

Servo Compensation Using a Floating Point Digital Signal Processor

by

Susan Jean Wittman

S.B., Aeronautical and Astronautical Engineering
Massachusetts Institute of Technology
(1986)

SUBMITTED TO THE DEPARTMENT OF AERONAUTICS AND
ASTRONAUTICS IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE IN AERONAUTICS AND ASTRONAUTICS
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
February, 1989

© Susan J. Wittman 1989

The author hereby grants to M.I.T. and the M.I.T. Lincoln Laboratory, permission to reproduce and to distribute copies of this thesis document in whole or in part.

Signature of Author _____
Department of Aeronautics and Astronautics
January, 1989

Certified by _____
Department of Aeronautics and Astronautics
Professor James M. ...
and Computer Science
Thesis Supervisor

Approved by _____
Norval P. Smith
Academic Supervisor

Accepted by _____
Chairman, Departmental Graduate Committee

MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

MAR 10 1989

LIBRARIES

WITHDRAWN
M.I.T.
LIBRARIES

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LINCOLN LABORATORY
LEXINGTON, MASSACHUSETTS 02173-0073

5 April 1989

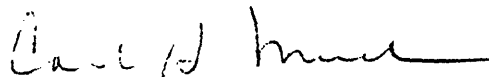
Area Code 617
863-3500

Chairman, Department Graduate Committee
Department of Aeronautics & Astronautics
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Dear Mr. Chairman:

The thesis submitted to your department by Susan J. Wittman as part of her master of science degree requirement titled "Servo Compensation Using a Floating Point Digital Signal Processor", has been released by the Department of Defense for general use. So, you may reproduce, publish and distribute the thesis as you see fit.

Very truly yours,



Carl H. Much
Group Leader,
Control Systems Engineering

CHM/apg

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LINCOLN LABORATORY
LEXINGTON, MASSACHUSETTS 02173-0073

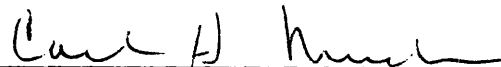
January 20, 1989

Chairman, Department Graduate Committee
Department of Aeronautical and Astronautical Engineering
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Dear Mr. Chairman:

I have reviewed the attached thesis of Susan J. Wittman on behalf of the MIT Lincoln Laboratory. The thesis is within the scope of the thesis proposal previously approved and in my opinion does not contain any material that is objectionable to the MIT Lincoln Laboratory. It is also approved for its technical content.

It is understood that the actual thesis document will be the permanent property of MIT and will be placed in the MIT Library within one month after the date of submission. However, the thesis will not be given general circulation until a review by the Department of Defense has been completed and the thesis released for general use. I will inform the Department of Aeronautical and Astronautical Engineering of the results of such review as soon as possible. It is also understood that Susan J. Wittman has granted MIT and the MIT Lincoln Laboratory the right to reproduce, publish, and distribute the thesis.



Carl H. Much
Group Leader, Control Systems Engineering
MIT Lincoln Laboratory

Servo Compensation Using a Floating Point Digital Signal Processor

by

Susan Jean Wittman

Submitted to the Department of Aeronautics and Astronautics
on January 20, 1988 in partial fulfillment of the requirements
for the degree of Master of Science

ABSTRACT

The capability of using a 32-bit floating-point digital signal processor (DSP32) in a closed loop feedback system is established. A dedicated input/output (I/O) board for interfacing with the serial port of the DSP32 was designed and built. Filter programs were written and tested to evaluate the I/O circuit and the DSP32. A simulated position loop was closed around an operational amplifier circuit simulating a rate loop. The DSP32 was used in the position of forward compensator in an actual closed loop, single axis, 40 Hz mirror position servomechanism. The DSP32 showed some promise of being flexible and fast enough for use in compensating higher bandwidth systems.

Thesis Supervisor:	Dr. James K. Roberge
Title:	Professor of Electrical Engineering and Computer Science
Technical Supervisor:	Dr. Norval P. Smith
Title:	Technical Staff Member The M.I.T. Lincoln Laboratory

Acknowledgement

I want to thank my thesis advisor, Professor James K. Roberge, for introducing me to the Lincoln Laboratory and for taking on this thesis. He provided direction and goals for this work even extending into the realm of “evil digital things”. His encouragement, assistance and concern went far beyond the scope of this thesis and were greatly appreciated.

I want to express deep gratitude to my technical supervisor, Dr. N. P. Smith, for all his help. He was a continuous source of ideas dealing with both experimental and theoretical approaches. Special thanks go to him for explaining the analog mirror servomechanism to me and for his help running and interfacing with the mirror system.

I wish to extend my sincere thanks to all the individuals in the Control Systems Group at the Lincoln Laboratory for taking the time to explain things to me and for being my friends. Special thanks go to the Group Leaders, Stobo Wright and Carl Much, for allowing me to work in their group.

I also want to thank John R. Coffee for sharing his expertise in the configuration and usage of L^AT_EX, the typesetter used to prepare this document.

The following AT&T registered trademarks are mentioned in this thesis:

WE[®] DSP32 Digital Signal Processor

WE[®] DSP32-DS Digital Signal Processor Development System

WE[®] DSP32-SL Support Software Library

This work was sponsored by the Department of the Air Force and performed at the MIT Lincoln Laboratory. The views expressed in this document are those of the author and do not reflect the official policy or position of the U.S. Government or the MIT Lincoln Laboratory. It is published for the exchange and stimulation of ideas.

Table of Contents

Abstract	3
Acknowledgement	3
Table of Contents	4
List of Figures	6
List of Tables	8
1 Introduction	9
1.1 Overview	9
1.2 The Mirror Servomechanism	10
1.3 The DSP32 Digital Signal Processor	12
1.3.1 Device Hardware	12
1.3.2 Device Software	14
2 Experimental Configuration	15
2.1 Overview	15
2.2 Hardware Configuration	15
2.2.1 The Development System	16
2.2.2 The Input/Output Board	17
2.3 Software Overview	22
2.3.1 The Software Support Library	22
2.3.2 A Programming Architecture	23
2.3.3 A Specific Software Example	24
3 Preliminary Designs	26
3.1 The First Design: A Low Pass Filter	26
3.2 The Second Design: A Notch Filter	28
3.2.1 The Analog Notch Filter	30
3.2.2 The Pole-Zero Matched Digital Notch Filter	30
3.2.3 The Bilinear Transform Notch Filter	31

4	Compensated Systems	34
4.1	The Op Amp Design	34
4.1.1	DSP32 Unity Gain Block	35
4.1.2	DSP32 Proportional-plus-Integral Block	38
4.2	The Overall Mirror Servomechanism	40
4.2.1	The Mirror Compensator Design	40
4.2.2	The Mirror Compensator Performance	42
5	Conclusions	46
5.1	DSP32 Results	46
5.2	Future Work	47
	References	49
A	DSP32 Information	50
B	Computer Programs	52
B.1	100 Hz Low Pass Filter	52
B.2	100 Hz Notch Filter	53
B.3	Unity Gain Block	53
B.4	Proportional-plus-Integral Compensator	54
B.5	Mirror Controller (Analog Gains)	55
B.6	Mirror Controller (159 Hz Rate Pole)	56
B.7	Mirror Controller (159 Hz Rate Pole and 70% Position Gain)	57
C	Detailed I/O Board Circuitry	59

List of Figures

1	Introduction	9
1.1	Analog Mirror Control Loops	10
1.2	Closed Loop Operational Amplifier System	11
1.3	Mirror Loop Digital Control Interfaces	12
2	Experimental Configuration	15
2.1	Hardware System Connections	16
2.2	I/O Board Input Circuitry	18
2.3	I/O Board Output Circuitry	19
2.4	I/O Board Timing and Control Circuitry	21
2.5	Master Sequencer Timing Diagram	21
2.6	Computer Code used for 100 Hz Notch Filter	25
3	Preliminary Designs	26
3.1	Computer Generated 100 Hz Low Pass Filter	29
3.2	DSP32 100 Hz Low Pass Filter	29
3.3	Magnitude Responses for Pole-Zero Notches	31
3.4	Computer Simulated 100 Hz Notch Filter Bode Plots	33
3.5	Actual DSP32 100 Hz Notch Filter Bode Plots	33
4	Compensated Systems	34
4.1	Op Amp Rate Plant	34
4.2	Plant Block Diagram	35
4.3	Closed Loop DSP32 Compensator Configuration	35
4.4	Closed Loop Op Amp Plant Frequency Response	36
4.5	Closed Loop Frequency Response With Unity Gain DSP32	37
4.6	Closed Loop Step Response With Unity Gain DSP32	37
4.7	Step Response of DSP32 Compensated Plant	40
4.8	Mirror Analog Control Gains	40
4.9	Input Multiplexor	41
4.10	Analog Compensator Mirror Step Response	43
4.11	Digital Compensator Mirror Step Response	43

4.12	Mirror Step Response – 159 Hz rate pole	44
4.13	Mirror Step Response – 159 Hz rate pole, 70 % position gain . . .	44
4.14	Analog Power Amplifier Drive Signal	45
4.15	DSP32 Power Amplifier Drive Signal	45
5	Conclusions	46
A	DSP32 Information	50
A.1	DSP32 Block Diagram	51
A.2	Memory Configurations	51
A.3	DSP32 Latencies	52
B	Computer Programs	53
C	Detailed I/O Board Circuitry	60

List of Tables

1	Introduction	9
2	Experimental Configuration	15
2.1	Software Tools	23
3	Preliminary Designs	26
3.1	Pole–Zero Notch Filter Gain Settings	31
4	Compensated Systems	34
4.1	DSP32 vs. Op Amp Compensator Step Characteristics	39

Chapter 1

Introduction

1.1 Overview

Digital signal processing chips are ideally suited to perform mathematically-intensive, repetitive algorithms. Conventional microprocessors which use microcode to do this processing tend to be slow. A dedicated hardware approach leads to inflexible particular solutions to various applications. The AT&T WE DSP32 Digital Signal Processor combines flexibility through programmability, with high speed [1].

Complex image processing and speech recognition are examples of areas where the DSP32 can be used effectively. This thesis involves using a DSP32 as a compensator in a closed loop servomechanism control configuration. The capability of using the DSP32 for control purposes in closed loop feedback systems is established.

This thesis is organized as follows. The remainder of this chapter discusses the mirror servomechanism configuration and the required background information on the DSP32. The hardware and software experimental configuration are discussed in Chapter 2. Chapter 3 contains the preliminary filter designs done using the DSP32. These programs laid the foundation for the more complicated controllers discussed in Chapter 4. The design and evaluation of two different compensated hardware systems is covered in Chapter 4. Chapter 5 sums up the results of the thesis and suggests future work.

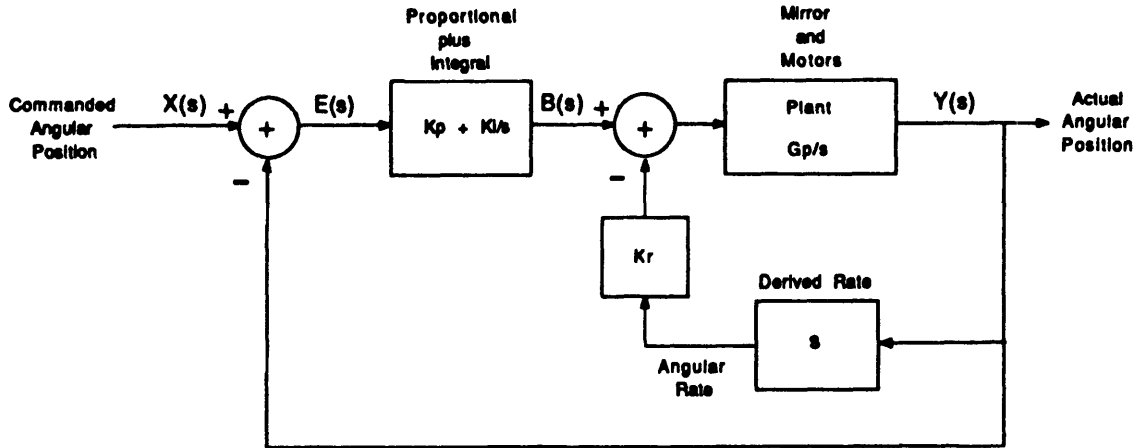


Figure 1.1: Analog Mirror Control Loops

1.2 The Mirror Servomechanism

The analog controller for a two-axis, three inch diameter mirror was designed and built by Dr. N. P. Smith, at the M.I.T. Lincoln Laboratory. Each axis consisted of a proportional-plus-integral position loop, and a proportional derived rate loop as shown in Figure 1.1.

The plant includes the mirror and its dynamics, the power amplifiers used to drive the motors, and the torque motors which were mounted on the shafts connected to the mirror. The elevation axis control system had a 40 Hz position loop bandwidth, with an approximately 120 Hz rate loop bandwidth.

It was found that the transfer function of the motor and mirror configuration (shown in Figure 1.1 as G_p) could be approximated as a dominant single pole of the form:

$$G_p(s) \approx \frac{1}{\tau s + 1}$$

The transfer function for the closed rate loop, shown in Figure 1.1 from input $B(s)$ to output $Y(s)$, is of the form:

$$\frac{Y(s)}{B(s)} = \frac{\frac{1}{s(\tau s + 1)}}{1 + \frac{K_r s}{s(\tau s + 1)}} = \frac{1}{s(\tau s + 1 + K_r)} \quad (1.1)$$

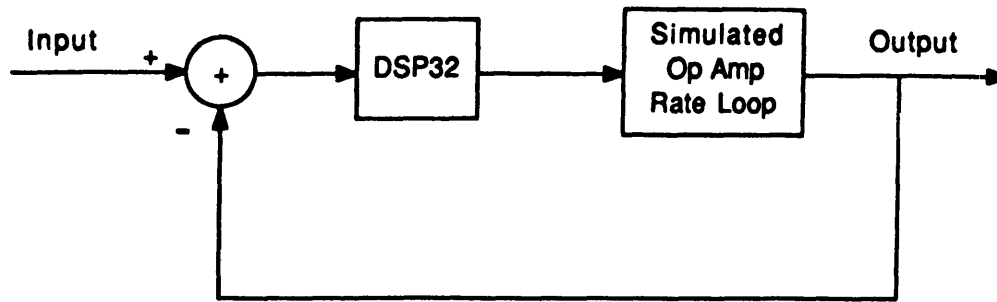


Figure 1.2: Closed Loop Operational Amplifier System

where

K_r = derived rate gain

τ = dominant rate loop lag

An operational amplifier (op amp) circuit simulating the transfer function in Equation 1.1 was built [2]. A simulated position loop was closed around this simulated rate loop (see Figure 1.2) to evaluate the DSP32 as a digital forward compensator. For comparison, an analog proportional-plus-integral compensator was also designed and implemented using the same control gains as the DSP32. Successful closed loop operation using the DSP32 demonstrated its capability as a digital compensator in closed loop control systems.

To evaluate the use of the DSP32 as a closed loop compensator, it was substituted for the analog compensator in the actual mirror servo system. Figure 1.3 shows the location of switches added to the the analog mirror controller that could be used to bypass the analog circuitry with the DSP32. The three switches made three distinct modes of operation possible. The first mode was total analog compensation. The second mode involved switching the analog proportional-plus-integral position compensation out of the loop and replacing it with the DSP32. The third mode of operation was total digital compensation. Both the analog position and rate compensation were disabled for total digital control.

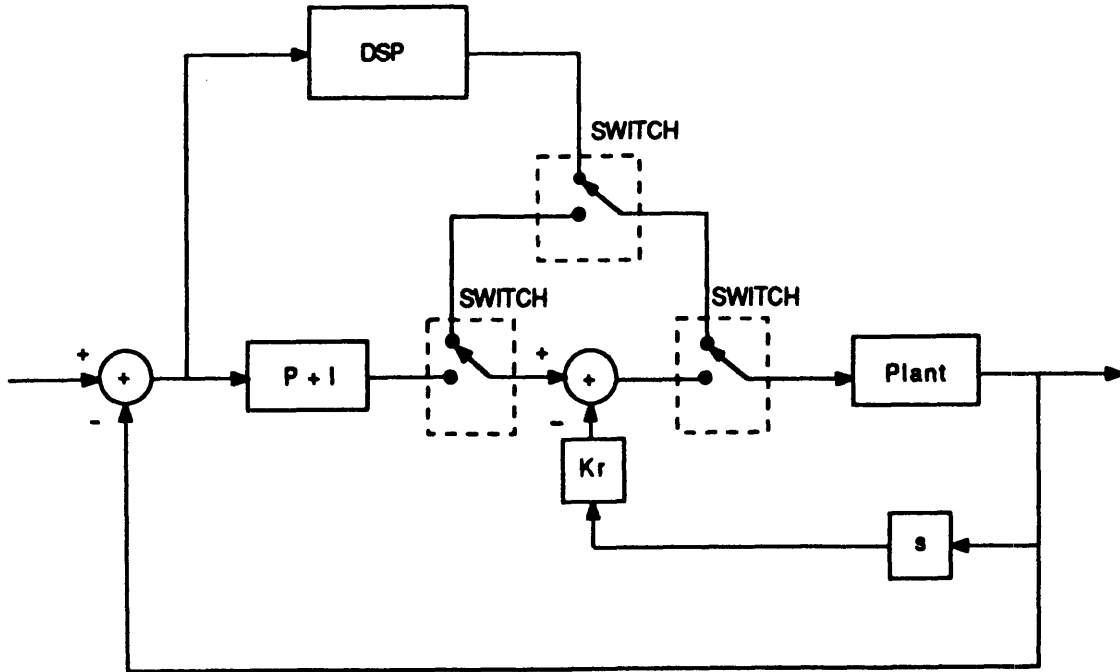


Figure 1.3: Mirror Loop Digital Control Interfaces

1.3 The DSP32 Digital Signal Processor

The DSP32 uses a hardware multiplier and an arithmetic logic unit to execute 32-bit floating-point mathematics and logic operations. A highly pipelined architecture allows the device to achieve throughput on the order of 8 million floating-point operations per second. The floating-point capabilities of the device allow much greater system dynamic range than that obtainable with fixed-point digital signal processors. This is especially important in control system design, where the effects of scaling, normalization, and overflow are critical to performance. The DSP32 is also the first 32-bit floating-point digital signal processor on the market.

1.3.1 Device Hardware

The version of the DSP32 device without direct external memory interfacing, which was used in this thesis, comes in a 40 pin dual-in-line package requiring

a single +5 volt power supply. The DSP32 consists of two main execution units: the control arithmetic unit (CAU) and the data arithmetic unit (DAU). Control, addressing, arithmetic and logic functions are supported by the 16-bit fixed-point CAU. The DAU is a 32-bit signal processing unit containing four 40-bit accumulators.

On-chip memory includes 2048 bytes of ROM and 4096 bytes of RAM, which can be addressed as 8-, 16-, or 32-bit words. Four main memory address configurations are possible, although only one of them has the RAM portion of memory assigned to the first block of addresses. (Appendix A shows all four configurations for reference.) The DSP32 follows the instructions it finds in memory starting with the lowest address, so it is important to designate RAM in this lowest address location. The memory is divided into two banks, which must be alternatively accessed to achieve maximum throughput.

Both serial and parallel ports are available for interfacing with external devices. Both of these ports also contain a direct memory access option for transfers between the input and output buffers and memory, without program intervention.

The DSP32 processor cycle is divided into four states: the instruction fetch, two memory reads and one memory write state. The high degree of pipelining used results in up to six different instructions in various stages of fetch-decode-execute during any particular processor cycle. Various latencies (described in Appendix A) due to the execution of instructions in the pipeline have to be included in the programming of the DSP32.

The Input/Output Control Register (IOC) sets the input/output (I/O) conditions for the DSP32 when interfacing with external devices. These conditions include information such as data lengths, clocking and synchronization.

1.3.2 Device Software

The DSP32 has two instruction sets. One set, called the data arithmetic instructions, is for signal processing algorithms. The other set, the control arithmetic instructions, is for control and logic instructions. The program coding strongly resembles assembly language.

A complete set of software tools is available for creating, debugging, and testing DSP32 application programs using the WE DSP32-SL Support Software Library [3]. These tools interface with MS-DOS¹ on an IBM computer and include programs for compiling, linking, loading, simulating and emulating (through the WE DSP32-DS Digital Signal Processor Development System) code [4].

¹MS-DOS[®] is a registered trademark of the Microsoft Corporation.

Chapter 2

Experimental Configuration

2.1 Overview

The experimental configuration was broken into two distinct parts: the hardware set-up and the software set-up. Both of these are described in detail in the following sections. The hardware configuration is defined here to be the physical devices used to program and run the DSP32 including the IBM XT, the development system, and the I/O interface board. All the programming methodologies and the resulting code downloaded to the DSP32, including the software tools used to generate and move the code, are included in the software configuration.

2.2 Hardware Configuration

The overall hardware system connections are shown in Figure 2.1. An IBM XT computer was used to do the programming and downloading to the DSP32 chip. The development system was connected to the IBM through a serial port. The development system was set up to function as an emulator connected to a dedicated I/O board via a ribbon cable. The I/O board provided a means of getting analog signals to and from the DSP32 chip located in the development system. The next two sections discuss first the development system and then the I/O board.

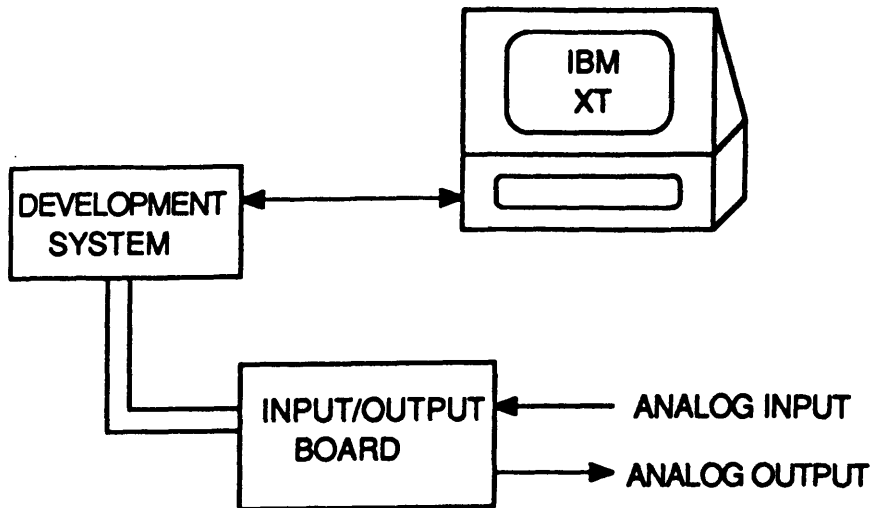


Figure 2.1: Hardware System Connections

2.2.1 The Development System

A WE DSP32 Digital Signal Processor Development System was purchased from AT&T. The development system contained a DSP32 device, memory, a microcomputer, and communication ports. The user interface to the development system was provided by the DSP32 simulator described in Section 2.3. The IBM XT computer hosted the DSP32 simulator software and linked to the development system.

The development system could be used to download and execute programs on the DSP32 device. Examination of the contents of DSP32 registers, accumulators, and memory was enabled. In-circuit emulation allowed testing of the system's analog I/O Interface, and subsequently enabled the DSP32 device contained in the development system to be used as a controller in closed loop applications.

2.2.2 The Input/Output Board

The precision codec¹ supplied in the development system contains low pass filters which cause unacceptable phase lags between input and output signals. A dedicated I/O interface board for the DSP32 digital signal processor with 12-bit analog-to-digital (A/D) and digital-to-analog (D/A) converters was designed and built so that the codec could be bypassed. A complete circuit diagram of the I/O board is included in Appendix C.

The DSP32 can input sampled data through both the serial and parallel ports. It was decided to interface with the serial port in order to leave the parallel port open for interfacing with other devices, such as a microprocessor, if desired. This approach also made it relatively easy to change the sampling rate of the data.

Three main functional units comprise the interface board: the input circuitry, the output circuitry, and the timing and control circuitry. The function of the input circuit and its topology is described in the following section. The output circuit is described next. The circuitry that controls the input and output circuits is described in last. The timing and control circuit is responsible for regulating all the signals on the I/O board and is basically the master sequencer that steps the I/O board through all its functions.

Input Circuitry

The overall input circuitry is illustrated in Figure 2.2. The main functional blocks are the sample and hold (S/H) circuit, the A/D converter and the parallel-to-serial shift register.

The LF398H monolithic integrated circuit is used as the S/H circuit. Its acquisition time is less than 10 μ sec and its logic input is TTL compatible. The

¹codec is the tradename of an integrated circuit which contains an A/D converter, anti-aliasing filters, a D/A converter and a smoothing filter.

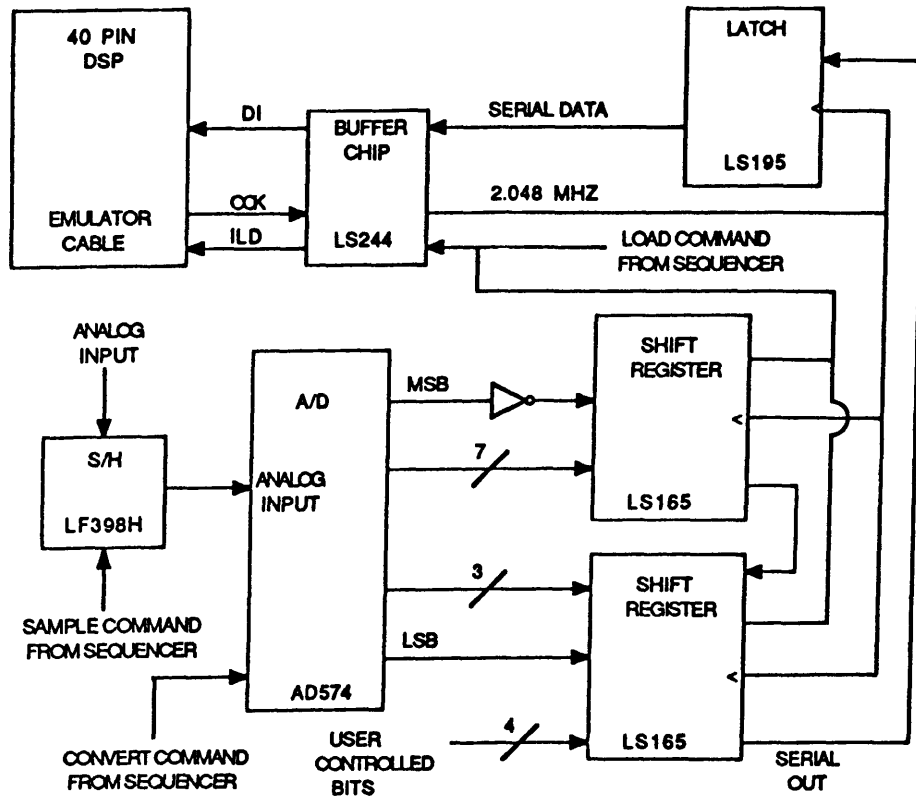


Figure 2.2: I/O Board Input Circuitry

AD574 was chosen as the A/D converter. It is a 12-bit successive approximation A/D converter with a 35 μ sec conversion time.

The S/H chip was connected to the A/D converter. The signals to sample the analog data line and to convert the sampled data to 12 data bits are generated by the master sequencer in the timing and control circuitry. The master sequencer also triggers the shift register to load the converted data. The parallel-in, serial-out shift register (two LS165's) shifts this data to a latch. Notice that the shift register is continuously shifting at the rate of OCK, which is 2.048 MHz, but that the data isn't loaded into the DSP32 until the input load command (ILD) comes from the master sequencer.

Since the DSP32 was programmed to accept 16-bit data streams, the four least significant bits could be user supplied to encode information to the DSP32.

The output circuit is functionally similar to the input circuit. The serial data stream from the DSP32 goes through a buffer and is shifted into a serial-in, parallel-out shift register (two LS165's). When the DSP32 data transfer to the shift register is complete the OSE signal from the DSP32 latches the data in the LS165's into latches (LS374's). The data is then converted by a DAC80 12-bit D/A converter to an analog signal. The inverters on the output data stream and on the most significant bit to the DAC80 are used to convert the two's complement format of the DSP32 into the offset binary format required by the DAC80.

Timing and Control Circuitry

The basic timing signals for the I/O board were derived from the DSP32 generated Output Clock (OCK). The I/O control word (IOC) was selected to produce a 2.048 MHz clock signal on OCK. This signal and all other signals to and from the DSP32 via the emulator cable were isolated from the DSP32 by a buffer chip, an LS244 tristate buffer. The timing and control circuitry is shown in Figure 2.4.

The main idea for controlling the I/O board was to build a master sequencer by dividing down the 2.048 MHz reference clock from OCK and then using it to trigger a shift register to shift a control signal to various parts of the circuit at appropriate times. Figure 2.5 shows a timing diagram of this sequencer's output.

Two dual 4-bit LS393 counter chips were used to divide down OCK by powers of two. The 128 kHz signal obtained by this method was used to "clock" the shift register.

The divided down clock signals between 1 kHz and 16 kHz were used as candidate sample rate sources. A set of switches was connected as an 8:1 selector so that different clock signals could be chosen. One of these switches was available

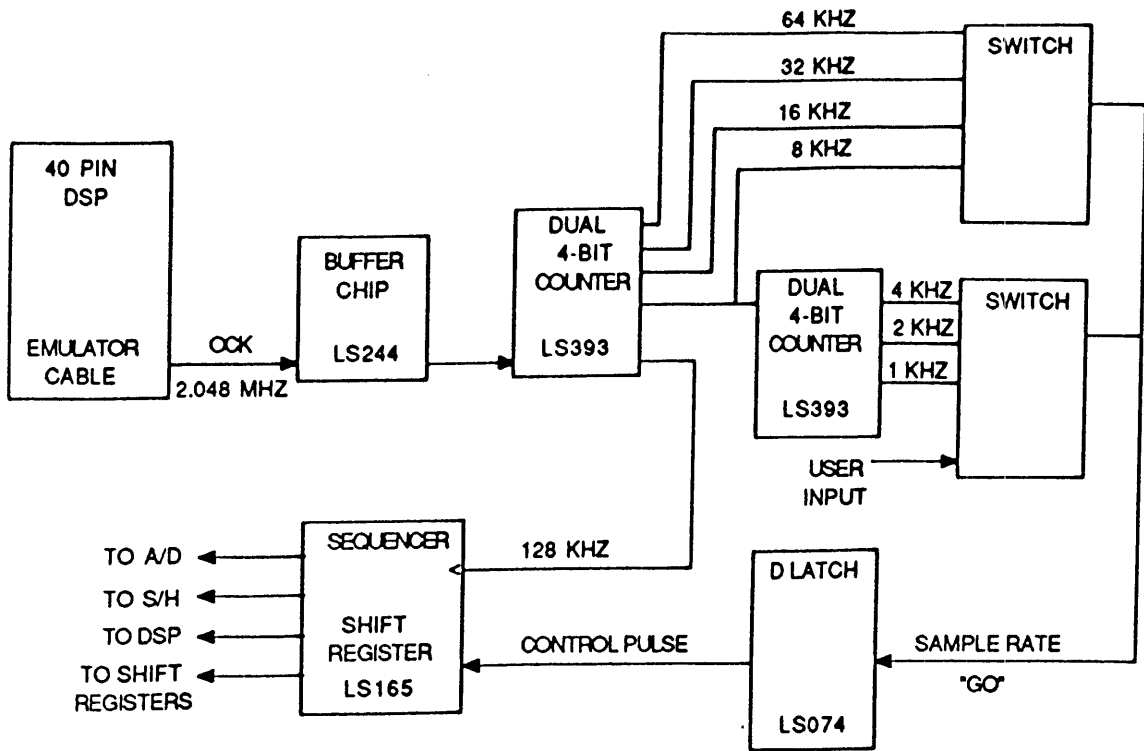


Figure 2.4: I/O Board Timing and Control Circuitry

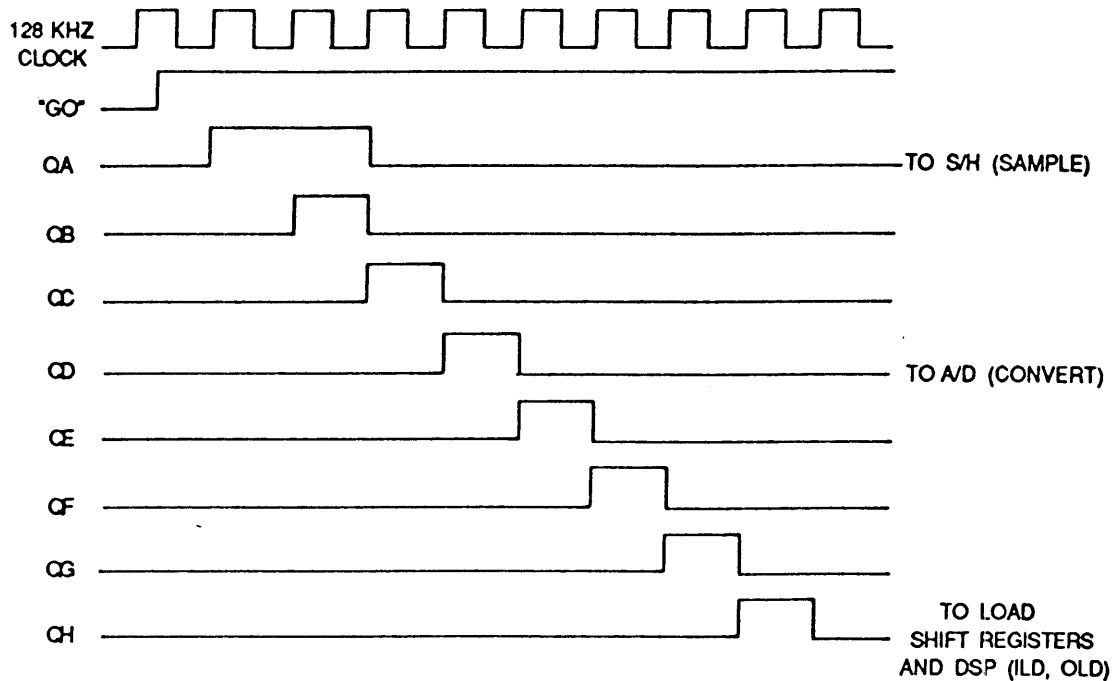


Figure 2.5: Master Sequencer Timing Diagram

for the user to supply a clock signal in case a specific frequency other than a divided down factor of the OCK signal was desired for sampling. The sampling rate signal thus supplied was designated as the "GO" signal and connected to a D-type latch, which in turn triggered the serial input of the shift register with a high logic level input control signal. This signal was then shifted through the shift register. By connecting the output pins of the shift register to the triggers of the different various parts of the I/O board, the shift register acted like a master sequencing controller. The A/D converter could be triggered after waiting the appropriate acquisition time for the S/H device, and the data could be shifted into the DSP32 after waiting for the A/D converter to settle.

2.3 Software Overview

This section discusses the various issues involved with the software associated with programming and using the DSP32. It begins with a description of tools available with the Software Support Library [3], continues by defining a typical programming architecture, and ends with a specific example of the software used to implement the 100 Hz notch filter described in Section 3.2. A complete list of the various programs downloaded to the DSP32 can be found in Appendix B.

2.3.1 The Software Support Library

The Software Support Library comes with an assembler and a link editor. The commands for invoking these tools, along with both the input files and the resulting output files are shown in Table 2.1. File extensions denote file types, where:

`.s` = assembly source code

`.o` = object code

`.l` = listing file

The final assembled and linked code has no file extension.

Table 2.1: Software Tools

Function	Invocation	Input File	Output File
assembler	dsp3as	sample.s	sample.o, sample.l
linker	dsp3ld	sample.o	sample
assembler/linker	dsp3make	sample.s	sample

The Software Support Library also contains a simulator. This simulator can be invoked to simulate programs on the IBM XT, or to download programs to the development system for emulation. Software breakpoints can be set in both modes of operation. The simulation program is invoked by the command “dsp3sim”. Adding the suffix “-d1” to the invocation tells the program to simulate the program on development system #1. The command “dsp3init 1” initializes the development system and establishes a communication port for downloading programs.

2.3.2 A Programming Architecture

Digital control systems often lead to transfer functions in the form of ratios of polynomials in z . A typical transfer function with second order polynomials is shown in Equation 2.1.

$$\frac{Y(z)}{X(z)} = \frac{A + Bz^{-1} + Cz^{-2}}{D + Ez^{-1} + Fz^{-2}} \quad (2.1)$$

The result of cross-multiplying out the terms of Equation 2.1 is

$$(D + Ez^{-1} + Fz^{-2})Y(z) = (A + Bz^{-1} + Cz^{-2})X(z)$$

Associating z^{-1} with a unit delay [5], and rearranging terms yields the following difference equation:

$$y_n = \frac{A}{D}x_n + \frac{B}{D}x_{n-1} + \frac{C}{D}x_{n-2} - \frac{E}{D}y_{n-1} - \frac{F}{D}y_{n-2} \quad (2.2)$$

The current output of the difference equation is y_n , with the previous output being designated as y_{n-1} . The current input to the transfer function is designated as x_n . Hence x_{n-1} and x_{n-2} designate the two previous inputs.

The DSP32 programming architecture lends itself very well to this difference equation format. A standard instruction involves both a multiply and an accumulate command. Equation 2.2 has four such instructions and one initial multiply and store instruction. Registers can be updated during these instructions, such that current inputs can become previous inputs on the next instruction cycle.

2.3.3 A Specific Software Example

The computer code shown in Figure 2.6, is meant to illustrate the major points involved with programming the DSP32 as a typical filter. It shows how straightforward the programming can be and demonstrates the modular approach taken in coding, throughout this thesis. The “dot” commands, `.rsect` and `.global`, are linker commands used to designate memory locations for the program to reside in the DSP32 memory and to define labels for sections of code for referencing throughout the program, respectively. Comments are surrounded by `/*` and `*/`.

The algorithm portion of the program is located in the first seven lines of the subroutine labeled “notch”. The rest of the program is mostly register initialization. The comments illustrate how closely the programming follows the difference equation format discussed in the previous section.

```

/*notch.s— 100 Hz Tustin Notch Filter */
.rsect    "lo_ram"
.global   sample, notch, coef, ym2, ym1, xm2, xm1, pause
ioc=0x986;          /* 16-bit I/O Board */
dauc=0;
r2=ym2; r3=xm2;    /* assign memory pointers */
r4=xm1; r5=ym1;
sample:   if(ibe) goto sample;      /* wait for sample */
          r1=coef;                  /* assign coef pointer */
notch:   a0=float(ibuf);            /* input X(n) */
          a1=*r1++**r2;             /* -F/D*Y(n-2) */
          a1=a1+*r1++**r3;         /* C/D*X(n-2)-F/D*Y(n-2) */
          a1=a1+(*r3=*r4)**r1++;   /* B/D*X(n-1)+... */
          a1=a1+(*r2=*r5)**r1++;   /* -E/D*Y(n-1)+... */
          a1=a1+(*r4=a0)**r1;      /* A/D*X(n)+... */
          *r5=a1=a1;               /* store Y(n-1) */
          goto sample;
out:     obuf=a1=int(a1);           /* new output */
.rsect   "hi_ram"                  /* coef = filter coef.*/
coef:    float -.894957983, .946778712, -1.889169001
          float 1.889169001, .948205686, 0.0
ym2:     float 0.0                  /* allocate memory
xm2:     float 0.0                  for storage */
xm1:     float 0.0
ym1:     float 0.0

```

Figure 2.6: Computer Code used for 100 Hz Notch Filter

Chapter 3

Preliminary Designs

Filter programs were written and tested to evaluate the I/O circuit designed and the DSP32.

3.1 The First Design: A Low Pass Filter

The first design attempted was a software low pass filter. It was decided to make the design breakpoint value equal to 100 Hz. A sampling rate of 8 kHz was chosen. This sampling rate was chosen relatively high compared to the break frequency in order to minimize the effects of the time delay associated with sampling in the frequency range of interest.

The sampling delay for this system comes from two sources. One is the sampling process itself which can be modeled as a zero-order [6] hold with a phase lag of $e^{-\frac{\pi r}{2}}$. The other source of time delay is the processing time. The DSP32 is configured to take an input in on one sample, process it and adjust the output on the next sample. This processing delay equals a pure time delay of the form e^{-sr} . Thus the total sampling time delay for this system is $e^{-\frac{3\pi r}{2}}$.

The analog transfer function of a one pole 100 Hz low pass filter is shown in Equation 3.1.

$$G(s) = \frac{2\pi * 100}{s + 2\pi * 100} \quad (3.1)$$

Pole-zero matching was chosen from among the various A/D transformations,

as the means for discretizing Equation 3.1. This involves setting

$$z = e^{s\tau} \quad (3.2)$$

to match the pole at $s = -2\pi * 100$, and placing a discrete zero at $z = -1$ to account for the implicit analog zero at $-\infty$. A sampling rate of 8 kHz corresponds to a sampling time of $\tau = .000125$ seconds.

$$G_D = k * \frac{2\pi * 100(z + 1)}{z - e^{-2\pi * 100 * .000125}}$$

The gain k was chosen to make the low frequency responses of the analog and digital functions match.

$$G_D(z)|_{z=1} = G(s)|_{s=0}$$

The resulting equation is shown in Equation 3.3.

$$G_D(z) = \frac{.037767375(1 + z^{-1})}{1 - .924465250z^{-1}} \quad (3.3)$$

It was useful to think of $G_D(z)$ as a difference equation for an output, y , in terms of an input, x , as follows:

$$y_n = .924465250y_{n-1} + .037767375x_n + .037767375x_{n-1}$$

This difference equation was programmed into the DSP32 for testing. The program is listed in Appendix B.

A computer simulation of G_D was run over the frequency range of 1 Hz to 1 kHz. The magnitude and phase results shown in Figure 3.1 match those of a 100 Hz low pass filter as expected. The phase angle is -45 degrees and the magnitude is down 3 dB at the breakpoint. This simulation does not include the phase lag associated with the sampling process, $e^{-\frac{3\pi\tau}{2}}$. At 100 Hz the phase lag of the sampling process is -6.75° . At 1 kHz the sampling process phase lag is -67.5° .

The HP3562A Dynamic Signal Analyzer was used to measure the frequency response of the DSP32 filter implementation. Figure 3.2 shows the actual frequency response of the DSP32 implementation. The results shown in Figure 3.2

indicate that the phase is -53 degrees and the magnitude is down a little more than 3 dB at 100 Hz. The discrepancy in phase between the actual frequency response and the computer generated frequency response is -8° which is mostly accounted for by the sampling phase lag of -6.75° . The actual phase is down to -172° at a frequency of 1 kHz, which is 14° less than the theoretical value. The model is more accurate at frequencies much less than the sampling frequency.

Even though the system is being over-sampled by a factor of 40 times the Nyquist rate, the phase lag contributed by the time delay associated with sampling is -6.75° at the desired break frequency of 100 Hz, which is not an insignificant factor. The amount of phase lag due to sampling time delays that can be tolerated for a particular system will have to be calculated in order to set an appropriate sampling rate for that system. The trade-off to be considered involves increasing the sampling rate at the cost of losing algorithm processing time. For "simple" systems like a 100 Hz low pass filter, running at high sampling rates is justified since the processor is being severely underworked and is spending most of its time waiting for the next sample to arrive.

3.2 The Second Design: A Notch Filter

The second design was a 100 Hz notch filter. A fairly high Q analog notch filter was targeted for implementation on the DSP32. The first attempts at discretization were done using the same pole-zero mapping as was used in the low pass filter design. Section 3.2.2 discusses these pole-zero mapping attempts. Notch filters with magnitude responses unsymmetrical with respect to the notch frequency were obtained and computer simulated. Different continuous-to-discrete transformations were tried in order to get a symmetrical magnitude frequency response.

The final implemented difference equation was arrived at by using the bilinear

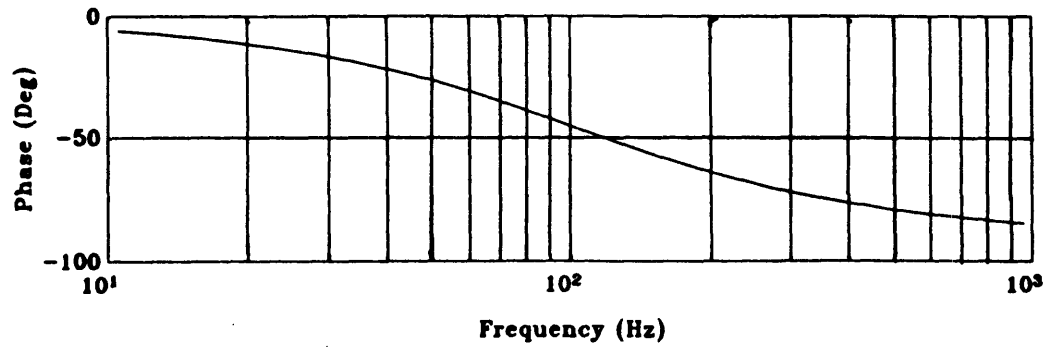
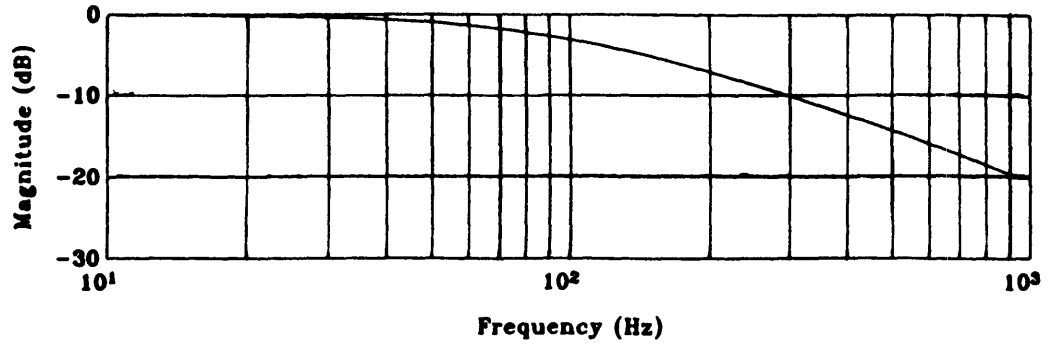


Figure 3.1: Computer Generated 100 Hz Low Pass Filter

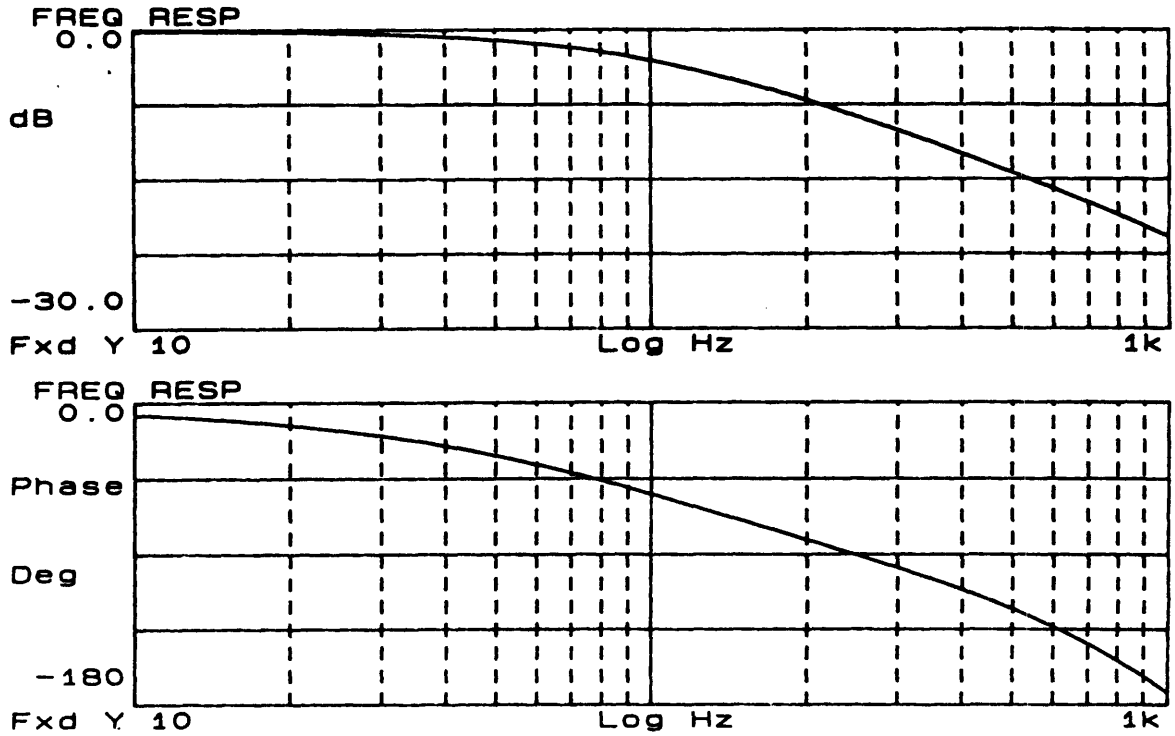


Figure 3.2: DSP32 100 Hz Low Pass Filter

transformation [6] with an 8 kHz sampling rate, where

$$s = \frac{2}{T} \left(\frac{1 - z^{-1}}{1 + z^{-1}} \right) \quad (3.4)$$

and

$$G_D(z) = G(s) \Big|_{s=\frac{2}{T} \left(\frac{1-z^{-1}}{1+z^{-1}} \right)} \quad (3.5)$$

3.2.1 The Analog Notch Filter

A two pole, two zero notch filter of the following form was chosen:

$$G(s) = \frac{\frac{s^2}{\omega_o^2} + \left(\frac{2\zeta_n}{\omega_o} \right) s + 1}{\frac{s^2}{\omega_o^2} + \left(\frac{2\zeta_d}{\omega_o} \right) s + 1}$$

where

$$\omega_o = 100 \times 2\pi,$$

$$\zeta_d = .707,$$

$$\zeta_n = .01$$

3.2.2 The Pole–Zero Matched Digital Notch Filter

Using the pole–zero mapping of Equation 3.2, $G(s)$ was transformed into the z -domain.

$$G_D(z) = k \left(\frac{1 - 1.990705833z^{-1} + .996863337z^{-2}}{1 - 1.786989192z^{-1} + .800799923z^{-2}} \right) \quad (3.6)$$

The gain of $G_D(z)$ was set to unity for various frequencies of interest by adjusting the gain k . The two frequency regions of interest were low frequencies and high frequencies.

The resulting values of k along with the resulting locations of poles and zeros are shown in Table 3.1. Notice the setting of the gain factor, k , does not influence the pole and zero locations of the transfer function. Changing the gain, k , simply translates the magnitude frequency response up and down as shown in

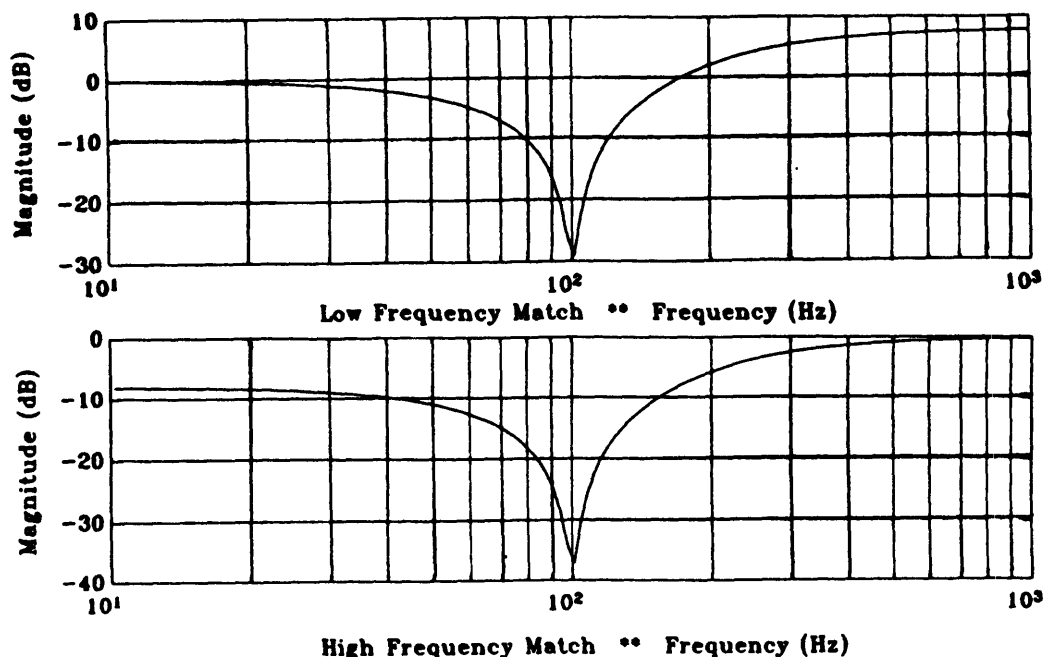


Figure 3.3: Magnitude Responses for Pole-Zero Notches

Figure 3.3. Hence both computer simulated gain settings and all other possible gain settings produce asymmetrical magnitude frequency responses about the notch frequency. This is because the gain k can only adjust the gain of the transfer function, $G_D(z)$, at one specified frequency. Setting the gain for low frequencies to a desired value does not guarantee the gain at other frequencies to be any desired value.

Table 3.1: Pole-Zero Notch Filter Gain Settings

Frequency Range	Gain, k	Z Domain Poles	Z Domain Zeros
Low Match	2.2429	$.8935 \pm .0497j$	$.9954 \pm .0783j$
High Match	.8998	$.8935 \pm .0497j$	$.9954 \pm .0783j$

3.2.3 The Bilinear Transform Notch Filter

A different continuous-to-discrete transformation was used in order to get a symmetrical magnitude frequency response. The bilinear (Tustin) transforma-

tion of Equation 3.4 was chosen. The resulting discretized equation,

$$G_D(z) = \frac{.948205686 - 1.889089965z^{-1} + .946719706z^{-2}}{1 - 1.889089965z^{-1} + .894925392z^{-2}} \quad (3.7)$$

produced the symmetrical frequency response shown in Figure 3.4. The z domain pole and zero locations associated with this $G_D(z)$ are slightly different than those of the pole-zero mapping transfer functions:

$$poles = .9445 \pm .0525j$$

$$zeros = .9961 \pm .0784j$$

The actual 100 Hz notch filter performance obtained from programming Equation 3.7 into the DSP32 is shown in Figure 3.5. The measured Bode performance is almost identical to the computer simulated performance. The high frequency phase rolls off due to the increasing significance of the sampling time delays at those frequencies.

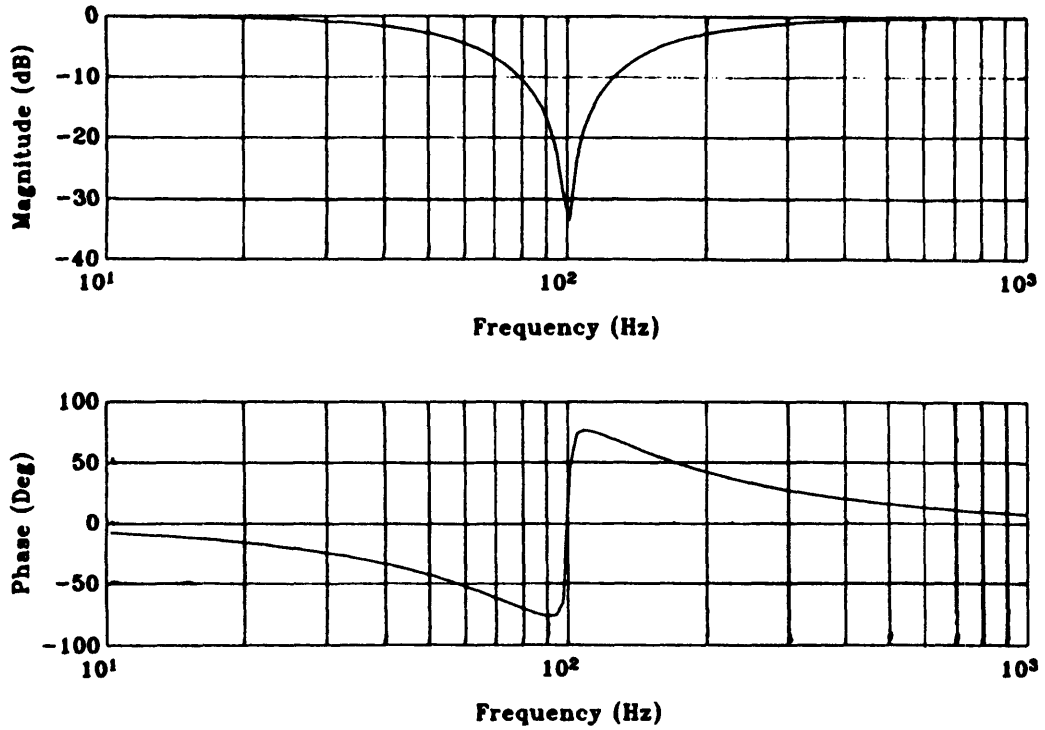


Figure 3.4: Computer Simulated 100 Hz Notch Filter Bode Plots

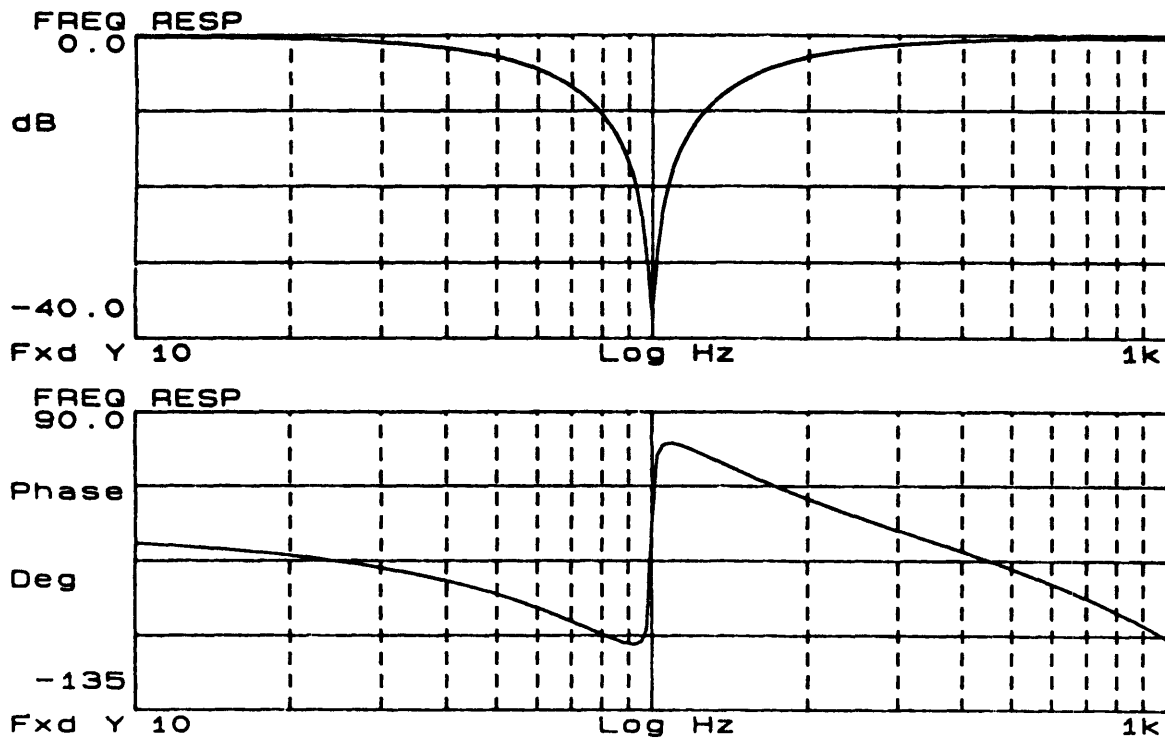


Figure 3.5: Actual DSP32 100 Hz Notch Filter Bode Plots

Chapter 4

Compensated Systems

4.1 The Op Amp Design

In order to simulate the rate loop of the mirror servomechanism (see Equation 1.1) which contained a dominant pole and an integrator, the op amp plant was designed to have a transfer function of the form:

$$\frac{Y(s)}{B(s)} = \frac{K}{s(\tau s + 1)} = \frac{100}{s(.001s + 1)} \quad (4.1)$$

The gain, K , was arbitrarily chosen to be 100 so that simply closing the plant loop without any compensation would result in closed loop poles at 18 Hz and 141 Hz. The op amp plant constructed is shown in Figure 4.1. The back-to-back zener diodes shown were used to limit the output of the op amp integrator to 9.1 volts.

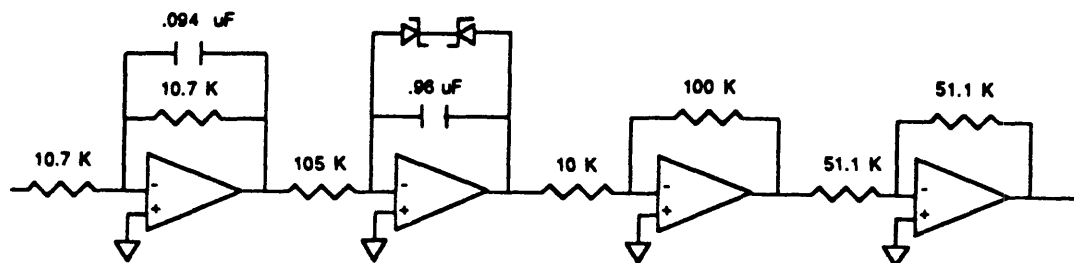


Figure 4.1: Op Amp Rate Plant

Associating each op amp shown in Figure 4.1 with its transfer function, leads to the block diagram representation of the plant shown in Figure 4.2.



Figure 4.2: Plant Block Diagram

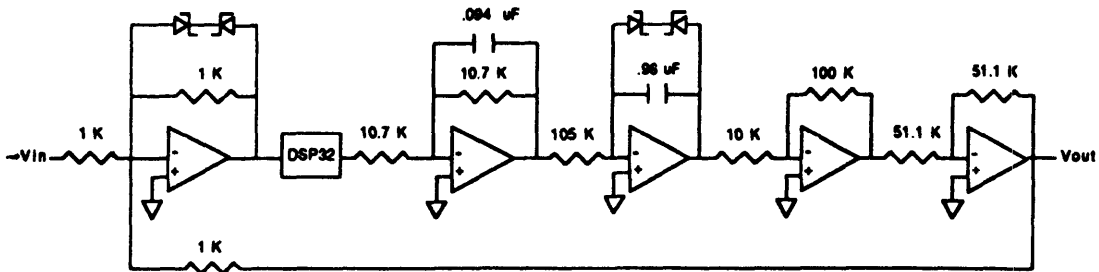


Figure 4.3: Closed Loop DSP32 Compensator Configuration

The overall plant transfer function was thus:

$$\frac{Y(s)}{B(s)} = \frac{99.2}{s(.001s + 1)}$$

In order to close the hardware loop, a summing amplifier was constructed. Zener diodes were used to voltage limit the output signal of the summer, which was used to drive the DSP32 via the I/O board. Since the A/D and D/A converters on the I/O board were set to operate in the ± 10 volt range, 9.1 volt zener diodes were chosen. The overall closed loop configuration is shown in Figure 4.3. Inverters are used to make the necessary sign changes.

To verify that the plant was behaving as expected, the loop was closed without the DSP32 and the closed loop frequency response was measured. The HP3562A Dynamic Signal Analyzer calculated the closed loop poles to be at 18 and 139 Hz from the frequency response data shown in Figure 4.4.

4.1.1 DSP32 Unity Gain Block

The next test performed was simply to place the DSP32 in the loop at the forward compensator location, programmed to be a unity gain block. (The

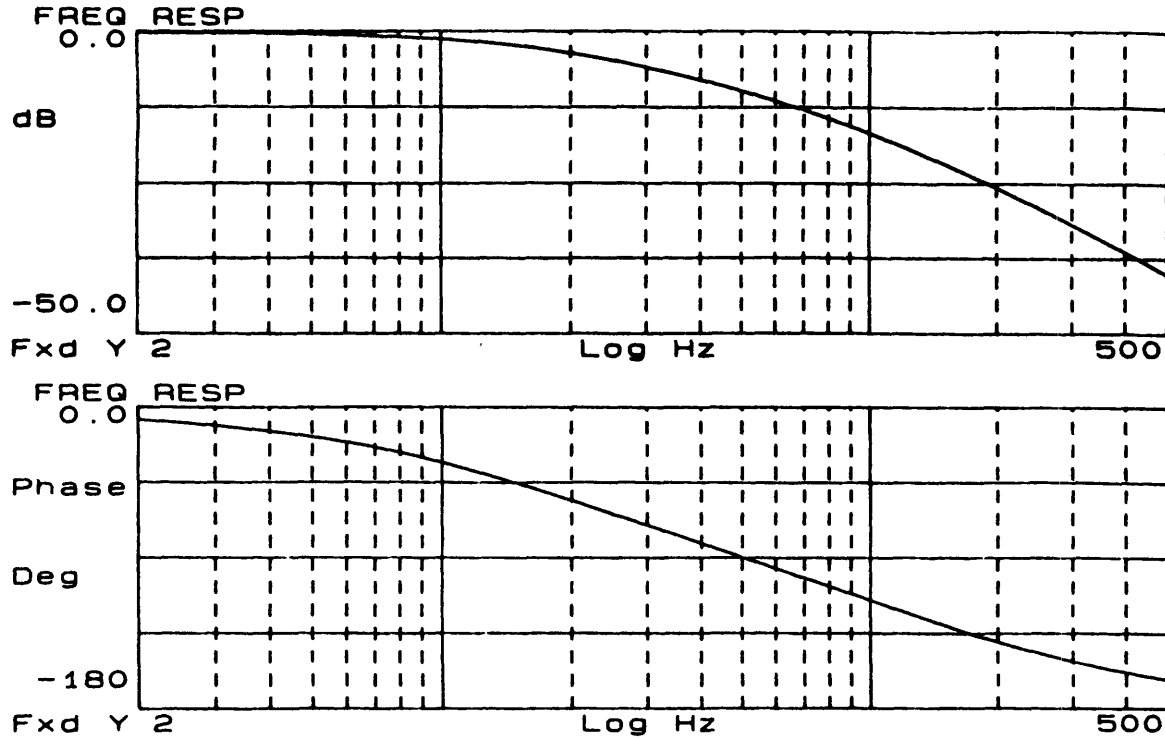


Figure 4.4: Closed Loop Op Amp Plant Frequency Response

program is listed in Appendix B.) In this way the effects of the sampling delay could be isolated from any effects caused by the digital processing associated with control algorithms. Figure 4.5 shows the frequency response obtained with the DSP32 unity gain block in the loop. The frequency response is almost identical to that shown in Figure 4.4 for the closed loop system without the DSP32 until higher frequencies where the phase roll-off associated with the sampling process starts to be significant.

The unity gain DSP32 system was driven by a 2.5 volt peak-to-peak 5 Hz square wave. The step response shown in Figure 4.6 shows an exponential response corresponding to the dominant closed loop pole at 18 Hz. The 10 to 90% risetime was approximately 20 ms for the step response measured both with and without the DSP32 in the loop.

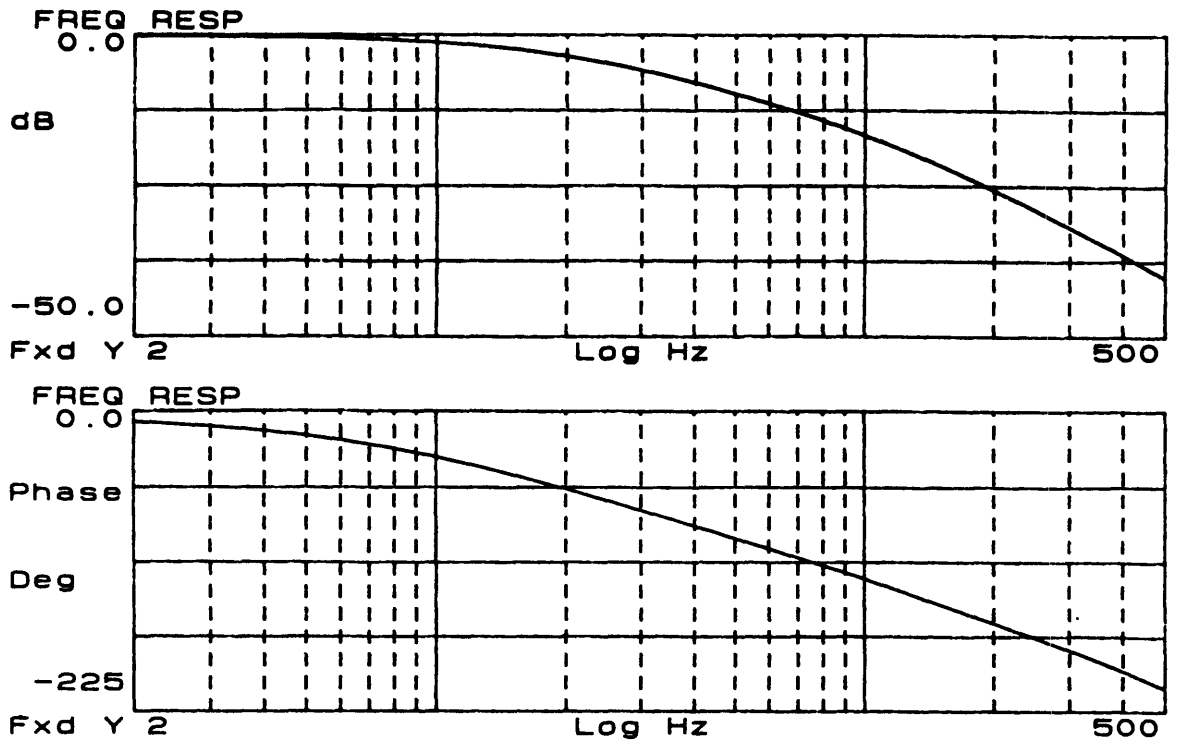


Figure 4.5: Closed Loop Frequency Response With Unity Gain DSP32

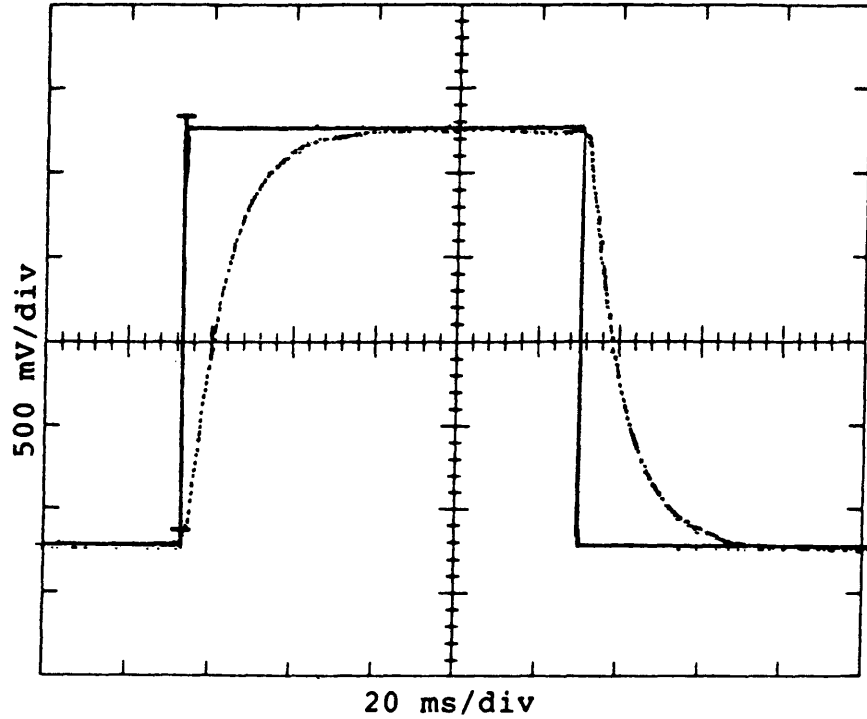


Figure 4.6: Closed Loop Step Response With Unity Gain DSP32

4.1.2 DSP32 Proportional-plus-Integral Block

In order to mimic the control architecture used in the analog mirror system, a proportional-plus-integral position compensation form was chosen with

$$G_{pi}(s) = \frac{B(s)}{E(s)} = \frac{K_p(1 + \tau_i s)}{s}$$

Placing the proportional-plus-integral compensator in the forward loop, the overall open loop transfer function is

$$G_{ol}(s) = \frac{Y(s)}{E(s)} = \frac{100K_p(1 + \tau_i)}{s^2(.001s + 1)} \quad (4.2)$$

The two remaining variables in $G_{ol}(s)$, corresponding to K_p and τ_i , were chosen to place the zero of $G_{ol}(s)$ at 16 Hz, and to make the open loop unity magnitude crossover point occur at approximately 50 Hz. Hence,

$$K_p = 315.296$$

$$\tau_i = .0099472$$

Using the bilinear transform of Equation 3.4, $G_{pi}(s)$ was transformed into $G_{Dpi}(z)$ shown in Equation 4.3.

$$G_{Dpi}(z) = \frac{3.156018 - 3.116606z^{-1}}{1 - z^{-1}} \quad (4.3)$$

This transfer function was programmed into the DSP32 using the method described in Section 2.3.2. Notice that in the notation of Equation 2.2, both $\frac{C}{D}$ and $\frac{F}{D}$ are zero for $G_{Dpi}(z)$.

A software limiter was added to the program to limit the output of the software integrator to ± 10 volts. The I/O board A/D and D/A converters were configured to accept and produce voltages in the ± 10 volt range, respectively. Sending the I/O board the digital data corresponding to more than 10 volts from the DSP32 would still result in the D/A producing 10 volts. More importantly, the integrator limiter was used to keep the software integrator from adding up to

Table 4.1: DSP32 vs. Op Amp Compensator Step Characteristics

Compensator	Peak Overshoot	Overshoot	Risetime	Settling Time
Op Amp	3.1 V	24%	3.4 ms	6.8 ms
DSP32	3.2 V	28%	3.6 ms	7.0 ms

excessively large values. Whenever the integrator added up to a number greater than the positive limiting value corresponding to +10 volts, it was reset to be the positive limiting value. The same strategy applied for negative sums with a negative limit.

In this way, when the integrator had reached the positive limit set on its register and a negative input was applied, the integrator would immediately start to subtract from its current value. Since this value was within the range of the I/O board, the effects would immediately be seen without a large time delay occurring while the integrator came out of saturation.

The step response for the closed loop system to a 2.5 volt step input is shown in Figure 4.7. The overshoot is approximately 28%, and the 10 to 90% risetime is 3.6 ms. For comparison, an op amp compensator with the same transfer function was built and tested. Table 4.1 compares the step response characteristics of the two compensators both driven by a 2.5 volt step input. The settling time shown in Table 4.1 refers to the time required for the output to enter and remain in a 10% envelope surrounding the steady state output value. The DSP32 performance as a closed loop compensator compared closely to the op amp compensator performance. The slightly lower stability of the DSP32 system can be attributed to the sampling delay present.

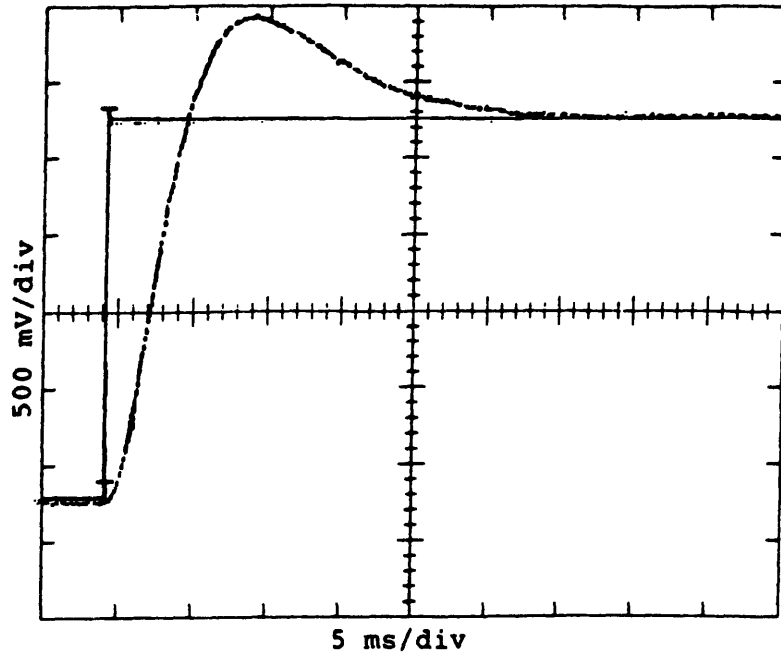


Figure 4.7: Step Response of DSP32 Compensated Plant

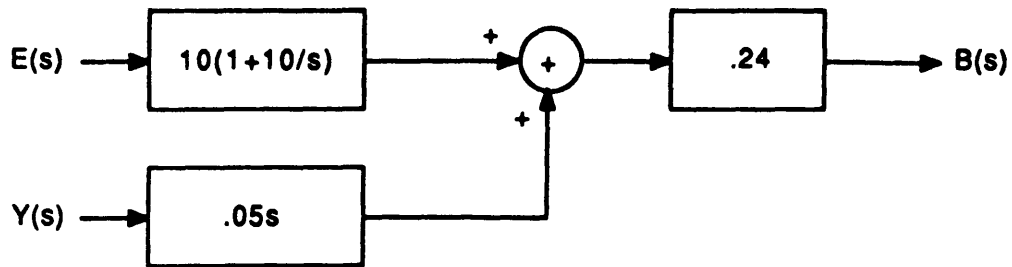


Figure 4.8: Mirror Analog Control Gains

4.2 The Overall Mirror Servomechanism

4.2.1 The Mirror Compensator Design

In order to program the DSP32 to control the actual mirror servomechanism, the analog control gains shown in Figure 4.8 were used. The notation used in Figure 4.8 refers to that used to define signals in the block diagram shown in Figure 1.1. $E(s)$ is the position error signal, $Y(s)$ is the output position signal, and $B(s)$ is the power amplifier drive signal.

Notice that Figure 4.8 illustrates two control inputs and one control output.

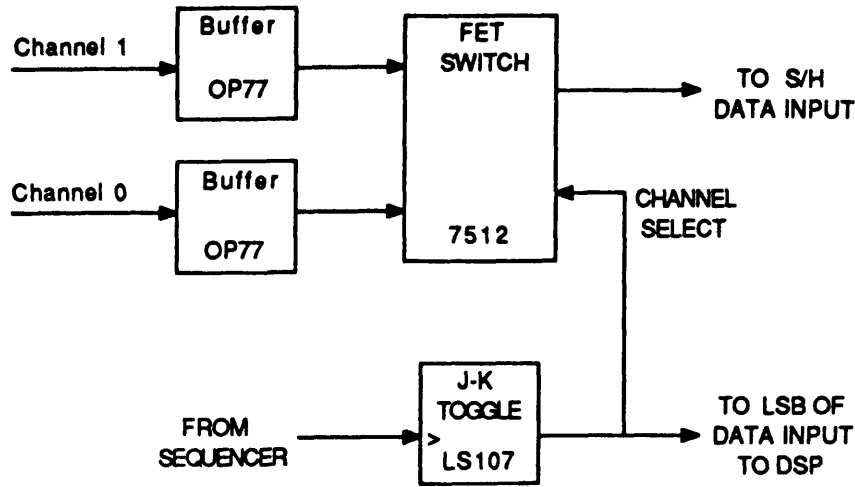


Figure 4.9: Input Multiplexor

In order to input these two different channels to the DSP32, a hardware multiplexor (see Figure 4.9) was designed and built for the front end of the I/O board. The channel information was encoded on the least significant bit sent to the DSP32 and decoded in the software. If the input channel consisted of $E(s)$, the program conditionally branched to the proportional-plus-integral part of the code. If the channel contained $Y(s)$, the program proceeded to the rate loop algorithm. In order to get an 8 kHz sampling rate on both channels, the overall sampling rate was increased to 16 kHz. The 8 kHz channel sample rate was achieved by alternately sampling the two channels.

The derived rate transfer function,

$$G_r(s) = \frac{.05s}{.003183s + 1}$$

was bilinearly transformed to yield,

$$G_{Dr}(z) = \frac{131.3025(1 - z^{-1})}{1 - .6717437z^{-1}}$$

The term in the denominator corresponds to a 500 Hz pole which was added to smooth the differentiation process by low pass filtering. The analog compensator had a similar filter at 500 Hz.

The proportional-plus-integral transfer function of the analog controller was similarly transformed to yield

$$G_{Dpi} = \frac{.006125(1 + z^{-1})}{1 - z^{-1}}$$

The DSP32 was programmed with these transfer functions and interfaced with the mirror through the hardware multiplexor. The analog .24 gain block was implemented with a potentiometer attenuator on the DSP32 output signal.

4.2.2 The Mirror Compensator Performance

Closed loop system testing was done with a 2.4 volt peak-to-peak square wave input. Figures 4.10 and 4.11 show the step responses of the analog compensator and the digital compensator, respectively. The DSP32 system demonstrates more overshoot (19 %) than the analog system (10 %). When the pole associated with the differentiator in the rate loop is moved to 159 Hz from 500 Hz, the overall system becomes more oscillatory as shown by the step response in Figure 4.12. The proportional-plus-integral position loop gain was reduced by 30 % while keeping the differentiator pole at 159 Hz to produce the step response shown in Figure 4.13. Lowering the position loop gain to 70 % that of the analog system, lowers the step response overshoot appreciably.

A comparison of the drive signal, $B(s)$, sent to the motor power amplifiers for both the analog system and for the digital system with the 159 Hz differentiator pole and can be made by looking at Figures 4.14 and 4.15. The step input shown on the top half of Figures 4.14 and 4.15 serves as a time reference for the drive signals shown. The digital signal with the 159 Hz filter shows much less noise than the analog signal with the 500 Hz filter.

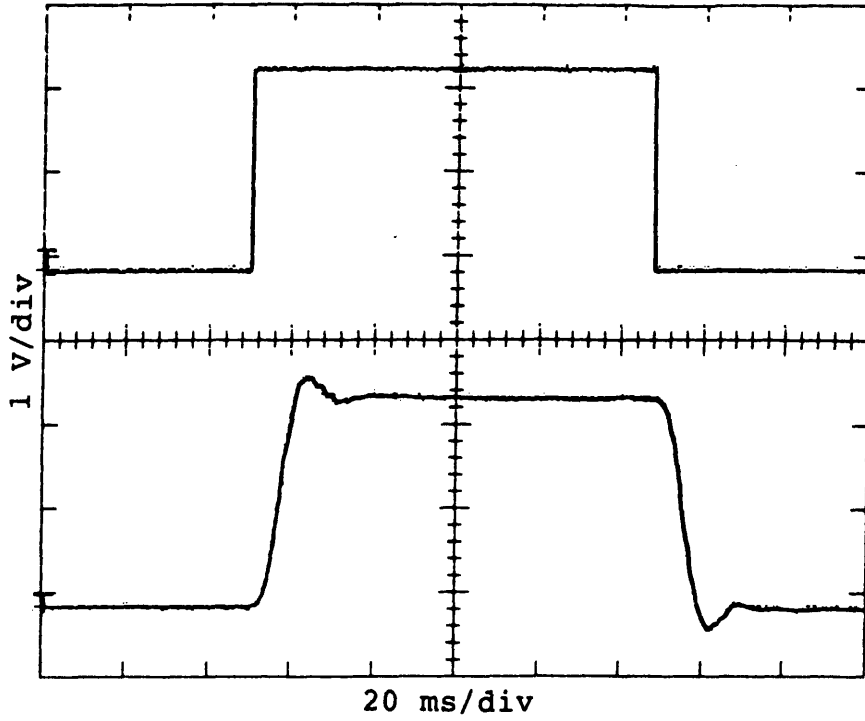


Figure 4.10: Analog Compensator Mirror Step Response

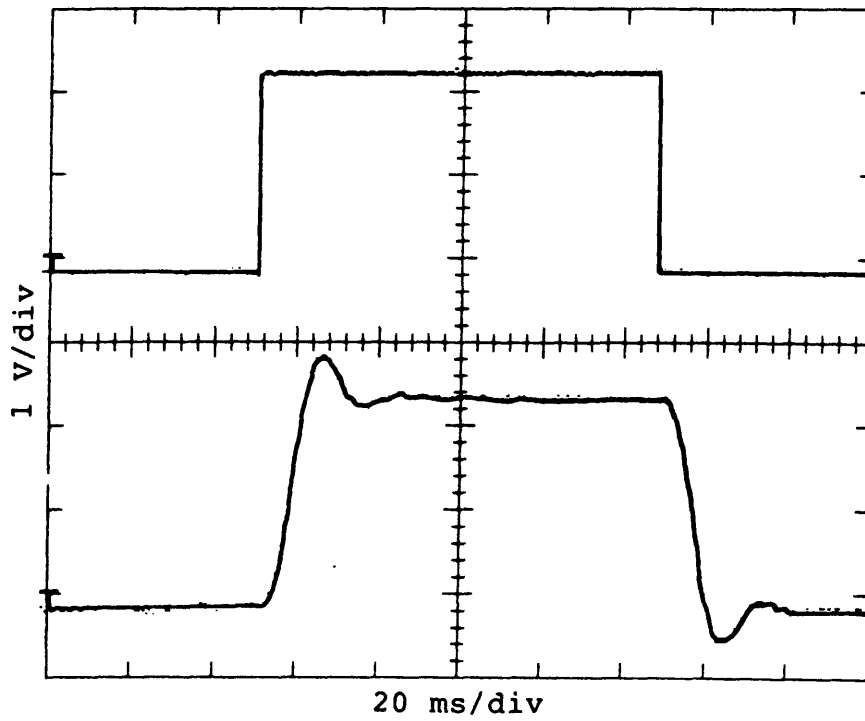


Figure 4.11: Digital Compensator Mirror Step Response

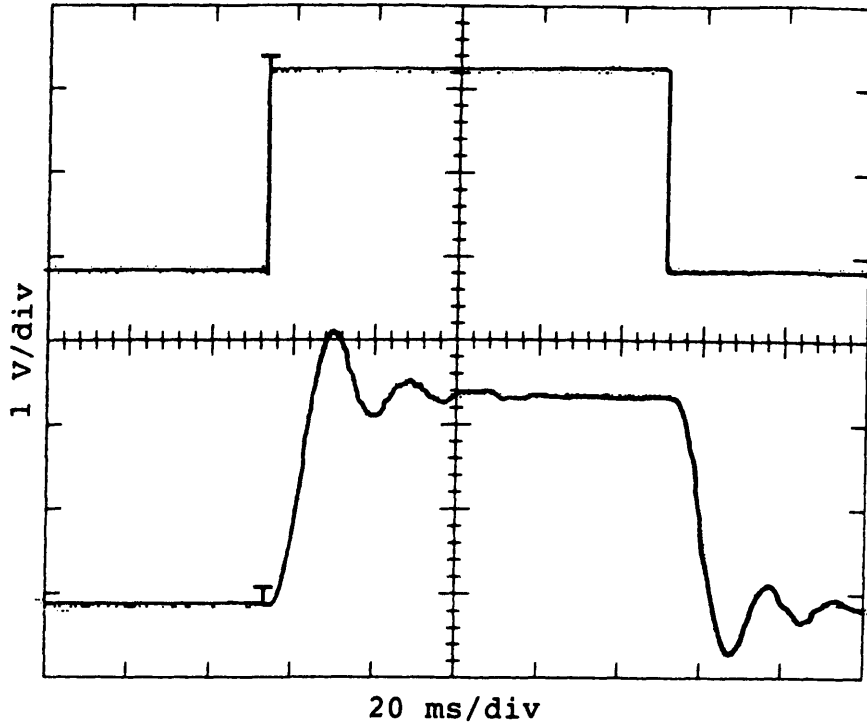


Figure 4.12: Mirror Step Response - 159 Hz rate pole

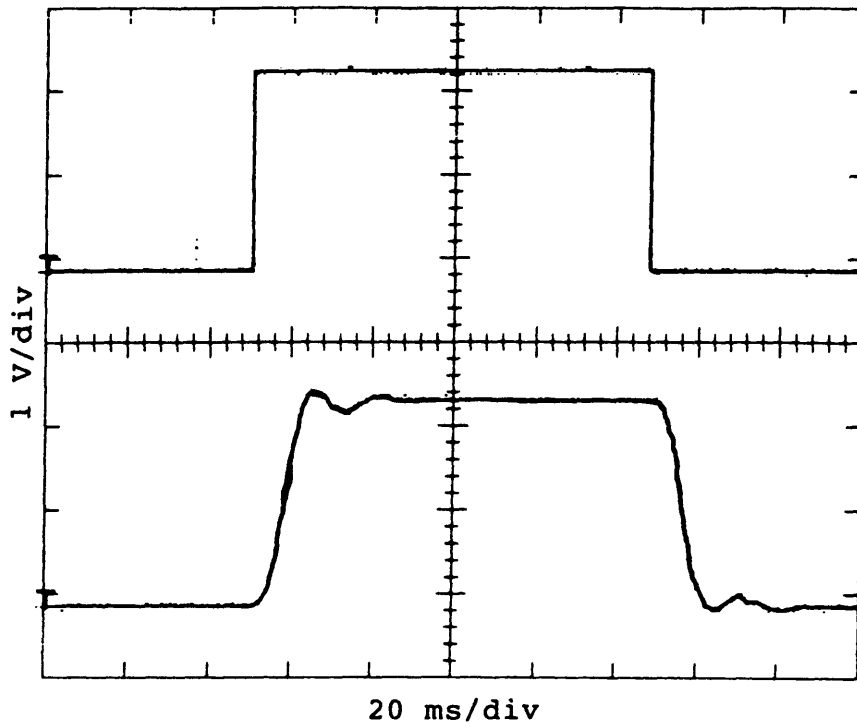


Figure 4.13: Mirror Step Response - 159 Hz rate pole, 70% position gain

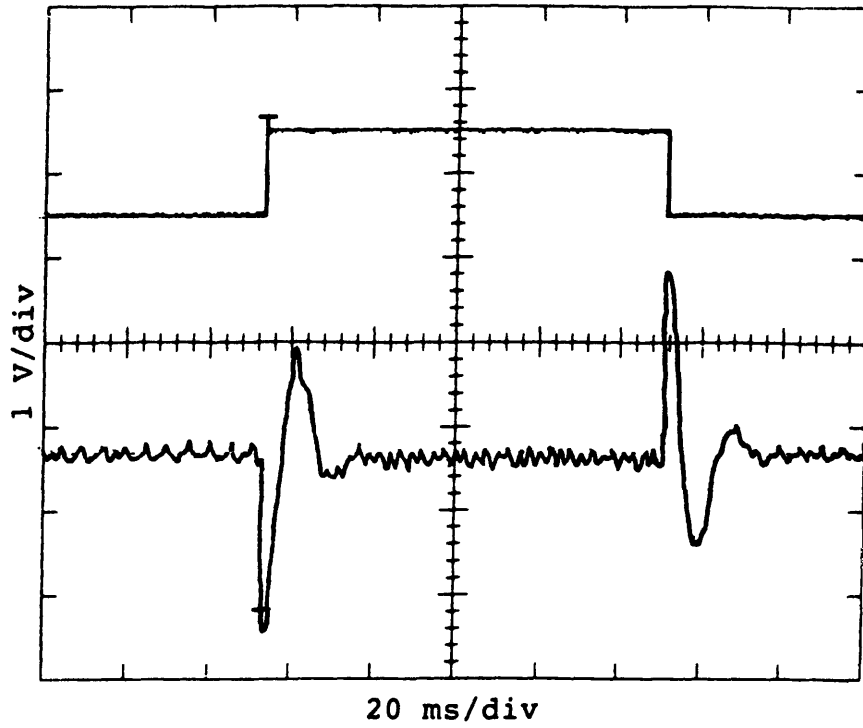


Figure 4.14: Analog Power Amplifier Drive Signal

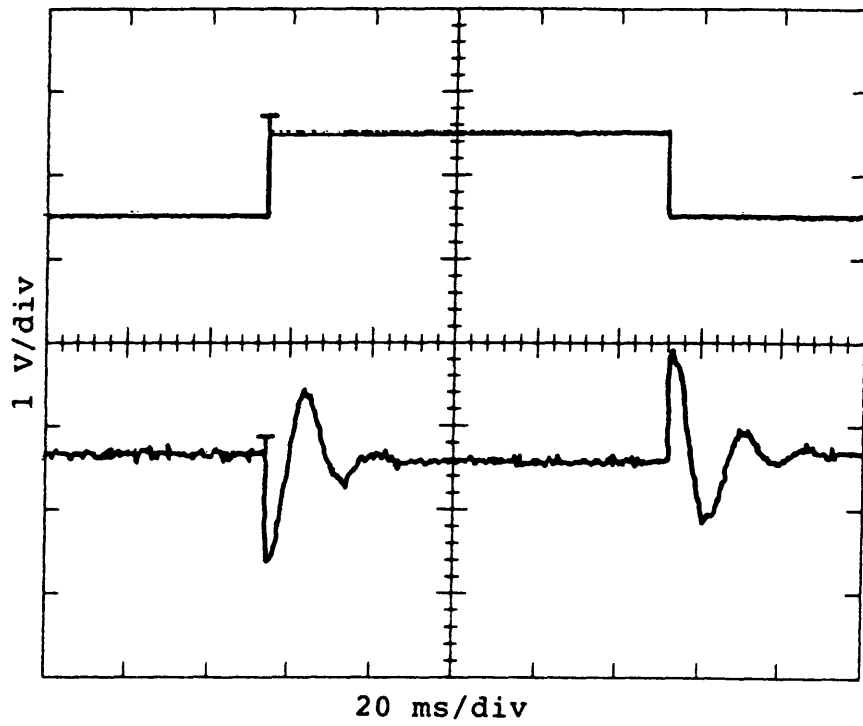


Figure 4.15: DSP32 Power Amplifier Drive Signal

Chapter 5

Conclusions

5.1 DSP32 Results

The DSP32 is capable of performing closed loop compensation. Once the hardware for interfacing to the serial port of the digital signal processor was built, the DSP32 proved to be a very flexible controller. The flexibility was derived from the programmability of the on chip RAM.

The DSP32 successfully compensated an operational amplifier model of the rate loop of the mirror as well as the closed loop mirror servomechanism. The step responses of the DSP32 compensated system and an analog compensated system were comparable. With the relatively high sampling rate of the digital system compared to the analog bandwidth desired, the pure time delay associated with the sampling process was a barely significant part of the response.

Four distinctly different control algorithms were implemented on the DSP32. The key to programming the algorithms is to break the code into modules. For example, placing a software limiter on the integrator in a proportional-plus-integral transfer function is relatively simple if the code is broken down into an integrator piece and a proportional piece. Large complex systems can be built by connecting many small functional groups together.

The beauty of using the DSP32 is its versatility. For example, once a certain two pole, two zero filter has been implemented, the filter coefficients can easily be modified to produce other filters of that form. Another place the programma-

bility of the DSP32 is advantageous is in the implementation of complex control algorithms. Long complex algorithms can be programmed and easily adjusted with software changes.

The high speed capabilities of the DSP32 really weren't exercised in this thesis. As noted in Section 3.1, the phase lag associated with the sampling process cannot be ignored, and yet many more programming steps could have been executed at the 8 kHz sampling rate used. Many different control loops running at different rates can be closed at the same time by appropriately multiplexing signals into the DSP32.

5.2 Future Work

In order to increase the sampling rate of the DSP32, it will be necessary to not only purchase faster A/D and D/A converters, but also to expand the length of the sequencer shift register by adding another shift register in series with the existing one. At the current sequencer clock rate of 128 kHz, each control pulse shift takes just over 7.8 μ s. This pulse shift time will decrease if the clocking rate is increased, making it impossible to wait the amount of time necessary for the current A/D converter to settle without more than the current 8 shifts available. The DSP32 should be capable of sampling up to about 50 kHz, whereas with the I/O board configured as it is now, the maximum sampling rate is 16 kHz for one channel, and 8 kHz for two channels.

To make the DSP32 useful for field applications, some sort of EPROM downloading scheme needs to be developed. The idea would be to program the EPROM with the control algorithms for the DSP32 and then to download the new algorithms at the site. A dedicated download board similar to the interface board would take care of the static RAM problem.

Finally, now that its capabilities as a closed loop compensator are documented a test needs to be performed to establish the system bandwidth obtain-

able. Theoretically, 500 to 1000 Hz systems should be feasible. The sampling speed of the DSP32 will have to be increased as suggested above and the size of the algorithm implemented will be limited.

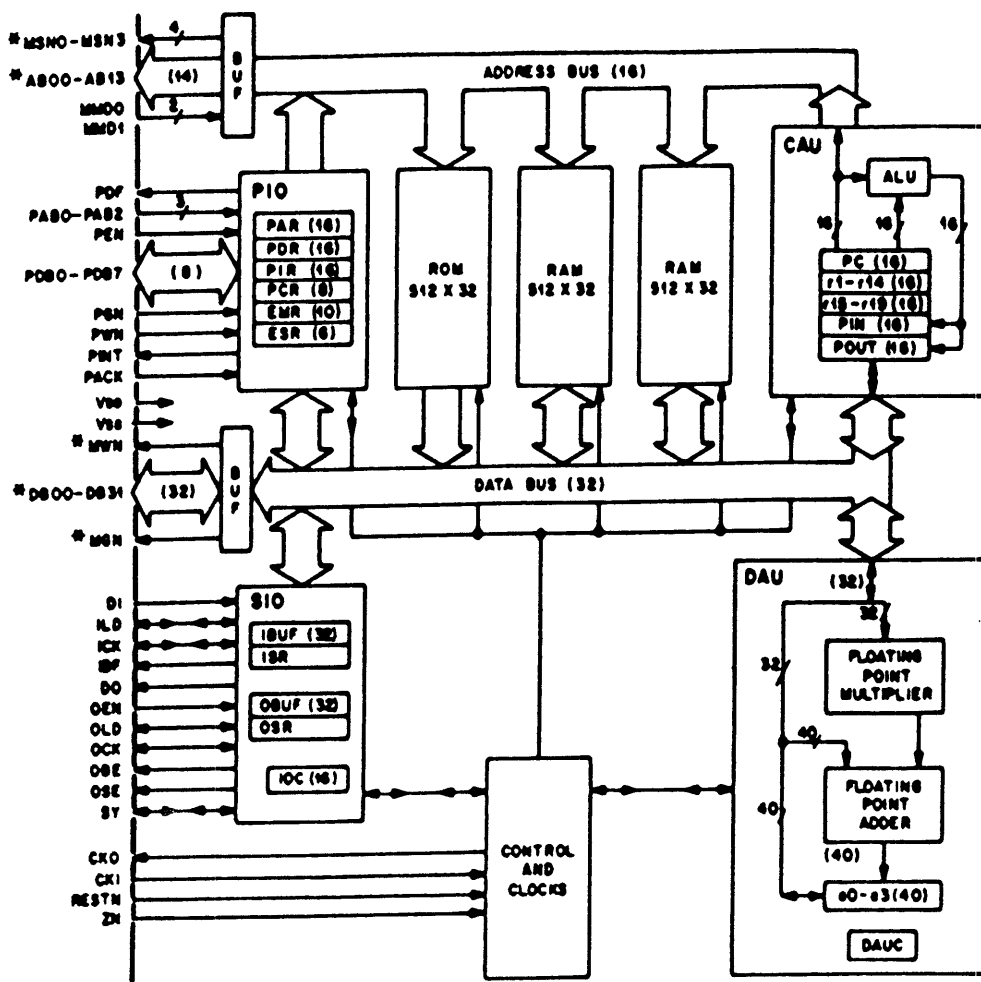
References

- [1] *WE^R DSP32 Digital Signal Processor Information Manual*. AT&T, USA, September 1986.
- [2] James K. Roberge. *Operational Amplifiers: Theory and Practice*. John Wiley & Sons, Inc., New York, New York, 1975.
- [3] *WE^R DSP32-SL Support Software Library User Manual*. AT&T, USA, July 1986. Preliminary Release.
- [4] *WE^R DSP32-DS Digital Signal Processor Development System User Manual*. AT&T, USA, December 1986.
- [5] Alan V. Oppenheim and Ronald W. Schaffer. *Digital Signal Processing*. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1975.
- [6] Katsuhiko Ogata. *Discrete-Time Control Systems*. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1987.

Appendix A

DSP32 Information

The following figures were taken from the WE DSP32 Digital Signal Processor Information Manual [1].



* AVAILABLE ON 100-PIN PGA PACKAGE ONLY

Figure A.1: DSP32 Block Diagram

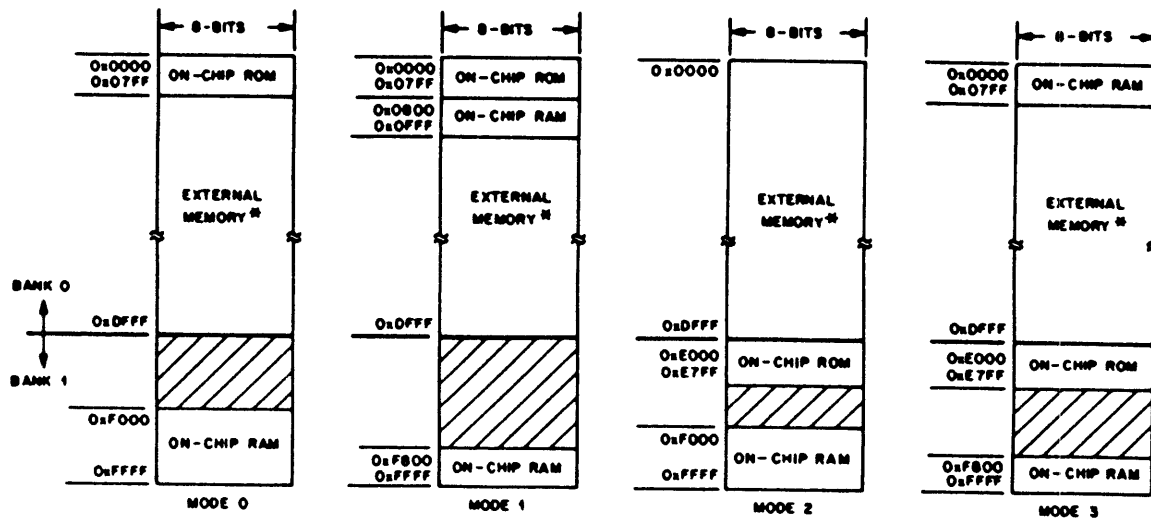


Figure A.2: Memory Configurations

DAU LATENCIES

- DAU Memory Writes (4)
- Accumulator as Multiplier Inputs (3)

CAU LATENCIES

- Conditional Branches on DAU Instructions (4)
- Delayed Branching

Figure A.3: DSP32 Latencies

Appendix B

Computer Programs

This appendix contains the computer programs used in this thesis. The following list gives the names of the programs and the sections that they correspond to in the thesis:

filter.s from Sections 2.3.2 and 3.1
notch8.s from Section 3.2
cl.s from Section 4.1
pilim.s from Section 4.1
mirpc.s from Section 4.2
mirtot.s from Section 4.2.2
larpi.s from Section 4.2.2

B.1 100 Hz Low Pass Filter

```
/*filter.s— 100 Hz Low Pass Filter */
.rsect "lo_ram"
.global sample,filter,coef,oldx,oldy,end
    ioc=0x986;
    dauc=0;
    r3=oldy;
    r4=oldx;
sampleif(ibe) goto sample;
    r1=coef;
filter: a0=float(ibuf);
        3*nop;
        a1=a0**r1++;
        a2=a1+*r1++**r4;
        nop;
        *r3=a3=a2+*r1**r3;
        nop;
```

```

        *r4=a0=a0;
        goto sample;
        obuf=a3=int(a3);
end:    4*nop;
.rsect ".hi_ram"
coef:   float .037767375,.037767375,.924465250
oldx:   float 0.0
oldy:   float 0.0

```

B.2 100 Hz Notch Filter

```

/*notch8.s— 100 Hz Tustin Notch Filter */
.rsect ".lo_ram"
.global sample,notch,coef,ym2,ym1,xm2,xm1,pause
        ioc=0x986;                /* 16 bit I/O Board */
        dauc=0;
        r2=ym2; r3=xm2;          /* assign memory pointers
        r4=xm1; r5=ym1;
sampleif(ibe) goto sample;      /* wait for sample */
        r1=coef;                 /* assign coef pointer */
notch:  a0=float(ibuf);          /* input X(n) */
        a1=*r1++**r2;            /* -F/D*Y(n-2) */
        a1=a1+*r1++**r3;        /* C/D*X(n-2)-F/D*Y(n-2) */
        a1=a1+(*r3=*r4)**r1++;  /* B/D*X(n-1)+... */
        a1=a1+(*r2=*r5)**r1++;  /* -E/D*Y(n-1)+... */
        a1=a1+(*r4=a0)**r1;     /* A/D*X(n)+... */
        *r5=a1=a1;             /* store Y(n-1) */
        goto sample;
out:    obuf=a1=int(a1);        /* new output */
.rsect ".hi_ram"                /* coef = filter coef.*/
coef:   float -.894957983,.946778712,-1.889169001
        float 1.889169001,.948205686,0.0
ym2:    float 0.0                /* allocate memory
xm2:    float 0.0                for storage */
xm1:    float 0.0
ym1:    float 0.0

```

B.3 Unity Gain Block

```

/*cl.s— unity transfer function

```

```

.rsect ".lo_ram"
.global sample,gain,pause;
    ioc=0x986;
    dauc=0;
sampleif (ibe) goto sample;
    nop;
gain:  a0=float(ibuf);
    goto sample;
pause: obuf=a0=int(a0);

```

B.4 Proportional-plus-Integral Compensator

```

/*pilim.s proportional + integral compensator */
.rsect ".lo_ram"
.global sample,algor,pause,coef,ym2,ym1,xm2,xm1,limit
    ioc=0x986;
    dauc=0
    r2=ym2;
    r3=xm2;
    r4=xm1;
    r5=ym1;
    r6=limit;
sampleif (ibe) goto sample;
    r1=coef;
algor: a0=float(ibuf);
    a1=*r1++**r2;
    a1=a1+*r1++**r3;
    a1=a1+(*r3=*r4)**r1++;
    a1=a1+(*r2=*r5)**r1++;
    a1=a1+(*r4=a0)**r1;
    *r5=a1=a1;
    4*nop
    a2=-*r6;                /*initialize a2 with neg. limit*/
    a0=-a1+*r6;            /*compare a1 with pos. limit*/
    a1=ifalt(*r6);        /*if a1+limit,replace with limit*/
    a0=a1+*r6;            /*compare a1 with neg. limit*/
    a1=ifalt(a2);         /*if a1-limit,replace with neg. limit*/
    4*nop
    goto sample;
pause: obuf=a1=int(a1);
coef:  float 0.0,0.0,-.03116606
    float 1.0,.03156018,0.0

```



```

ym2: float 0.0
xm2: float 0.0
xm1: float 0.0
ym1: float 0.0
limit: float 8000.0

```

B.5 Mirror Controller (Analog Gains)

```

/*mirpc.s analog gains*/
.rsect ".lo_ram"
.global sample,pcoef,rcoef,pym1,pxm1,limit,rxm1,rym1,chan,ploop,rloop;
    ioc=0x986;
    dauc=0
    r4=pxm1;r5=pym1;r6=limit;
    r7=rxm1;r8=rym1;
sampleif (ibe) goto sample;
    r1=pcoef;
chan: r9=ibuf;4*nop;
    r10=r9/2;
    if (cs) goto rloop; nop;
ploop: a0=float(ibuf);
    a1=*r1++**r4;
    a1=a1+*r5**r1++;
    a1=a1+(*r4=a0)**r1++;
    4*nop
    a2=-*r6;                /*initialize a2 with neg. limit*/
    a0=-a1+*r6;            /*compare a1 with pos. limit*/
    a1=ifalt(*r6);        /*if a1+limit,replace with limit*/
    a0=a1+*r6;            /*compare a1 with neg. limit*/
    a1=ifalt(a2);        /*if a1-limit,replace with neg. limit*/
    4*nop;
    *r5=a1=a1;            /*store ym1*/
    4*nop;
    a1=a1+*r4**r1;        /*proportional branch*/
    4*nop;
    goto sample;
    nop;
rloop: r2=rcoef;
    a2=float(ibuf);
    a3=*r7**r2++;
    a3=a3+*r8**r2++;
    a3=a3+(*r7=a2)**r2;

```

```

    *r8=a3=a3;
    4*nop;
    a0=a1+a3;
    4*nop;
    goto sample;
    obuf=a0=int(a0);
pcoef: float .00625,1.0,.00625,10.0
rcoef: float -131.3025,.6717437,131.3025
pxm1: float 0.0
pym1: float 0.0
rxm1: float 0.0
rym1: float 0.0
limit: float 32767.0

```

B.6 Mirror Controller (159 Hz Rate Pole)

```

/*mirtot.s analog gains and 159 Hz rate pole*/
.rsect ".lo_ram"
.global sample,pcoef,rcoef,pym1,pxm1,limit,rxm1,rym1,chan,ploop,rloop;
    ioc=0x986;
    dauc=0
    r4=pxm1;r5=pym1;r6=limit;
    r7=rxm1;r8=rym1;
sampleif (ibe) goto sample;
    r1=pcoef;
chan:  r9=ibuf;4*nop;
    r10=r9/2;
    if (cs) goto rloop; nop;
ploop: a0=float(ibuf);
    a1=*r1++**r4;
    a1=a1+*r5**r1++;
    a1=a1+(*r4=a0)**r1++;
    4*nop
    a2=-*r6;                /*initialize a2 with neg. limit*/
    a0=-a1+*r6;            /*compare a1 with pos. limit*/
    a1=ifalt(*r6);         /*if a1+limit,replace with limit*/
    a0=a1+*r6;             /*compare a1 with neg. limit*/
    a1=ifalt(a2);          /*if a1-limit,replace with neg. limit*/
    4*nop;
    *r5=a1=a1;             /*store ym1*/
    4*nop;
    a1=a1+*r4**r1;        /*proportional branch*/

```

```

    4*nop;
    goto sample;
    nop;
rloop: r2=rcoef;
    a2=float(ibuf);
    a3=*r7**r2++;
    a3=a3+*r8**r2++;
    a3=a3+(*r7=a2)**r2;
    *r8=a3=a3;
    4*nop;
    a0=a1+a3;
    4*nop;
    goto sample;
    obuf=a0=int(a0);
pcoef: float .00625,1.0,.00625,10.0
rcoef: float -47.05882,.882353,47.05882
pxm1: float 0.0
pym1: float 0.0
rxm1: float 0.0
rym1: float 0.0
limit: float 32767.0

```

B.7 Mirror Controller (159 Hz Rate Pole and 70 % Position Gain)

```

/*larpis analog gains,159 Hz rate pole, .7*gain */
.rsect ".lo_ram"
.global sample,pcoef,rcoef,pym1,pxm1,limit,rxm1,rym1,chan,ploop,rloop;
    ioc=0x986;
    dauc=0
    r4=pxm1;r5=pym1;r6=limit;
    r7=rxm1;r8=rym1;
sampleif (ibe) goto sample;
    r1=pcoef;
chan: r9=ibuf;4*nop;
    r10=r9/2;
    if (cs) goto rloop; nop;
ploop: a0=float(ibuf);
    a1=*r1++**r4;
    a1=a1+*r5**r1++;
    a1=a1+(*r4=a0)**r1++;

```

```

4*nop
a2=-*r6;          /*initialize a2 with neg. limit*/
a0=-a1+*r6;      /*compare a1 with pos. limit*/
a1=ifalt(*r6);   /*if a1+limit,replace with limit*/
a0=a1+*r6;       /*compare a1 with neg. limit*/
a1=ifalt(a2);    /*if a1-limit,replace with neg. limit*/
4*nop;
*r5=a1=a1;       /*store ym1*/
4*nop;
a1=a1+*r4**r1++; /*proportional branch*/
4*nop;
a1=a1**r1;
4*nop;
goto sample;
nop;
rloop: r2=rcoef;
a2=float(ibuf);
a3=*r7**r2++;
a3=a3+*r8**r2++;
a3=a3+(*r7=a2)**r2++;
4*nop;
a3=a3**r2;
4*nop;
*r8=a3=a3;
4*nop;
a0=a1+a3;
4*nop;
goto sample;
obuf=a0=int(a0);
pcoef: float .00625,1.0,.00625,10.0,.7
rcoef: float -47.05882,.882353,47.05882,1.0
pxm1: float 0.0
pym1: float 0.0
rxm1: float 0.0
rym1: float 0.0
limit: float 32767.0

```

Appendix C

Detailed I/O Board Circuitry

The following page contains the detailed circuit schematic for the entire I/O board. It includes all three subsystems discussed in Section 2.2.2.

