, Signal Processing III: Theories and Applications, Elsevier Science Publishers B.V., North-Holland, 1986, pp. 1105-1108

[38] Norman C. Beaulieu and Cyril Leung, *On the Performance of Three Suboptimum Detection Scheme for Binary Signaling*, IEEE Transactions on Communications, vol. COM-33, no. 3, March 1985, pp. 241-245

[39] Subbarayan Pasupathy, *Correlative Coding: A Bandwidth-Efficient Signaling Scheme,* IEEE Communications Society Magazine, July 1977, pp. 4-11

[40] Peter Kabal and Subbarayan Pasupathy, *Partial-Response Signaling*, IEEE Transactions on Communications, vol. COM-23, no. 9, September 1975, pp. 921-934

[41] William C. Lindsey and Chak Ming Chie, *A Survey of Digital Phase-Locked Loops,* Proceedings of the IEEE, vol. 69, no. 4, April 1981, pp. 296-317

[42] Texas Instruments, *Digital Phase-Locked Loop Design Using SN54/74LS297,* Dallas, Texas, 1986

[43] George Troullinos, Peter Ehlig, Raj Chirayil, Jon Bradley, and Domingo Garcia, Texas Instruments, *Theory and Implementation of a Splitband Modem Using the TMS32010,* 1986

[44] David Taylor, Alex Genusov, and Rami Friedlander, *DSPs Finally Find Floating Point,* The Electronic System Design Magazine, January 1988, pp. 81-86

[45] Bob Williams, Jim Prater, *Digital Filtering: The Right Stuff for Video,* The Electronic System Design Magazine, January 1988, pp. 91-94

[46] J.B. O'Neal, *Overview of Power Distribution Line Carrier Communication Systems,* Conference Record IEEE Global Telecommunication Conference, pp. 464-467, November 28 - December 1 1983, San Diego, CA

[47] K.W. Whang, G.C. Cagle, and S.W. Smart, *The Power Distribution System as a Communication medium for Load Management and distribution Automation,* Conference Record IEEE Global Telecommunication Conference, pp. 478-482, November 28 - December 1 1983 San Diego, CA

[48] G. Bowling, *The Power Distribution System as a Communication Medium to Control of Power,* Conference and Exposition, IEEE Record 79CH1377-1 Register 5, March 19-21 1979(also Rockwell International internal report)

[49] R.A. Arrington, *Load Management and Feeder Automation Through the Use of*

the Power Line Carrier, Rockwell International, March 19, 1981

[50] J.B. O'Neal, *Modeling the Noise on the Substation Power Distribution Bus at Frequencies from 1-20 KHz*, Paper of Center for the Communications and Signal Processing, North Carolina State University, CCSP-wp-84/2, 1984

[73] H.J. Trussell, and J.D. Wang, *Cancellation of Harmonic Noise in Distribution Line Communication*, IEEE Transaction on Power Apparatus and Systems, to appear

[74] Brain Harry, *A Macro Assembler for the TMS32010 and TMS32020 written in the C Programming Language,* CCSP, North Carolina State University, 1988

[75] Todd Cook, *A TMS32010/32020 Simulator Written in the C Programming Language*, CCSP, North Carolina State University, 1987

[76] James B. Johnson and Steve Kassel, *The Multibus Design Guidebook, Structure, Architecture, and Applications*, 1984

[77] National Semiconductor Corporation, *Series 3200, Development Board Monitor Reference Manual*, June 1984

[78] Plessey Microsystems, *PSM512A Error-Correcting DRAM User's Guide,* 1982

Appaendix A Digital Communication Workstation User Manaul

TABLE OF CONTENTS

1. Introduction

The technological advance in Digital Signal Processing(DSP) has made it possible to implement communication systems in all-digital form. This is due to the fact that both the theoretical and practical aspects of the field have made tremendous progress during the past decade. As more DSP algorithms are being discovered, better tools are also being developed to implement these algorithms. One of the most important technological breakthroughs in electronics is the high-speed digital signal processor. Essentially high speed microprocessors/microcomputers and digital signal processors are designed specifically to perform computation intensive digital signal processing algorithms. These processors can execute millions of DSP operations per second by taking advantage of the advanced architecture, parallel processing, pipelining, and dedicated DSP instruction sets. These capabilities and the low cost of VLSI technology to manufacture these single-chip processors allows complicated DSP algorithms to be implemented in a tiny silicon chip with an affordable price. Digital signal processors have the advantages of reliability, reproducibility, compactness, programmability, and efficiency. The programmability makes digital signal processors very attractive for (1) system upgrades, in the case of advancements is DSP algorithms, and (2) multitasking where different tasks can be performed with the same device by simply changing its program. For the above and many other advantages, digital signal processors are being widely used in areas of general purpose digital signal processing, telecommunications, voice and speech, graphics and imaging, control, instrumentation, and the military. The purpose of the digital communication workstation is for a communication system designer and researcher to implement real time communication systems in an all-digital form by utilizing

advanced DSP technology. The workstation makes digital signal processors' attractive and powerful features easily accessible to the user by an advanced system architecture and a user friendly interface. With the workstation, communication systems can be designed and realized with minimum engineering effort. This manual is intended to give the user a good understanding of the workstation's hardware and software tools. It enables the user to utilize the workstation's available resources effectively. It also provides instructions on how to operate the workstation. The user is advised to study the references listed at the end of the manual to enhance the understanding of this manual.

The manual is organized into five sections. Section one is a brief introduction. Section two introduces the user to the workstation's hardware and describes important DSP hardware components in detail. The available software associated with the workstation's DSP applications is discussed. Section three uses a digital sinewave generator as an example to demonstrate the basic procedure of how to implement a DSP system on the workstation. Section four discusses the real-time interaction between the user and the workstation. Section five looks at each of the workstation's hardware components and examines the ways to manage these resources.

2. System Overview

The following sections describe in detail the hardware construction of the workstation and introduces the software tools available to operate the workstation efficiently. The system block diagrams shown in Appendix H of this manual will enhance understanding of this section.

2.1 Hardware Description

The architecture of the workstation is best described as a multicomputing configuration. It uses a Multibus I interface as the system bus. There are presently three processing units(boards) on the bus. Each of three boards has a microprocessor, local I/O channels, and shared-local memory. The fourth board is a memory expansion board. The National Semiconductor Corporation DB32016 single board computer serves as the bus master and performs system control functions of the workstation. Two Digital Signal Processing boards are built around Texas Instrument's TMS320 DSP family single-chip microcomputers. These two boards are dedicated to perform actual digital signal processing, operating at 20MHz. A Plessey PSM512A Multibus Error-Correction DRAM board has been installed to expand the system memory capacity by 256Kbytes.

2.1.1 System Bus - Multibus Interface

The Multibus system bus was originally developed at Intel Corporation in 1975 and later evolved to become the IEEE Standard 796-1983. Often referred to as Multibus/IEEE-796, it is a commercial-quality industry-standard open architecture bus for the use in microprocessor-based systems. The bus supports independent memory and I/O address spaces. There are 16 megabytes directly addressable memory and 64K I/O ports. Devices with either 8-bit or 16-bit(or both) data path(s) can transfer data over the bus using the asynchronous bus cycle protocol. The master-slave bus structure concept allows devices of different speeds to be interfaced via the bus handshaking mechanism. Data transfer rates up to five million transfers(bytes or words) per second are conceivable.

The Multibus interfacing system also provides three subsidiary busses in addition

to the system bus. They are the iSBX bus, iLBX bus, and MULTICHANNEL bus. However, they are not implemented in the workstation design because of the cost and complexity considerations. Instead, a dedicated I/O channel interface, P2 Data Channel, is introduced in place of the iLBX bus to enhance the performance of the digital signal processing units. It is configured using the form-factor of a standard 60-pin Multibus P2 connector. The signal definitions are described in Appendix A of this manual. The bus backplane is a MUPAC Multibus cage with five P1 bus slots.

2.1.2 Bus Master - DB32016 Single Board Computer

The DB32016 single board computer is a complete microcomputer using the National Semiconductor Series 32000 family of advanced microprocessors. It is equipped with an EPROM-based firmware, MON16 monitor, which provides a native debug and execution environment for programs developed on a host computer. The National's C cross software package, residing on a Micro-VAX II, provides a high level programming environment for the user to develop software for the DB32016. The Multibus interface permits the DB32016 microcomputer system to be used on the industry standard Multibus system bus as the bus master for the workstation. The board includes a NS32016 CPU, a NS32082 Memory Management Unit(MMU), a NS32081 Floating-Point Unit(FPU), and a NS32202 Interrupt Control Unit(ICU). The 128 Kbytes of on-board dynamic RAM is configured as local memory to the DB32016 board. A PROM-based firmware MON16 monitor, which resides on two 2732 PROMs with a 450 ns access time, is installed on the DB32016 board. The jumper setting requirements are shown in Table 1.

TABLE 1 JUMPER SETTING FOR PROM INSTALLATION

W6	9-10
W11	1-2, 5-6
W3	3-4, 7-8

Two serial I/O ports are provided via an 8251A USART(Universal Synchronous Asynchronous Receiver Transmitter). These ports establish the user-to-workstation-to-host-computer interface through two RS232 connections(DTE and DCE) configured in transparent mode. A standard terminal-display with a keyboard is connected to serial port 0, designated as J2, while serial port 1, designated as J3, is connected to a Micro-VAX II as the host computer. Both ports are operating at a 9600 baud speed. The jumper settings are listed in Table 2.

TABLE 2 JUMPER SETTING FOR PORT 0 AND PORT 1

PORT 0 (J2)	W31 W32 W33	4-5 1-3, 2-4 4-5
PORT 1 (J3)	W26 W27 W28	1-2, 3-4 1-2, 3-4 1-2, 3-4

Port 0 is configured as a DCE(data communication equipment) with handshaking and port 1 is configured as a DTE(data terminal equipment) without handshaking. The dip switch S3 is set as follows: position 4 closed, position 3 closed, position 2

closed, and position 1 open for 9600 baud; position 5 closed for FPU; position 6 closed for MMU; and position 7 open to inhibit testing of 8255APPI connected to J1(see TABLE A-1. DIP SWITCH SETTINGS(S3) in *Series 32000 DB32016 Development Board User's Manual* for a complete list of baud rate settings). The DB32016 board is used as the bus master on the Multibus. Common bus request(CBRQ) signal to the Multibus is enabled by jumper W9 2-3. The bus arbitration is resolved by allowing the DB32016 to have the exclusive access to the bus at all time. This enables the DB32016 to access the dual port memories on the other boards at any time and to perform system controls. The DB32016 is plugged in the first slot(at bottom) in the Multibus backplane cage and the bus priority signal BPRN is pulled low(connected to ground) on the Multibus backplane.

The power requirement is 5 volts at 7.5 amperes, 12 volts at 0.05 amperes, and negative 12 volts at 0.05 amperes.

2.1.3 Memory Expansion Board - Plessey PSM512A

The Plessey PSM512A Multibus Error-Correction DRAM board is used as a memory expansion for the workstation. It contains a 256 Kbytes random-access-memory mapped to the Multibus interface at the address of 28000 - 67FFF(Hex). The PSM512A has the build-in on-board error detection/correction capability. The board is intended to be the memory expansion for the DB32016, making a 384 Kbytes contiguous memory space for the bus master.

2.1.4 DSP-A Board - DSP32010

This board consists of a TMS32010 16/32-bit single-chip microcomputer with a four kilo-words(two bytes per word) static memory. Four TMM2018D 2048x8-bit

Toshiba MOS memory chips are used. These fast memory chips(with 45 ns access time) are essentially cache memory for the TMS32010 CPU to execute programs at the full speed(200ns clock cycle) without wait state. These memory chips are mapped to the full address space of the TMS32010(from 000 to FFF) and also mapped to the Multibus memory space from A0000 to A1FFE(hex). Because the TMS32010 addresses memory 16 bits at a time while Multibus does 8 bits at a time, it takes twice the memory space to access the memory from the Multibus than from the TMS32010. For this reason the memory can only be accessed by word(16-bit) because the Multibus byte-swapping scheme was not implemented in the hardware design.

There are two 16-bit bi-directional parallel I/O ports. DSP-A-I/O(TMS32010 refers to as port PA4) is designated as the Multibus I/O port for the board. It maps to the I/O space of the Multibus address 8C0008(hex). Two status ports are dedicated to perform asynchronous hardware interrupts to the TMS32010 and software interrupts to the bus-master(DB32016). The status ports are named as Receive Ready(RCVRDY) and Transmit Ready(XMTRDY), with the Multibus addresses 8C0000(hex) and 8C0002(hex), respectively. Data written to the input port from the bus-master causes an interrupt to the TMS32010 through XMTRDY port. Upon servicing the interrupt, the TMS32010 executes an IN instruction to read the data from the input port and sets RCVRDY simultaneously. The bus-master can read RCVRDY port to determine the response from the TMS32010. The RCVRDY can be reset by writing a "1" to the port from the Multibus interface. When the TMS32010 writes data to the output port, XMTRDY is set automatically. The XMTRDY can be polled by the bus-master to synchronize the data transfer. Port PA6 is a 16-bit bi-directional I/O port dedicated to data transfers between the DSP32010 and DSP32020 boards. In addition, there are two 14-bit I/O ports(both are designated as PA5 by the

TMS32010) which are connected to the A/D and D/A.

The TMS32010 can be reset by the bus-master by writing a "1" to the port at the address C00030. To release the TMS32010, write a "0" to the same address. Since the TMS32010 does not generate Multibus addresses, the DSP32010 board cannot access the off-board memory space and I/O ports.

Power requirement for the board is 5 volts at 3.1 amperes.

2.1.5 DSP-B Board - DSP32020

The DSP32020 board contains a TMS32020 single-chip microcomputer with eight kilo-words static cache memory. The memory maps to the lower part of the 64K address space of the TMS32020(0000-1FFF) and is addressable from the Multibus(E0000-E3FFE). As with DSP32010, the board memory can be accessed by word(two bytes) only. Similar to DSP32010 board, this board also has two 16-bit bi-directional I/O ports. Referred to as PA1 by the TMS32020 and PA6 by the TMS32010, it is designed to allow data transfer between DSP32020 and DSP32010 boards. It operates in asynchronous mode with hardware interrupt and handshaking. The DSP32010 board usually initiates data transfer by writing data to the port. This action causes a lowest level hardware interrupt to TMS32020 via BIO signal. The TMS32020 reads data from the port and resets P2RDY to inform the TMS32010 that the data has been read. Since the TMS32020 resets P2RDY every time PA1 is addressed, P2RDY can also be used to indicate that data has been written to the port by the TMS32020. PA2 is a Multibus I/O port and it maps to Multibus I/O address 8D000A(hex) with a status port 8D0002, which is used by the Multibus to synchronize data transfer between the DB32016 and DSP32020. The synchronization is accomplished by using a hardware interrupt register. This interrupt register

interrupts the TMS32020 at three levels, depending on the vector written to the register by the bus-master.

TABLE 3 DSP-B-IR Interrupt Register(three bits)
(Multibus Address 8D0002)

Vectors(binary)	Interrupts
110	int0
101	int1
011	int2
111	no interrupt

The bus-master can reset the TMS32020 momentarily by addressing 8D000F. On the other hand, the TMS32020 can be held in a RESET or HOLD state indefinitely by the bus-master through register control at the Multibus address 8D0001.

TABLE 4 Processor Control Register(two bits)
(Multibus Address 8D0001)

Vector(binary)	Functions
11	run
10	hold
01	reset

There is a 4-bit configuration register CFR that is write-only by the TMS32020. It

enables the TMS32020 to configure the on board cache memory either as the program or data memory for itself. The memory is organized into four blocks with 2K x 16 each. A board reset signal from the bus master(i.e. writing 01(binary) to the PCR at 8D0001) configures all four blocks into the program memory. Because of the fact that the TMS32020's external data memory starts from 0400(hex), only the last three blocks(0800-1FFF) can be exclusively used as data memory. For the first block, the lower half(0000-03FF) is inaccessible by the TMS32020 when it is configured as data memory. While configured as the program memory, the first block maps to the TMS32020 program address from 0000 to 07FF, the second block from 0800 to 0FFF, the third block from 1000 to 17FF, and the forth block from 1800 to 1FFF. As with the DSP32010 board, the DSP32020 board cannot access off-board memories and I/O ports.

The power requirement is 5 volts at approximately 2.5 amperes.

2.1.6 Multibus Power Supply - SPL130-4000

The Multibus backplane is powered by a SPL130-4000 switching power supply, manufactured by Power One Corporation. The power supply provides four output voltages as listed in TABLE 5. The SPL130-4000 provides adequate power for all the

TABLE 5 SPL130-4000 Switching Power Supply

voltage	current(max)
+5V	22A
-5V	1.5A
+12V	3A
-12V	1.5A

Multibus boards in the workstation and for future additional board(s) with moderate power requirements.

2.1.7 Data Acquisition Interface -A/D and A/D Converters

The A/D and D/A converters use advanced data acquisition devices which have high resolution and fast conversion speed. The A/D converter samples analog input at a 32 KHz rate and converts to 14-bit offset binary digital words. The D/A converter does the reverse operation at the same speed. Two 16KHz lowpass analog filters are used, one before the A/D for anti-aliasing and one after D/A. The converters are powered by a different power supply. The interface between the converters and the DSP32010 board is driven by TTL gates.

2.2 Software

The basic software tools that support the workstation's operations are contained in two software packages as described below.

2.2.1 GENIX

The operation of the workstation is controlled through an interactive debugger program in the GENIX software pack- age. This symbolic debugger, along with other cross-support tools, currently resides on a Micro-VAX II running Ultrix. The user can perform the following functions with the debugger: C, Assembly, and mixed-language program debugging on DB32016 single board computer; loading programs from the host system to the workstation; using symbolic on the DB32016; multimodule program debugging on the DB32016; radix specifications; indirect files and history files usage; on- line help facility; and disassembler to give DB32016 assembly output. The user can create C and/or assembly program source files using the editor(s) on the host system. Then the C and Assembler for the DB32016 supplied with GENIX may

be used to compile/assemble the source program to produce object code. Finally, the cross-support linker is used to generate executable code for the DB32016 to perform various workstation control functions as well as signal processing tasks.

2.2.2 Macro-Assembler

A Macro preprocessor and the TMS320 assemblers are used to develop machine code for the TMS320s. The macro preprocessor allows the user to define macro definitions for commonly used TMS320 assembly routines. These macro definitions can be expanded to produce in-line assembly codes. It uses TI's assembly syntax format with certain improvements. Loop structure can be used also to generate repeated in-line codes. The assemblers generate executable code for the TMS320s in simple Intel checksum format. In order to load the TMS320 code to the DSP boards' memory, the executable codes must be compiled and linked into a program to run on the DB32000. The Series 32000 symbolic debugger is then used to load the program into the DB32016's memory; the program is run on the DB32016. The program actually reads the TMS320 executable codes included in the program as pure data and writes these codes into the TMS320s' memory via Multibus dual-port memory access. In section 4.1.6 a faster loading technique is described.

3. DSP Application - an Example

This chapter describes basic procedure on how to efficiently use the workstation as a design and research vehicle to implement DSP system(s). The following sections use an example of a digital sine wave generator to demonstrate how to formulate building blocks for a particular system and implement these blocks with various resources on the workstation. It is not necessarily a best design approach in

terms of efficiency. It merely shows the flexibility of the workstation.

3.1 Formulation of Application Building Blocks

A table lookup technique is used to achieve the best possible speed to generate a sine wave by means of digital signal processing. The sample values of a sine wave are computed by the DB32016's CPU, then these values are stored as 16-bit integer numbers in the memory. The TMS32020 uses a modulo wrap-around technique to step through the table values and to transfer the samples to the DSP32010. The DSP32010 performs the necessary timing control to synchronize with the D/A converter to output samples at a 32KHz rate. Thus, the sine wave generator is constructed with three building blocks: (1) sine table generation by the DB32016, (2) table lookup by the DSP32020, and (3) output synchronization by the DSP32010. The coding of these blocks are described in the following section.

3.2 Program Coding

The program routines are coded in C and assembly languages. The mixed language programming allows the user to achieve concurrent-processing.

3.2.1 Sine Table Generation

In order to obtain accurate table values, floating-point arithmetic is used to sample a one-period of sine wave at the desired rate. This rate ultimately determines the frequency resolution of the sine wave generator. These table values are computed with a double precision algorithm, i.e. the table values are computed as double precision floating-point numbers before converting them to integer numbers. The coding is made simple by using C programming language and the *sin()* function

provided with the C cross compiler. In order to store the table values in the DSP32020's dual-port memory an assembly routine is written for the NS32016 to access the memory via the Multibus address mapping. This assembly routine writes the entire sine table into the DSP32020's program memory. This allows the DSP32020 to directly obtain table values from the program memory using a TBLR instruction.

To compile the C program,

(1) type

% **nmcc -c** *sine.c* **<cr>**

which produces an object file *sine.o*.

(2) Type

% **nasm -o** *tblout.o tblout.s* **<cr>**

to assemble the assembly routine*(tblout.s)* and to produce the object code(*tblout.o*).

(3) Type

% **nmeld -T 9600 -e main -o** *sinetbl.out sine.o tblout.o* **-lm -lc <cr>**

to link the C program and the assembly routine along with the C library and math library. The option for the Multibus address **9600**(hex) is necessary because (1) the RAM on the DB32016 board starts at 8000(hex), and (2) the default memory protection tables are built in the RAM memory starting at the address 8000(hex). (see TABLE 1.1 Monitor Address Map in reference [6])

3.2.2 Table Lookup

A sine wave is generated by simply stepping through the table values at a constant rate, wrapping around at the end of the table. The fast wrap around is accomplished by making the table size a power of two and by using bit-masking to

perform the modulo operation on the current table index and the table size. This scheme can be implemented efficiently in assembly language on the TMS320. The table lookup routine can be executed on the DSP32020 in less than four microseconds. To assemble for TMS32020, type

% **tasm20** *tblsine* **<cr>**

to produce the executable code(*tblsine.obj*) in Intel checksum format. Command **tasm20** assumes *.mac* as the file extension for the source file.

3.2.3 Output Synchronization

The output synchronization is achieved by utilizing the TMS32010's BIO interrupt signal generated by the A/D sampling pulse. Upon receiving a BIO interrupt, the TMS32010 reads a sample from the P2 Data Channel port and outputs the sample to the D/A converter. An interrupt to the TMS32020 is initiated by executing a dummy OUT instruction to the P2 Data Channel port, causing the TMS32020 to perform the next table value lookup operation. A dummy IN instruction is then executed to reset the BIO signal and the TMS32010 enters a wait loop for the next sampling instance. To assemble *in2out.mac*, type

% **tasm10** *in2out* **<cr>**

the output file, *in2out.obj*, is the executable code in Intel checksum format.

3.3 Program Loading and Execution

This section describes the code loading procedures for the DB32016, DSP32010, and DSP32020 boards.

3.3.1 Workstation Initialization

After power up, the following logo appears on the user terminal:

R_VERSION2.0010FEB83.

Type an exclamation point(!) followed by a carriage return. An asterisk(*) will replace the ***R*** in the first column. Now type **omt** to set the monitor operation mode to transparent(see section 3.3.1 Reset Acknowledge Command in reference [6]). From this point on the user communicates with the host computer. After properly logging in the host computer, the user may invoke the debugger by typing **dbg16**. The debugger must be initialized every time it is invoked. The necessary initialization steps include (1) selecting the communication link, and (2) setting up memory protection. The communication channel is established by selecting the link command of the debugger. Since the DB32016 is installed with MMU, the memory access and protection must be established accordingly. Specifically, the dual-ported memory residing on the two DSP boards must be enabled(validated) before being accessed by a user program. The initialization setup is performed by writing a command file *dbg16.ini* and the **dbg16** executes it automatically every time the debugger is invoked.

3.3.2 Loading DB32016

To load DB32016 board, simply type

--> **b** *sinetbl.out*/**sp=20000 <cr>**

at the debugger's prompt. The debugger reads the executable file sinetbl.out from the host computer's mass storage device and writes it into the DB32016's RAM. The necessary CPU register setups, such as base register and stack pointer module register etc., are performed by the debugger. After loading is completed, type

--> **g <cr>**

to start the execution.

3.3.3 Loading DSP32010

To load the executable code into the DSP32010's program memory, first the Intel checksum format object code is included in an assembly program as the data declaration for the DB32016. Then the assembly program simply moves the data, i.e. the TMS32010's opcode, from the DB32016's RAM into the DSP32010's program memory through the Multibus interface. Type

--> **load10** *in2out.obj* **<cr>**

to produce *in2out.out*. The **load10** command performs (a) code conversion from the Intel checksum to the National's assembly program's data declaration, (b) writing of an assembly program to move code, (c) assembly of the program, and (d) linking of the object code for the DB32016. Two intermediate files are created at the same time: *in2out.s*(the assembly program) and *in2out.o*(the object file from the assembler). The output file in2out.out from the load10 command can be loaded into the DB32016's RAM by the debugger in the same manner as described in the previous section. When the program is executed on the DB32016, the code is moved into DSP32010's program memory and is ready for execution by the TMS32010. The command sequence

% dbg16 <cr>

--> b *in2out.out*/**sp=20000 <cr>**

--> g <cr>

will complete the loading process.

3.3.4 Loading DSP32020

Loading the DSP32020 board is exactly the same as for the DSP32010 board except that the **load20** command is used in the place of **load10**. Type

% **load20** *tblsine.obj* **<cr>**

this produces *tblsine.s, tblsine.o*, and *tblsine.out*. Then load the file tblsine.out with the debugger:

% **dbg16 <cr>**

--> **b** *tblsine.out*/**sp=20000 <cr>**

then type

--> **g <cr>**

to execute.

4. Run Time User-Workstation Interactions

The run time operations of the workstation are facilitated in two ways: (1) the debugger is the most convenient and reliable tool to control the workstation operations. Its capabilities, briefly listed in section 2.2, can be utilized in various ways; (2) the user can develop his/her own software for a particular environment. This software can range from simple control programs to sophisticated operating systems for the workstation.

4.1. Using the debugger as a Control Tool

The debugger relies on a mon16 monitor and a host computer to carry out its command executions. The mon16 monitor is a ROM-based firmware installed on the DB32016 single-board computer. The debugger itself resides on a MicroVAX-II with the Ultrix operating system. Using the debugger is simply a convenient way of

accessing the monitor utilities. The user can easily perform the following operations: resetting the DSP boards, interrupting the DSP boards, examining/altering memory contents, using an indirect command file for automatic command execution, collecting data with history files, and loading programs into the workstation for executions. These operations are described in the following sections.

4.1.1. Reset/Start DSP32010/DSP32020 Boards

The DSP32010 board can be reset by using the debugger's replace command. To reset type

--> r 0C00030/nv <cr>

--> 1 <cr>

The DSP32010 board will be held in a reset state until the user issues another replace command and write a "0" to the address 0C00030. To release DSP32010 from reset state, type

--> r 0C00030/nv <cr>

--> 0 <cr>

where 0C00030 is the address of an I/O port on the DB32016 board. This port is a write-only 1-bit register that sends out a lock signal to the Multibus interface, i.e. the LOCK signal is held low as long as the register contains a "1". Since the LOCK signal is recognized as the reset by the DSP32010 board, it is one of the two ways to reset the DSP32010. The TMS32010 on the DSP32010 will also be reset momentarily any time there is a memory access to the board's program memory from the Multibus interface. The DSP32020 has a 2-bit control register at the Multibus address 8D0001(hex). The contents of the register determines the board status(see TABLE 4). To reset the DSP32020, type

--> **r 8D0001/nv <cr>**

--> **1 <cr>**

The red LED on the board lights, indicating the TMS32020 is being held in reset state. Type

--> **r 8D0001/nv <cr>**

--> **3 <cr>**

to release the TMS32020 from the reset state. The green LED lights, indicating a run-state. The TMS32020 can be placed in a HOLD state forcing it to give up the data and address busses of the board. During the HOLD state the yellow LED is on. The register control board-status mechanism enables the user to perform a prolonged dual port memory access to both the DSP32010 and DSP32020 boards. The DSP32010 board will also be reset momentarily any time the bus master requests a memory access to the board.

4.1.2. Interrupting DSP32010/DSP32020

The TMS32010, the CPU of the DSP32010 board, has a maskable hardware interrupt which is connected to the DSP-A-I/O port. To interrupt the DSP32010 board, the user simply writes data to the DSP-A-I/O port at the Multibus address 8C0008(hex). Type

--> **r 8C0008 <cr>**

--> **(any 16-bit data) <cr>**

This generates a hardware interrupt to the TMS32010. It is a simple and convenient way to interrupt the DSP32010's operation causing the TMS32010 to execute predefined interrupt routines. Section 5.5.3 will discuss how to write interrupt service routines. Interrupting the DSP32020 is accomplished by writing a vector(3-bit data)

to the DSP-B-IR port, an interrupt register mapped to the Multibus interface at address 8D0002. Type

--> **r 8D0002/nv <cr>**

--> **(any 3-bit data) <cr>**

TABLE 3 defines the vectors corresponding to the three levels of interrupt to the TMS32020.

4.1.3. Examine/Alter Memory Contents

The user can examine and alter the workstation memory contents and all the memory mapped I/O ports. The debugger's print and replace commands are all that the user needs to know for this purpose. To examine a memory location, or a memory mapped I/O port, type

--> **p *address* <cr>**

The debugger will print out the contents at the location specified by ***address***. To change the contents of a memory location, type

--> **r *address* <cr>**

old_value? *new_value* <cr>

The **old_value** at location specified by ***address*** will be replaced by ***new_value***. The verification of the ***new_value*** is also performed. To alter without the verification type

--> **r *address*/nv <cr>**

? *new_value* <cr>

This is useful for certain write-only I/O port accesses.

4.1.4. Indirect Command File Programming

The user can avoid the tedious typing of frequently used command sequences by

placing the commands in a file, then letting the debugger read commands from that file and execute them. If the file is named with .ind extension, the user can execute the command file after entering the debugger by typing

--> **@*filename* <cr>**

The commands are not given to the debugger directly by the user, thus the name indirect file. An indirect command file can be created by typing in all the commands with an editor or saving the commands with debugger's history file facility. The following example shows how to use the second method to create an indirect command file. A history file simply logs all the interactions between the user and the debugger and saves the history of a debugging session in a file on the host computer. The user can use the history file to save all the commands that are required to load all the sinewave generator's modules into the workstation. Later on, the sinewave generator can be loaded with a single command. After entering the debugger, type

--> **sh *sinegen.ind* <cr>**

--> **b *tblsine.out*/sp=20000 <cr>**

--> **g <cr>**

--> **b *sinetbl.out*/sp=20000 <cr>**

--> **g <cr>**

--> **b *in2out.out*/sp=20000 <cr>**

--> **g <cr>**

--> **sh/o <cr>**

The first command opens a file named *sinegen.ind*. The following six command lines are recorded in the file and the last command line closes the file. Later on, the user can re-execute the commands stored in *sinegen.ind* file by typing

--> **@*sinegen* <cr>**

4.1.5. Collecting Data with History Files

The history file facility can also be used to collect data from the workstation and save it on the host computer's mass storage device. For example

--> **sh** *datafile.txt* **<cr>**

--> **p addressrange <cr>**

...

--> **sh/o <cr>**

The file *datafile.txt* can be further processed to extract the useful information. This technique frees the user from the concern of I/O transfers between the workstation and the host computer.

4.1.6. Fast Program Loading for TMS320

The opcode for TMS320s can be loaded into the memory much faster by using the debugger's **replace** command. Taking advantage of the indirect command file, a series of replace commands along with the appropriate parameters are saved in a file. Loading is a simple matter of invoking the command file from the debugger. In order to accurately load a complete program to either the DSP32010 or DSP32020, reset(or hold) the CPU(TMS32010 or TMS32020) during the loading period. This will ensure that the CPU does not run out of the control. The user can use the **fld10** and **fld20** commands described in Appendix G.

4.2. User Written Control Programs

The user can develop programs for the DB32016 to perform more interesting control functions such as assembler/disassembler for the TMS320s, graphical displays, and a multitasking dispatcher. The following sections discuss these

possibilities.

4.2.1. Assembler/Disassembler for TMS320s

An assembler/disassembler for the TMS320s would be a useful feature for the workstation. This feature may be implemented on the DB32016 with the C programming language. The ability of the bus master DB32016 to directly access the program memory of the TMS320s makes the assembler/disassembler's real-time implementation a straight forward matter. The assembler would obtain assembler mnemonics from the user via the terminal keyboard, translate them into corresponding opcode, and write the opcode into the program memory via the Multibus interface. The disassembler would read from the TMS320s' program memory via the Multibus interface access, translate the opcode into mnemonics, and display to the user on the terminal. The specific program memory location access may be accomplished by assembly programming on the DB32016, i.e. an assembly subroutine for DB32016 would be used to access the TMS320s' program memory explicitly.

4.2.2. Graphical Display

Graphical display capability of the workstation would enable the user to monitor real-time signals at various points in the DSP programs, which are normally inaccessible by external instruments. It would also allow the user to view the DSP analysis results of the signals in a meaningful manner. This feature could be implemented in software on the DB32016. GNUPLOT could be transported to the DB32016 with modifications to the external file access routines. Instead of the file access, the data would be obtained directly from memories.

4.2.3. Multitasking Dispatcher

A multitasking dispatcher would allow multiple tasks(programs) to reside in the memories at the same time and let the user choose the task(s) by picking from the list on the terminal, turning the workstation into a DSP operating console.

5. Workstation Resource Management

This section describes how to use the various workstation's resources in an efficient way for DSP operations.

5.1. A/D and D/A converters

The A/D and D/A converters are both 14-bit offset binary data acquisition devices. The sampling rate is 32 KHz. It is possible to connect other compatible A/D and D/A converters with different sampling rates to the workstation. Since the TMS320s perform numerical operations in two's complement format, the samples from(or to) the A/D(or D/A) converter must undergo the format conversion. Macro routines are available for this purpose. The A/D converter interrupts the DSP32010 at every sampling interval. The DSP32010 board acknowledges the interrupt through the TMS32010's BIO operation and resets the interrupt bit with an IN input operation.

5.2. Dual Ported Memories

The 4K program memory on the DSP32010 board and 8K on the DSP32020 board are both mapped to the Multibus address space. These fast memory chips are used for the program memory for the TMS32010 and TMS32020 on their respective boards. Block data transfer can also be carried out by the bus master using the DMA

technique. The I/O port may be used to synchronize the DMA operation. For the DSP32020 board, the memory can be configured as either the program or data memory, depending on the configuration register's content. Upon reset, all memory blocks on the DSP32020 board are configured as the program memory for the TMS32020.

5.3. Multibus I/O ports and Interrupts to TMS320s

The Multibus I/O ports allow interrupt driven data transfer between the TMS320s and the DB32016 through the Multibus interface.

5.4. P2 Data Channel, the I/O port between DSP320s

The P2 I/O port interface is used to enhance the workstation's DSP capability by incorporating this dedicated channel between the DSP32010 and DSP32020. The processor synchronization in DSP applications is accomplished through the use of this port.

5.5. Bus Master - DB32016

A manual reset and interrupt to the DB32016 can be performed by the user via two push button switches. There are three red LEDs that can be programmed for status display purposes.

5.6. External Memory Board

The 256 Kbyte memory on the PSM512A board greatly increases the workstation's ability to run sophisticated software.

On power up, due to the random data bits in the RAMs, errors will occur if

attempts are made to access the memory. The user must therefore write data to all address locations after power up, before reading data from the memory if spurious errors are to be avoided. This can be achieved be using the fill command of the debugger in the debugger initialization file dbg16.ini.

6. Workstation's Future Expansion

Because of the availability of a variety of the Multibus compatible components from various sources, the workstation could be expanded easily with adds-on off-the-shelf boards.

The workstation could become fully stand-alone by adding a mass storage device, such as a hard disk. The Multibus compatible disk drive controllers are available for many types of disk drives. An advanced graphics display controller could be integrated into the workstation by adding an appropriate display device with a Multibus compatible board.

Appendix A Multibus Data Sheet

Pin Assignment of Bus Signal on 796 Bus Board Connector (P2)

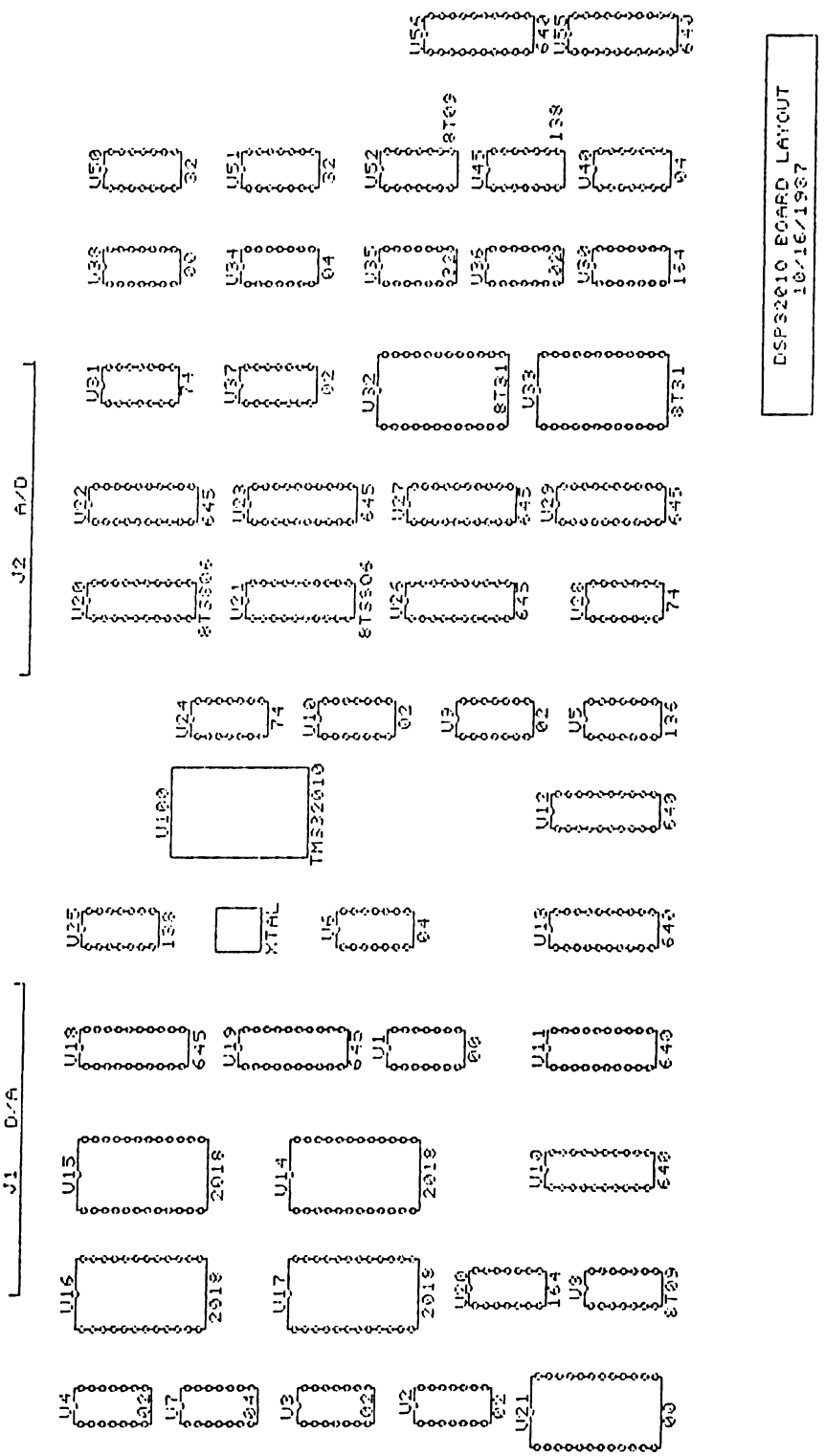
	Pin	Component Side		Pin	Circuit Side	
		Mnemonic	Description		Mnemonic	Description
	1	DAT15		2	GND	
	3	DAT14		4	GND	
	5	DAT13		6	GND	
	7	DAT12		8	GND	
	9	DAT11		10	GND	
	11	DAT10		12	GND	
	13	DAT9		14	GND	
	15	DAT8		16	GND	
	17	DAT7	Data	18	GND	Signal GND
	19	DAT6	Bus	GND		
	21	DAT5		22	GND	
	23	DAT4		24	GND	
	25	DAT3		26	GND	
	27	DAT2		28	GND	
	29	DAT1		30	GND	
	31	DAT0		32	GND	
Port	33	RD*	Read	34	GND	
Control	35	WT*	Write	36	GND	Signal GND
Signals	37	RDY*	Ready	38	GND	
	39		Not Used	40		Not Used
	41		Not Used	42		Not Used
	43		Not Used	44		Not Used
	45		Not Used	46		Not Used
	47		Not Used	48		Not Used
	49		Not Used	50		Not Used
	51		Not Used	52		Not Used
	53		Not Used	54		Not Used
	55		Not Used	56		Not Used
	57		Not Used	58		Not Used
	59		Not Used	60		Not Used

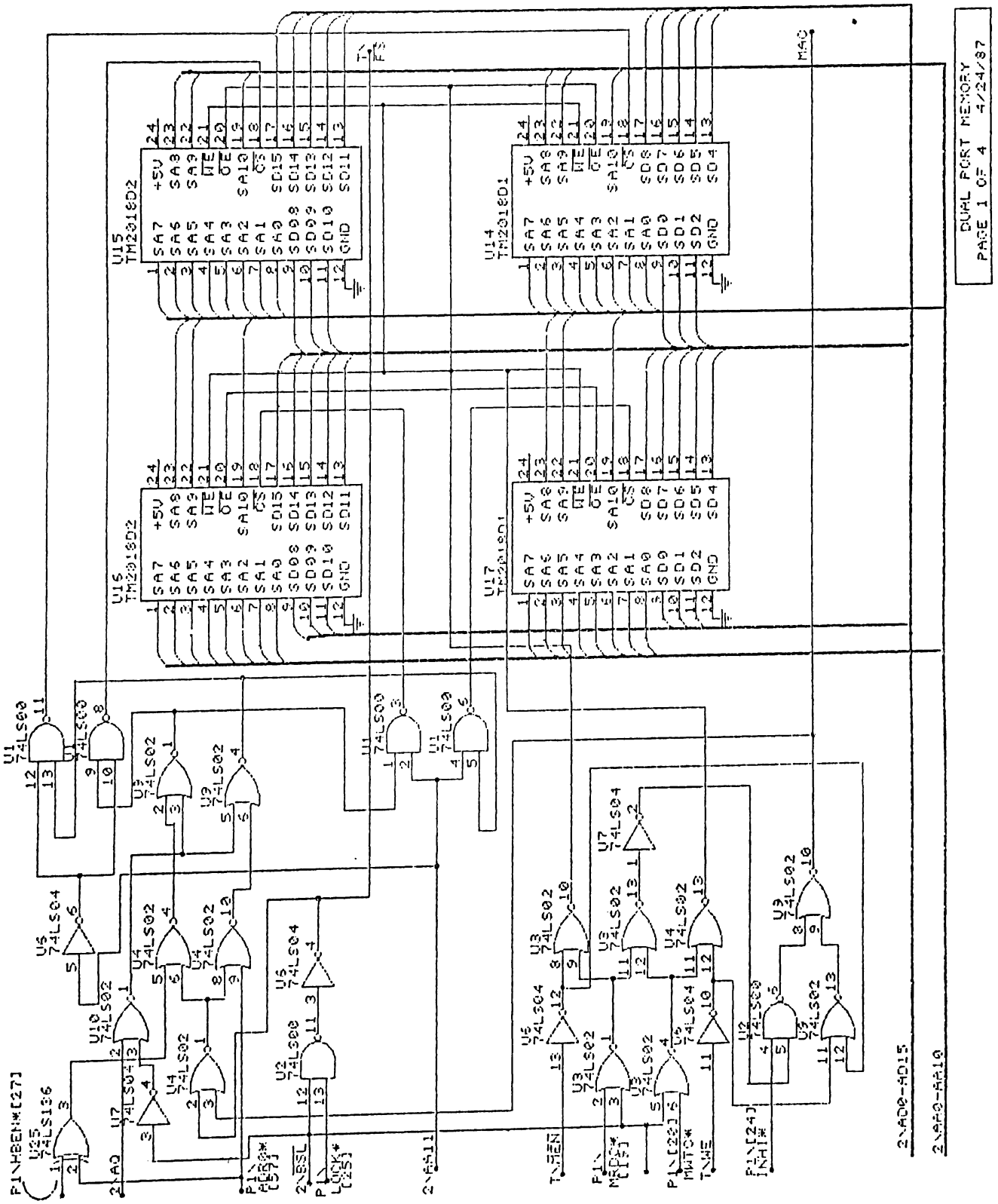
Appendix B Workstation Memory Map

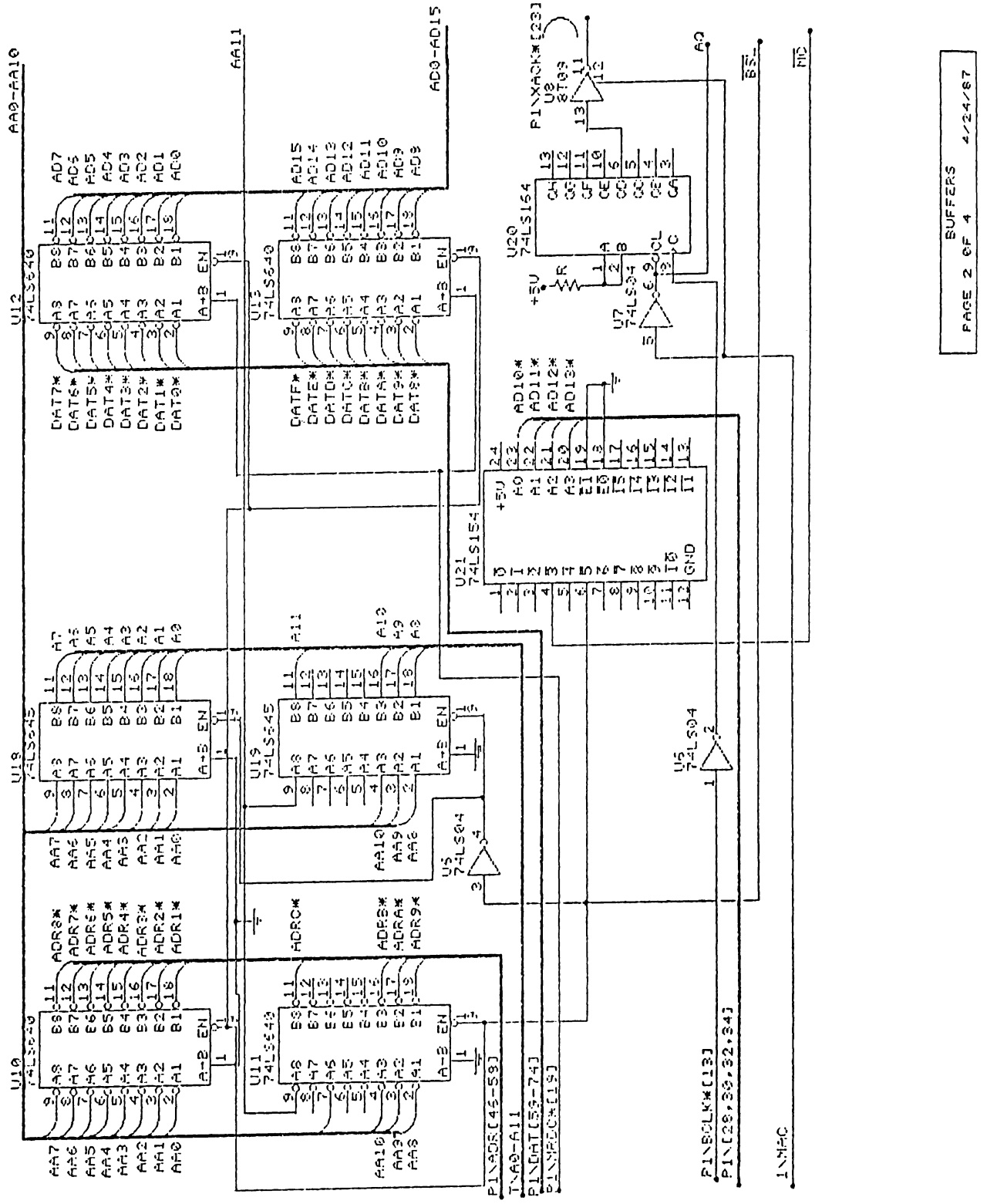
Address Space Cross Reference Table

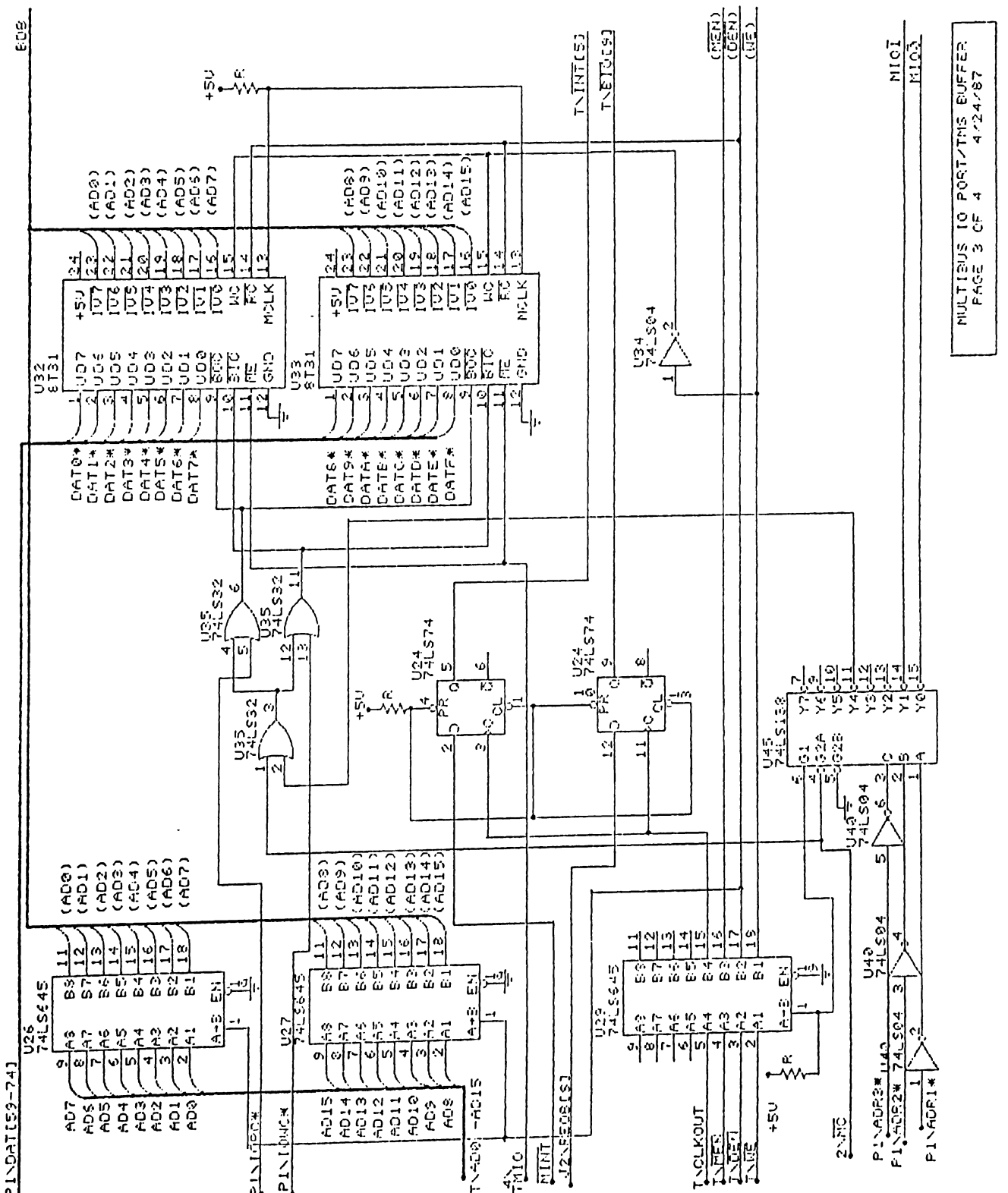
Description	Address(Hex)			Schematic Reference
	DB32016	DSP32010	DSP32020	
DB32016 on-board ROM/EPROM space	000000-007FFF	-	-	U15-U18 on Db32016
DB32016 on-board RAM	008000-027FFF	-	-	
DB32016 off-board RAM	028000-067FFF	-	-	PSM 512A
Dual Port Memory	0A0000-0A1FFF	000-FFF	-	U14-U17 on DSP32010
Dual Port Memory	0E0000-0E3FFF	-	0000-1FFF	U1-U8 on DSP32020
RCV_RDY status port	8C0000	-	-	U316 on DSP32010
XMT_RDY status port	8C0002	-	-	U318 on DSP32010
Multibus-DSP32010 I/O	8C0008	4 (PA4)	-	U32-U33 on DSP32010
DSP32020 Interrupt Register	8D0000	-	-	U28 on DSP32020
DSP32020 Hold/Reset Register	8D0001	-	-	U27 on DSP32020
Multibus-DSP32020 I/O Port-status Register	8D0002	- -	- -	U259 on DSP32020
Multibus-DSP32020 I/O Port	8D000A	-	2 (PA2)	U33-U34 on DSP32020
DSP32020 Sys-Reset Signal	8D000F	-	-	U19 on DSP32020
Seriel Port J2 port 0	C00000-C00004	-	-	
Override; asserts Multibus LOCK/ signal if bit-0 = 1	C00030	- -	- -	
DS LEDs	C00032 C00034 C00036			
Seriel Port Diagnostic	C00038	-	-	
Seriel Port J3 port 1	C00040-C0004F	-	-	
Programmable Timer	C00050-C0005F	-	-	
BLX J4 MCS0/	C00060-C0006E(even)	-	-	
BLX J4 MCS1/	C00070-C0007E(even)	-	-	
BLX J4 MCS1/	C00061-C0006F(odd)	-	-	
DB32016 on-board ROM/EPROM space	D00000-D0FFFF -	- -	- -	-
ICU ports	E000000-E0FFFFF	-	-	
A/D Interface	-	5(PA5)		
D/A Interface	-	5(PA5)		
P2 Data Channel I/O	-	6(PA6)	1(PA1)	
DSP32020 Configuration Register	-	-	0(PA0)	

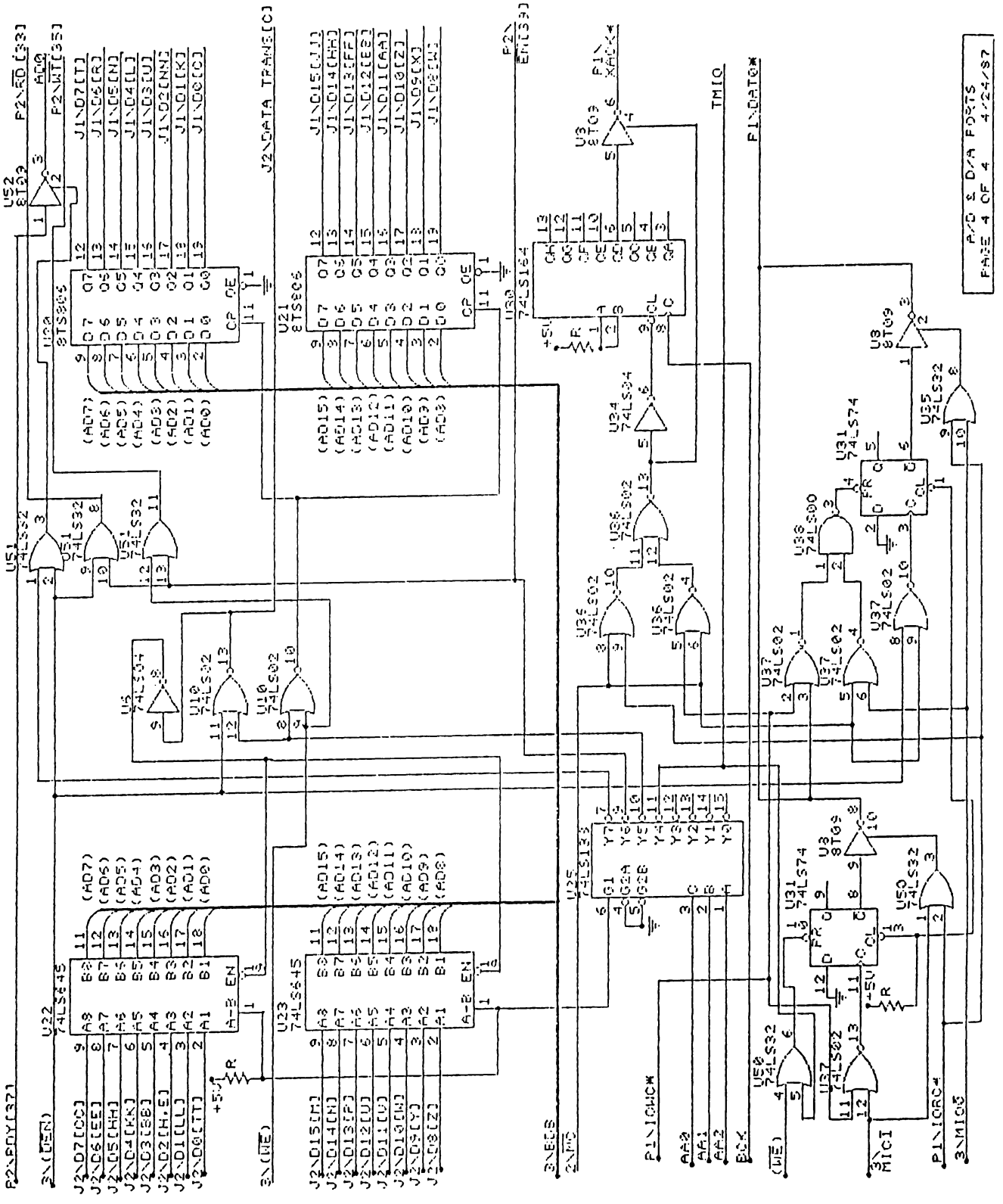
Appendix C DSP32010 Data Sheet



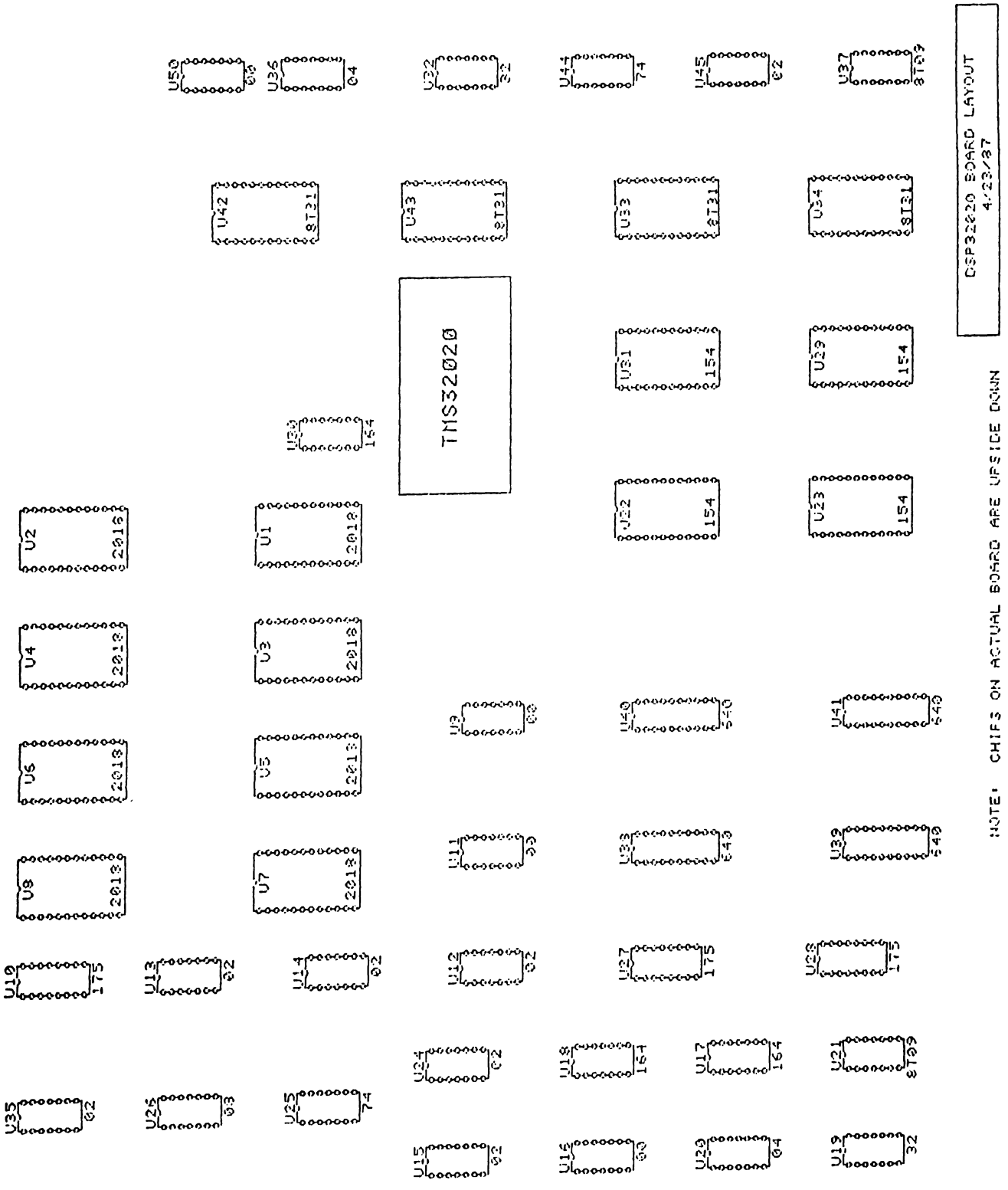

DUAL PORT MEMORY
PAGE 1 OF 4 4/24/87
U15 TM2016D2
U16 TM2016D2
U14 TM2016D1
U17 TM2016D1
+5V
GND
2VAD0-AD15
2VA0-A10

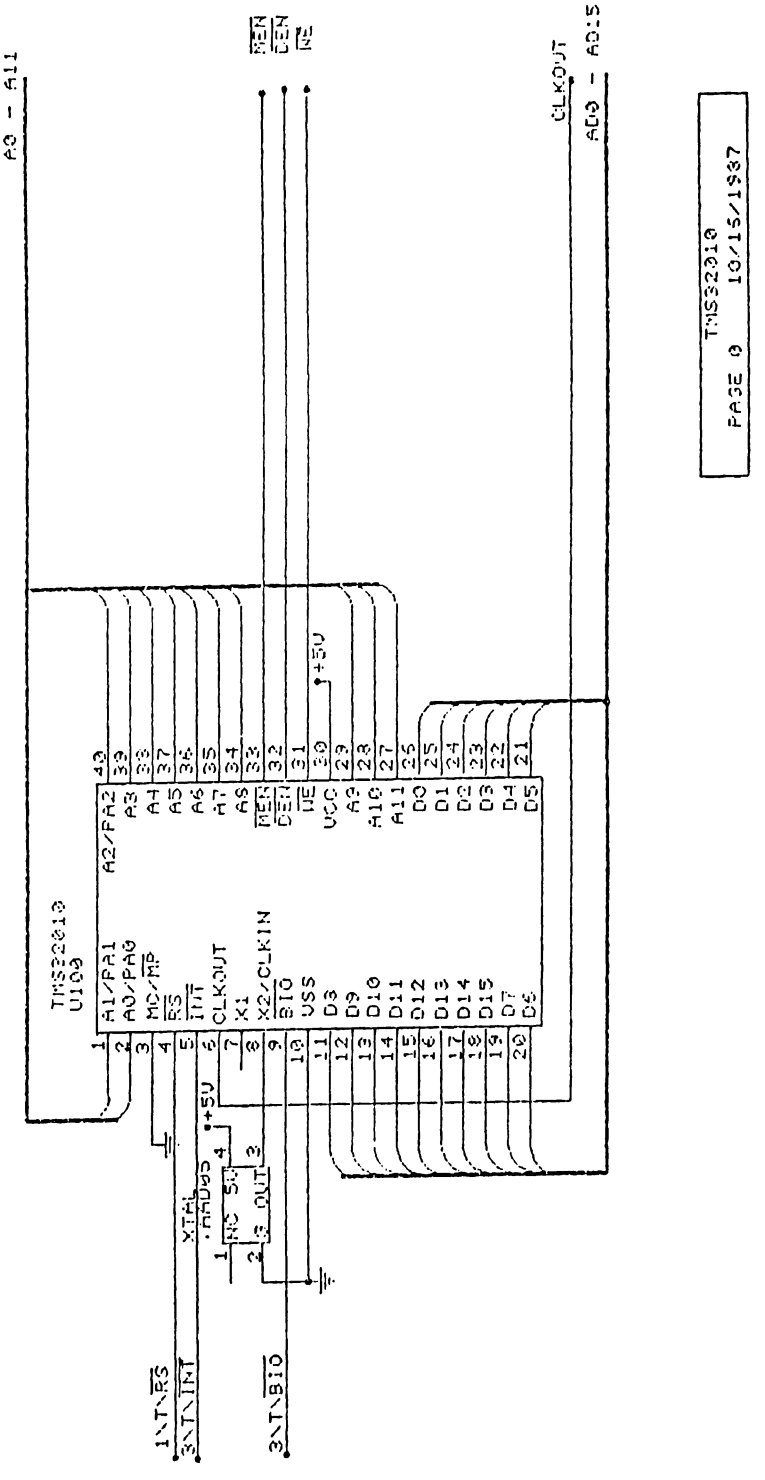

BUFFERS
PAGE 2 OF 4
4/24/87

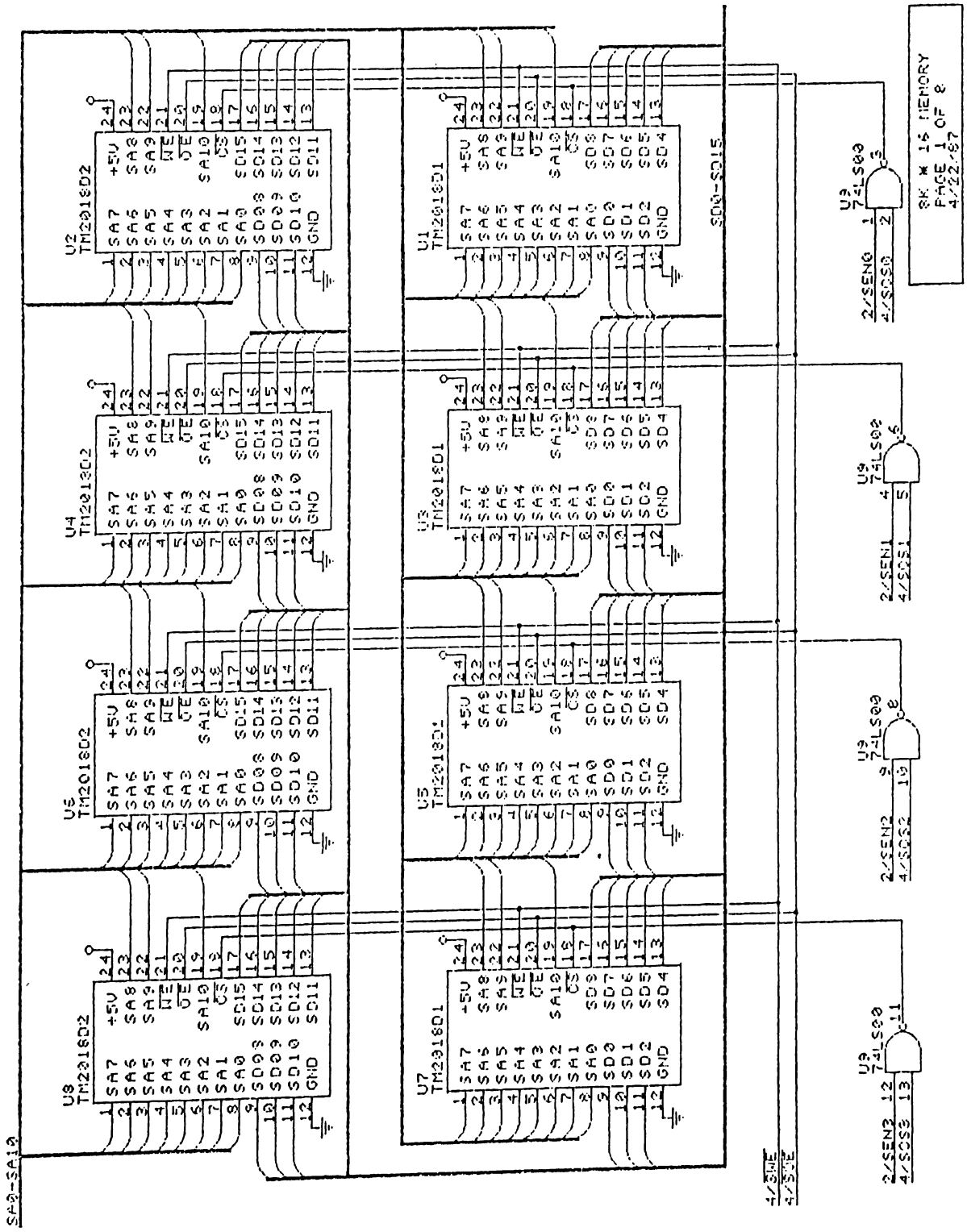

MULTIBUS IO PORT/TMS BUFFER
PAGE 3 OF 4 4/24/87
U26 74LS645
U27 74LS645
U29 74LS645
U32 8T31
U33 8T31
U34 74LS04
U35 74LS32
U24 74LS74
U45 74LS138
U40 74LS04
+5V
R
MCLK
MIOI
MIOO
TINT[5]
TNET[5]
(MEN)
(OEN)
(WE)
TCLKOUT
TMES
TMEN
TME
2.MC
P1\ADR3*
P1\ADR2*
P1\ADR1*

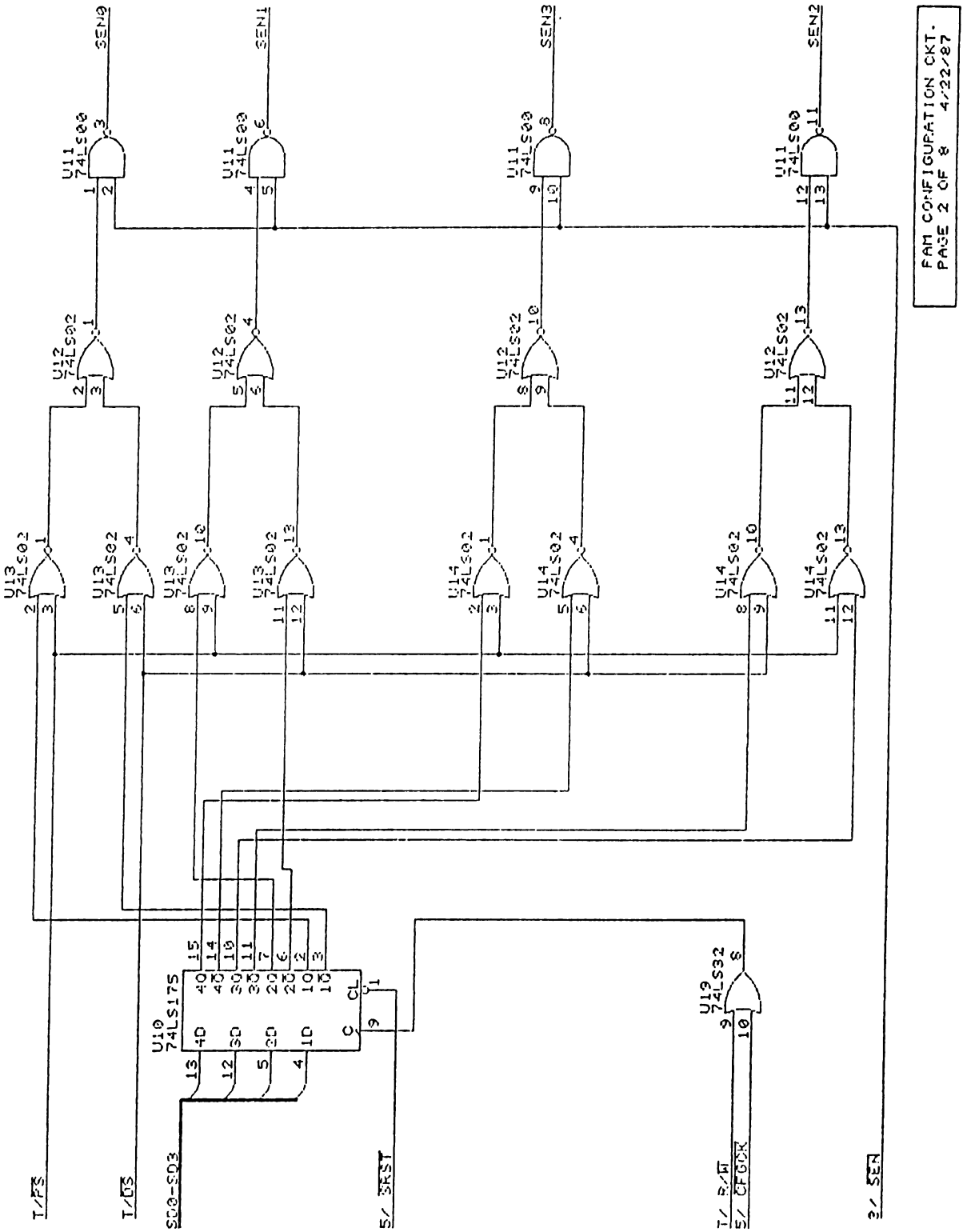

A/D & D/A PORTS
PAGE 4 OF 4 4/24/87
J2DATA TRANS[0]
P2ENT[39]
P1DATA[0]
TMIO
P2RDY[32]
P2RD[33]
P2WR[35]
J2D7[CC]
J2D6[EE]
J2D5[HH]
J2D4[KK]
J2D3[BB]
J2D2[H,E]
J2D1[LL]
J2D0[TT]
J2D15[MM]
J2D14[NN]
J2D13[PP]
J2D12[UU]
J2D11[VV]
J2D10[WW]
J2D9[YY]
J2D8[ZZ]
P1DIGWORD
P1DIGRD
3MICI
3MICO
3BEEB
2PHS
3(WE)
3(DEN)
+5V
R

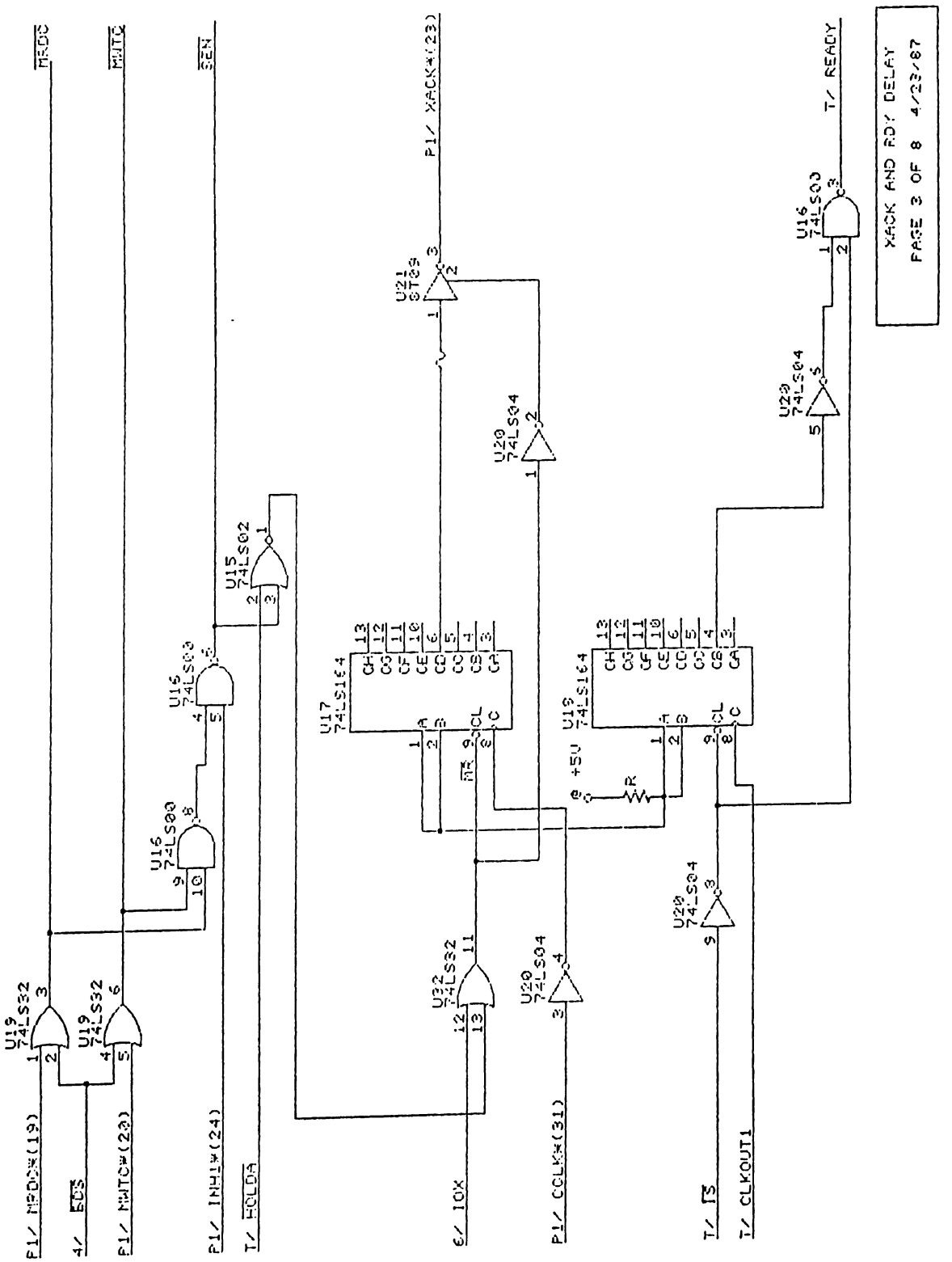
Appendix D DSP32020 Data Sheet

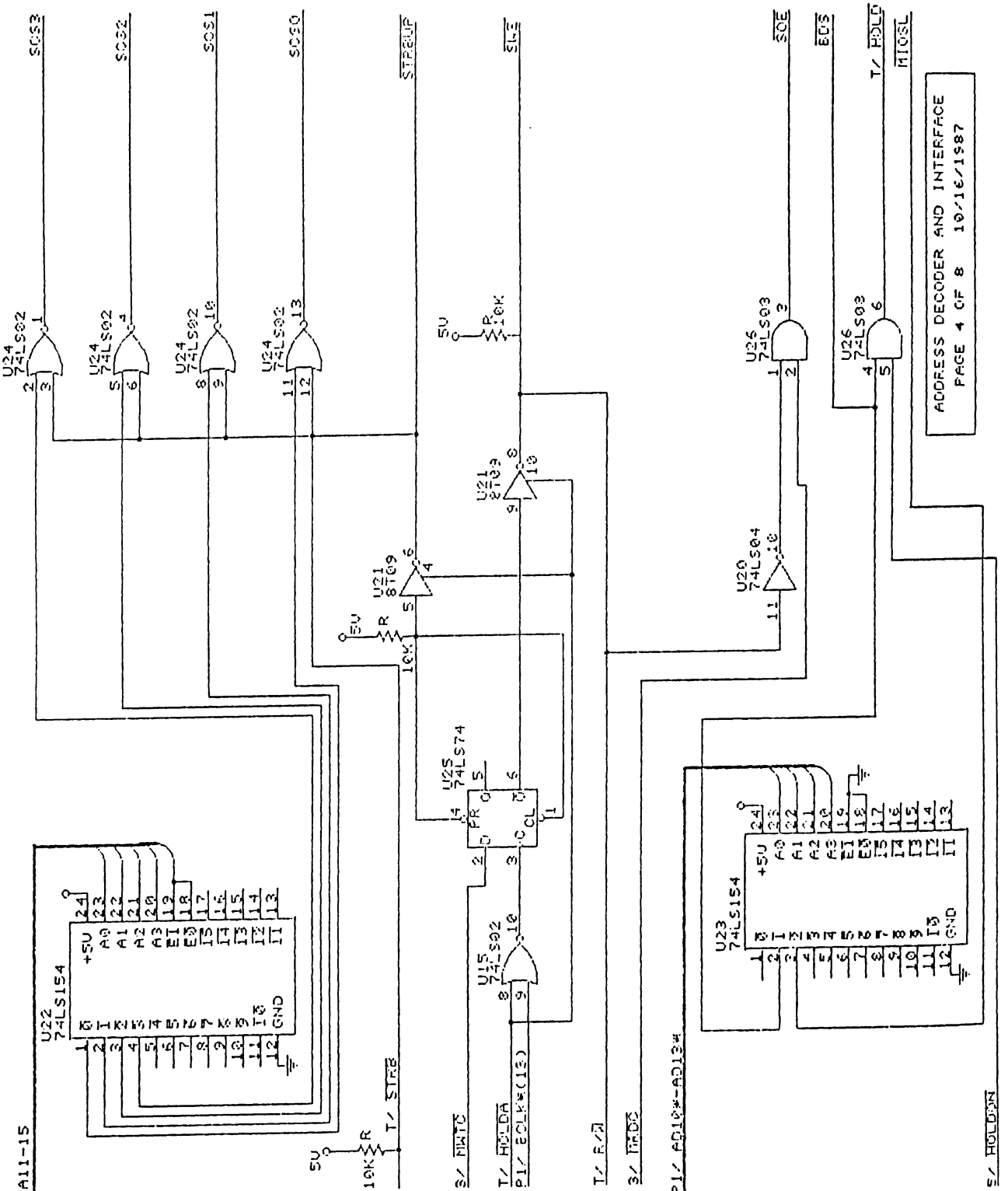


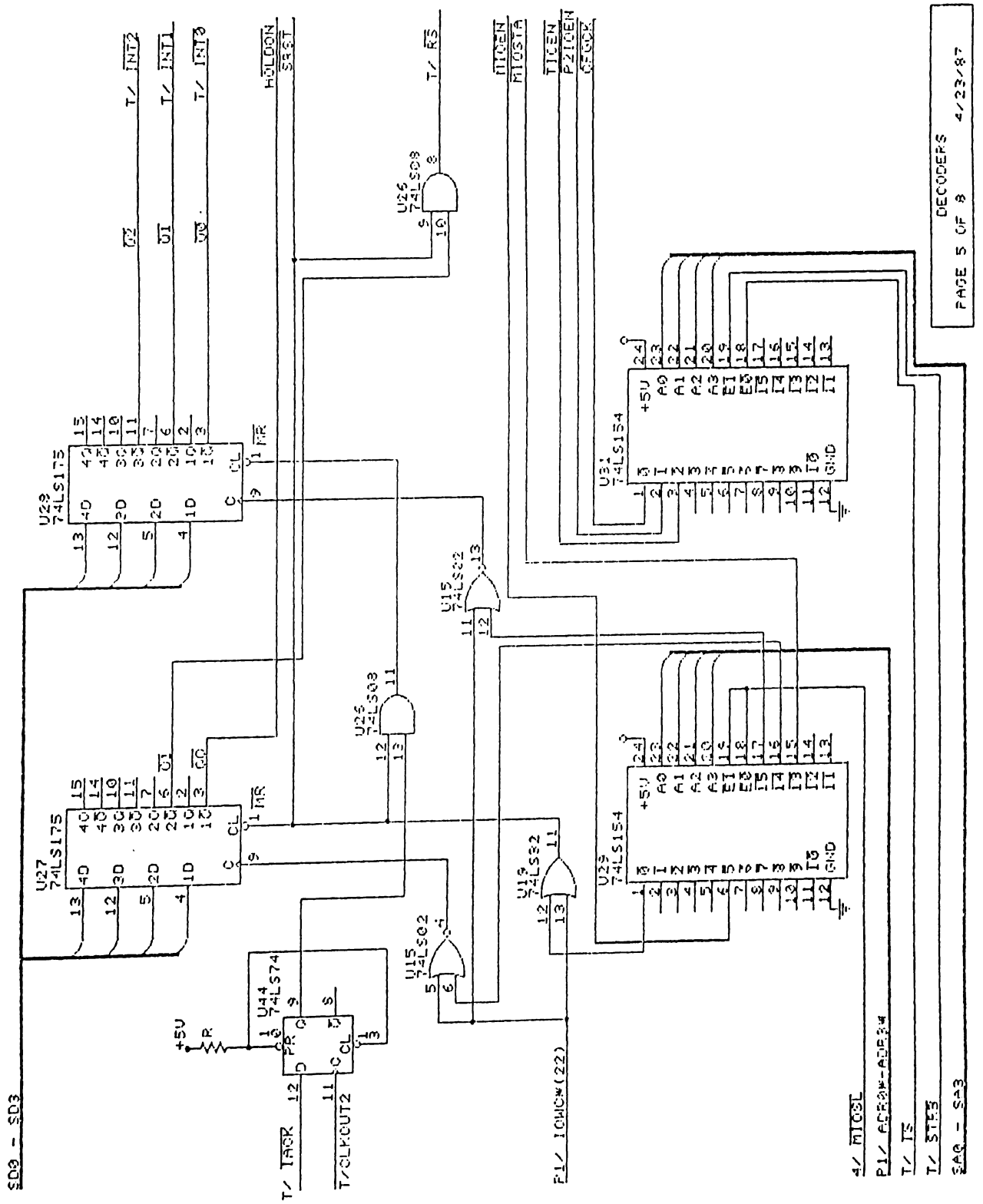

A0 - A11
MEN
CEN
WE
CLKOUT
AD0 - AD15
TMS32010
PAGE 0 10/15/1987
TMS32010
U100
A1/PA1
A2/PA2
A0/PA0
MC/MP
DS
IAK
CLKOUT
X1
X2/CLKIN
BIO
VSS
D8
D9
D10
D11
D12
D13
D14
D15
D7
D6
A3
A4
A5
A6
A7
A8
VCC
A9
A10
A11
D0
D1
D2
D3
D4
D5
+5V
XTAL
INTRES
INTINT
INTBIO

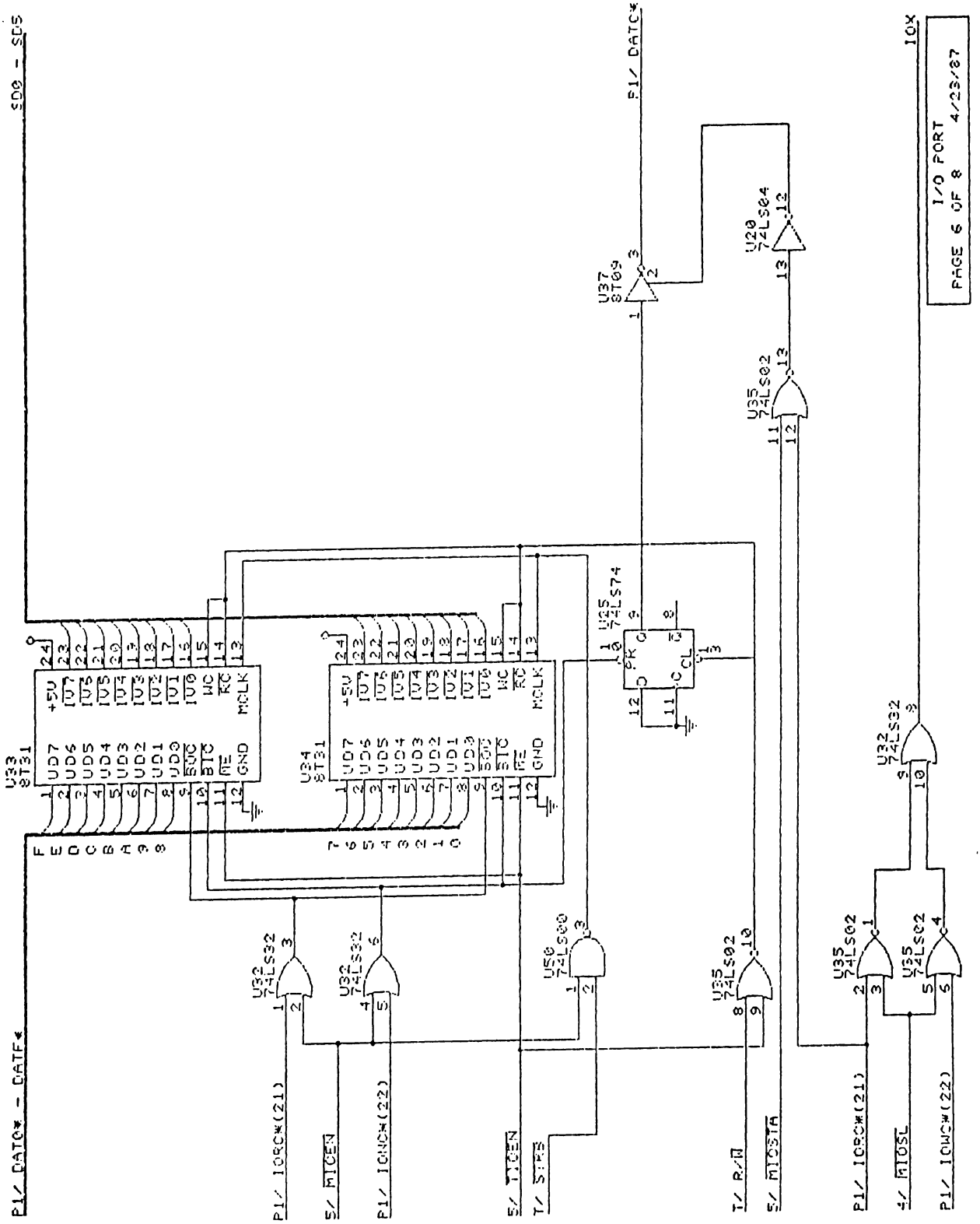

8K * 16 MEMORY
PAGE 1 OF 8
4/22/87
SD0-SD15
SA0-SA12
U1 TM2016D1
U2 TM2016D2
U3 TM2016D1
U4 TM2016D2
U5 TM2016D1
U6 TM2016D2
U7 TM2016D1
U8 TM2016D2
U9 74LS00
2/SEN0
2/SEN1
2/SEN2
2/SEN3
4/SOE
4/SWE

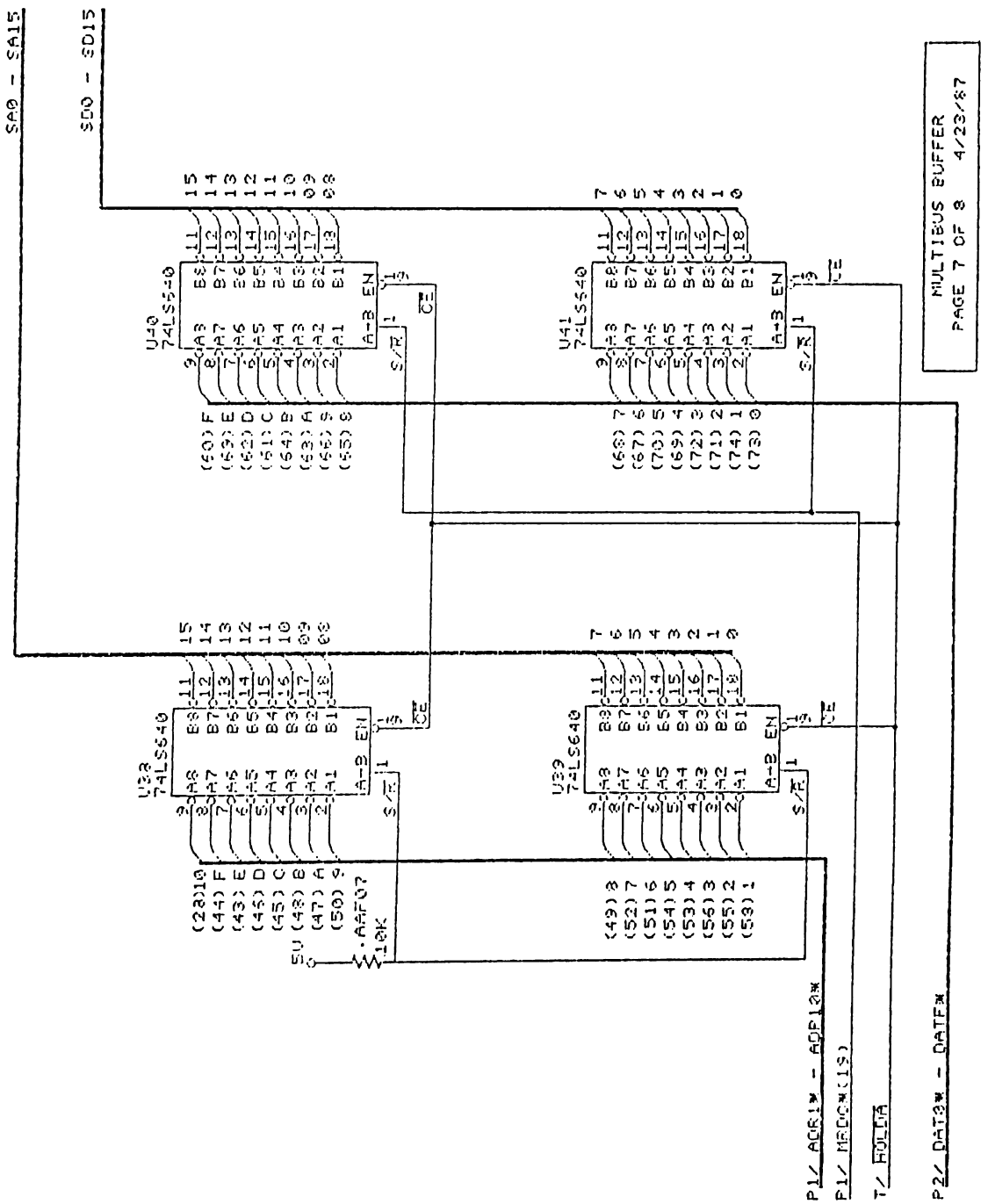

FAM CONFIGURATION CKT.
PAGE 2 OF 8 4/22/87
SEN0
SEN1
SEN3
SEN2
U11 74LS00
U12 74LS02
U13 74LS02
U14 74LS02
U10 74LS175
U19 74LS32
T/ FS
T/ DS
SD0-SD3
S/ SRST
T/ B/W
S/ CFCLK
S/ SEN

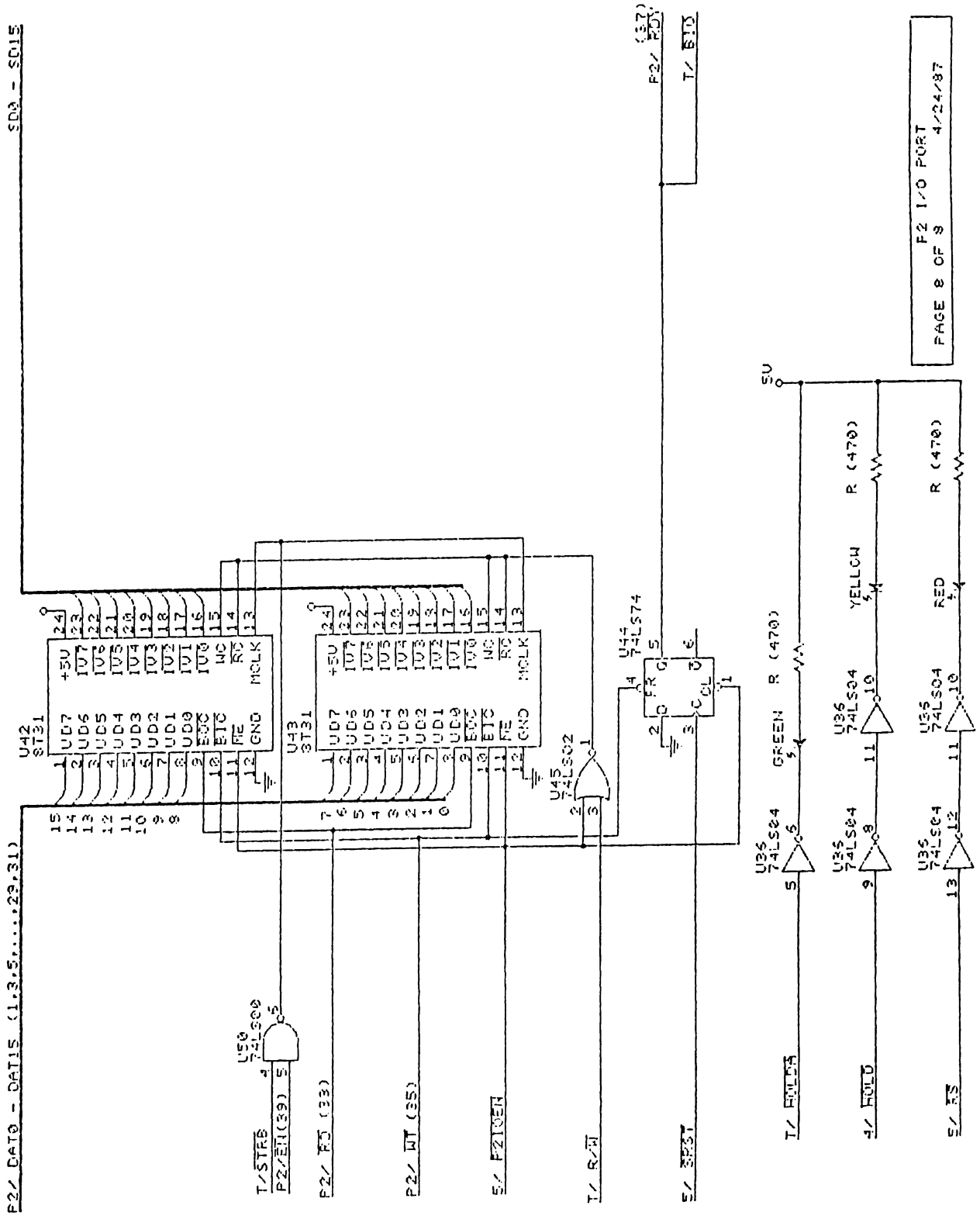

XACK AND RDY DELAY
PAGE 3 OF 8 4/23/87
MSDS
MUTC
SEN
P1/ XACK*(23)
T/ READY
P1/ MSDS*(19)
4/ EOS
P1/ MUTC*(20)
P1/ INH1*(24)
T/ HOLDA
E/ 10K
P1/ CCLK*(31)
T/ CLKOUT1
+5V
R

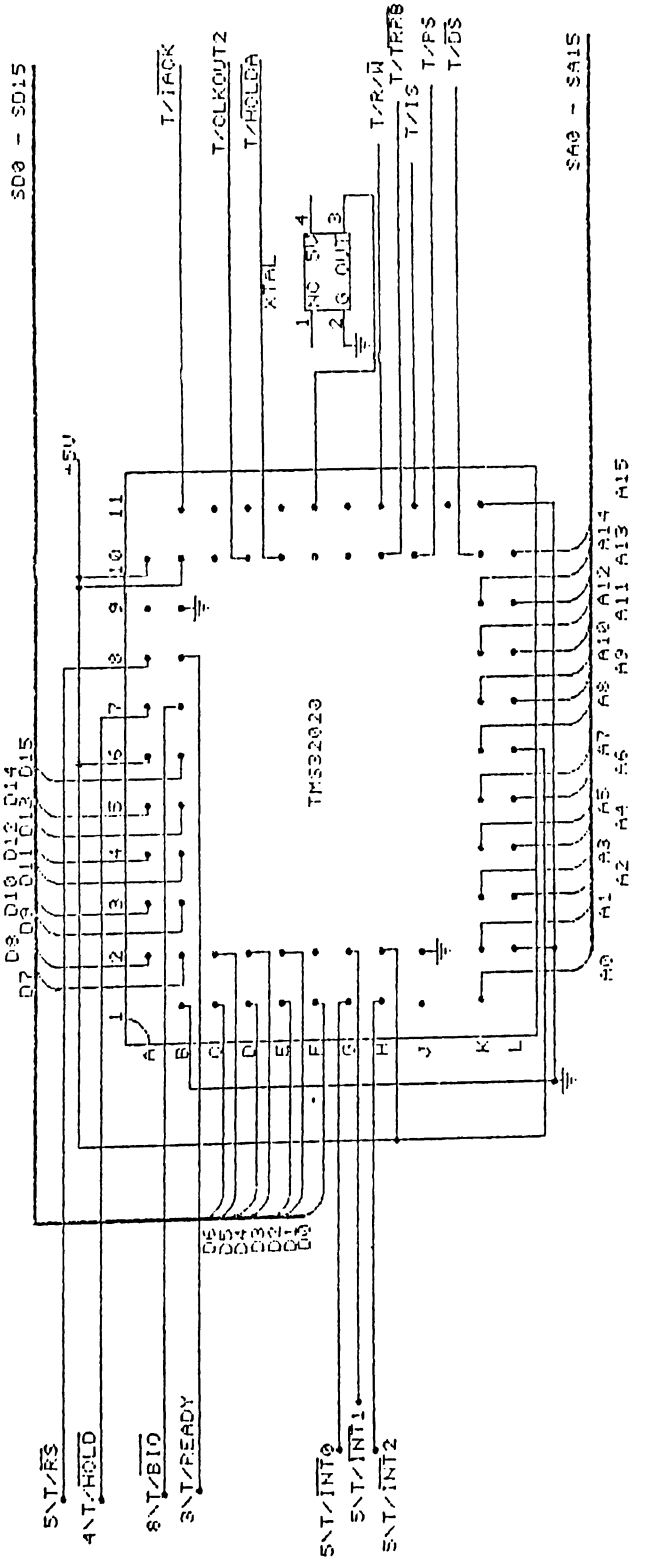

ADDRESS DECODER AND INTERFACE
PAGE 4 OF 8 10/16/1987
SCS3
SCS2
SCS1
SCS0
STRBUP
SLE
SOE
EDS
T/ HOLD
HLDSL
U24 74LS02
U21 8T09
U20 74LS04
U26 74LS03
U25 74LS74
U15 74LS02
U22 74LS154
U23 74LS154
+5V
GND
R 10K
SA11-15
T/ STRB
S/ MATC
T/ HOLDA
P1/ ECLK*(13)
T/ R/W
S/ MEMC
P1/ AD10*-AD12*
S/ HOLDIN


DECODERS
PAGE 5 OF 8
4/23/87
SD0 - SD3
T/ INT2
T/ INT1
T/ INT0
HOLDON
SRST
T/ RS
MICEN
MIOSTA
PICEN
PIOSTA
CPUCK
U29 74LS175
U27 74LS175
U26 74LS08
U15 74LS02
U44 74LS74
U31 74LS154
U19 74LS32
+5V
R
T/ TAOR
T/CLKOUT2
P1/ IOWCW(22)
4/ MIOSL
P1/ ADR0W-ADR3W
T/ IS
T/ STBS
SA0 - SA3


SD0 - SD5
P1/ DATC* - DATF*
P1/ DATC*
IOX
I/O PORT
PAGE 6 OF 8 4/23/87
U33 8T31
U34 8T31
U37 8T09
U20 74LS04
U35 74LS02
U25 74LS74
U32 74LS32
U30 74LS32
U36 74LS00
P1/ IORC*(21)
P1/ IOWC*(22)
S/ MICEN
T/ STB5
T/ R/W
S/ MIOSTA
S/ MIOSL


SA0 - SA15
SD0 - SD15
MULTIBUS BUFFER
PAGE 7 OF 8 4/23/87
U40 74LS640
U41 74LS640
U38 74LS640
U39 74LS640
A-B EN
S/R
P1/ ADR1M - ADR15M
P1/ MRDCM(15)
T/ HOLDA
P2/ DAT0M - DAT7M


SD0 - SD15
P2/ DAT0 - DAT15 (1,3,5,...,29,31)
U42 ST31
U43 ST31
U45 74LS02
U44 74LS74
U50 74LS00
T/STRB
P2/EN(39)
P2/ RS (33)
P2/ WT (35)
S/ P2TOEN
T/ R/W
S/ RESET
F2/ RDY (37)
T/ BTS
+5V
GREEN R (470)
YELLOW R (470)
RED R (470)
U36 74LS04
T/ HOLDA
T/ HOLD
S/ IS
F2 I/O PORT
PAGE 8 OF 9
4/24/87


SD0 - SD15
SA0 - SA15
+5V
T/IACK
T/CLKOUT2
T/HOLDA
XTAL
T/R/W
T/TRFB
T/IS
T/PS
T/DS
TMS32020
5NT/RS
4NT/HOLD
8NT/BIO
3NT/READY
5NT/INT0
5NT/INT1
5NT/INT2

Appendix E DSP32010 to A/D and D/A digital Interface Data Sheet

The interface handshaking is implemented on A/D to DSP32010 only. DSP32010 to D/A interface assumes correct synchronization between A/D and D/A converters, i.e., they are controlled by the same timing circuitry.

On the A/D interface, REQ-B signal is latched into a high state each time a new sample(14 bits parallel data) is ready for DSP32010 to process. DSP32010 resets REQ-B signal by putting a active low pulse through DATA-TRANS signal on the interface.

Appendix F Program Listing

The following pages contain the source programs of a sinewave generator described in this manual. They are

sine.c - This program is to be executed on DB32016. It computes sine values at equally spaced angles between 0 and 360 degrees.

tblout.s - This assembly routine is called by *sine.c* to transfer the sine values from DB32016's memory to DSP32020's dual port memory.

tblsine.mac - This is a TMS32020 assembly program. It performs table lookup on the sine table.

in2out.mac - This TMS32010 assembly program carries out the synchronization between the sinewave generator and D/A converter.

```
/*-------------------------------------------------------
*/
/*      This program, "sine.c", computes sine value, stores them in array
        'sinetbl[]', and calls subroutine "movtbl()" to transfer the table
         to DSP32010
*/


#include      <math.h>
#define       N       64              /* size of the table */
#define       A       8000.0          /* maximum amplitude */

main()
{
int     i,                                      /* loop index       */
        sinetbl[N+1];                           /* sine table       */
float   tmp;
```

```
        sinetbl[0]=N;
        tmp=6.283185307/N;
        for (i=1; i<=N; i++)
                sinetbl[i]=A*sin((i-1)*tmp);

        movtbl(sinetbl);

}
```

```
;-------------------------------------------------------------
;
; "tblout.s" moves the sine table to DSP32020
; to assemble: nasm -o movtbl.o movtbl.s
;
_movtbl::
        movd    h'e0800,r1      ; table address on DSP32020
        movd    8(sp),r0 ; address of sine table on DB32016
        movd    0(r0),r2 ; first entry is the size of table
        addqd   1,r2            ; send table size also
        movqd   0,r3
loop:   movw    0(r0),0(r1)
        addqd   4,r0
        addqd   2,r1
        addqd   1,r3
        cmpd    r3,r2
        ble     loop
        rxp     0
```

```
;-------------------------------------------------------------------
;
;       "tblsine.mac" performs table lookup on a sine table stored on
;       DSP32020. The table values are provided by DB32016 through the
;       Multibus interface
;
        .list off
        .include "/usr/users/hwa/macutls/utilg.mac"
        .list on
ZERO=0
OFFSET=50
ONE=51
DELTA=10
YOUT=11
INDEX=12
TBLOFF=13   ; table starting address
```

```
MODMASK=14          ; modulo mask
;
        procinit20      "GO","INTSEV","GO","GO","GO","GO","GO","GO"
NA:     .word   1025    ; table starting address in program RAM
;
GO:     LDPK  7         ; eighth page of data memory
;
;       binary to offset initialization
        LACK  1
        SACL  ONE
        LAC   ONE,13
        SACL  OFFSET
;
        EINT            ; enable interrupt
WTINT:      B       WTINT; use interrupt to start
;
SINGEN:     EINT
        LACK  NA    ; this is the only way to get numbers bigger than 255
        TBLR  TBLOFF        ; into data RAM
        LAC   TBLOFF
        SUB   ONE   ; compute address for N in program RAM
        TBLR  MODMASK     ; read N into data RAM *** N MUST BE POWER OF BASE 2 ***
        LAC   MODMASK
        SUB   ONE   ; compute and
        SACL  MODMASK     ; store modulo mask
;
LOOP:   ADD   TBLOFF        ; calculate effective table index
        TBLR  YOUT  ; read a sine value into data memory
        LAC   YOUT
        ADD   OFFSET        ; convert to offset binary
        SACL  YOUT
WTIO:   BIOZ  WTIO  ; I/O wait
        OUT   YOUT,PA1; output a sample
        IN    YOUT,PA1; dummy input
        LAC   INDEX
        ADD   DELTA; increment INDEX
        AND   MODMASK     ; modulo N
        SACL  INDEX
        B     LOOP
;
; Interrupt service
INTSEV:     IN      DELTA,PA2; get DELTA
        B       SINGEN
;
```

```
;---------------------------------------------------------------
;
;       "in2out.mac" performs sinewave generator and D/A sampling rate
```

```
;       synchronization

        .list off
        .include "/usr/users/hwa/macutl/utilg.mac"
        .list on

        proginit10      "go","go"       ; initialize TMS32010
go:     ldpk    0                       ; use the first page of data memory
wio:    bioz    wio                     ; polling BIO
        in      1,pa6                   ; get data from DSP32020
        out     1,pa5                   ; output to D/A
        in      1,pa5                   ; dummy input to reset BIO signal
        out     1,pa6                   ; interrupt DSP32020 to get new data
        b       wio
```

Appendix G Miscellaneous Commands and Files

This appendix describes some useful commands for the workstation operations.

tasm10/tasm20 These two command files allow the user to run the macro and the assembler with a single command.

```
"tasm10"

#
if ($#argv == 0) then
        echo "usage [-m] [-ln] files..."
else
        if ($1 == '-m' | $1 == '-M') then
                set MACLIS = '-l'
                shift
        else
                set MACLIS = ' '
                endif
        if ($#argv > 1) then
                set ASMLIS = $1
                shift
        else
                set ASMLIS = ' '
                endif
        if ($#argv != 1) then
                echo "usage [-m] [-ln] files..."
        else
                mac $MACLIS $1
                masm10 $ASMLIS $1
                endif
```

```
endif

-----------------------------------------
"tasm20"


#
if ($#argv == 0) then

        echo "usage [-m] [-ln] files..."
else

        if ($1 == '-m' | $1 == '-M') then

                set MACLIS = '-l'
                shift
        else

                set MACLIS = ' '
                endif
        if ($#argv > 1) then

                set ASMLIS = $1
                shift
        else

                set ASMLIS = ' '
                endif
        if ($#argv != 1) then

                echo "usage [-m] [-ln] files..."
        else

                mac $MACLIS $1
                masm20 $ASMLIS $1
                endif
endif
```

load10/load20 the following two command files convert TMS320 object code into DB32016 assembly programs and make them executable files.

```
-------------------------------
"load10"
```

```
#
if ($#argv == 0) then

        echo "usage: load10 obj_file [start_addr(hex)]"
else

        if (-f $1) then

                set filename = $1:r
                if ($#argv == 1) then

                        intasm10 <$1 >$filename.s
                else

                        intasm10a $2 <$1 >$filename.s
                endif
                echo ">> " $filename".s"
                nasm -o $filename.o $filename.s
                echo ">> " $filename".o"
                nmeld -T 9000 -o $filename.out $filename.o
                echo ">> " $filename".out"
        else

                echo "file not found: " $1
        endif

endif


----------------------------------------
"load20"


#
if ($#argv == 0) then

        echo "usage: load20 obj_file [start_addr(hex)]"
else

        if (-f $1) then

                set filename = $1:r
                if ($#argv == 1) then

                        intasm20 <$1 >$filename.s
                else

                        intasm20a $2 <$1 >$filename.s
                endif

                echo ">> " $filename".s"
                nasm -o $filename.o $filename.s
```

```
                echo ">> " $filename".o"
                nmeld -T 9000 -o $filename.out $filename.o
                echo ">> " $filename".out"
        else

                echo "file not found. " $1
        endif

endif
```

fld the program allows fast loading of DSP320 boards by using debugger's *replace* command.

"fld.c"

```
#include <stdio.h>
main(argc, argv)
int argc;
char *argv[];
{
        /* This program convert  simple Intel check-sum records */
        /* to a GENIX dbg16 debugger command file using 'replace'
          command to put opcode into TMS320s' memory directly   */
        /*
                George Hwa
                September 28, 1987

                Usage: fld [starting_address] [v]
                - starting_address is the memory address, specified is
                hexdecimal, where the opcode will be loaded,
                default is 0.

                - v is the verify option, default is no verify.   */

        char    tc,cc,iline[80] ;
        int     i,address;

        address = 0 ;
        if( argc >= 2 )
                sscanf(argv[1],"%x", &address);
        printf("sr 2h0);
        scanf("%s",iline) ;        /* discard the first line  */
```

```
            scanf("%c",&tc) ;
            while( tc != ':')
            {       if(tc == 'B')
                    {       if(argc >= 3 )
                                    printf("r 0%x 0, address);
                            else
                                    printf("r 0%x/nv 0, address);
                            address =  address + 2 ;

                            for(i=1 ; i<=4 ; i++)
                            {       scanf("%c",&cc) ;
                                    printf("%c",cc) ;
                            }
                            printf("U) ;
                    }
                    else if(tc == '8' | tc == '7') { scanf("%s",iline) ; }
                    scanf("%c",&tc) ·
            }

    }
```

For example, to load and program "test.obj" into DSP32010, type

% fld a0000 <test.obj >test.ind <cr>

% dbg16 <cr>

--> r 0c00030/nv <cr> ; put DSP32010 board in reset state

? 1 <cr> ; using Multibus LOCK signal

--> se/o <cr> ; turn off echo

--> @test <cr> ; execute 'test.ind' to load the program

--> r 0c00030/nv <cr> ; release DSP32010 from the reset state

? 0 <cr>

To load a program into DSP32020, the starting address should be "e0000". DSP32020 can be held in HOLD state while loading is in progress. To put DSP32020 in HOLD state, type

--> r 8d0001/nv <cr>

? 2 <cr> To release DSP32020 from the HOLD state, type

--> r 8d0001/nv <cr>

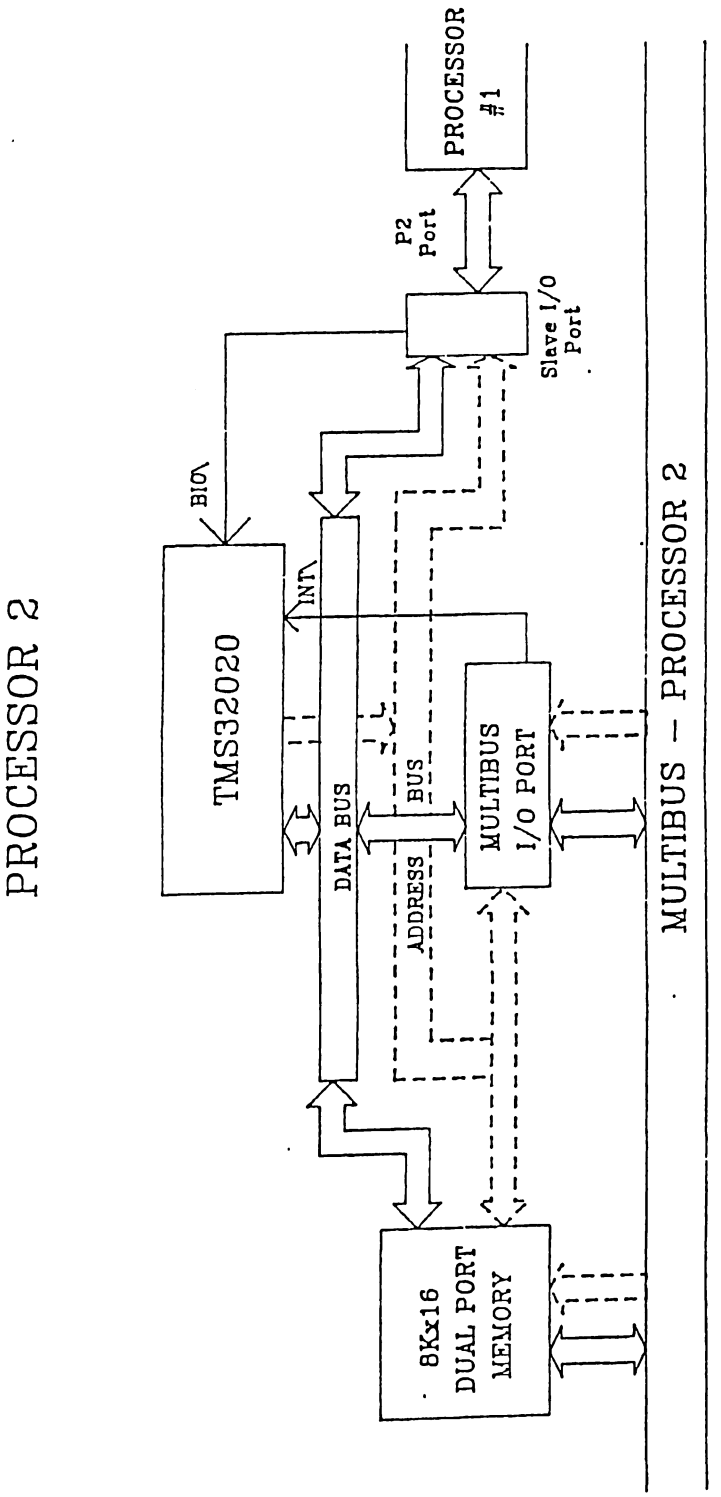
? 3 <cr>

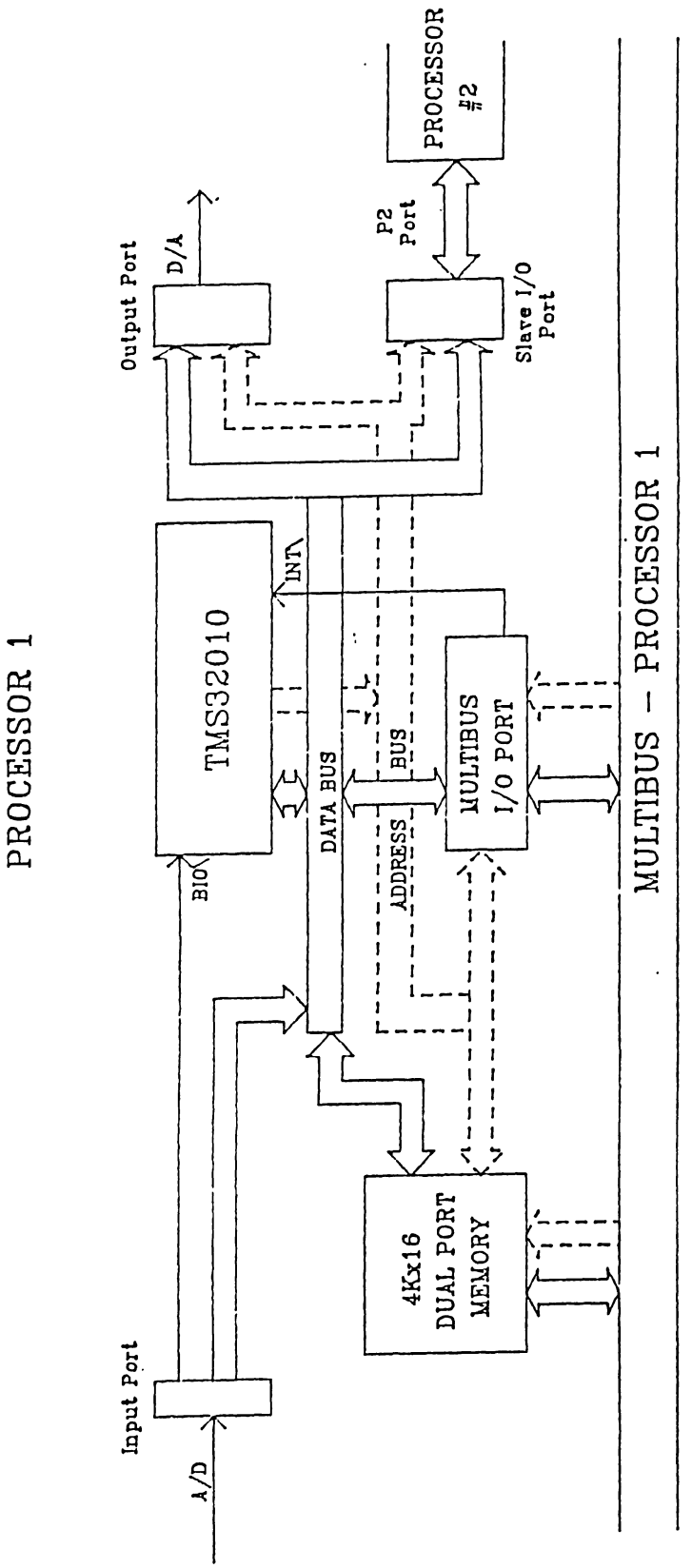
dbg16.ini this is an initialization command file for the debugger.

```
"dbg16.ini"

sl tty    ; select terminal line
sr 4h     ; select radix 4 byte hex
r .mmsr;
80000   ;
r .msr   ;
1b80000;
r .ptb0  ;
8400     ;
pc 28000..67fff/v/sw/p  ;
mf 28000..67fff  ; clear Plessey memory
0         ;          board
pc 0a0000..0a1ffe/v/sw/p          ;
pc 0e0000..0e3ffe/v/sw/p          ;
r 8d000f          ; reset DSP320-20 board
0
```

Appendix H System Block Diagrams




PROCESSOR 1
TMS32010
INT
BIO
DATA BUS
ADDRESS
BUS
MULTIBUS I/O PORT
4Kx16 DUAL PORT MEMORY
Input Port
A/D
Output Port
D/A
P2 Port
Slave I/O Port
PROCESSOR #2
MULTIBUS - PROCESSOR 1

REFERENCES

[1] Texas Instruments: "TMS32010 User's Guide, 16/32-bit Digital Signal Processor", *May 1983*

[2] Texas Instruments: "TMS32020 User's Guide, Digital Signal Processor Products", *Copyright © 1985*

[3] Texas Instruments: "Digital Signal Processing Applications with the TMS320 Family, Theory, Algorithms, and Implementations", *Copyright © 1986*

[4] National Semiconductor Corporation: "Series 32000 Databook", *Copyright © 1984* [5] National Semiconductor Corporation: "Series 32000, GENIX Symbolic Debugger Reference Manual", *October 1984*

[6] National Semiconductor Corporation: "Series 32000, Development Board Monitor Reference Manual", *June 1984*

[7] National Semiconductor Corporation: "Series 32000, GENIX Cross-Support Software Programmer's Manual, Volume 1 & 2", *October 1985*

[8] National Semiconductor Corporation: "Series 32000, DB32016 Development Board User's Manual", *May 1985*

[9] National Semiconductor Corporation: "Series 32000, Instruction Set Reference Manual", *June 1984*

[10] James B. Johnson & Steve Kassel: "The Multibus Design Guidebook, Structure, Architectures, and Applications", *Copyright © 1984 by McGraw-Hill, Inc.*

[11] Intel: "Multibus Handbook", Copyright © 1983, Intel Corporation

[12] Plessey Microsystems: "PSM 512Λ Error-Correcting DRAM User's Guide", Copyright © 1982 The Plessey Company Limited

Appendix B Adpative Filter Program Listing

```
                .list off
                .include "ut1p.mac"
                .include "utilg.mac"
                .list on
;               2/5/87 Adaptive Hormanic Cancellation
;               George C. Hwa
;               CCSP - NCSU
;
;               This routine is written to implement AHC on TMS32020 board of the
;               workstation.
;
;               miscellaneous data reside in page 7 of block 1 on chip memory
;
p2port=1
delayqueue=2
wlmsblock=3
xlmsblock=4
;
zero=0
one=1
data=2
count=3
maskad=4
offset=5
error=6
mu=7
delay=8
muerr=9
ed=10
innerprod=11
tap=12
;
; page 4 is dedicated to delay queue
; page 5 is used for XLMS coefficients
; page 6 is used for WLMS coefficients
; page 7 is for miscellaneous data and direct addressing is used
;
                proginit20      "go","go","go","go","go","go","go","go"
;
go:             sovm                            ; set overflow mode
                spm             0               ; product output no shift
                ssxm                            ; sign-extension
;
                ldpk            7               ; page 7 for direct addressing
                zac
                sacl            zero
```

```
                lack        1
                sacl        one
                lalk        256             ; gain factor
                sacl        mu
                lack        10              ; delay
                sacl        delay
                lalk        54              ; number of taps
                sacl        tap
                INITMASKAD  one,offset,maskad
;
                lrlk        xlmsblock,640   ; starting address for XLMS block(page 5)
                lrlk        wlmsblock,768   ; starting address for WLMS block(page 6)
                lalk        639
                sub         delay
                sacl        ed
                lar         delayqueue,ed   ; starting address for delay queue(page 4)
                lar         0,delay         ; used for auto increment "ar4" for delay pipe
;
; clear XLMS block
                larp        xlmsblock
                zac
                rpt         tap
                sacl        *+
                mar         *-,wlmsblock
;
; clear WLMS block
                rpt         tap
                sacl        *+
                mar         *-
;
;
; filter loop
wio:            bioz        wio             ; wait for BIOZ interrupt
;

                in          *,p2port        ; input a sample
; complement bits
                lac         *

                out         error,p2port    ; output the processed sample

; offset binary to two's complement conversion
                and         maskad
                sub         offset
                sacl        *0+             ; ar2=end of delay queue
;
; set up decorrelation delay; samples are shoveled down the dealy pipe
;
                rpt         delay
                dmov        *-
                mar         *+,xlmsblock
```

```
;
; use LMS algorithm to model an unknown filter
;
            mpyk        0
            zac
            lar         1,tap ; ar1 as the loop counter

trans:      ltd         *-,wlmsblock    ; load T, move data, accumulate, decrement ar4, and
larp ar3
            mpy         *-,1            ; multiply and larp ar1
            banz        trans,*-,xlmsblock ; branch on ar1=0, or decrement ar1, and larp ar4
            mar         *+
            mar         *+,wlmsblock
            mar         *+,delayqueue
            apac                        ; add the last product
            sach        innerprod,4
;
; error calculation for LMS muerr = mu*error
            lac         innerprod

            sub         *,0,wlmsblock
            sacl        error
            lt          mu              ; load gain factor
            mpy         error
            pac
            sach        muerr,4
;
; LMS adapt to adjust coeficients wl = wl + muerr*xl
;
            lt          muerr
            lar         1,tap
lmsad:      lac         *,12,xlmsblock
            mpy         *+,wlmsblock    ; muerr*xl
            spac                        ; add to wl
            sach        *+,4,1
            banz        lmsad,*-,wlmsblock
            mar         *-,xlmsblock
            mar         *-
            mar         *-,delayqueue
;
            lac         innerprod
            add         offset          ; convert predicted signal to offset binary
            sacl        innerprod

            lac         error
            add         offset          ; convert error to offset binary
            sacl        error
            b           wio
```