

To the Graduate Council:

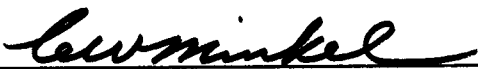
I am submitting herewith a thesis written by Randall Thomas Schonhoff entitled "Development of a Digital Signal Processing Laboratory." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.


Robert E. Bodenheimer, Major Professor

We have read this thesis and
recommend its acceptance:




Accepted for the Council:


Vice Provost
and Dean of The Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at the University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the head of the Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature



Date

11/22/85

DEVELOPMENT OF A DIGITAL SIGNAL
PROCESSING LABORATORY

A Thesis
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

Randall Thomas Schonhoff

December 1985

DEDICATION

This thesis is dedicated to my parents.

ACKNOWLEDGMENTS

There are several people that assisted me in the production of this thesis. In particular, I am grateful to Bobby Bodenheimer for his help in documenting the diagnostics of the TI computer, and Fred Dyer for his help in producing the oscilloscope pictures.

Most of all, I am indebted to my major professor Robert E. Bodenheimer, and the committee members Donald Boldin and Robert W. Rochelle for their criticism and encouragement during the entire production process.

ABSTRACT

Digital Signal Processing (DSP) is fast becoming the desired method for measurement and control operations due to the increased use of computers in these applications and the advantages of performing operations on digital signals as opposed to analog signals. As the use of Analog-to-Digital converters (ADC) and Digital-to-Analog converters (DAC) increase, the graduating student is faced with a need to understand these systems and have some experience in the operation of these devices in DSP applications.

The specific objectives of this thesis were: 1) to explore the capabilities of the DAC and ADC systems on the TI 990, 2) to explore the diagnostic procedures necessary for proper maintenance and calibration of the ADC and DAC subsystem, and 3) to provide sample DSP programs that can be written by the student in the laboratory in conjunction with coursework.

To achieve the first objective, an Input/Output (I/O) panel was constructed to allow access to the many channels on the ADC and DAC modules. This panel was then used in the diagnostic procedure and the sample DSP programs required of the second and third objectives.

FORTRAN was chosen as the language to use in all software development because it was available on the TI computer, it has several support packages available that

accessed the ADC and DAC system, and it is a common language in the engineering curriculum. The example programs that were developed are discussed in Chapter II, and the source code is available in the appendix. These programs consisted of 1) a digital oscilloscope, 2) a Fourier Series analysis program, 3) a program to characterize lesser resolution ADC's, and 4) a program to perform Fast Fourier Transforms and filtering. An operation amplifier summing junction was constructed for the third laboratory and an eight bit ADC for the fourth laboratory. The schematic for these devices is available in the appendix.

Results indicate that the TI system can be used very effectively in the laboratory analysis of some DSP applications. The only limitations that were evident were those due to the nature of the TI 990 being a multi-user system, some time shifting of the sampled signal was observable due to the system scheduling and running other users during the sampling process. Also, to use the system for analysis of high frequency signals, an assembly language routine would need to be developed.

TABLE OF CONTENTS

CHAPTER	PAGE
INTRODUCTION	1
I. TEXAS INSTRUMENTS 990 DATA SYSTEM	4
Capabilities	5
Analog-to-Digital Converter Module	5
Digital-to-Analog Converter Module	7
FORTRAN-78 ISA Extension Package	9
Construction of I/O Panel	13
II. STUDENT LABORATORIES	18
Digital Oscilloscope	19
Procedure	20
Results	21
Fourier Series Analysis	23
Procedure	26
Results	27
ADC System Characterization	31
Procedure	35
Results	38
Fast Fourier Transform and Filtering	43
Procedure	45
Results	48
III. MAINTENANCE OF THE TI 990 SYSTEM	58
Diagnostic Operational Control System (DOCS)	59
Analog-to-Digital Check Routine	62
Digital-to-Analog Check Routine	66
Results	67
IV. CONCLUSION	69
Results	69
Future Developments	71
LIST OF REFERENCES	72
APPENDIXES	75
APPENDIX A. TUTORIAL ON TI 990 FORTRAN	76
APPENDIX B. DIGITAL OSCILLOSCOPE PROGRAM	85
APPENDIX C. FOURIER SERIES PROGRAM	91
APPENDIX D. ADC CHARACTERISTIC PROGRAM	96
APPENDIX E. FAST FOURIER TRANSFORM PROGRAM	104
APPENDIX F. COMMON SUBROUTINES	111
APPENDIX G. SCHEMATIC OF EIGHT-BIT ADC	123
VITA	125

LIST OF FIGURES

FIGURE	PAGE
1.1. TI 990 ADC Options and Specifications	6
1.2. TI 990 DAC Options and Specifications	8
1.3. Input/Output Panel for TI System	15
2.1. Digital Oscilloscope Lab Results	22
2.2. Waveforms for the Fourier Series Laboratory	25
2.3. Fourier Series Sawtooth Wave - Fundamental and Six Harmonics	28
2.4. Fourier Series Square Wave - Fundamental and Eleven Harmonics	29
2.5. Fourier Series Arbitrary Wave - Fundamental and Twenty-one Harmonics	30
2.6. Analog-to-Digital Conversion Errors	33
2.7. ADC0804LCN Specifications	34
2.8. Block Diagram of the Eight-bit ADC Under Test	36
2.9. 0804 ADC Timing Diagram	37
2.10. Top View of Eight-bit ADC Project	37
2.11. Laboratory Results Showing Characteristics of 0804 ADC	40
2.12. Common Low-pass Filter Characteristics	46
2.13. Operational Amplifier Unity Gain Summing Junction	48
2.14. 1.2 kHz Signal Summed with 320 Hz Signal	49
2.15. Fourier Transform of 1.2 kHz Signal Summed with 320 Hz Signal	51
2.16. Inverse Fourier Transform of High-pass Filtered Signal - Less the DC Offset	52
2.17. Fourier Transform of High-pass Filtered Signal - Less the DC Offset	53

FIGURE	PAGE
2.18. Low-pass Filtered Signal - Results are 320 Hz Signal Less any DC Offset	55
2.19. High-pass Filtered Signal - Results are 1.2 kHz Signal Less any DC Offset	56
3.1. Front Panel of the TI 990 System	60
3.2. Loading the DOCS System	62
3.3. DACHK Sample Listing	63
3.4. ADCHK Sample Listing	63
A.1. TI 990 Directory Structure	78
A.2. Assign SYNONYM Command	79
A.3. FORTRAN Compile Screen	80
A.4. Example of LINK Control File	81
A.5. Example of LINK Screen	82
A.6. Assign LUNO Command	83
A.7. Example of XFTE Command	84
G.1. Schematic of Eight-bit 0804 ADC System	124

INTRODUCTION

Traditionally, the measurement of analog signals and the resultant analog control system outputs of a signal processing application have been performed by analog circuits. Radar, television, and telephone are a few examples of devices that originally utilized analog components in their design, but now are more frequently being constructed with digital signal processing (DSP) devices. These analog circuits were prone to drifts in performance due to temperature and aging, factors that do not affect digital products.

This shift to digital processing in areas that were traditionally analog is caused by 1) many functions are difficult or impossible to perform with analog components, 2) digital systems are more reliable, and 3) digital devices can be easily reprogrammed for other functions.

Several of the applications that are utilizing DSP are listed in Table 1. As technology advances the processing power and speed of the microprocessor, many new DSP applications will be explored that previously required a large mainframe. Also, semicustom chips are being designed for specific DSP applications. [1]

The objective of this thesis was to explore the possibilities of using the TI 990 system in a laboratory support environment to a Digital Signal Processing course, or

TABLE 1. DIGITAL SIGNAL PROCESSING APPLICATIONS

GENERAL PURPOSE DSP

- Digital Filtering
- Correlation
- Fast Fourier Transforms
- Adaptive Filtering
- Waveform Generation
- Seismic Processing
- Electronic Countermeasures

TELECOMMUNICATIONS

- Spread Spectrum Comm.
- Data-Encryption
- Signal Modulation
- Data Compression

HIGH SPEED CONTROL

- Robotics
- Missile Guidance

IMAGE PROCESSING

- Pattern Recognition
- Image Enhancement
- Radar and Sonar

INSTRUMENTATION

- Spectrum Analysis
- Phase-locked Loops
- Transients Analysis

SPEECH PROCESSING

- Voice Store and Forward
- Speech Synthesis
- Speech Recognition

SOURCE: Cole, B.C., "Signal Processing: A Big Switch to Digital", Electronics, McGraw-Hill, p. 43, August 26, 1985.

as a adjunct to the current Analog/Digital Converter course. The equipment and software available on the TI system is discussed in Chapter I. This includes a 12 bit ADC, a 12 bit DAC, and several support routines in FORTRAN to access these devices.

Several example laboratory exercises are developed in Chapter II. These examples are programmed in FORTRAN, and the source code is available in the appendix. Chapter III examines the maintenance and calibration of the Analog/Digital system. Software available from TI was used to perform the testing and calibration of the equipment. The conclusions and recommendations are contained in Chapter IV along with some ideas for future development. A Tutorial on developing applications in FORTRAN is available in Appendix A.

CHAPTER I

TEXAS INSTRUMENTS 990 DATA SYSTEM

Texas Instruments' 990/12 Data System is a 16 bit minicomputer constructed of medium scale integrated components. The system that was used in this thesis supported two 80 Megabyte (Mb) disk drives, 11 TI 911 terminals, one 300 LPM (line per minute) printer, and one TI 810RO low speed printer. Additionally, the system had 256 Kilobytes (Kb) of memory and supported two PROM programmers and two floppy disk drives. Currently the system was being used as a laboratory support to a class in microprocessor programming and interfacing.

The Texas Instruments' 990 Data System supports the addition of an Analog-to-Digital (ADC) and Digital-to-Analog (DAC) modules. These allow the 990 system to be used for sampling analog signals with the ADC, and in turn, to control external devices with the DAC. The modules are boards that plug into the computer's backplane and are referenced using standard 990 CRU instructions.

Both modules are 12 bit devices, which means they are capable of resolving the selected input or output to a resolution of 2^{12} or 1 out of 4096 parts. For a range of -10.24 to +10.24 volts, this implies that each step or resolution is 5.0 mV.

Capabilities

Analog-to-Digital Converter Module

Four different configurations are available for the Analog-to-Digital converter module (TI part number 949024-000X), where the "X" in the part number indicates the number of 16 single-ended channels on the board. The module is available with 16, 32, 48, and 64 single-ended channels, or with 8, 16, 24, and 32 differential-ended channels. Selection of single-ended or differential-ended is accomplished by a jumper on the board. The module that was used in this thesis was configured with 64 channels set for single-ended conversions.

Several other features are selectable by DIP switches on the board. Four different ranges, and three different coding selections which were available are shown in Figure 1.1. A range of -10.24 to +10.24 volts and two's complement coding were selected for use throughout this thesis. Current levels can also be measured by installing pad resistors on the board. All changes to the DIP switches require the removal of the board from the backplane.

Please note that all channels on the ADC board are configured identically by the settings chosen for the DIP switches. One cannot have channels with different ranges or coding methods. This restriction exists because the ADC is physically realized as one converter with a 16 position

<u>INPUT</u> <u>RANGE (volts)</u>	<u>SWITCH</u>	
	<u>S2-1</u>	<u>S2-2</u>
0.0 to 10.24	A	B
0.0 to 5.12	A	A
-10.24 to 10.24	B	B
-5.12 to 5.12	B	A

<u>CHANNEL</u> <u>CONFIGURATION</u>	<u>SWITCH</u>				
	<u>S1-1</u>	<u>S1-3</u>	<u>S1-4</u>	<u>S2-3</u>	<u>S2-4</u>
Single-Ended	A	A	A	A	A
Differential	B	B	B	B	B

<u>DIGITAL OUTPUT</u> <u>CODE</u>	<u>INPUT</u> <u>RANGE</u>	<u>S1-2</u>
Straight Binary	Unipolar	A
Offset Binary	Bipolar	A
Two's Complement	Bipolar	B

<u>Accuracy</u>	
Resolution	12 bits
Relative Accuracy	$\pm 0.025\%$ FSR @ 27.5 KHz throughput rate
Monotonicity	Guaranteed 0°C to 70°C

<u>Signal Dynamics</u>	
Maximum Throughput	27,500 channels/sec
Sample/Hold aperture uncertainty	5 nsec

FIGURE 1.1. TI 990 ADC Options and Specifications.

SOURCE: Texas Instruments, Module 990 Computer Analog/Digital Converter Module, Digital/Analog Converter Module Installation and Operation Manual, part number 944774-9701, p. 14-15, July 1977.

multiplexor connected to the input stage. Then, to expand to 64 single-ended channels, another input multiplexor is added in multiples of 16.

An additional feature available is a conversion trigger strobe for the ADC. This strobe is selectable under software control to begin a conversion on the falling edge of the signal attached to this input. Also, the ADC can be set under software control to generate an interrupt on the completion of the current conversion. Because the conversion rate of the ADC is so rapid, the interrupt feature is not necessary for normal use, but is useful when the asynchronous trigger option is employed.

Digital-to-Analog Converter Module

Four different configurations were available for the Digital-to-Analog converter module (TI part number 949029-00X). These configurations use the same identifying character in the part number as the ADC module. The four modules available have 1, 2, 3, or 4 output channels. DIP switches determine the configuration of each channel independently of the other channels. Switches select the analog output range and the digital code that the DAC uses to interpret the voltage that the user desires to generate. These ranges and codes are specified in Figure 1.2. All changes to the DIP switches require the removal of the board from the backplane.

ANALOG OUTPUT		SWITCH			
<u>RANGE (volts)</u>	<u>CURRENT (mA)</u>	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>
-10.24 to 10.24	-20.48 to 20.48	B	B	A	A
0.0 to 10.24	0.0 to 20.48	B	A	B	B
-5.12 to 5.12	-10.24 to 10.24	B	A	A	A
0.0 to 5.12	0.0 to 10.24	A	A	B	A

NOTE: One switch pack for each channel (S2-S5)

		SWITCH			
<u>DIGITAL CODE</u>	<u>OUTPUT RANGE</u>	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>
Straight Binary	Unipolar	B	B	B	B
Offset Binary	Bipolar	B	B	B	B
Two's Complement	Bipolar	A	A	A	A
		0	1	2	3
		Channel #			

		OUTPUT CHARACTERISTICS	
<u>ACCURACY</u>		<u>VOLTAGE</u>	<u>CURRENT</u>
Resolution		12 Bits	12 Bits
Relative accuracy		.012% FSR	.035% FSR
<u>ANALOG OUTPUT</u>			
Load current	5 mA	----	
Load resistance	----		500 Ω
<u>SPEED</u>			
Settling time	10 μ sec max		110 μ sec max
Slew rate	10 V/ μ sec		----

FIGURE 1.2. TI 990 DAC Options and Specifications

SOURCE: Texas Instruments, Module 990 Computer
Analog/Digital Converter Module, Digital/Analog
Converter Module Installation and Operation Manual,
part number 944774-9701, p. 16-17, July 1977.

Each channel is implemented on the board as a separate Digital-to-Analog converter. Because all channels are desired to sustain the voltage or current selected for them, each DAC is implemented as an independent converter.

In addition to the voltage output, each channel also simultaneously generates a current output that is dependent on the configuration chosen for the voltage output. Due to the fast conversion rate of the DAC, no interrupt or strobe trigger options are available. The complete specifications for the DAC module are listed in Figure 1.2.

FORTRAN-78 Instrument Society of America Extension Package

To access the ADC and DAC modules using the FORTRAN language, Texas Instruments supplied with the 990 system a set of subroutines called the ISA Extension Package. This package allows for the sequential or random reading of each ADC input channel, and the random writing to each DAC output channel.

Each of the three subroutines returns in the integer value IERR a 1 if the call executed properly. Any other number would indicate that some error occurred in the execution of the subroutine. To read the ADC input channels the two subroutines provided are the AISQW (sequential channel read), and the AIRDW (random channel read). DAC channels are accessed by the AOW (random channel write) subroutine.

To read random channels of the ADC, use the subroutine call :

```
CALL AIRDW (NUMBER, ADDRESS, SAMPLE, IERR)
```

NUMBER is an INTEGER*2 variable (referred to henceforth as integer) that specifies the number of conversions to perform. ADDRESS is a two-by-NUMBER size matrix that specifies which ADC module and which channel on the ADC module from which to read. The data is arranged as :

```
ADDRESS(1,I) = >0400 + >0040
```

```
ADDRESS(2,I) = Channel Number
```

where I is an index pointer through the ADDRESS matrix and the SAMPLE matrix. Notice that the address of the module is specified as >0400 which identifies which expansion box, and >0040 which is the CRU address of the module. Both of these addresses are set by DIP switches and are recorded in the system documentation.

SAMPLE is a 2-by-NUMBER integer array that is to contain the results of the conversions. These integer values must then be converted to REAL*2 or REAL*4 (henceforth referred to as REAL) numbers. This conversion depends on the configuration of the ADC module (i.e. range and binary code). A subroutine named ADSCL (ADC scale) was written that performs these conversions and is used throughout the sample laboratories in Chapter II. Refer to Appendix F for a listing of this subroutine.

The format of the subroutine call that reads sequential channels of the ADC is :

```
CALL AISQW (NUMBER,CRUADDR,SAMPLE,IERR)
```

Starting with channel zero, this subroutine call will read sequentially the number of channels specified in the variable NUMBER, and store the results in the matrix SAMPLE. Both of these variables are the same as for the AIRDW subroutine call previously discussed. Because this call only reads sequentially starting with channel zero, only the address of the module to be read from needs to be specified. Therefore, the variable CRUADDR is an INTEGER*2 number that specifies the address that applies to the entire sequential read operation.

The format of the subroutine call that writes random channels to the DAC is :

```
CALL AOW (NUMBER,ADDRESS,SAMPLE,IERR)
```

Integer variables NUMBER and ADDRESS are integer variable that are used in the same context as the subroutine call AIRDW explained above. The format of the array ADDRESS is:

```
ADDRESS(1,I) = >0400 + >0060
```

```
ADDRESS(2,I) = Channel Number
```

where as before, the integer I is an index pointer. Notice that the expansion box value >0400 is the same as that specified for the ADC, but that the CRU address is different. The integer values in the array SAMPLE must be specified in

accordance with the characteristics of the DAC module (i.e. range and binary code). A subroutine named DASCL (DAC scale) was written that performs this scaling from a real to integer value dependent on the configuration of the DAC module.

One limitation for the applications that the ADC and DAC modules are suitable for is the speed that the modules can convert analog to digital or digital to analog. Because of the overhead of programming in a higher level language such as FORTRAN, the fastest throughput that was measured for the ADC module is approximately 16,000 conversions/sec, which is considerably slower than the 27,500 channels/sec specified earlier in the chapter. Later in Chapter III, which covers the diagnostic routines, one of the tests performed on the ADC module is whether the converter is capable of 20,000 conversions/sec. Therefore, this rate should be obtainable if an assembly language input routine was written.

A 16,000 samples/sec conversion rate was achieved by a call to the ISA subroutine AIRDW with an array size of 1,000. For calls to the subroutine with an array size of one or a single conversion, a throughput of only 6,000 conversions/sec was possible. Analysis of the time required for many calls to the subroutine with different array sizes indicates that 70 to 90 μ secs were spent in the execution of the FORTRAN statements supporting the subroutine call.

The maximum throughput rate for the DAC module was

measured as 50 kHz, which was for an array size of 1,000. Small array sizes once again degraded the speed of the system due to the inherent overhead of the FORTRAN statements, which was 60 to 70 μ secs. This conversion rate of 20 μ secs for a large array size is still twice the 10 μ secs speed that is indicated in the DAC specifications of Figure 1.2 on page 8.

For applications where the ADC channel is sampled with a large array and then the sampled data, after some manipulations, is output to a DAC channel, one should note that the throughput of the DAC is approximately three times faster than the ADC. Therefore, the output waveform is increased in frequency by a factor of three. One possibility for pacing the output data is to code into the software a small delay loop.

A short program was coded that accepts one value from the ADC input channel and writes that value directly to the DAC output channel. By programming this loop to execute as fast as possible, a throughput rate of six kHz was achieved. From the Nyquist theorem, the fastest waveform that could be converted would be a three Khz signal. This excludes any type of digital encoding of speech or music. By coding this read/write loop in TI assembly language, a much higher throughput could be achieved. The FORTRAN Programming Manual mentioned in reference [6] contains an excellent example of calling Macro routines from FORTRAN.

Construction of the I/O Panel

No external cabling is available for the DAC ADC module from Texas Instruments because the diverse number of applications that the devices could be used for would require a large selection of cables. Therefore, the first step in utilizing the devices involved the construction of a input/output panel. All channels for both modules were brought out to connectors on the I/O panel so that all of the capabilities of the devices could be available for later use.

Coax-cable was initially chosen for the connections from the board to the panel, but the header on the connector to the board did not allow sufficient clearance for a bundle of 64 coax-cables. Therefore, ribbon cable, which was readily available in stores, was selected for the connections. The main factor in choosing coax was to reduce the noise pickup that would be encountered in the use of flat ribbon cable.

Figure 1.3 shows the layout of the I/O panel that was constructed. Each ADC channel that is not in use can be selectively grounded by the column of DIP switches on the ADC section of the panel. This technique provided some additional noise immunity along the ribbon cables' 12 foot length. To facilitate testing, all DIP switch outputs can be either grounded, or connected to a single voltage (or current) source. This allows for all ADC inputs to be

University of Tennessee Digital Signal Processing Laboratory

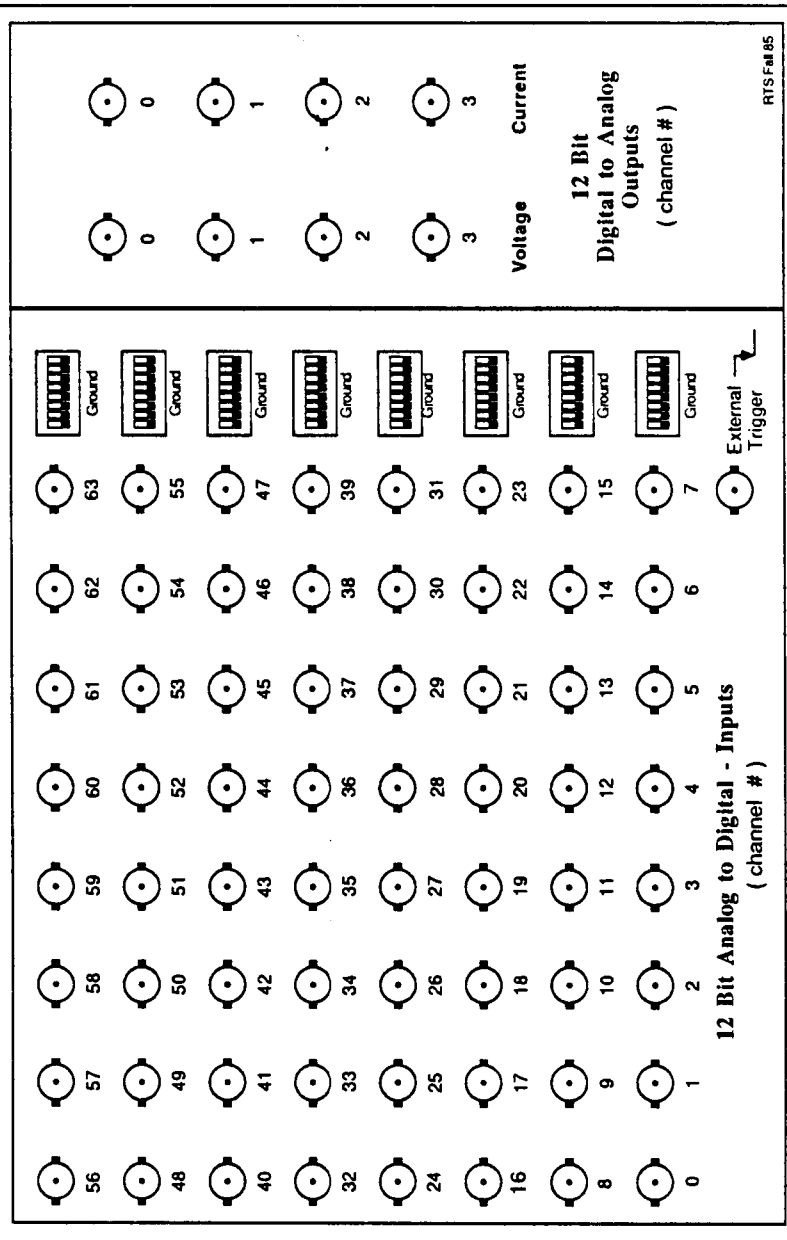


FIGURE 1.3. Input/Output Panel for TI System.

attached to the same voltage source, and in turn, eliminates the necessity for the individual who is calibrating the unit to attach a reference voltage source to each individual ADC input in sequence.

If the ADC module is configured for differential-ended inputs, then the channel that is vertically adjacent makes up the pair of inputs that are required. For example, channel zero and eight make up a differential channel, and channel 16 and 24 make up another differential channel.

To measure the noise that is attributed to the ribbon cable, the noise content of the outputs directly from the ADC and DAC modules edge connector was compared to the noise content at the end of the 12 foot ribbon cable. First the DAC's output channels will be analyzed with the ribbon cable removed. When driven to a ground potential, the output channels on the DAC had a noise spike every 28 μ secs with a width of less than 1 μ sec. This spike varied in amplitude from 10 mV's to 200 mV's. When one of the adjacent channels was driven with a square wave of the largest possible amplitude and speed, no crosstalk between the channels was observed.

With the ribbon cable connected and the noise measurement performed at the I/O Panel's end, the same noise spike was observed with the same amplitude. When the adjacent channel was driven with the large, fast square wave,

approximately three to ten millivolts of crosstalk were observed. When the DAC is set for the -10.48 to +10.48 volt range, the LSB has a value of 5 mV. Therefore, the noise could contribute up to 2 bits of error to the output signal.

To measure the ADC's noise content, the input channel was grounded and a short program was executed that continuously read that channel's input value and displayed the largest variance from the correct value. A variance of at most 2 bits was seen during the testing. For applications requiring precise measurements, several readings can be taken and the results averaged to remove any random noise or the spike that was discussed earlier. Of course, this multiple sampling will degrade the throughput of the systems.

CHAPTER II

STUDENT LABORATORIES

These example laboratories were used as a method of determining the capabilities of the ADC and DAC modules on the TI 990 system. The laboratory programs are coded entirely in the FORTRAN language and utilize the IAS Extensions Subroutines to access the ADC and DAC modules. Each project was chosen to present a different use or capability of the modules in digital signal processing applications.

Because the terminals were not available with a graphical output mode, the output device for several of the labs was an oscilloscope. By driving the X (sweep), Y (amplitude), and Z (blanking) inputs from the DAC module, the user was provided with a stable, flicker-free display of the data. This method provided immediate feedback to the user after each operation on the data.

As an alternative to driving the Z (blanking) input on the oscilloscope, the software that displays the waveforms could return the Y (amplitude) input to zero. This provided an extra trace on the oscilloscope screen at zero potential, but only required two of the DAC modules channels. This would allow two programmers, each with an oscilloscope, to use the system simultaneously. Because the oscilloscope was primarily used without a need for the timebase, the second output device could be a simple deflection driven CRT.

Appendix A through F contain the source code for all the laboratory projects. Subroutines which are used in only one program are included in that program's appendix. If the subroutine is used by more than one program, it is included in Appendix G.

Digital Oscilloscope

This laboratory involves the simulation of a digital oscilloscope using the TI 990 ADC's and DAC's. Samples from the ADC of the analog signal are converted and the results are displayed on the deflection CRT or an oscilloscope. The oscilloscope is just used as the display device; its timebase is not used in any capacity.

To display the signal, the DAC module drives the (X) sweep, (Y) magnitude, and (Z) blanking inputs to the oscilloscope. Because this arrangement is used in several of the remaining labs, and because the concept of this lab is rather straightforward, the digital oscilloscope project is chosen as the first one in the series. Routines that are developed here can be used in several of the remaining labs.

Several features of an oscilloscope were incorporated into this lab. For example, the digital scope (DScope) can be set to trigger on the rising or falling edge of the signal. Also, the voltage at which this trigger mechanism becomes active can also be specified. One is often interested in the

shape of the signal just prior to the triggering event. A delay mechanism implements this feature, and the period prior to the trigger is specified in samples. Different frequency waveforms can be displayed by varying the timebase that the DSCOPE samples at, and in turn, this will alter the display. Finally, the maximum and minimum voltages that are present in the waveform are displayed on the screen after each sample.

Procedure

Since the time required to sample the analog signal, perform the triggering process, find the maximums and minimums, and set up the waveform to the correct timebase for display was on the order of one second, the scope cannot run in a "free-triggered" mode as a normal oscilloscope. Instead, the user is required to initiate the sampling process from the keyboard. Once started, the program displays missed trigger on the screen until a valid trigger is seen.

To achieve the fastest throughput, the DSCOPE program always samples in blocks of 1024 samples. Samples are taken in this large of a block primarily so that the program can capture waveforms of a higher frequency than if it converted one value at a time. The program then looks across half of this block of 1024 samples to see if there is a match to its triggering criterion. Only half of this block is searched because if the trigger is found after this point, there may

not be enough data left in the block to make a meaningful display.

The program always displays 128 samples per trace on the oscilloscope, but the timebase mechanism determines how many samples are skipped over between samples. Low frequency signals that would only display a portion of a complete cycle can be expanded to 128 points over 512 sample points to be expanded across the rest of the block of data and produce a meaningful display.

Once the trigger event is found in the block, the program next determines if a delay has been selected. If so, the program moves the sampled data into the display data buffer a delay number of samples from the trigger point prior to the trigger. Those samples that are prior to the trigger event are shown on the oscilloscope as signals of half intensity.

Results

Because the conversion rate for a 1024 block of samples is 16,000 samples/sec, it requires approximately 64 msec to fill the block. Since only 128 of these samples are displayed at once, the time that is represented on the oscilloscope is 8 msec. From the Nyquist Theorem, one knows that the maximum frequency that can be displayed is 8,000 Hz, but the practical limit that produces acceptable results is 3 kHz. If samples are skipped in the block (i.e. the 128 samples to display are

expanded across the block), the slowest frequency that produces a full cycle is 15.6 Hz. Figure 2.1 shows the trace that is generated on the screen of the oscilloscope for a one kHz signal.

The triggering mechanism worked very well and allowed the trigger event to be specified exactly. Displaying the maximum and minimum voltages on the terminal eliminates the necessity of interpreting the display, and is also accurate to 5 mV.

One enhancement that would provide a needed feature would be the inclusion of a frequency measuring device. Counting the number of transitions or peaks would work for a sine wave, but would be useless on a more arbitrary waveform. This would be an ideal application for the Fast Fourier Transform (FFT), and could possibly be added to the program after the laboratory that develops the FFT.

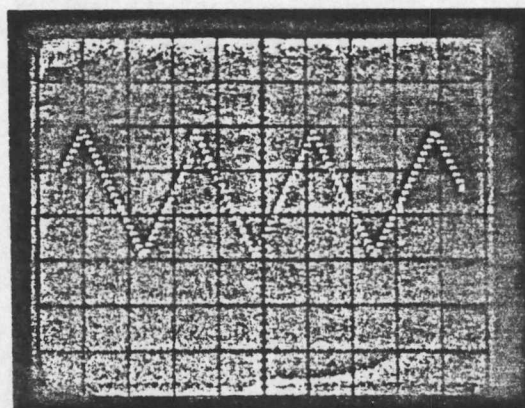


FIGURE 2.1. Digital Oscilloscope Lab Results.

Fourier Series Analysis

Many of the applications in DSP involve the synthesis of an analog signal or a determination of the frequency content. Both of these processes are based on an understanding of the Fourier Series. This laboratory project involves the synthesis of several signals from sine and cosine waveforms of different amplitudes and frequencies. By completing this project prior to Lab 4, a better understanding of the Fast Fourier Transform should result.

All signals, if approximated as periodic over an interval, can be represented as a combination of sine and cosine waveforms of varying frequencies. If the periodic interval is represented as T , then

$$f(t+nT) = f(t) \quad .$$

Assuming $f(t)$ is periodic, the Fourier Series is represented by :

$$f(t) = a_0 + \sum_{n=1}^{\infty} \{a_n \cos(nt) + b_n \sin(n\pi t)\} \quad .$$

The value of the constants a_0 , a_n , and b_n can be determined analytically using the Euler formulas. These are given as :

$$a_0 = \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) \, dt \quad ,$$

$$a_n = \frac{2}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) \cos \frac{2n\pi t}{T} dt \quad n=1,2,3\dots ,$$

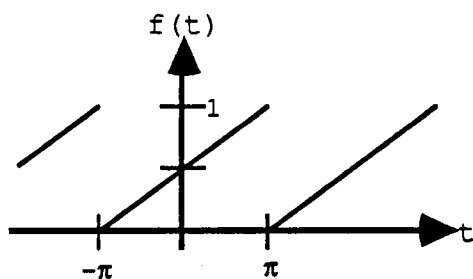
$$\text{and} \quad b_n = \frac{2}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) \sin \frac{2n\pi t}{T} dt \quad n=1,2,3\dots .$$

As an example of the application of the Euler formulas, consider the sawtooth waveform shown in Figure 2.2a. Because the waveform is odd, one knows that there are no cosine terms, hence, all the coefficients represented by a_n equal zero. The value for the coefficient a_0 is determined to be :

$$\begin{aligned} a_0 &= \frac{1}{2\pi} \int_{-\pi}^{\pi} \left(\frac{1}{2} + \frac{t}{2\pi} \right) dt \\ &= \frac{1}{2\pi} \left[\frac{1}{2} t \right]_{-\pi}^{\pi} + \frac{1}{2\pi} \left[\frac{1}{2\pi} \left(\frac{t^2}{2} \right) \right]_{-\pi}^{\pi} = \frac{1}{2} . \end{aligned}$$

Values for the b_n coefficients are found to be :

$$\begin{aligned} b_n &= \frac{2}{2\pi} \int_{-\pi}^{\pi} \left(\frac{1}{2} + \frac{t}{2\pi} \right) \sin \frac{2n\pi t}{2\pi} dt \\ &= \frac{1}{2\pi} \left[-\cos(nt) \right]_{-\pi}^{\pi} + \frac{1}{2\pi^2} \left[\frac{1}{n^2} \sin(nt) - \frac{t}{n} \cos(nt) \right]_{-\pi}^{\pi} \\ &= -\frac{1}{n\pi} \cos(n\pi) \quad n = 1,2,3\dots . \end{aligned}$$



$$a_0 = 0.5$$

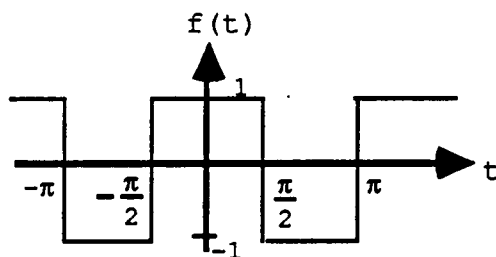
$$a_n = 0.0$$

$$b_n = -\frac{1}{n\pi} \cos(n\pi) \quad n=1,2,3\dots$$

Odd: $f(t) = -f(t)$

Period = $T = 2\pi$

a) Sawtooth Wave



$$a_0 = 0.0$$

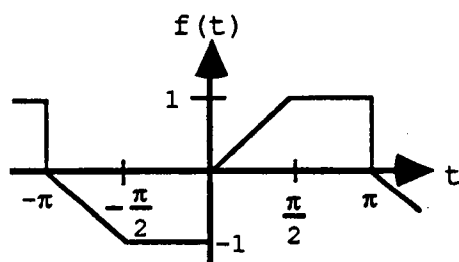
$$a_n = \frac{4}{n\pi} \sin\left(\frac{n\pi}{2}\right) \quad n=1,2,3\dots$$

$$b_n = 0.0$$

Even: $f(t) = f(-t)$

Period = $T = 2\pi$

b) Square Wave



$$a_0 = 0.0$$

$$a_n = \frac{2}{n^2\pi^2} \{\cos(n\pi) - 1\} \quad n=1,2,3\dots$$

$$b_n = \frac{1}{n\pi} \left\{ \frac{4}{n\pi} \sin\left(\frac{n\pi}{2}\right) - \cos(n\pi) + 1 \right\} \quad n=1,2,3\dots$$

Not Even or Odd

Period = $T = 2\pi$

c) Synthesized Wave

FIGURE 2.2. Waveforms for the Fourier Series Laboratory.

Therefore, the software developed for this lab should allow the user to specify a single constant, a_0 , and then several waveforms that are characterized by their amplitude and phase angle. All of these waveforms should be summed and the results shown on the oscilloscope. As more of the harmonics in the series are included, the resultant waveform should increasingly take on the shape of the desired waveform.

Two additional waveforms are shown in Figure 2.2b and 2.2c with their associated coefficients. Notice that the b_n coefficient for the square wave is equal to zero because the wave is even. The waveform in Figure 2.2c is an arbitrary shape which is neither even nor odd; hence, the a_n and b_n coefficients are both non-zero.

Procedure

The oscilloscope should be attached so that the DAC drives the X (sweep), Y (magnitude), and Z (blanking) inputs. This arrangement allows for the waveform that results from the summation of the sine and cosine terms to be displayed. When the program is first begun, the user is prompted for the constant a_0 . Next, the user is repeatedly prompted to either build another waveform, add to the current waveform, print a plot or table of the values, or exit the program.

Using a method of testing the keyboard for input before actually entering a wait state to receive input from the keyboard, the program is allowed to display the results of the

last operation on the oscilloscope until the user requests another action. This method of displaying the last result until the user requests another action is used throughout the remainder of the labs.

Results

The coefficients for the waveforms shown in Figure 2.2 were used as the inputs to the program. The resultant waveforms were plotted on the printer and those results are shown in Figure 2.3, 2.4, and 2.5. For the sawtooth wave, six coefficients were specified. The square wave and the arbitrary shaped wave were specified with 11 and 21 coefficients, respectively.

An enhancement to the program would be for the software to prompt for the number of coefficients to display in the series, and then have the program generate that number of coefficients itself. This would result in a waveform that is a much closer approximation to the desired waveform. The user should keep in mind that as the number of coefficients the program calculates increases, a roundoff error will eventually become a problem as the coefficients tend toward zero.

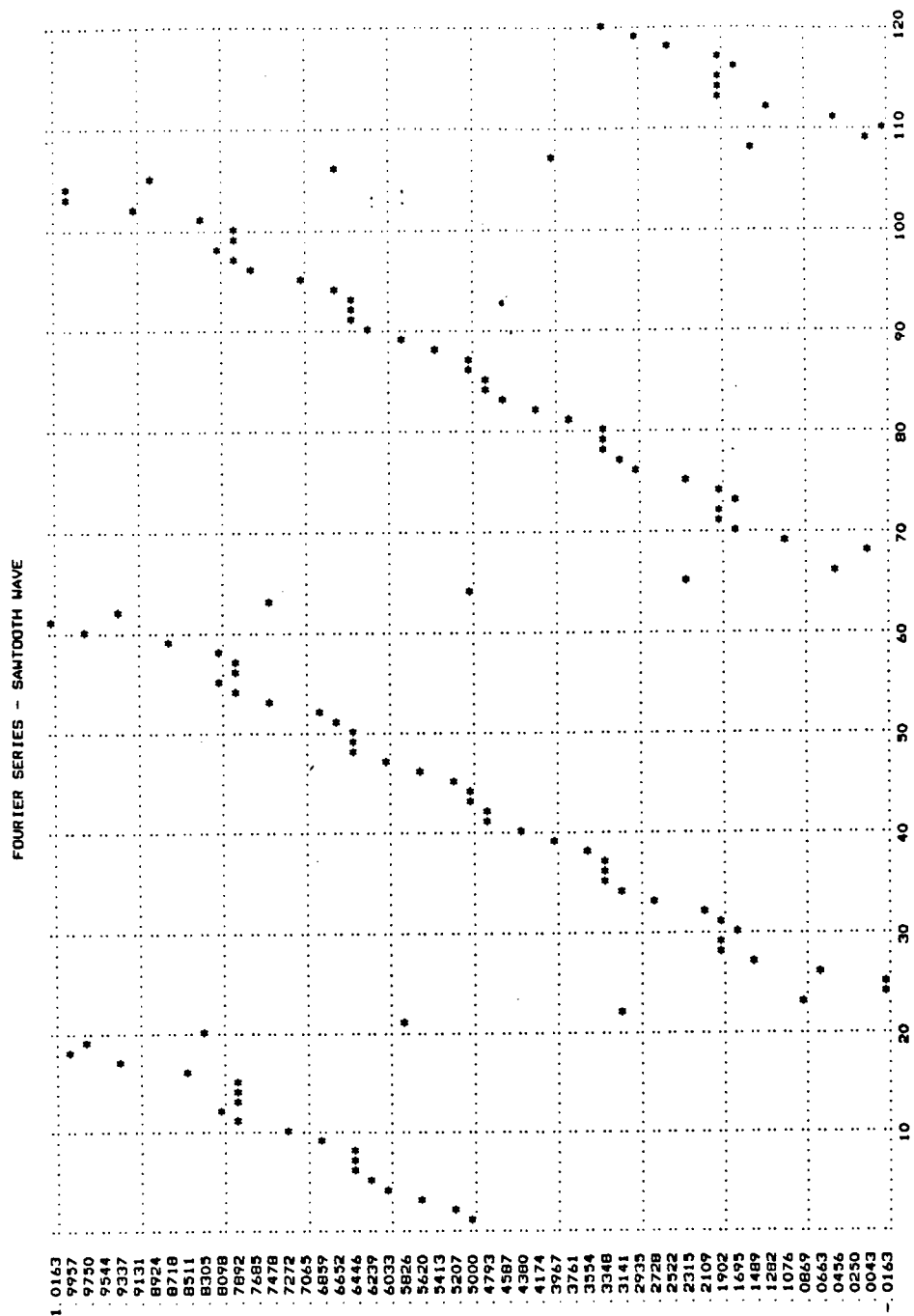


FIGURE 2.3. Fourier Series Sawtooth Wave - Fundamental and Six Harmonics.

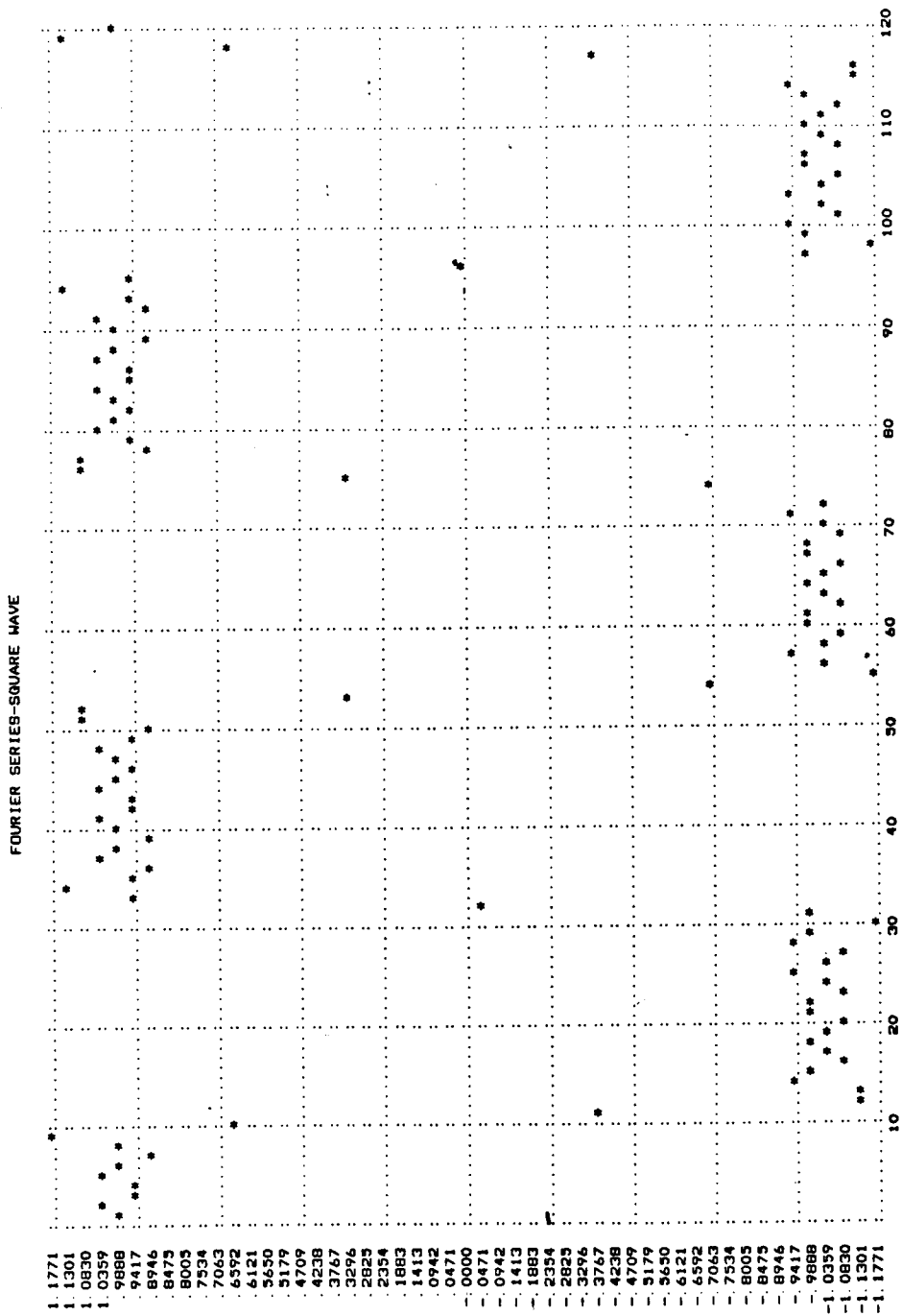


FIGURE 2.4. Fourier Series Square Wave - Fundamental and Eleven Harmonics.

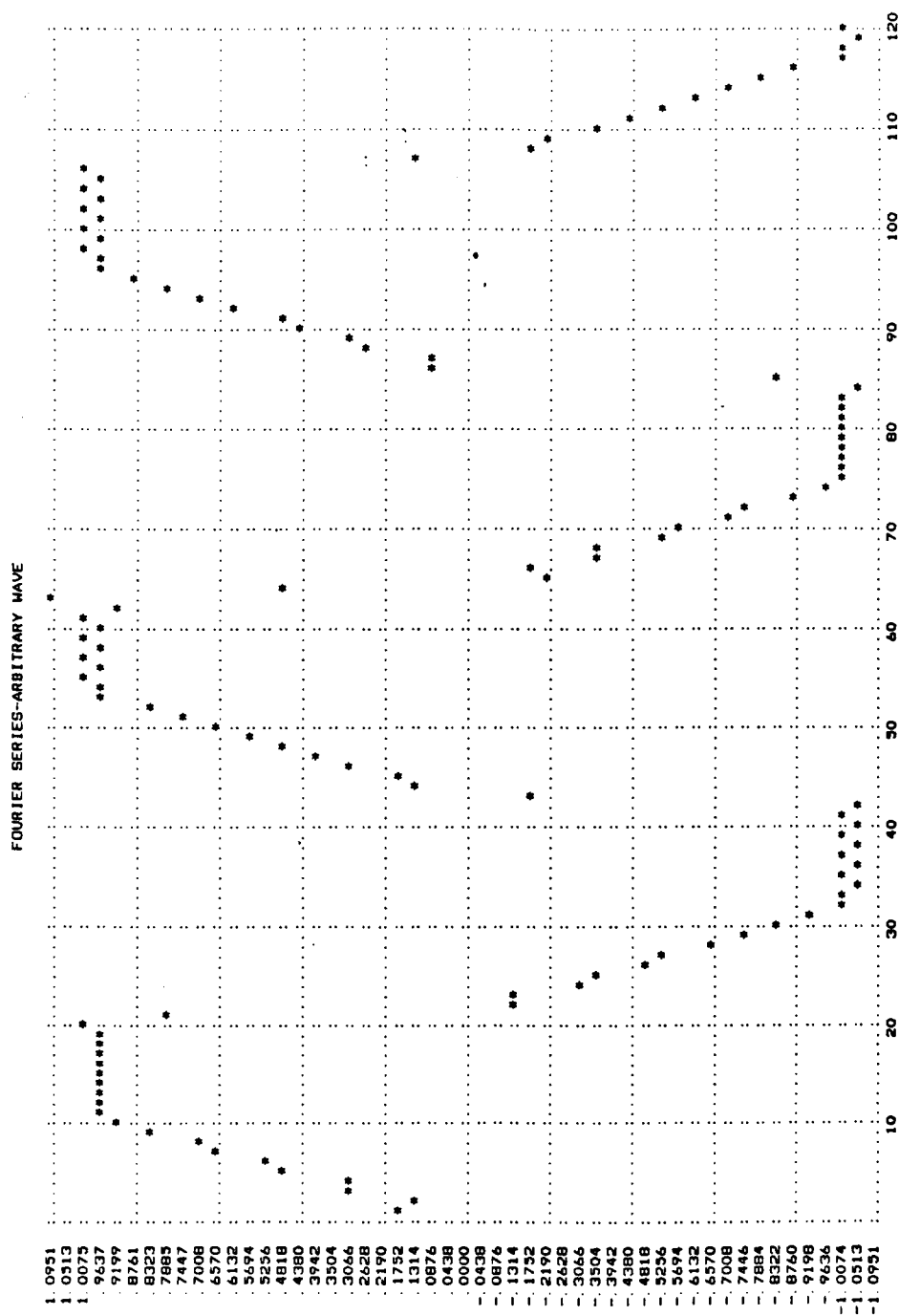


FIGURE 2.5. Fourier Series Arbitrary Wave - Fundamental and Twenty-one Harmonics.

ADC System Characterization

One is often interested in determining the characteristics of another Analog-to-Digital system. By using a host Analog/Digital system with high resolution driving a device under test of lesser resolution, several characteristics of the device under test can be observed. This project will entail using the ADC and DAC module on the TI 990 system to test an eight-bit single chip Analog-to-Digital conversion device.

The single chip successive approximation ADC chosen was the ADC0804LCN because of its general availability, inexpensive cost, ease of use, and performance limitations. Much more can be gained from the lab if the performance limitations of the device under test are easily observable. Therefore a device with a total error of ± 1 LSB (least significant bit) was chosen instead of devices with total errors of $\pm 1/2$ LSB, or less.

The terminology used in specifying ADC's must be explained before the errors associated with conversion of an analog to a digital value can be appreciated. Most performance values in the specification of an ADC are based on the number of bits or resolution of the converter. This is used directly to determine the accuracy of the converter and the channel width. For eight bits, 2^8 or 256 different binary values can be represented. Therefore, for a converter with

0.0 to 5.12 volt input range, the relative accuracy and channel width is:

$$\text{Relative Accuracy} = 1/256 * 100 = .39\% \text{ of FS (full scale)}$$

$$\text{Channel Width} = 5.12 * .39\% = 20 \text{ mV}$$

As the analog input voltage increases from its smallest value to the full scale value, the transfer function shown in Figure 2.6a is produced for an ideal ADC. Notice that the analog input voltage is characterized at the center of the channel width. The absolute error that is produced for this ideal converter is shown in the error plot of Figure 2.6a. Each digital 'step' is 1 LSB wide, which produces a maximum error of $\pm 1/2$ the LSB or channel width. This error of $\pm 1/2$ the LSB is also referred to as the inherent quantization error. This produces an accuracy of ± 0 LSB because the transfer function changes to the next digital value at exactly $A_{-1} + (A - A_{-1})/2$. In other words, the only error present is the inherent quantization error.

Next, consider the transfer function for a converter with an accuracy of $\pm 1/2$ LSB shown in Figure 2.6b. Please note that the analog input no longer produces a transition to the next digital value at exactly $A_{-1} + (A - A_{-1})/2$. Instead, the transition can occur up to a maximum of $\pm 1/2$ LSB from this edge point.

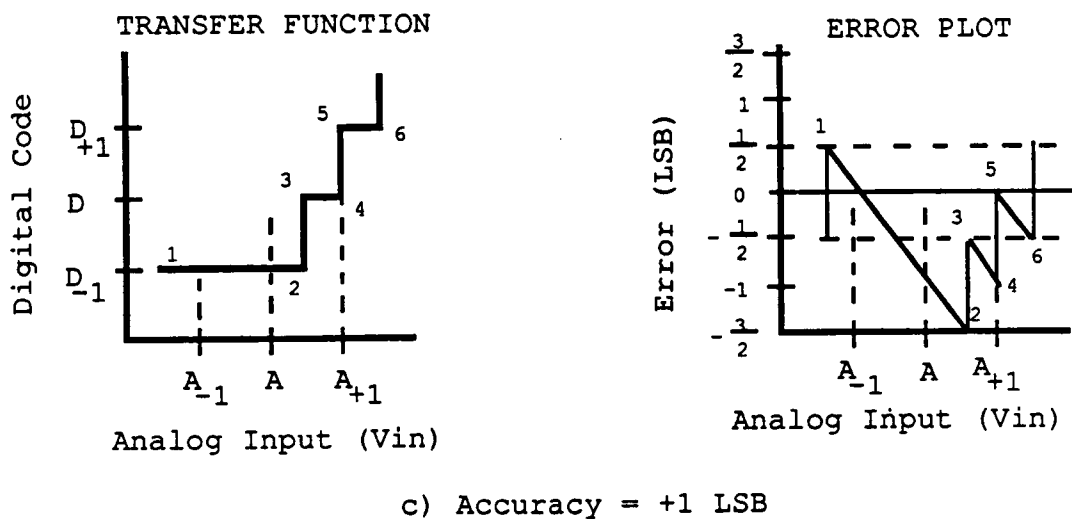
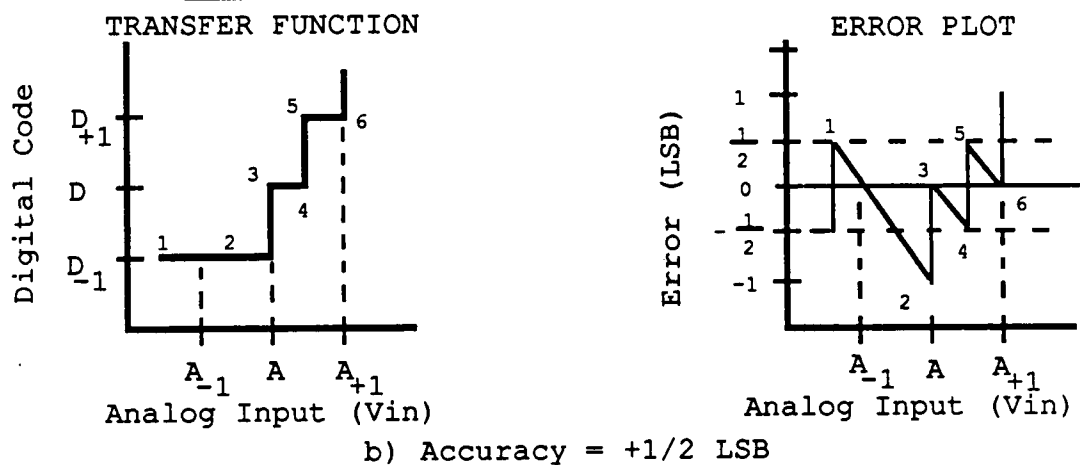
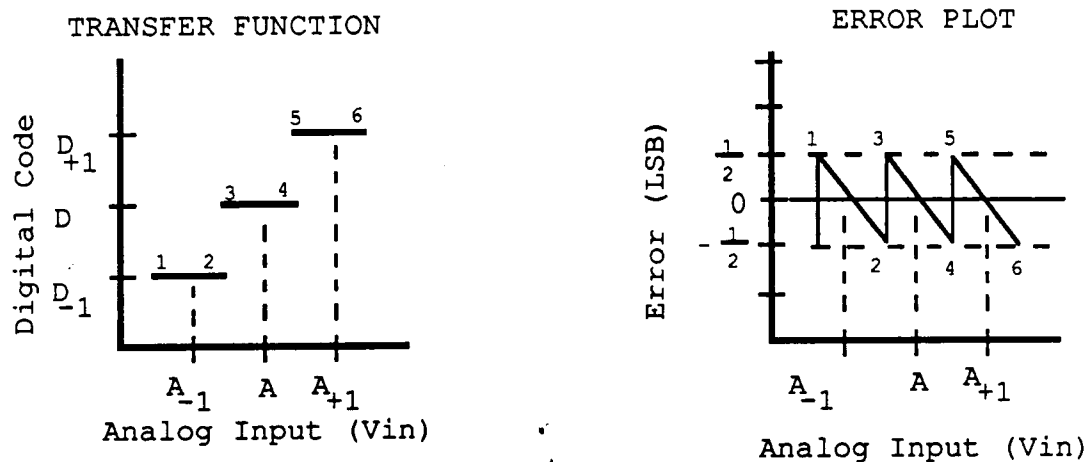


FIGURE 2.6. Analog-to-Digital Conversion Errors.

The ADC0804LCN is specified to have an error of no greater than ± 1 LSB. This means the manufacturer guarantees that if an analog value is applied at one of the transition steps ± 1 LSB, that the ADC will output the correct code [2]. The transfer function for this large of an error is shown in Figure 2.6c. Notice that a digital code could be missed entirely with this large of an error; hence, this condition is called a missed code. The complete characteristics of this converter are listed in Figure 2.7.

Total Unadjusted Error	± 1 LSB		
Relative Accuracy	.39 % of FS		
Channel Width (0.0 to 5.12)	20 mVolt		
	<u>Minimum</u>	<u>Maximum</u>	<u>Units</u>
Vcc	5.0	6.0	volts
Voltage at Any Input		Vcc+0.3	volts
Clock Frequency f(clk)		800	kHz
Conversion Time Tc		93	μ secs

Values for Constructed ADC

R clk	10 K		
C clk	150 pF		
	<u>Calculated</u>	<u>Measured</u>	<u>Units</u>
f(clk) (pin 4)	600	575	kHz
Tc	135	160	μ sec

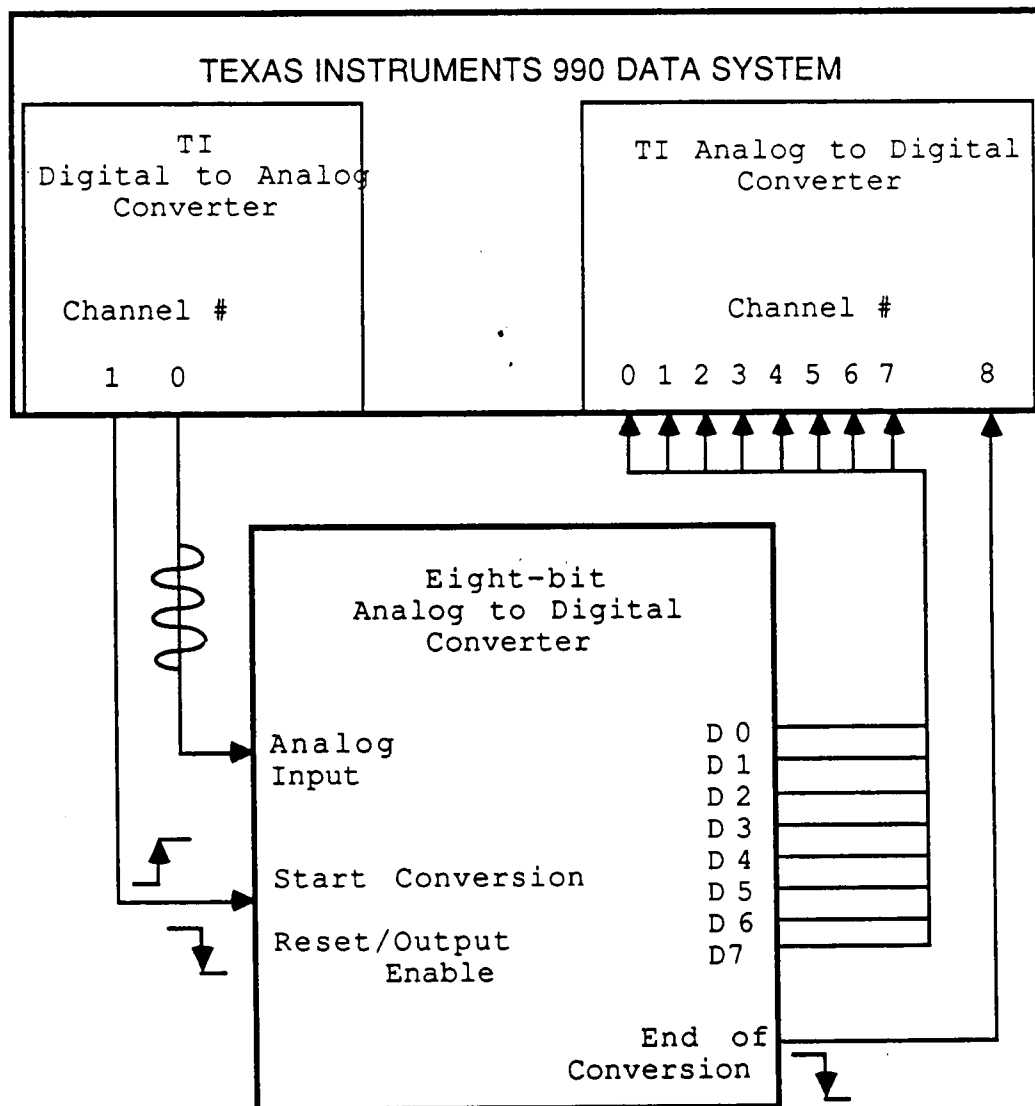
FIGURE 2.7. ADC0804LCN Specifications.

Procedure

By using the TI 990's 12 bit DAC module to produce an analog signal to serve as the input to the eight-bit ADC under test, then the accuracy, linearity, and differential linearity of the device can be measured. The block diagram for this project is shown in Figure 2.8. Notice that the TI 990's DAC also produces the Start Conversion signal for the 0804 ADC. This signal must be produced as a TTL signal (0.0 to 5.0 volts). Also, the results of the 0804 ADC conversion are to be read by the TI 990's ADC module. This includes the eight data outputs and the End of Conversion signal.

Since the TI 990 was controlling the inputs to the 0804 ADC and reading the resultant data and control line, a thorough understanding of the timing constraints of the 0804 chip was necessary. The timing diagram in Figure 2.9 shows the entire cycle involved in stimulating the 0804 ADC and reading the results.

An overhead view of the construction details of the 0804 ADC project is shown in Figure 2.10. The ADC was built on perforated board and mounted inside an aluminum casing, which was attached to the chassis ground. To provide for easy attachment to the I/O panel that was discussed in Chapter I, the inputs and outputs to the ADC were constructed as coax BNC connectors. A nine-volt battery powered the device, but this could easily be adapted to use a nine-volt power supply.



NOTE: The outputs D0 to D7 and EOC are interpreted by the software as digital signals.

FIGURE 2.8. Block Diagram of the Eight-bit ADC Under Test.

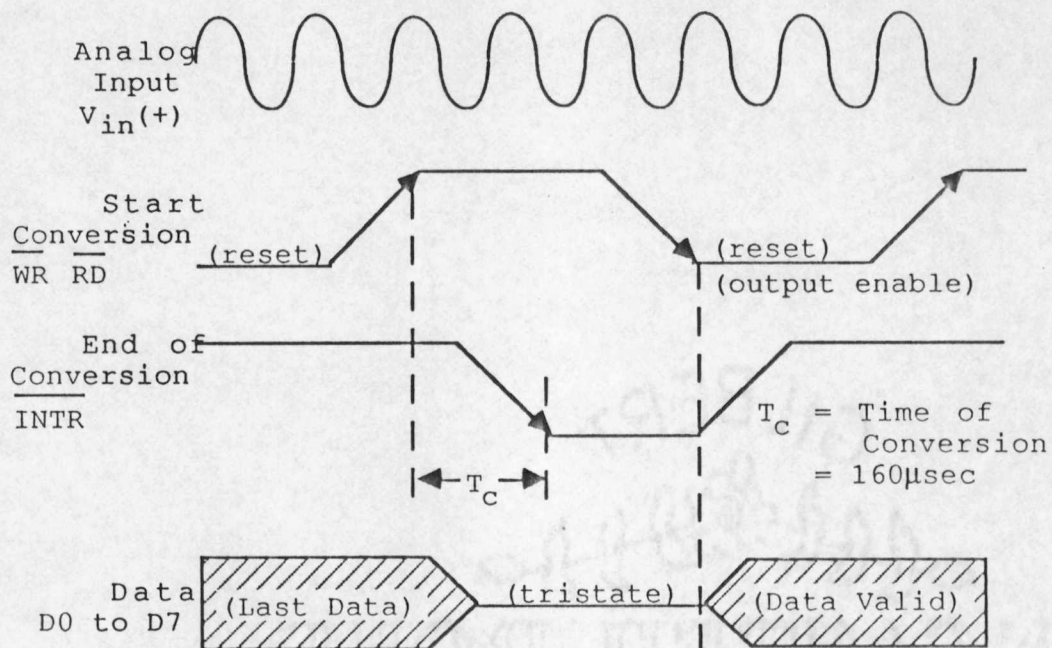


FIGURE 2.9. 0804 ADC Timing Diagram.

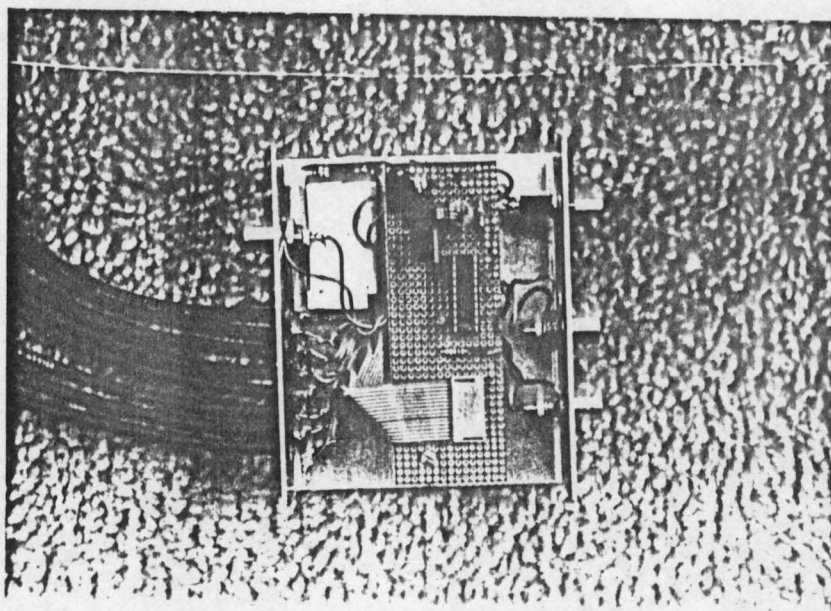


FIGURE 2.10. Top View of Eight-bit ADC Project.

Because the TI 990 DAC outputs are capable of -10.24 to 10.24 volts, caution must be exercised when initially attaching the 0804 ADC to the DAC channels. A maximum input voltage at any pin of $V_{CC} \pm 0.3$ volts is specified in the data sheet for the 0804 ADC. One method of guaranteeing that the maximum voltage parameter is not exceeded is to design any program to be used with the 0804 ADC to condition the TI 990 DAC to voltage levels which are within these limits. After these safe limits have been set, the program should inform the user to attach the 0804 ADC device.

Results

Before gathering any test data with the 0804 ADC, the device was calibrated using a digital voltmeter. An LM336 precision voltage zener diode is the reference that is used by the 0804 ADC to perform conversions. This zener diode is manufactured with an adjustment tap available for external use. A 10K ohm potentiometer was connected across this tap, and the device was calibrated for 2.56 volts. Because the 0804 ADC has an auto-zeroing ability, no offset error adjustments were necessary. Figure G.1 in Appendix G contains a complete schematic for the system.

The first test measurement performed on the 0804 ADC was a determination of the conversion speed of the device. By observing the END OF CONVERSION signal on the oscilloscope with the START CONVERSION signal attached to the external

trigger of the oscilloscope, the duration of the conversion process could be observed. The manufacturer specifies that the clock frequency can be analytically determined from the equation

$$f_{clk} = 1/((1.1) * R * C)$$

For the component values listed in Table 2.1, page 34, the frequency is found to be 600 kHz. An analog to digital conversion requires at most 81 clock cycles on 135 μ secs. The measured conversion time of 160 μ secs compared closely with the calculated conversion time of 135 μ secs.

Several errors of the DB04 ADC were revealed by the software program that is listed in Appendix D. With the TI 990 DAC configured for a range of -10.24 to 10.24, a channel width of 5 mVolts is produced. Because the 0804 ADC has a channel width of 20 mVolts, each consecutive four analog samples should be converted to the same digital value by the 0804 ADC. Therefore, 1024 analog output values were converted by the 0804 BDC into 256 digital values. The results of running this program over the entire range of 0.0 to 5.12 volts for the 0804 ADC is shown in Figure 2.11.

There are six main columns in the figure, with four headings to each column. The column CODE indicates which digital code that the analog signal was converted to by the 0804 ADC. Since approximately four analog signals should convert to the same digital value, a good performance

indicator is the number of analog signals the 0804 ADC grouped together. This total is shown under the column headed # (NUMBER). Each analog signal that is output by the TI DAC and returned by the 0804 ADC as a digital value is stored in an array. SMALL and LARGE indicate the smallest and largest digital codes in the array that were quantized into the same digital CODE indicated for that specific row.

The first item to notice in Figure 2.11 is the star "*" that is beside one of the codes. This indicates where the 0804 ADC failed to always increase the digital code with an increase in the analog input, a condition known as monotonicity. Normally, the data sheet for the device in use will specify a guarantee of monotonicity if that characteristic is present. No such guarantee is stated for the 0804 ADC.

Originally, the program was run with only one data value being read for each of the 1024 binary values that the TI 990 DAC was generating. This produced an average of six to eight "*" or non-monotonic values. When 50 data points were read for each binary value of the TI 990 DAC, this dropped to an average of zero to one "*" per table. By taking the average of the 50 values and using that result, any random noise present in the system is eliminated. This condition was discussed in Chapter I.

Separating lines have been drawn on the tabular data in

Figure 2.11 to differentiate where each 32 digital steps occur of the 256 digital steps for the 0804 ADC. The column labeled LARGEST can then be used as an indication of the gain error of the device. For example, if a positive gain error was present, then fewer analog input steps would be required to attain the gain associated with the digital code of 32. As can be seen from Figure 2.11, each 32 step increase in CODE results in 127 to 128 analog steps, which indicates a flat gain.

Non-linearity is the departure from a linear transfer curve. Quantizing, zero, and scale errors are not included in the non-linearity specification [3]. Figure 2.11 shows a maximum deviation from the center value for all 256 levels of 4 bits or ± 1 LSB. This maximum occurs at several locations, but for example, consider the SMALL value returned for CODE equal to 255 (Full Scale). The correct digital code for the center-level is $(255 \times 4) + 2$ or 1022. This is a deviation of $4 \times 5\text{mV}$ or 20 Mv of non-linearity, which is ± 1 LSB of the 0804 ADC.

Differential non-linearity indicates the difference between actual analog voltage change and the ideal (± 1 LSB) voltage change for the ADC. This difference can always be as much as twice the non-linearity, but no more [3]. Using this criterion, the largest difference between actual analog voltage change and the digital channel width can be determined

from Figure 2.11. For every four consecutively increasing analog inputs to the 0804 converter, a change to the next digital code should occur. Figure 2.11 indicates that the converter actually changes levels for a minimum of three and a maximum of five inputs. Therefore, the differential non-linearity was calculated as $\pm 1/4$ LSB.

Fast Fourier Transform and Filtering

Many operations, such as filtering, are much easier and efficient to perform on the frequency spectrum of a signal than on the time domain. One accepted method of transforming sampled data into the frequency domain involves the use of the Fast Fourier Transform (FFT). The sampled data is transformed into the frequency domain using the FFT and some filtering, smoothing, or shifting operation is then applied. Then the results are transformed back to the time domain by an inverse application of the FFT. These results may then be output as the modified sample data.

The Fourier Transform and its inverse are defined for a real and continuous function $f(x)$ as:

$$F(u) = \int_{-\infty}^{\infty} f(t) \exp[-j2\pi ut] dt$$

$$\text{and } f(t) = \int_{-\infty}^{\infty} F(u) \exp[j2\pi ut] dt \quad .$$

For the discrete case:

$$F(u) = \frac{1}{N} \sum_{t=0}^{N-1} f(t) \exp[-2j\pi ut/N] \quad u=0,1,2\ldots,N-1$$

$$\text{and } f(t) = \sum_{u=0}^{N-1} F(u) \exp[2j\pi ut/N] \quad u=0,1,2\ldots,N-1$$

Because a numerical solution using these equations on a data set with N elements requires N^2 complex multiplications and additions, the time to solve for the Fourier Transform can become very lengthy, even with a small data set. One approach to decreasing the number of complex operations is to decompose the equation for the Fourier Transform into :

$$F(u) = \frac{1}{2} \left[F_{\text{even}}(u) + F_{\text{odd}}(u) \exp[-j2\pi u/2M] \right],$$

$$F_{\text{even}}(u) = \frac{1}{M} \sum_{x=0}^{M-1} f(2x) \exp[-j2\pi ux/M] \quad u=0,1,2\ldots,M-1,$$

$$\text{and } F_{\text{odd}}(u) = \frac{1}{M} \sum_{x=0}^{M-1} f(2x+1) \exp[-j2\pi ux/M] \quad u=0,1,2\ldots,M-1,$$

$$\text{where } M = \frac{N}{2}.$$

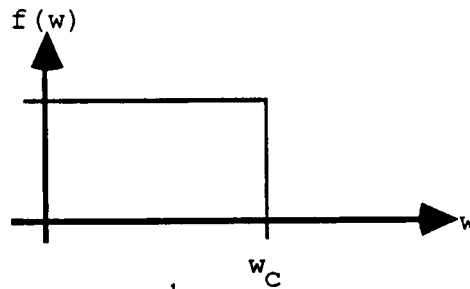
This decomposition requires only $N \log_2 N$ complex operations and is called the Fast Fourier Transform (FFT). To perform the inverse of the FFT requires only that the complex conjugate of the complex frequency domain data set be taken and the FFT be applied to this data set. [4]

Filtering the sampled data set implies an adjustment or modification to the data set. In DSP, the most common filtering performed involves the emphasis or de-emphasis of some frequency in the spectrum. This is most easily accomplished in the frequency domain and is one driving force for the application of the FFT. Three of the most common low-pass filters are shown in Figure 2.12 with their transfer functions. By rearranging the terms, these transfer functions can also be applied as high-pass filters. [5]

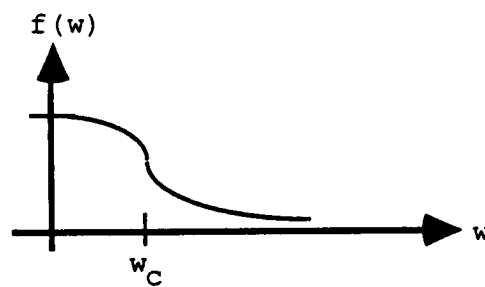
One of the most desirable features of DSP is that the filtering process allows such complete control in shaping the result. Many filters, such as the ideal filter, are not physically possible using analog components. Several other type filters such as comb and notch that are easily realized in software, are much more difficult to design using analog devices and components.

Procedure

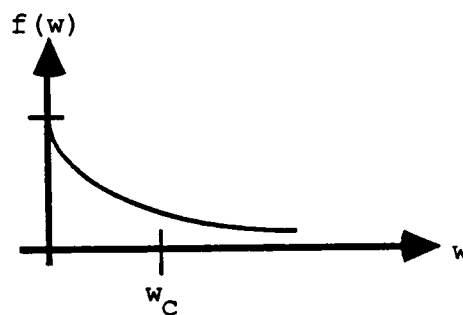
One application that can readily be applied to the program developed for this laboratory is the separation of two sine waves of different frequencies into their base



a) Ideal Low-pass Filter



b) Butterworth Low-pass Filter



c) Exponential Low-pass Filter

FIGURE 2.12. Common Low-pass Filter Characteristics

components. To provide for the addition of several waveforms, and use this result as the input to the filter program, the unity gain operational amplifier shown in Figure 2.13 was constructed. Two different frequency sine waves were combined into one waveform by this amplifier, and the results were decomposed by the program into the original two frequencies. This approach allowed for the magnitude and frequency of the components to be controlled completely and independently of each other.

As in the previous laboratory, the TI DAC's were used to drive the X (sweep), Y (amplitude), and Z (blanking) inputs on the oscilloscope. The signal was sampled with one of the TI 990's ADC inputs.

When the program TRANSF was first started, the user enters the number of samples to take on the ADC input for each trace. The user was then prompted on the terminal screen to select a choice of the following operations:

- 1) sample the signal;
- 2) Fast Fourier Transform (FFT) the signal;
- 3) inverse FFT the signal;
- 4) filter the signal;
- 5) or print the results.

Results from each operation were displayed on the oscilloscope until a new operation was selected by the user.

The FFT subroutine that was used in the laboratory was

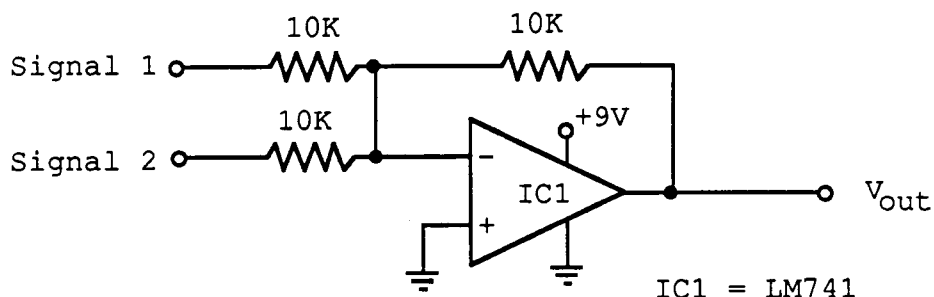


FIGURE 2.13. Operational Amplifier Unity Gain Summing Junction.

an adaptation of a program written by Cooley, Lewis, and Welch [6]. The FFT subroutine and the main program, TRANSF, are both listed in Appendix E.

Results

One frequency of 1.2 kHz and a second frequency of 320 Hz were attached to the inputs of the summing junction. Both waveforms required the addition of an offset because the operational amplifier derived its voltage from a single +9.0 volt supply. The lower frequency was assigned a larger voltage swing than the higher frequency so that both frequencies could be easily observed in the resultant waveform. This waveform is shown in Figure 2.14 with the amplitude values on the vertical axis, and the data point value on the horizontal axis. Because of the limitation in width of 132 columns per line of print-out, the plot of the waveform only shows the first 120 of the 128 samples taken.

The approximate frequency of the waveform can be derived

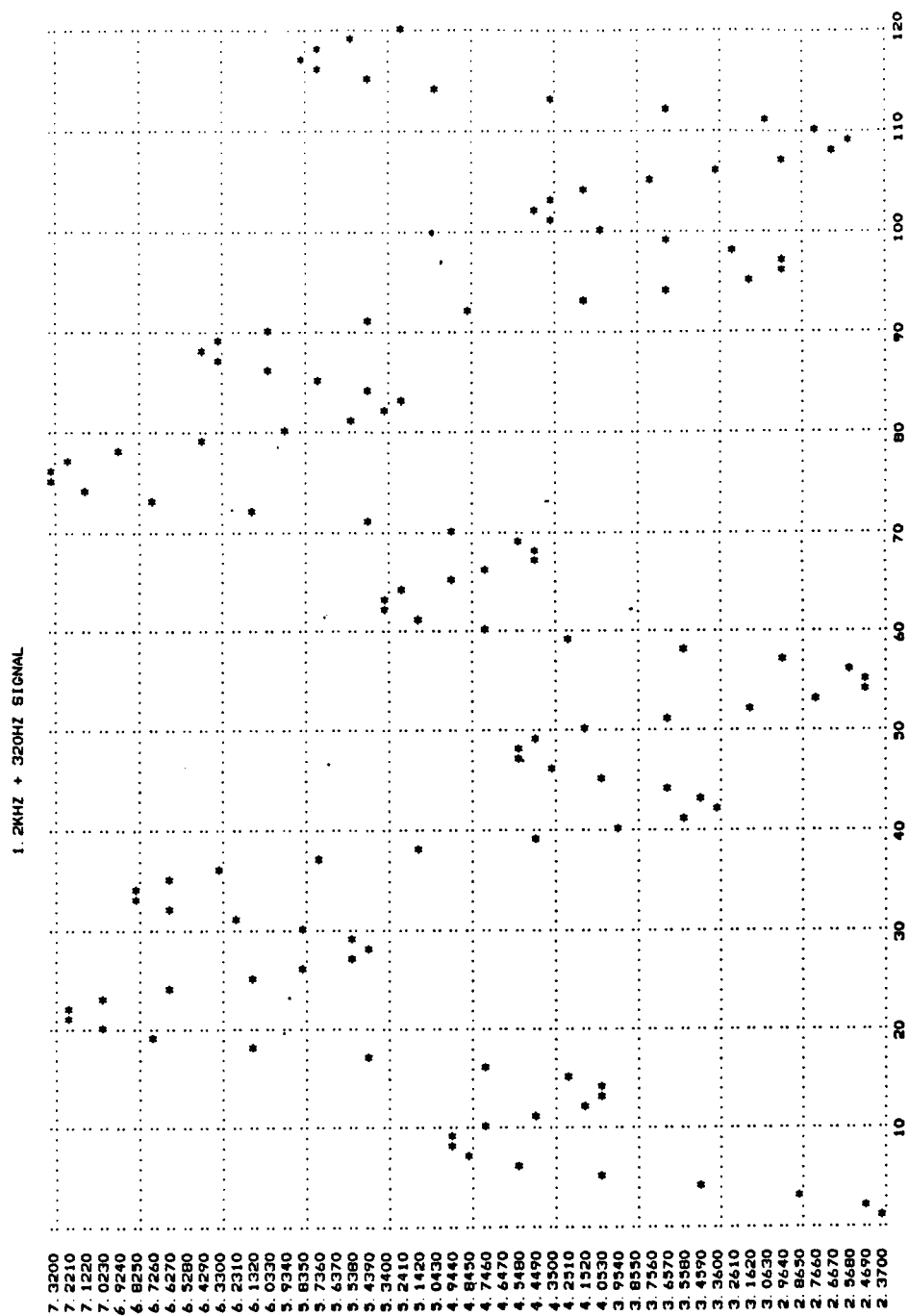


FIGURE 2.14. 1.2 kHz Signal Summed with a 320 Hz Signal.

from the horizontal axis from the knowledge that the sample rate of the ADC was 16,000 samples/sec. Because 1.2 kHz is not an even multiple of 320 Hz, the two waveforms roll in and out of phase with each other, making it difficult to determine the exact number of data points per cycle. Over several cycles of the waveform one can observe that there are approximately 45 to 55 data points per cycle. This produces a frequency of 290 to 355 Hz.

Figure 2.15 shows the results of applying the FFT algorithm to the sampled data. The results of the FFT process are normalized to a value of 5.0 for display, but internally retain their actual value for all calculations. Because of the width limitation, only the first 120 of 128 frequency divisions are shown. The large spike with a value of 5.0 is the voltage offset of the waveform. By the application of an ideal high-pass filter, this large spike and the associated offset voltage were removed. This filtering process was applied with a cutoff frequency ω_c of two, thereby removing all data points at or below sample number two. The inverse FFT was then applied and the results are shown in Figure 2.16. Notice that the waveform no longer contains an offset voltage. Also, the peak to peak voltage swing is still 4.95 volts.

When the FFT transform is again applied to this waveform with no offset, the plot in Figure 2.17 results. Once again the results are normalized to 5.0, but now the two frequency

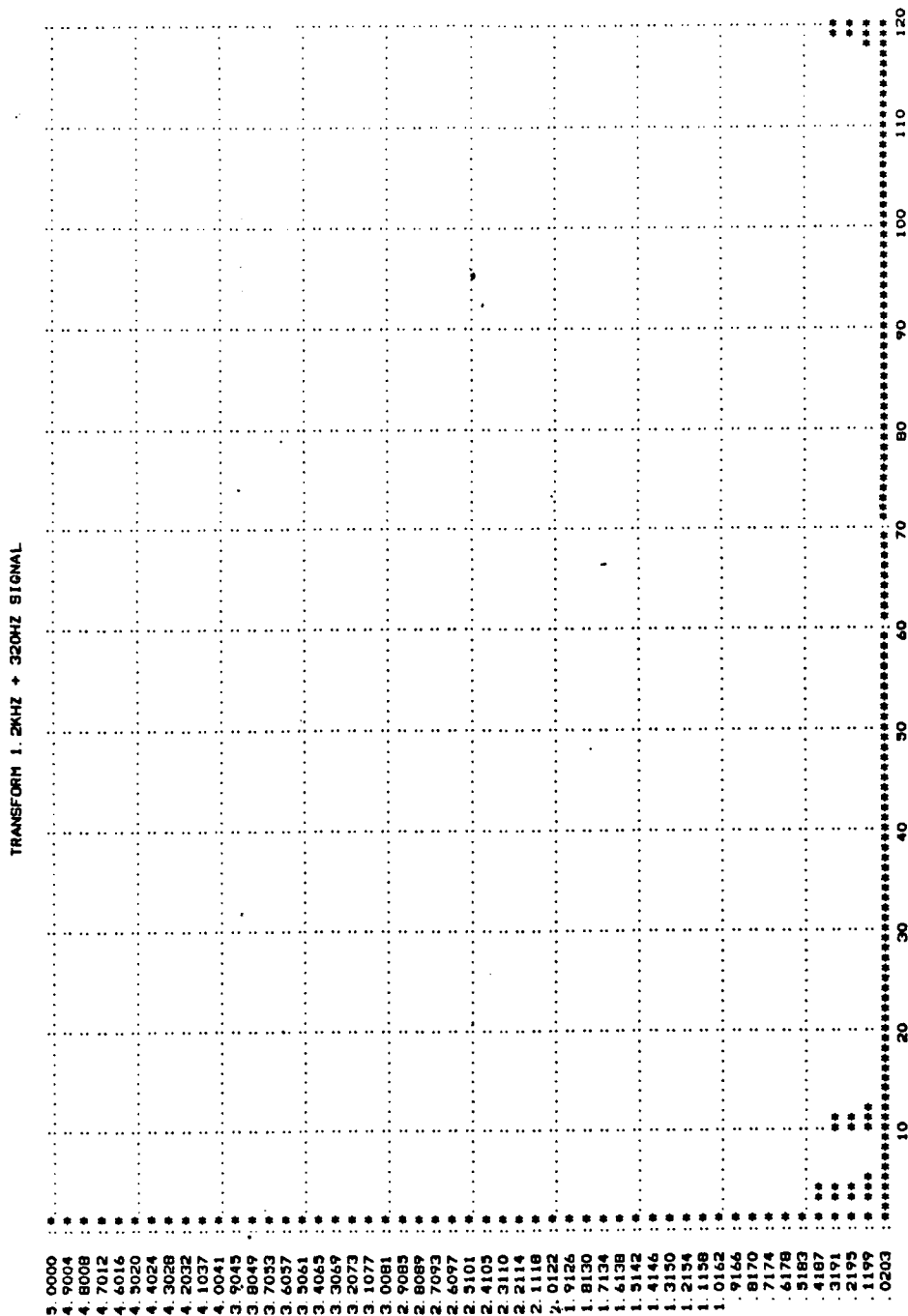


FIGURE 2.15. Fourier Transform of 1.2 kHz Signal Summed with 320 Hz Signal.

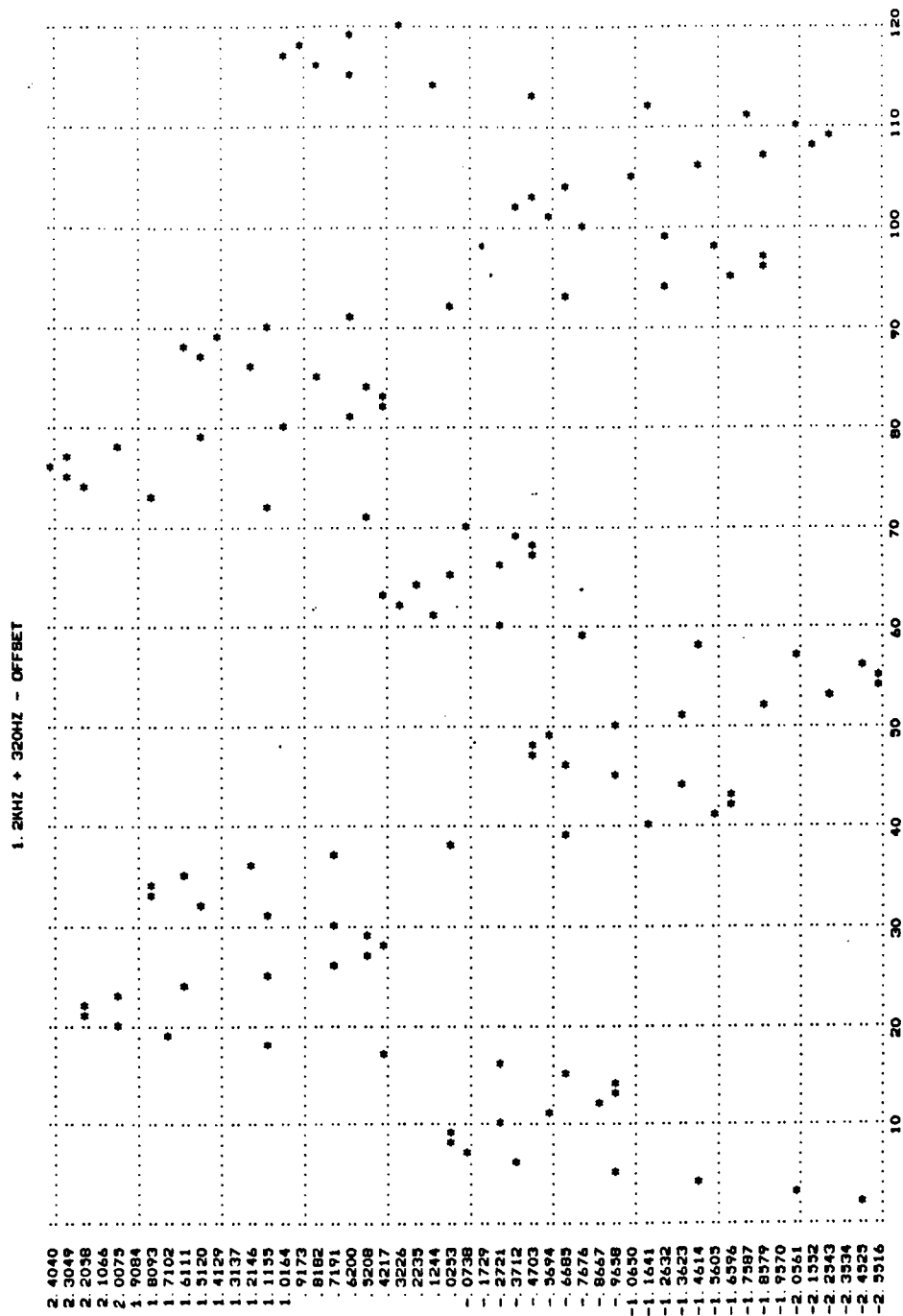


FIGURE 2.16. Inverse Fourier Transform of High-pass Filtered Signal - Less the DC Offset.

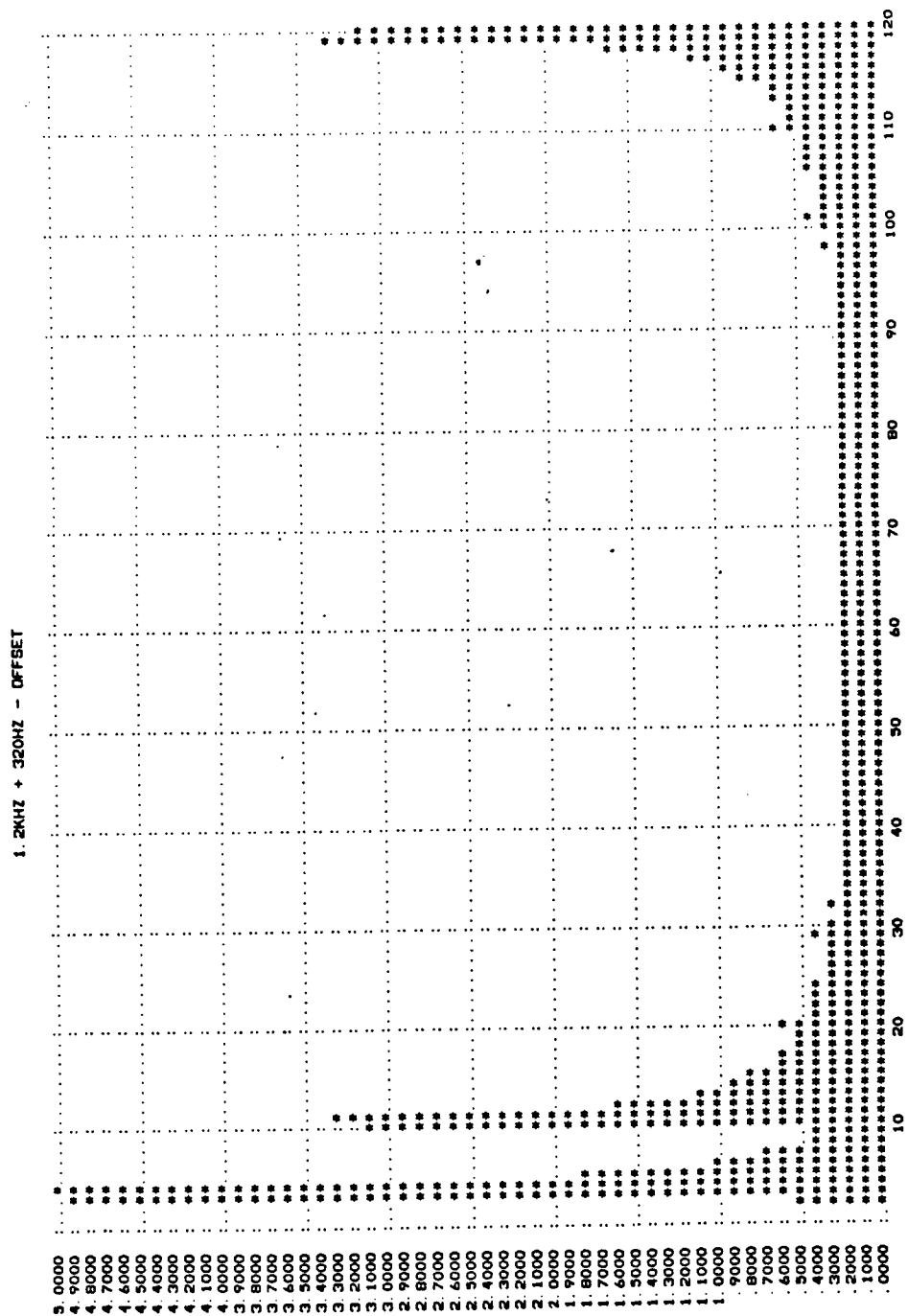


FIGURE 2.17. Fourier Transform of High-pass Filtered Signal -
Less the DC Offset.

spikes are very prominent. Because the FFT was taken on a data set with 128 points, and the sample rate of the ADC was 16,000 samples/sec, each vertical column on the frequency plot represents an incremental increase of $16,000/128$ or 125 Hz. Notice that there is no entry on the plot for the zero frequency. This zero frequency or offset was removed in the first filtering operation. Also, notice that the first spike occurs at a frequency of 2.5×125 Hz or 313 Hz, and the second at a frequency of 9.5×125 Hz or 1,188 Hz. This agrees very closely with the measured values of the frequencies.

Assuming for now that the higher frequency is noise riding on the lower frequency, elimination of the higher frequency would be desirable. By applying an ideal low-pass filter to this signal and then the inverse FFT operation, the waveform shown in Figure 2.18 is produced. Although some distortion is visibly present, by using the two peaks of the waveform and the 16,000 samples/sec sample rate, the frequency can be calculated as 290 Hz. Also, the amplitude is calculated to be 1.58 volts peak-to-peak.

Now assuming that the low frequency signal is rumble (for example noise in music from phonograph motor) and the high frequency is the signal of interest, an ideal high-pass filter could be used. Figure 2.19 shows the result of this operation. A frequency of 1,180 Hz is calculated from 16,000 samples/sec divided by one cycle per 13.6 samples. An

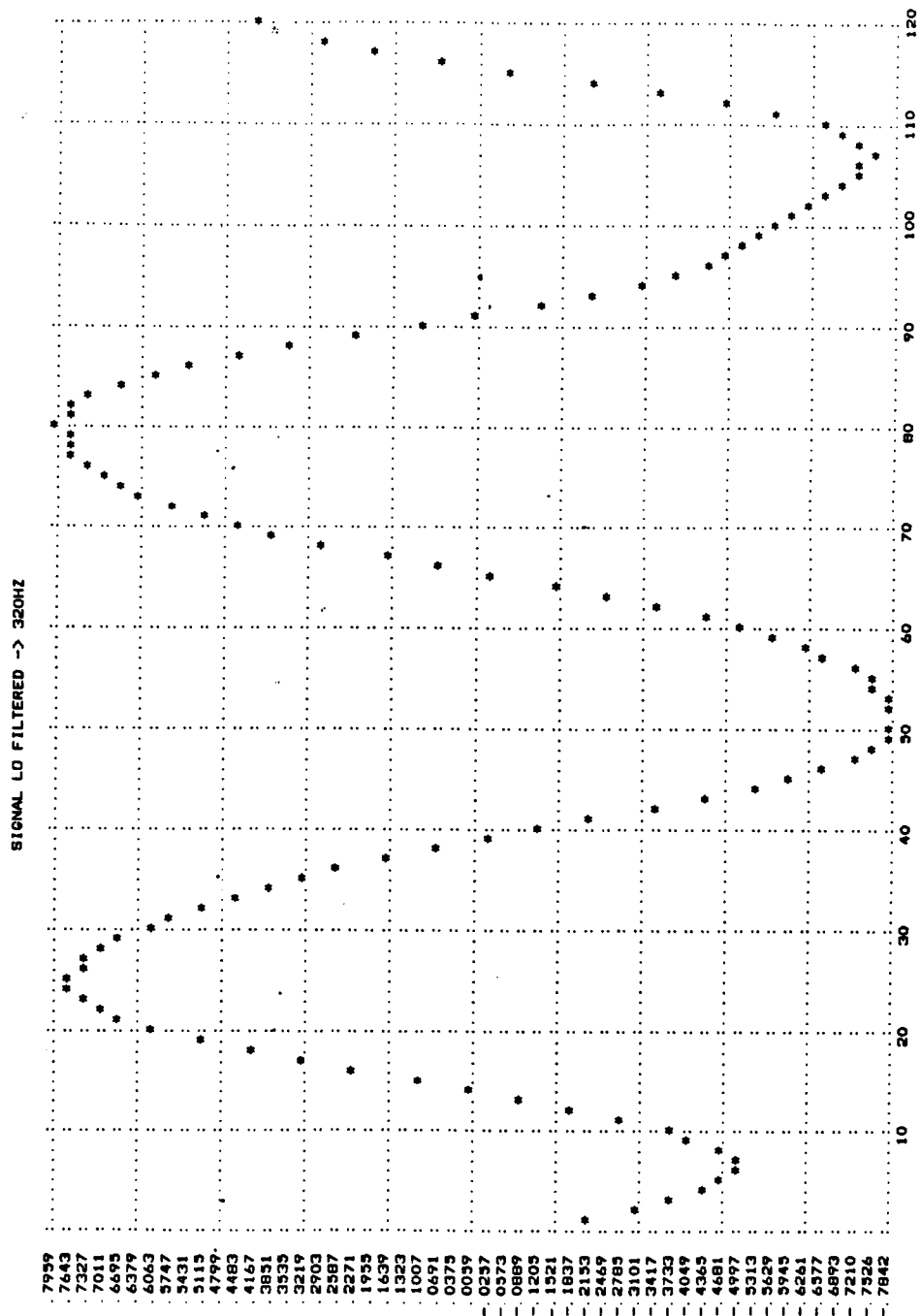


FIGURE 2.18. Low-pass Filtered Signal - Results in a 320 Hz
Signal less any DC Offset.

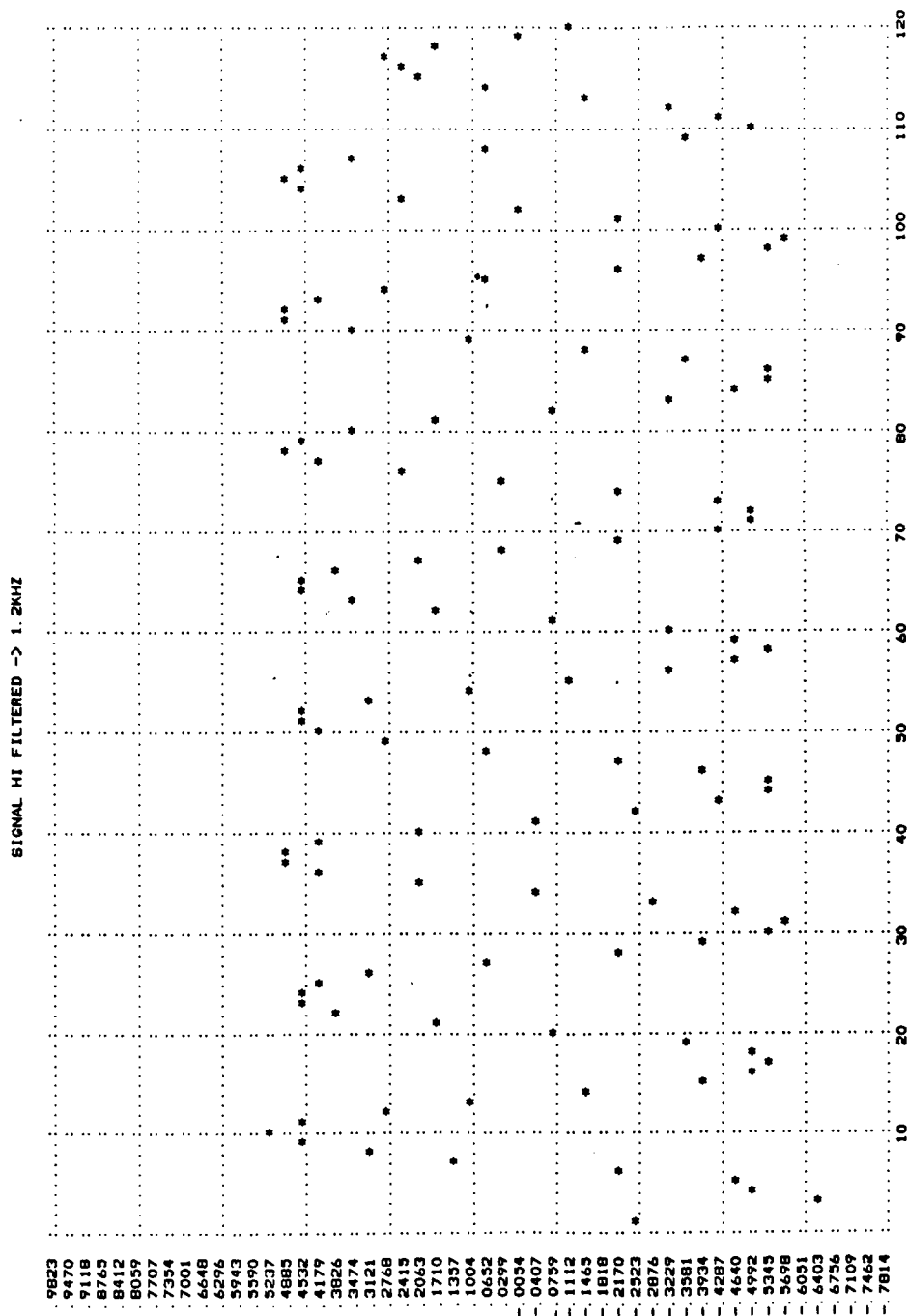


FIGURE 2.19. High-pass Filtered Signal - Results in a 1.2 kHz
Signal less any DC Offset.

amplitude of 1.1 volts can be measured from the figure. Because the amplitudes for the component signals added together when both were simultaneously at their maximum, a wave with peak-to-peak amplitude of 5.36 volts was produced. This compares closely with the peak-to-peak amplitude of 5.0 volts for the original waveform.

CHAPTER III

MAINTENANCE OF THE TI 990 ADC AND DAC

Texas Instruments supplies software tools to use for the calibration and maintenance of the entire TI 990 Data System. These software tools are designed to run under a diagnostics operating system called Standard Diagnostic Operational Control System (DOCS). The programs that run under DOCS support all of the hardware that is supplied with the TI 990 System.

DOCS is available in four different configurations. MAXI DOCS is the version that will be used in the calibration of the ADC and DAC modules. The two routines that run under the DOCS system are the program ADCHK and DACHK, which check the Analog-to-Digital and the Digital-to-Analog converter, respectively. These programs can be invoked to calibrate the modules, or alternately, to verify the correct operation of the devices.

Some additional equipment is necessary for the calibration of the DAC and ADC modules. Both modules require the use of a modified CRU extender board and a small screwdriver. The extender board allows the user to adjust the potentiometers on the module using the screwdriver during the calibration phase. Neither of these are necessary for the verification phase.

For the Analog-to-Digital converter, the equipment

required is a DC Voltage Standard from Analogic (part number AN3100). A similar device that provides 0.0 to +10.24 volts DC with a resolution of 0.1 millivolt can be substituted.

Calibration of the Digital-to-Analog converter requires a five and one-half digit digital voltmeter such as the Fluke 8800A digital voltmeter. The zero voltage adjustment can be performed using a digital voltmeter with less digits, but the gain adjustment requires the accurate measurement of a large voltage, such as 10.23 volts.

Texas Instruments also recommends the use of a functional/test cable harness for each module. The features that these cables provided are included on the I/O panel that was constructed in conjunction with this thesis. A switch is included on the DAC test cable harness to allow either the current or voltage to be applied to the four test leads - one for each channel. Separate current and voltage connectors are provided on the I/O panel. The ADC test cable harness included a push button to manually activate the External Trigger, and a switch to select either +5.0 volts or the voltage standard (VSTD) to be applied to all the channels. Both functions are available on the I/O panel.

Procedure

Floppy disk drives were the load media for the DOCS system software used in this thesis. Running the test and

calibration software is performed as follows.

1. Boot the DOCS system on the floppy disk drive.
2. Load and configure the DOCS system.
3. Load the individual test/calibration program for the particular module.
4. Perform the test/diagnostic routine.

Step one above, booting the floppy disk drives, requires the operator to enter several values and commands into the front panel of the TI 990 system. A representation of the front panel is given in Figure 3.1. When one of the push buttons on the panel is pressed, there is an audible tone generated.

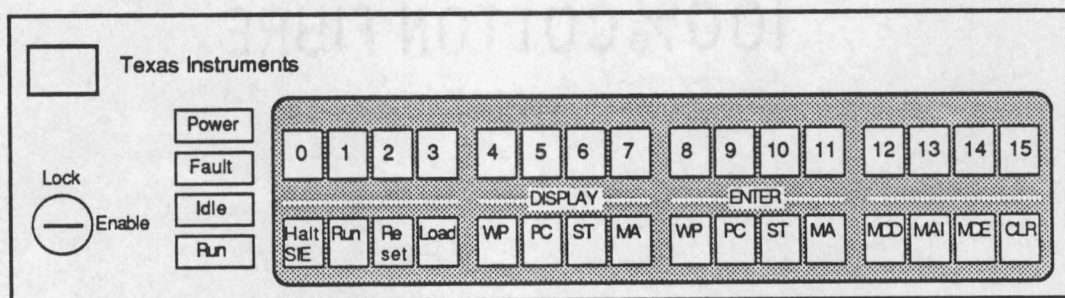


FIGURE 3.1. Front Panel of TI 990 System.

The procedure for booting the floppy disk drives is as follows.

1. Verify that no users are on the system.

2. Press the HALT/SIE button on the front panel.
3. Press the RESET button.
4. Press the CLR button.
5. Insert the write protected floppy disk "DOCSVOL4" into the left drive. (Note: The disk is write protected when the tab is removed.)
6. Set the data switches to >0082 (hexidecimal) or 0000 0000 1000 0010 (binary).
7. Press the MA button in the ENTER section.
8. Press the MDD button.
9. Set the data switches to >F820 (hexidecimal) or 1111 1000 0010 0000 (binary).
10. Press the MDE button.
11. Press the MAI button.
12. Set the data switches to >0800 (hexidecimal) or 0000 1000 0000 0000 (binary).
13. Press the MDE button.
14. Press the LOAD button.

After several seconds, all the LED's should become active on the front panel. When the lights quit blinking, the DOCS system has completed loading. Pressing the "Y" key on the terminal will inform the DOCS system of the terminal to use as the interactive device. Figure 3.2 is a listing of the screen contents during the initialization of the DOCS system.

```

** DYNAMIC DOCS                VERSION 339/84          TI 990/12
TERMINAL BASE ADDRESS - 0100
TERMINAL INTERRUPT LEVEL? DEF=0A - <RETURN>
ERROR MESSAGE DEVICE (0=ME,1=PRINTER) DEF=0 - <RETURN>
ARE SELECTED ERROR MESSAGE DEVICES OK? (DEF=1) -
<RETURN>

DOCS TYPE?
(0=MIN1 1=MAXI 2=BCSRUN 3=BCSEDT 4=DEBUG) DEF=1 -
<RETURN>
LOADING VERB MODULE MXVERB1
LOADING VERB MODULE MXVERB2
WRITE PROTECT DISK, HIT RETURN <RETURN>
-> DONE
LOADING VERB MODULE    > DONE

ENTER CONTROL OPTIONS SEPARATED BY COMMAS:
DEF=E,H,N - <RETURN>
LOAD TEST? (DEF=1) - 0
VERB? - .LD      (This will produce a directory listing)
FILE # OR NAME? -

```

FIGURE 3.2. Loading the DOCS System.

This listing ends with the DOCS system prompting for "File # or Name? -". At this point, either the Analog-to-Digital program's name (ADCHK) or the Digital-to-Analog program's name (DACHK) should be entered. Figure 3.3 is a sample listing of the ADCHK program's results, and Figure 3.4 is a listing of the DACHK program's results.

Analog-to-Digital Check Routine

The test of the ADC consists of seven parts which can be run as one test cycle, or as individual modules. Control of the test environment is achieved by the use of what the TI documentation calls "verbs". These "verbs" are actually

```

FILE # OR NAME? - DACHK
LOAD BIAS? DEF = 0047EE - <RETURN>

DACHK          DIGITAL-TO-ANALOG TEST FOR ANALOGIC VERSION
ENTER D/A CRU ADR (DEF=0020) - 0460

ENTER DISPLAY MODE (0=VOLTS, 1=MA) DEF=00 - <RETURN>
ENTER RANGE (0=5V 1=10V) DEF=1 - <RETURN>
ENTER # OF CHANNELS; DEF = 4(D) - <RETURN>
ENTER CODE (NB,OB, OR TC) DEF = TC - <RETURN>
      ( This selects Natural Binary, Offset Binary,
        or Two's Complement )
EXECUTE EA VERB? (DEF=1) - 0
VERB? -

```

FIGURE 3.3. DACHK Sample Listing.

```

FILE # OR NAME? - ADCHK
LOAD BIAS? DEF = 0047EE - <RETURN>

ADCHK          ANALOG-TO-DIGITAL TEST FOR ANALOGIC VERSION
ENTER A/D TOLERANCE (DEF=0001) - <RETURN>
ENTER PWR. FREQ (0=60HZ, 1=50HZ) (DEF=0) - <RETURN>
ENTER A/D CRU ADR (DEF=0020) - 0440
ENTER A/D INT. LEVEL (DEF=0F) - B
ENTER RANGE (0=5V, 1=10V) DEF=01 - <RETURN>
ENTER # OF CHAN; DEF=64(D) - <RETURN>
ENTER CODE (NB,OB, OR TC) DEF=TC - <RETURN>
EXECUTE EA VERB ? (DEF=1) - 0

VERB? -

```

FIGURE 3.4. ADCHK Sample Listing.

keywords to the DOCS program to perform a certain action. The possible verbs for the ADCHK routine are as follows:

- E1 - This performs a functionality test of the ADC module and ensures that all the control bits in the CRU register can be set and reset.
- E2 - Requests that the operator apply two unique voltages on two different channels, then verifies that the module can read both and obtain the correct conversion.
- E3 - Performs an even/odd channel test by attaching all the even channels and odd channels to two unique voltages, then reads all the channels to check for correct operation.
- E4, E5, and E6 - Apply maximum, midrange, and minimum voltage, respectively, to all channels and verify that all conversions are correct.
- E7 - Check that the ADC can perform 20,000 conversions in one second. The results of the test are conveyed to the operator; no measurement is required.
- DA - Continuously display the voltage at the specified channel on the terminal.
- RC - Take one sample from each channel and display the results.
- CB - The calibration routine. First set the ADC input

(any channel) for zero offset. Then adjust the gain for correct maximum value.

The verbs L1 through L7 will execute the specific test number E1 through E7 in a continuous loop. To exit the loop, the user should type the "@" key.

The calibration verb "CB" requires a few additional observations. All of this discussion is for a ADC board configured for a range of +10.24 volts. When the verb is first executed, the operator is prompted for a tolerance value between zero and 20, a channel number, and whether to perform an offset or gain calibration. The offset calibration requires an analog voltage halfway between 0.000 and 0.005, and the gain requires a voltage halfway between +10.230 and +10.235 volts. Notice that the analog voltage applied is centered on a transition point for the ADC. The routine then converts 100 samples and checks for a distribution that is within the tolerance amount supplied by the operator. If the routine encounters a conversion value other than one of the two values on either side of the transition point, the routine prints this value out to the screen. If only the two values that bracket the transition point are seen, then the routine prints the percentage distribution on the terminal. This allows the operator to adjust the potentiometer on the ADC module with "hands-off" the terminal.

Digital-to-Analog Check Routine

Since the DAC module on the system can be configured such that each channel has a unique range, this routine should be executed once for each configuration. No changes should be made to the channels that do not match the parameters currently selected by the software.

There are four verbs to use in testing the DAC module. They are as follows:

E1 - A functionality check performed in three parts. A check that all channels can output full scale is performed in Part One. Part Two checks for minimum voltage on all channels. A test of channel independence is performed in Part Three. Each individual channel in turn is driven to the maximum voltage while the others are at the minimum.

E2 - Ripples a logic one through a field of all zeroes. Channel output is verified to match a number on the screen with a voltage meter.

E3 - Ripples a logic zero through a field of all ones.

CB - Calibration routine for offset and gain adjustments.

The calibration routine first drives all the channels to a value of zero volts. The operator then adjusts the offset potentiometer for each channel on the module so that the meter reads zero volts. Then the routine drives all the channels to

a maximum positive voltage, and the operator adjusts the gain potentiometer until the voltmeter reads the maximum voltage.

Results

Both boards were very difficult to adjust without the extender board that allowed for the potentiometers on the module to be adjusted while the test was running. Instead, one needed to remove the module from the backplane at each step, make an adjustment on the potentiometer, and after returning the module to its slot, check the results of the adjustment. Because a board should never be removed from the backplane while the power is on, the system was shutdown and restarted for each adjustment. The direction and number of turns that each potentiometer was adjusted were recorded and used as a guide in the subsequent adjustments. An extender board should be considered a must when the next calibration is performed.

Also, the voltage source that was used to calibrate the ADC module was much too sensitive to changes, making it very difficult to adjust to the desired voltage. A less sensitive voltage source would make this process easier.

Initially, the DAC outputs were to be used as the voltage source to drive the ADC inputs during the calibration process, but the DOCS routines all perform a software RESET when started. During this software reset, the DAC output

voltages all change to the most negative voltage resulting in this approach being invalid. If the extender board was available then the calibration of the ADC's could be performed under a program written in FORTRAN that would allow the DAC output to be used as the voltage reference. By using the DAC outputs as the voltage source, the reference voltage could be digitally specified to a resolution of ± 1 LSB.

CHAPTER IV

CONCLUSIONS

This thesis has explored the use of the TI 990 system in a digital signal processing (DSP) course or as an additional laboratory facility to the Analog/Digital converter course. FORTRAN was presented as the language of use on the system, and through the use of the ISA Extension Subroutines furnished with the system, a full utilization of the ADC and DAC modules was possible. Several example laboratory projects were developed in the DSP field and one project explored the characterization of a separate single chip ADC device.

Results

The FORTRAN language provided a very good computational tool for the sample projects. Besides being easy to work with, the language executed reasonably fast and provided good results on most of the labs.

The absence of a graphics output device to display the many waveforms generated by the labs was initially thought to be a drawback, but in retrospect, this demanded a more thorough understanding of the DAC system to output these results to the oscilloscope. Several subroutines were developed to output the results of the labs onto the printer as either tabular data or as a graphical chart.

One noticed problem occurred only occasionally during the high speed sampling of a signal using an ADC channel. There would be a pronounced time skew of about 23 μ secs in the waveform that was shown on the oscilloscope. Figure 2.1 on page 22 shows a representation of this skew. One assumes that the TI operating system has an internal clock that interrupts 60 times per second which produces this skew. The interrupt would take priority over the job running, and while the system serviced the very short interrupt, the input signal would not be sampled at the next appropriate interval. This same interrupt appears on the oscilloscope during output of the waveform as a particularly bright dot on the display.

When the program is continuously displaying the waveform on the oscilloscope, the system is loaded very heavily. Even though the display is occurring much faster than necessary for no flicker to be apparent in the display, there is no way to pace the display of the waveform. The shortest duration that the program will pause using one of the supplied FORTRAN subroutines is for one second. If this pause could be shortened to several milliseconds, then another program could execute while the pause is occurring.

As was mentioned in Chapter I, the user can sample the ADC channel at two possible rates, depending on the size of the array used in the subroutine call. The ADC module provides a trigger input that could be used to control the

rate at which it samples. By utilizing this feature, the user could adjust a signal generator's output to control the sample rate of the ADC's channels. Because this trigger feature is only useful if used in conjunction with the interrupt line on the ADC, it cannot be used with the FORTRAN support subroutines.

Future Developments

Several of the drawbacks of the system mentioned above could be overcome by the development of a device driver for the ADC and DAC modules. This driver would significantly reduce the loading on the system during the input and output of analog data from the system by not requiring that the device be driven continuously. Also, since it would be coded in assembly language, the driver should be capable of much faster speeds than are available using the FORTRAN support routines.

DSR's (Device Service Routines) provide the interface between peripheral devices and tasks running on the 990 computer [7]. The TI Systems Programming Guide provides an excellent reference for writing a DSR in Chapter 5. This enhancement would provide much greater functionality to the already impressive capabilities of the TI 990 system.

LIST OF REFERENCES

LIST OF REFERENCES

1. P. Mock, Digital Processors Up Efficiency, Electronics Week, McGraw-Hill, p. 71, November 5, 1984.
2. CMOS Databook, National Semiconductor, Catalog Number 62-1375, page 3-21, 1977.
3. Linear Databook, National Semiconductor, Santa Clara, California, pp. 8-28 to 8-47, 1982.
4. R. C. Gonzalez, P. Wintz, Digital Image Processing, Addison-Wesley, Reading, Massachusetts, page 80, 1977.
5. R. C. Gonzalez, P. Wintz, Digital Image Processing, Addison-Wesley, Reading, Massachusetts, pp. 140-148, 1977.
6. Chi-Tsong Chen, One Dimensional Digital Signal Processing, Marcel Dekker, New York, page 102, 1979.
7. TI Systems Programming Guide, Part Number 946250-9705*G, Texas Instruments, Chapter 5 and 6, 1982.
8. Model 990 Computer Analog/Digital Converter Module Installation and Operation Manual, TI Part Number 944774-9701, Texas Instruments, Dallas, Texas, July 1977.
9. FORTRAN 78 Reference Manual, TI Part Number 2268681-9701, Texas Instruments, Dallas, Texas, October 1982.
10. FORTRAN 78 ISA Extensions Manual, TI Part Number 2268696-9701*A, Texas Instruments, Dallas, Texas, December 15, 1983.
11. FORTRAN 78 Programmers Guide, TI Part Number 2268679-9701*C, Texas Instruments, Dallas, Texas, December 15, 1983.
12. General Diagnostic Information Manual, Volume 1, TI Part Number 945400-9701*M, page 3-6, Texas Instruments, Dallas, Texas, July 1984.
13. ROM Loader, TI Part Number 2270534-9701*B, Texas Instruments, page 2-4, August 1984.

14. Unit Diagnostics Handbook, Volume 5, TI Part Number 945400-9705*D, Texas Instruments, DACHK Chapter, Dallas, Texas, August 1, 1982.
13. Unit Diagnostics Handbook, Volume 5, TI Part Number 945400-9705*D, Texas Instruments, ADCHK Chapter, Dallas, Texas, August 1, 1982.

APPENDIXES

APPENDIX A

TUTORIAL ON USE OF THE 990 SYSTEM

TUTORIAL ON TI 990 FORTRAN

The FORTRAN language that is available with the TI 990 system conforms to ANSI standard X3.9-1978 (commonly called FORTRAN-77) which was formulated in 1978. There are several extensions to this core standard that are available in the language and also in subroutine libraries that are furnished with FORTRAN. First, the directory structure is examined, then the compiling, linking, and execution of the program is discussed.

Directory Structure

The directory structure that is used in the TI 990 system lends itself very well to the segmentation of source, object, and task images. Source code can be further subdivided into main programs and subroutine libraries. Figure A.1 depicts the structure suggested by the TI documentation. MAIN is the segment where the users directory starts.

Notice that the directory MAIN contains the subdirectories DIRLAB and DIRLIB, and the files FILES1. When a listing of the contents of a directory are requested by the command "LD", the files are listed first, and then the subdirectories within that directory are listed. In turn, the subdirectories DIRLAB and DIRLIB both contain further subdirectories and files as shown in Figure A.1. Therefore,

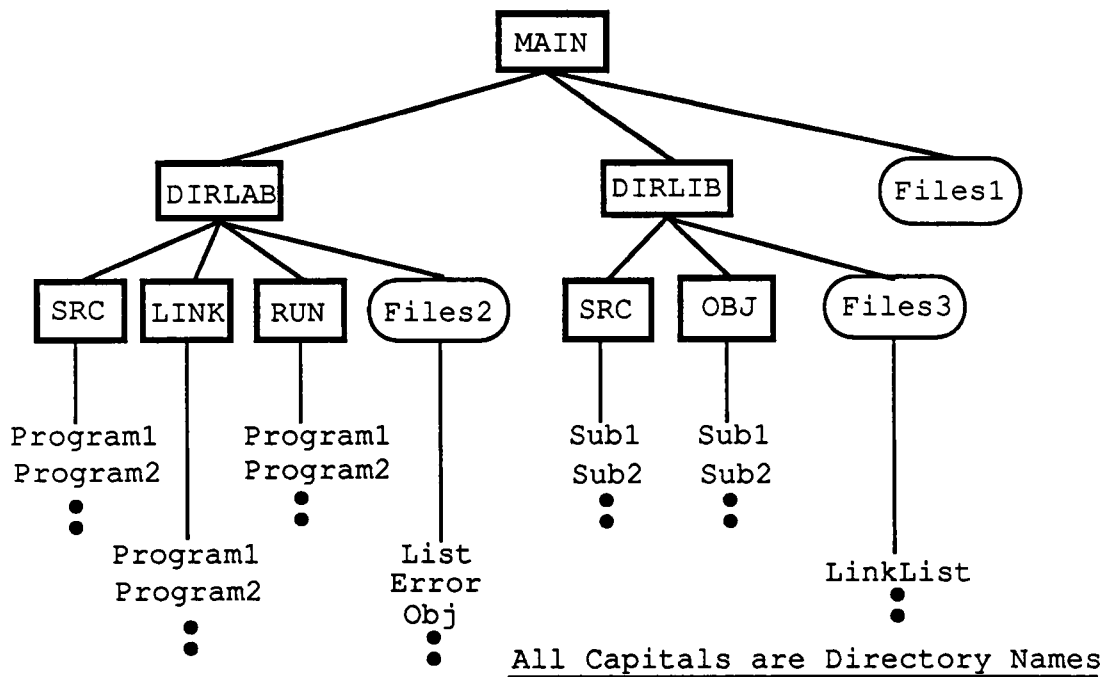


FIGURE A.1. TI 990 Directory Structure.

all subroutines can be stored in a commonly referenced area, known to the operating system as MAIN.DIRLIB.SRC for the source code, and MAIN.DIRLIB.OBJ for the object module. This name that is used to access files is called the PATHNAME and can be up to 79 characters in length.

The directory MAIN is actually a short name for the full name that specifies the beginning of the users directory. This long name, for example, is of the form :

VOL1.CLASS.CLS010

Short names are created by the use of the AS command to the operating system. This command assigns a short name or

SYNONYM to a long complete name. Synonyms remain in effect from session to session, or until modified by the user. Figure A.2 gives an example of the AS command. In this and all following screen displays, the text typed by the user is underlined.

To lessen the typing required during the compilation and linking processes, one would assign a SYNONYM to the two directories DIRLAB and DIRLIB. A suggested SYNONYM for VOL1.CLASS.CLS010.DIRLAB would be LABS, and for VOL1.CLASS.CLS010.DIRLIB would be LIBS.

```
ASSIGN SYNONYM VALUE
  SYNONYM : LIBS
  VALUE : VOL1.CLASS.CLS010.DIRLIB
```

FIGURE A.2. Assign SYNONYM Command.

Compilation

Once the source code has been written using the editor on the system, the program is ready to be compiled. The command XF78 invokes the FORTRAN-78 compiler and an example screen is shown in Figure A.3.

Notice in Figure A.3 that the compile listing, object module, and error listing are all stored as files in the directory LABS. An additional directory for these items was not deemed necessary; hence, these listings are only valid for the last compile that occurred. If the compile process

```
EXECUTE FORTRAN-78 COMPILER <VERSION 1.3.0 83334>
SOURCE ACCESS NAME : LABS.SRC.PROGRAM1
OBJECT ACCESS NAME : LABS.OBJ.PROGRAM1
LISTING ACCESS NAME : LABS.LIST
ERROR ACCESS NAME : LABS.ERROR
      OPTIONS : E
      PRINT WIDTH : <return>
      PAGE LENGTH : <return>
EXECUTION MODE (F/B) : E
```

FIGURE A.3. FORTRAN Compile Screen.

produces any errors, the compiler advises the operator on the screen and the error listing can be checked immediately to identify those errors.

Also, the OPTIONS selection in the figure has several possible entries. An "F" indicates that the FORTRAN source code is in "free format" and does not strictly adhere to the FORTRAN standards concerning which columns contain the statement number, continuation mark, etc.

Execution modes can be specified as either "F" or "B" for foreground or background, respectively. The foreground job is always the process that is active on the terminal. If "B" is chosen, another job (the background job) performs the compile process and the terminal is immediately free for other functions, such as editing.

If the compile procedure indicates that any errors are present, then the source code must be modified to correct them. Otherwise, if no errors are present, then an object

module file will be created in the directory using the pathname specified by OBJECT ACCESS NAME prompt.

Linking

Once the main program compiles without error, the program's object module is ready to be LINKED with any external subroutine object modules into a TASK. A CONTROL file must first be created that instructs the LINKER how to arrange these modules. This CONTROL file is created using the editor on the system. Figure A.4 shows a CONTROL file for a typical main program "PROGRAM1" that is linked with subroutines "SUB1" and "SUB2".

```
NOSYMT
FORMAT IMAGE,REPLACE
LIBRARY .FORT78.OSLOBJ
LIBRARY .FORT78.STLOBJ
LIBRARY .FORT78.ISAOBJ
TASK PROGRAM1

INCLUDE LABS.OBJ
INCLUDE LIB.OBJ.SUB1, LIB.OBJ.SUB2
END
```

FIGURE A.4. Example of LINK CONTROL File.

Much of the contents of the CONTROL file should not be varied, and requires a thorough understanding of the LINKING process to appreciate. The line that begins with "TASK" is important because this should match the name of the program that the user is attempting to create. Also, the "INCLUDE"

statement should contain the pathname to the main programs object module and any other subroutines that are called by the main program. Once this CONTROL file has been created, a suggested storage location is in the subdirectory LINK which is a co-directory of the source files. (See Figure A.1.)

The actual linking process is initiated by the XLE command, and a sample screen is shown in Figure A.5. In this figure, the LINKED OUTPUT ACCESS NAME is the actual runnable TASK image. Notice the position of the subdirectory RUN in Figure A.1. Because this process is usually time-consuming, it always runs as a background task.

If any errors are detected in the LINKING process, they are reported to the screen. The user can then review the specific error in the LINKLIST file.

```
EXECUTE LINKAGE EDITOR
      CONTROL ACCESS NAME : LABS.LINK.PROGRAM1
LINKED OUTPUT ACCESS NAME : LABS.RUN.PROGRAM1
      LISTING ACCESS NAME : LABS.LINKLIST
      PRINT WIDTH : <return>
      PAGE LENGTH : <return>
```

FIGURE A.5. Example of LINK Screen.

Program Execution

Before the TASK can be executed, a logical unit number (LUNO) must be assigned to the program's RUN file. This is

accomplished by the AL command shown in Figure A.6. By typing a RETURN in response to the LUNO prompt, the user is requesting the system to assign the next available LUNO to the program's RUN file. After typing RETURN at the last prompt, the system displays on the terminal the LUNO that was assigned to the program. The LUNO is expressed as a hexadecimal number, which can be identified by a ">" prefix.

This assignment of a LUNO must only be performed once for each program during a session. The assignment stays in effect for each succeeding cycle of editing, compiling, and linking. However, the assignment does not always stay in effect after the user logs off the system. The best procedure is to release all the LUNO's at the beginning of each session by use of the RAL command.

```
ASSIGN LUNO
      LUNO : <return>
ACCESS NAME : LABS.RUN.PROGRAM1
PROGRAM FILE : Y
```

FIGURE A.6. Assign LUNO Command.

Once the LUNO has been assigned to the RUN file, the user is ready to execute the program. Figure A.7 is an example of the use of the XFTE command.

Notice in Figure A.7 that the pathname is not specified for the TASK NAME OR ID prompt. Instead, use the name given in the TASK build CONTROL file shown in Figure A.1. Any

errors that occur during the execution of the program is displayed on the screen after exiting the program if the ERROR ACCESS NAME is specified as shown.

```
EXECUTE FORTRAN TASK IN FOREGROUND
PROGRAM FILE LUNO : >## (from previous AL command)
TASK NAME OR ID : PROGRAM1
ERROR ACCESS NAME : <return>
INPUT : <return>
OUTPUT : <return>
```

FIGURE A.7. Example of XFTF Command.

APPENDIX B
DIGITAL OSCILLOSCOPE PROGRAM

```

*** SCOPE
*** 18-SEP-85
*** R. SCHONHOFF

*** ROUTINE THAT READS DATA FROM THE ADC SYSTEM AND OUTPUTS THE
*** SAMPLED DATA THROUGH THE DAC SYSTEM TO A SCOPE FOR DISPLAY.
*** SHOULD SIMULATE A DIGITAL OSCILLOSCOPE, ALLOWING SELECTIVE
*** TRIGGERING AND DELAY LINE.

*** PROGRAM EXPECTS THAT ADC CHANNEL 1 IS THE ANALOG INPUT CHANNEL
*** ATTACH THE DAC CHANNELS AS FOLLOWS:
***     1 - NOT USED
***     2 - BLANKING ON OSCILLOSCOPE
***     3 - SWEEP ON OSCILLOSCOPE
***     4 - MAGNITUDE ON OSCILLOSCOPE

*** ARRAYS
      DIMENSION SAMPLE(1024), ADDR1(2,1024), SCOPE(1024), ADDR2(2,1024)
      DIMENSION RBUFF(256), CBUFF(256), RTBUFF(1024)
*** TYPE OF ALL VARIABLES
      COMPLEX CBUFF
      REAL RBUFF, TEMPR, TEMPR1, TEMPR2, MAX, RTBUFF
      REAL TRIGV, MAXV, MINV
      INTEGER*2 TRIGE, DELAY, POINT, TIMEB, SDOT, MINUS2
      INTEGER*2 SAMPLE, ADDR1, SCOPE, ADDR2, NUMSPT, ICHAR, TEMPI
      INTEGER*2 IERR, ISTAT, TYPE, ORDER, CUTOFF, LN, PLOTTYPE
      CHARACTER RETURN*1, SELECT*1, HEADER*30
      CHARACTER TRIGA*1, EDGE*7

*** FIRST, CLEAR THE SCREEN
50 CALL CLRCRT

      TRIGV = 0.0
      TRIGE = 1
      TIMEB = 1
      DELAY = 0
      NUMSPT = 1024
      CALL DASCL (3, 2, -2.0, MINUS2, IERR)

*** NEED TO SETUP SOME TABLES
      CALL PLWAIT
      DO 70, I=1, NUMSPT
          ADDR1(1, I) = >40 + >0400
          ADDR1(2, I) = 1
C          ADDR1(2, I) = ADCCHNL
70 CONTINUE
*** NOW, GO DIRECTLY TO THE SAMPLE PART OF THE PROGRAM
      GOTO 200

*****
*** MAIN LOOP
*** THIS SECTION DISPLAYS THE DATA SAMPLED ONTO THE SCOPE
*** AND WAITS FOR INPUT FROM THE USER TO PERFORM ANOTHER
*** TASK
100 CALL CLRCRT
      EDGE = 'RISING '
      IF (TRIGE.EQ.0) EDGE = 'FALLING'

      DISPLAY (9, 110, LINE=3, POSITION=1) MAXV, MINV
110 FORMAT (' MAXIMUM VOLTAGE = ', F8.4, ' MINIMUM = ', F8.4)
      DISPLAY (9, 115, LINE=4, POSITION=1) TRIGV, EDGE

```

```

115  FORMAT (' TRIGGER VOLTAGE = ',F8.4,'      TRIGGER EDGE = ',A7)
      DISPLAY (9,120,LINE=5,POSITION=1) DELAY
120  FORMAT ('          DELAY = ',I4,' SAMPLES')

      DISPLAY (9,130,LINE=7,POSITION=1)
130  FORMAT (' (S)AMPLE, (T)RIGGER, (D)ELAY, TIME(8)ASE, ',
& 'G)UIT      ')
      ICHAR = 1
      CALL BUFIN (10,1,SELECT,ICHAR)

***  DISPLAY THE SAMPLED DATA ON THE SCOPE
150  CALL AOW (512,ADDR2,SCOPE,IERR)
      IF (IERR.NE.1) GOTO 750
***  CHECK THE CHANNEL NUMBER FOR USER INPUT
      ISTAT = IUNIT(10)
      IF (ISTAT.EQ.1) GOTO 150

      IF (SELECT.EQ.'S') GOTO 200
      IF (SELECT.EQ.'T') GOTO 400
      IF (SELECT.EQ.'D') GOTO 500
      IF (SELECT.EQ.'B') GOTO 300
      IF (SELECT.EQ.'G') GOTO 800
      GOTO 100

*****
***  SAMPLE
***  THIS SECTION SAMPLES THE ADC SYSTEM AND CONVERTS THE
***  SAMPLED DATA INTO REAL NUMBERS
200  CALL PLWAIT
205  CALL AIRDW (NUMSPT,ADDR1,SAMPLE,IERR)
      IF (IERR.NE.1) GOTO 700
***  CONVERT THE INTEGER SAMPLES INTO REAL NUMBERS.
***  OF COURSE, THE REAL NUMBER DEPENDS ON THE SETUP OF THE
***  ADC SYSTEM.
      DO 210, I=1,1024
          CALL ADSCL (3,2,SAMPLE(I),TEMPR,IERR)
          IF (IERR.NE.1) GOTO 720
          RTBUFF(I) = TEMPR
210  CONTINUE

***  NEED TO LOOK FOR THE TRIGGER VALUE (ONLY IN FIRST HALF OF 1024)
      PRIMED = 0
      SDOT = 0
      DO 230, I=1, 512
          IF (PRIMED.EQ.1 .AND. TRIGE.EQ.0) GOTO 215
          IF (PRIMED.EQ.1 .AND. TRIGE.EQ.1) GOTO 220
          IF (TRIGE.EQ.1 .AND. RTBUFF(I).LT.TRIGV) PRIMED = 1
          IF (TRIGE.EQ.0 .AND. RTBUFF(I).GT.TRIGV) PRIMED = 1
          GOTO 225
215  IF (RTBUFF(I).LE.TRIGV) SDOT = I
          IF (SDOT.NE.0 .AND. SDOT-DELAY.LE.0) GOTO 223
          GOTO 225
220  IF (RTBUFF(I).GE.TRIGV) SDOT = I
          IF (SDOT.NE.0 .AND. SDOT-DELAY.LE.0) GOTO 223
          GOTO 225
223  PRIMED = 0
          SDOT = 0
          GOTO 230
225  IF (SDOT.NE.0) GOTO 240
230  CONTINUE

```

```

232  DISPLAY (9,235,LINE=23,POSITION=1)
235  FORMAT ( ' MISSED TRIGGER ' )
      GOTO 205

***  FOUND THE TRIGGER IF HERE. TRIGGER SAMPLE IS IN ARRAY AT
***  LOCATION "SDOT"
240  DISPLAY (9,245,LINE=23,POSITION=1)
245  FORMAT ( '              ' )

      MAXV = -10.4
      MINV = 10.4
***  NEED TO SUBTRACT OFF THE NUMBER OF SAMPLES TO JUMP PER
***  ITERATION "TIMEB", AND THE "DELAY" AMOUNT.
      POINT = SDOT - TIMEB - DELAY
      DO 260, I=1,128
          POINT = POINT + TIMEB
          RBUFF(I) = RTBUFF(POINT)
          IF (RBUFF(I).GT.MAX) MAXV = RBUFF(I)
          IF (RBUFF(I).LT.MIN) MINV = RBUFF(I)
260  CONTINUE

***  NOW, PLACE THE DATA INTO A FORM TO BE DISPLAYED
      CALL PUTDAC(128,RBUFF,SCOPE,ADDR2,4,8,2)
***  GO BACK INTO THE ARRAY AND MAKE THE FIRST "DELAY" NUMBER OF
***  POINTS DISPLAY AT HALF-INTENSITY
      IF (DELAY.LE.0) GOTO 100
      DO 270, I=1,DELAY
          POINT = I*4
          SCOPE(POINT) = MINUS2
270  CONTINUE

***  RETURN TO THE MENU SELECTION AND DISPLAY PART OF PROGRAM
      GOTO 100

*****
***  TIMEBASE
***
***  THIS SECTION GETS THE TIMEBASE FROM THE USER
300  CALL CLRCRT-
      DISPLAY (9,310,LINE=10,POSITION=1)
310  FORMAT ( ' TIME PER TRACE ? ' )
      DISPLAY (9,320,LINE=12,POSITION=5)
320  FORMAT ( ' 1=8MS 2=16MS 3=32MS ' )
      ACCEPT (9,325,LINE=10,POSITION=21) TIMEB
325  FORMAT (I2)
      IF (TIMEB.LE.0 .OR. TIMEB.GT.3) GOTO 300
      GOTO 240

*****
***  TRIGGER
***
***  THIS SECTION SETS THE TRIGGER VOLTAGE AND WHETHER TO TRIGGER ON
***  THE RISING EDGE OR THE FALLING EDGE
400  CALL CLRCRT
      DISPLAY (9,410,LINE=10,POSITION=1)
410  FORMAT ( ' TRIGGER VOLTAGE ' )
      ACCEPT (9,415,LINE=10,POSITION=21) TRIGV
415  FORMAT (F8.4)
420  DISPLAY (9,430,LINE=12,POSITION=1)
430  FORMAT ( ' TRIGGER ON (R)ISING OR (F)ALLING EDGE ' )
      ACCEPT (9,435,LINE=12,POSITION=42) TRIGA
435  FORMAT (A1)

```

```

IF (TRIGA.NE. 'R' .AND. TRIGA.NE. 'F') GOTO 420
TRIGE = 1
IF (TRIGA.EQ. 'F') TRIGE = 0
GOTO 100

```

```

***   DELAY
***
***   SECTION SETS THE DELAY AMOUNT IN NUMBER OF SAMPLES
500   CALL CLRCRT
      DISPLAY (9,510,LINE=10,POSITION=1)
510   FORMAT (' TRIGGER DELAY (NUMBER OF SAMPLES) ')
      ACCEPT (9,520,LINE=10,POSITION=36) DELAY
520   FORMAT (I4)
      IF (DELAY.LT.0 .OR. DELAY.GT.128) GOTO 500
      GOTO 100

```

```

***   FASTFT
***
***   THIS IS THE SECTION THAT CALLS THE EXTERNAL SUBROUTINE "FASTFT"
***   TO PERFORM A FAST FOURIER TRANSFORM ON THE SAMPLED DATA
***   FIRST, PLACE DATA IN THE COMPLEX ARRAY
600   DO 610 I=1,NUMSPT
      CBUFF(I) = CMPLX(RBUFF(I),0.0)
610   CONTINUE
      DISPLAY (9,615,LINE=16)
615   FORMAT (' BEGIN FFT SUBPROGRAM ')
      LN = NINT( (ALOG(REAL(NUMSPT)))/(ALOG(REAL(2))) )
      DISPLAY (9,616,LINE=20) LN
616   FORMAT (I5)
      CALL FASTFT (CBUFF,LN,MAX)
***   NOW, PLACE THE MAGNITUDE OF THE RESULTS BACK IN THE ARRAY
618   DO 620, I=1,NUMSPT
      TEMPR1 = REALCX(CBUFF(I))
      TEMPR2 = AIMAG(CBUFF(I))
      TEMPR = SQRT((TEMPR1*TEMPR1) + (TEMPR2*TEMPR2) )
      RBUFF(I) = (TEMPR/MAX) * 5.0
620   CONTINUE
***   PLACE THE VALUES IN RBUFF INTO THE SCOPE BUFFER FOR DISPLAY
      CALL PUTDAC (NUMSPT,RBUFF,SCOPE,ADDR2,4,8,2)
***   LASTLY, SET A FLAG THAT WILL DETERMINE THE PLOT TYPE
***   FROM WHETHER WE NEED A FREQUENCY OR TIME PLOT
      PLOTTYPE = 2
      GOTO 100

```

```

C   THIS IS THE ERROR SECTION
700   DISPLAY (9,710,LINE=22) IERR
710   FORMAT (' ERROR NUMBER ',Z8,' OCCURRED IN CALL TO AIRDW')
      GOTO 800

720   DISPLAY (9,730,LINE=22) IERR
730   FORMAT (' ERROR NUMBER ',Z8,' OCCURRED IN CALL TO "ADSCL"')
      GOTO 800

750   DISPLAY (9,760,LINE=22) IERR
760   FORMAT (' ERROR NUMBER ',Z8,' OCCURRED IN CALL TO AOW')
      GOTO 800

```

```

770  DISPLAY (9,780,LINE=22) IERR
780  FORMAT ( ' ERROR NUMBER ',Z8, ' OCCURRED IN CALL TO "DASCL"' )
      GOTO 800

```

```

C    NOW, STOP THE PROGRAM
800  DISPLAY (9,810,LINE=23,POSITION=1)
810  FORMAT ( ' WAITING FOR <RETURN> TO TERMINATE ' )
      ACCEPT (9,820,LINE=23,POSITION=36) RETURN
820  FORMAT (A2)

```

```

      STOP
      END

```

```

***  START OFF WITH A CLEAN SCREEN AND A HEADER
SUBROUTINE CLRCRT
2000  DISPLAY (9,2010,LINE=1,POSITION=20,ERASE)
2010  FORMAT ( ' FAST FOURIER TRANSFORM ' )
      RETURN
      END

```

```

***  ROUTINE THAT CLEARS SCREEN AND THEN DISPLAYS A
***  PLEASE WAIT MESSAGE
SUBROUTINE PLWAIT
2100  CALL CLRCRT
      DISPLAY (9,2110,LINE=10,POSITION=20)
2110  FORMAT ( ' PLEASE WAIT .... ' )
      RETURN
      END

```

APPENDIX C
FOURIER SERIES PROGRAM

```

***  FSERIES
***  12-SEP-85
***  R. SCHONHOFF

***  ROUTINE THAT ALLOWS FOR INDIVIDUAL HARMONICS IN A FOURIER
***  SERIES TO BE INDIVIDUALLY CALCULATED AND DISPLAYED FOR
***  INSPECTION. THEN, IF DESIRED, THE WAVEFORM WILL BE ADDED TO
***  THE TOTAL WAVEFORM.

***  THE RESULTS OF THESE OPERATIONS CAN BE OUTPUT TO THE PRINTER
***  IN EITHER A PLOT OR AS A TABLE.

      DIMENSION SUM(256), EACH(256)
      DIMENSION BUFF(1024), ADDR(2,1024)
      REAL SUM,EACH,TEMPR, AMP,PHASE,TWOPI,CONST
      INTEGER*2 TEMPI,NUMSPT,BUFF,ADDR,ISTAT,ICHAR
      CHARACTER TYPE*1,SELECT*1,HEADER*30,SEL1*1

      TWOPI = 6*3.14159
      GOTO 50

***  COMES HERE ON A RESTART.  NEED TO ZERO THE SUM ARRAY
30    CALL PLWAIT
      DO 40, I=1,256
          SUM(I) = 0.0
40    CONTINUE

50    CALL CLRCRT
      DISPLAY (9,60,LINE=5)
      FORMAT (' NUMBER OF SAMPLES IN EACH HARMONIC ')
      ACCEPT (9,65,LINE=5,POSITION=38) NUMSPT
      FORMAT (I4)
      IF (NUMSPT.EQ.0) GOTO 50

***  ASK FOR THE CONSTANT (MAY BE ZERO )
      DISPLAY (9,70,LINE=6)
      FORMAT (' FOURIER SERIES CONSTANT ')
      ACCEPT (9,75,LINE=6,POSITION=26) CONST
      FORMAT (F8.5)
***  NOW NEED TO INSERT THIS CONSTANT IN SUM ARRAY
      CALL PLWAIT
      DO 80, I=1,NUMSPT
          SUM(I) = CONST
80    CONTINUE

***  IMMEDIATELY GO AND DISPLAY THIS CONSTANT VALUE
      GOTO 550

***  PUT UP THE MENU
***  THIS ALLOWS FOR THE DATA TO BE PRESENTED ON THE SCOPE
***  UNTIL THE USER MAKES A SELECTION
100   CALL CLRCRT
      DISPLAY (9,110,LINE=5,POSITION=1)
      FORMAT (' (N)EW, (A)DD LAST, (P)LOT, (T)ABLE, (R)ESTART, (Q)UIT ')
      ICHAR = 1
      CALL BUFIN (10,1,SELECT,ICHAR)

      TEMPI = NUMSPT*4
150   CALL AOW (TEMPI,ADDR,BUFF,IERR)
      IF (IERR.NE.1) GOTO 750

```

```

      ISTAT = IUNIT(10)
      IF (ISTAT.EQ.1) GOTO 150

      IF (SELECT.EQ.'N') GOTO 200
      IF (SELECT.EQ.'A') GOTO 300
      IF (SELECT.EQ.'P') GOTO 650
      IF (SELECT.EQ.'T') GOTO 600
      IF (SELECT.EQ.'R') GOTO 30
      IF (SELECT.EQ.'Q') GOTO 800
      GOTO 100

*****
***  NEW HARMONIC
***  ACCEPT THE INPUT OF THE PARAMETERS THAT SPECIFY A NEW HARMONIC
200  CALL CLRCRT
      DISPLAY (9,210,LINE=6)
210  FORMAT (' (S)INE OR (C)OSINE WAVE      ')
      ACCEPT (9,215,LINE=6,POSITION=26) TYPE
215  FORMAT (A1)
      IF ( (TYPE.NE.'S') .AND. (TYPE.NE.'C') ) GOTO 200

220  DISPLAY (9,230,LINE=7)
230  FORMAT (' AMPLITUDE                    ')
      ACCEPT (9,235,LINE=7,POSITION=12) AMP
235  FORMAT (F8.4)

240  DISPLAY (9,250,LINE=8)
250  FORMAT (' PHASE ANGLE (RADIANS)        ')
      ACCEPT (9,255,LINE=8,POSITION=24) PHASE
255  FORMAT (F8.4)

***  NOW CALCULATE THE VALUES FOR THE SPECIFIED HARMONIC
      CALL PLWAIT
260  DO 270, I=1,NUMSPT
          TEMPR = (TWOPI/REAL(NUMSPT)) * REAL(I)
          IF (TYPE.EQ.'C') EACH(I)=AMP*(COS(TEMPR*PHASE) )
          IF (TYPE.EQ.'S') EACH(I)=AMP*(SIN(TEMPR*PHASE) )
270  CONTINUE

***  NOW, PLACE THE VALUES CALCULATED IN THE BUFF ARRAY TO BE DISPLAYED
      CALL PUTDAC (NUMSPT,EACH,BUFF,ADDR,4,8,2)

***  ALL FINISHED, RETURN TO THE DISPLAY AND PROMPT SECTION
      GOTO 100

*****
***  ADD
***
***  THIS SECTION ADDS THE EACH ARRAY TO THE SUM ARRAY
500  CALL PLWAIT
      DO 510, I=1,NUMSPT
          SUM(I) = SUM(I)+EACH(I)
510  CONTINUE
***  NOW, PLACE THE SUM ARRAY INTO THE DISPLAY BUFFER
550  CALL PUTDAC(NUMSPT,SUM,BUFF,ADDR,4,8,2)
***  RETURN TO THE PROMPT AND DISPLAY SECTION
      GOTO 100

*****

```

```

*** PRINT
***
*** THIS OUTPUTS THE MAGNITUDE VALUES IN THE BUFF ARRAY
*** TO THE PRINTER
600 CALL CLRCRT
    DISPLAY (9,610,LINE=5)
610 FORMAT (' HEADER FOR DATA COLUMN ')
    ACCEPT (9,615,LINE=5,POSITION=25) HEADER
615 FORMAT (A30)
    DISPLAY (9,620,LINE=7)
620 FORMAT (' (T)OTAL WAVEFORM OR (H)ARMONIC ')
    ACCEPT (9,625,LINE=7,POSITION=36) SEL1
625 FORMAT (A1)

    CALL PLWAIT
    IF (SEL1.EQ. 'T') CALL PRTRBUF(11,SUM,NUMSPT,HEADER)
    IF (SEL1.EQ. 'H') CALL PRTRBUF (11,EACH,NUMSPT,HEADER)
    GOTO 100

*****
*** THIS SECTION PLOTS THE WAVEFORM ON THE PRINTER
630 CALL CLRCRT
    DISPLAY (9,660,LINE=5)
660 FORMAT (' HEADER FOR PLOT ')
    ACCEPT (9,665,LINE=5,POSITION=18) HEADER
665 FORMAT (A30)
    DISPLAY (9,670,LINE=7)
670 FORMAT (' (T)OTAL WAVEFORM OR (H)ARMONIC ')
    ACCEPT (9,675,LINE=7,POSITION=36) SEL1
675 FORMAT (A1)

    CALL PLWAIT
    IF (SEL1.EQ. 'T') CALL PRTPLT (11,SUM,NUMSPT,HEADER,1)
    IF (SEL1.EQ. 'H') CALL PRTPLT (11,EACH,NUMSPT,HEADER,1)
    GOTO 100

*****
C THIS IS THE ERROR SECTION

750 DISPLAY(9,760,LINE=22) IERR
760 FORMAT (' ERROR NUMBER ',Z8,' OCCURRED IN CALL TO AOW')
    GOTO 800

770 DISPLAY (9,780,LINE=22) IERR
780 FORMAT (' ERROR NUMBER ',Z8,' OCCURRED IN CALL TO "DASCL"')
    GOTO 800

*****
C NOW, STOP THE PROGRAM
800 DISPLAY (9,810,LINE=23,POSITION=1)
810 FORMAT (' WAITING FOR <RETURN> TO TERMINATE ')
    ACCEPT (9,820,LINE=23,POSITION=36) RETURN
820 FORMAT (A2)

    STOP
    END

```

```
***   START OFF WITH A CLEAN SCREEN AND A HEADER
SUBROUTINE CLRCRT
2000   DISPLAY (9,2010,LINE=1,POSITION=20,ERASE)
2010   FORMAT ( ' FOURIER SERIES SUMMER ' )
      RETURN
      END
```

```
***   ROUTINE THAT CLEARS SCREEN AND THEN DISPLAYS A
***   PLEASE WAIT MESSAGE
SUBROUTINE PLWAIT
2100   CALL CLRCRT
      DISPLAY (9,2110,LINE=10,POSITION=20)
2110   FORMAT ( ' PLEASE WAIT .... ' )
      RETURN
      END
```

APPENDIX D

ADC CHARACTERISTIC PROGRAM

```

*** ADCHAR
*** R. SCHONHOFF
*** 1-OCT-85

*** ROUTINE THAT WILL CHARACTERIZE A LESSER RESOLUTION ADC
*** SYSTEM THAT IS CONNECTED TO THE TI 990 ADC/DAC SYSTEM.

*** THIS PROGRAM WILL DRIVE THE DAC CHANNEL 0 WITH A SIMULATED
*** ANALOG VOLTAGE AND USE THE ADC CHANNELS 0 THROUGH 7 TO READ
*** IN THE TTL LOGIC VOLTAGE. ADC CHANNEL 8 IS ATTACHED TO THE
*** END OF CONVERSION SIGNAL FROM THE 0804 ADC. THE DAC CHANNEL 1
*** IS USED AS THE START CONVERSION LEVEL.

        DIMENSION ARRAY(258,3)

        INTEGER*2 ANALOG, STARTC, ENDCON
        INTEGER*2 IVAL, I, IERR, LOOPCNT, MAXLOOP, BYTE
        INTEGER*2 ARRAY, ILOW, SIXTH
        INTEGER*2 I1, I2, I3, I4, N1, N2, N3, N4, S1, S2, S3, S4, L1, L2, L3, L4
        INTEGER*2 I5, I6, N5, N6, S5, S6, L5, L6
        REAL RVAL

        CHARACTER SELECT*1, RETURN*1, HEADER*30
        CHARACTER*1 C1, C2, C3, C4, C5, C6
        COMMON ANALOG, STARTC, ENDCON, LOOPCNT, MAXLOOP

*** SOME SETUP
        MAXLOOP = 100
*** SET UP THE CHANNELS THAT WILL BE USED
        ANALOG = 4
        STARTC = 8
        ENDCON = 7

*** START WITH CLEAR SCREEN
100 CALL CLRCRT

*** PUT UP THE MENU
        DISPLAY (9,120,LINE=10)
120 FORMAT (' (S)INGLE VALUE, (T)EST, (P)RINT TABLE, (Q)UIT ')
        ACCEPT (9,125,LINE=10,POSITION=45) SELECT
125 FORMAT (A1)

        IF (SELECT.EQ.'S') GOTO 300
        IF (SELECT.EQ.'T') GOTO 400
        IF (SELECT.EQ.'P') GOTO 500
        IF (SELECT.EQ.'Q') GOTO 800
        GOTO 100

*** THIS SECTION ASKS THE USER FOR THE ANALOG VALUE
*** TO OUTPUT ON THE TI 990 DAC SYSTEM. THIS VALUES IS
*** THEN READ BACK AFTER BEING CONVERTED BY THE ADC SYSTEM
*** UNDER TEST. THE RESULTANT VALUE IS THEN DISPLAYED
300 CALL CLRCRT

        DISPLAY (9,310,LINE=10)
310 FORMAT (' VALUE TO OUTPUT TO THE TEST ADC ')
        ACCEPT (9,315,LINE=10,POSITION=34) RVAL
315 FORMAT (F8.4)

*** CONVERT THE REAL VALUE TO AN INTEGER VALUE

```

```

      CALL DASCL (3,2,RVAL,IVAL,IERR)

*** CALL THE SUBROUTINE THAT WILL HANDLE THE START CONVERSION
*** AND END-OF-CONVERSION PULSES. WILL RETURN THE OUTPUT
*** OF THE ADC SYSTEM UNDER TEST IN THE INTEGER "BYTE"
      CALL DOCONV(IVAL,BYTE)
*** CALCULATE THE VOLTAGE THAT IT REPRESENTS (.02 VOLTS = 5/256)
      RVAL = (.02)*REAL(BYTE)

*** CHECK IF THE ADC UNDER TEST "TIMED OUT" (I.E. NEVER SENT
*** BACK THE "END OF CONVERSION" SIGNAL.
      IF (LOOPCNT.LT.MAXLOOP) GOTO 330
      DISPLAY (9,320,LINE=12)
320   FORMAT (' ADC UNDER TEST NEVER SET "END OF CONVERSION" ')
      GOTO 340

*** DISPLAY BACK TO USER THE RESULTS
330   DISPLAY (9,332,LINE=12) BYTE
332   FORMAT (' RETURNED VALUE = ',I6)
      DISPLAY (9,335,LINE=14) RVAL
335   FORMAT (' VOLTAGE      = ',F6.3)

*** NOW WAIT FOR RETURN FROM USER BEFORE RETURNING TO MENU
340   DISPLAY (9,342,LINE=22)
342   FORMAT (' RETURN TO CONTINUE ')
      ACCEPT (9,345,LINE=22,POSITION=21) RETURN
345   FORMAT (A1)
      GOTO 100

*****
*** (T)EST THE ADC UNDER TEST
***
*** ROUTINE THAT OUTPUTS 1024 DIFFERENT ANALOG VOLTAGES
*** TO THE ADC UNDER TEST AND EXAMINES THE 256 DIFFERENT
*** VALUES THAT IT GENERATES FROM THE INPUTS
400   CALL PLWAIT

*** FIRST, NEED TO ZERO OUT "ARRAY"
*** NOTICE THAT ARRAY SIZE IS 258. BECAUSE 258/6 COLUMNS
*** EQUALS A EVEN NUMBER OF ROWS OF DATA
      DO 410, I=1,258
          ARRAY(I,1) = 0
          ARRAY(I,2) = 0
          ARRAY(I,3) = 0
410   CONTINUE

*** NOW, DETERMINE THE INTEGER VALUE THAT WILL PRODUCE AN OUTPUT
*** OF ZERO VOLTS ON THE DAC OUTPUTS. THIS WILL SERVE AS THE
*** STARTING VOLTAGE FOR 1024 ADDITIONAL ONES.
      CALL DASCL (3,2,0.0,ILOW,IERR)
      ILOW = ILOW -1

*** OUTPUT THE NEXT 1024 CONSECUTIVE 1024 VALUES ON THE DAC CHANNEL
*** AND GET THE BYTE BACK FROM THE ADC UNDER TEST
      DO 430, I=1,1024
          IVAL = ILOW + I
          CALL DOCONV (IVAL,BYTE)
*** SINCE THE RETURNED INTEGER VALUE WILL BE USED TO STORE
*** DATA INTO ARRAY'S, IT'S POSSIBLE VALUE OF ZERO MUST BE
*** CHANGED TO A VALUE OF ONE.
          BYTE = BYTE + 1

```

```

*** FIRST ELEMENT IN ARRAY TELLS HOW MANY TEST OUTPUTS WHERE
*** CHARACTERIZED INTO THIS PARTICULAR "ONE OF 256" LEVELS.
      ARRAY(BYTE,1) = ARRAY(BYTE,1) + 1
*** NOW PLACE THE LARGEST AND SMALLEST VALUES
      IF (ARRAY(BYTE,2).EQ.0) ARRAY(BYTE,2) = I
      IF (ARRAY(BYTE,3).LT.I) ARRAY(BYTE,3) = I
430  CONTINUE
*** ALL DONE, SO RETURN
      GOTO 100

```

```

*** (P)RINT TABLE
***
*** ROUTINE THAT PRINTS THE VALUES THAT WERE SAVED IN THE
*** (T)EST ROUTINE.
500  CALL CLRCRT

```

```

      DISPLAY (9,510,LINE=10)
510  FORMAT (' HEADER FOR TABLE ')
      ACCEPT (9,515,LINE=10,POSITION=21) HEADER
515  FORMAT (A30)
      CALL PLWAIT

```

```

*** PRINT THE HEADER
      WRITE (11,530) HEADER
530  FORMAT ('1',T50,A30)
      WRITE (11,532)
532  FORMAT (' ')
      WRITE (11,535)
535  FORMAT (' ',6('CODE # SMALL LARGE '))
      WRITE (11,537)
537  FORMAT (' ')

```

```

*** PRINT THE TABLE
      SIXTH = 43
      DO 550, I=1, SIXTH
          C1 = ' '
          C2 = ' '
          C3 = ' '
          C4 = ' '
          C5 = ' '
          C6 = ' '
          I1 = I
          I2 = I+SIXTH
          I3 = I+(2*SIXTH)
          I4 = I+(3*SIXTH)
          I5 = I+(4*SIXTH)
          I6 = I+(5*SIXTH)
          N1 = ARRAY(I1,1)
          N2 = ARRAY(I2,1)
          N3 = ARRAY(I3,1)
          N4 = ARRAY(I4,1)
          N5 = ARRAY(I5,1)
          N6 = ARRAY(I6,1)
          S1 = ARRAY(I1,2)
          S2 = ARRAY(I2,2)
          S3 = ARRAY(I3,2)
          S4 = ARRAY(I4,2)
          S5 = ARRAY(I5,2)
          S6 = ARRAY(I6,2)

```

```

*** CHECK FOR DIFFERENTIAL NON-LINEARITY
      IF (I1.EQ.1) GOTO 538
      IF (S1.NE.L1+1) C1 = '*'
      IF (S2.NE.L2+1) C2 = '*'
      IF (S3.NE.L3+1) C3 = '*'
      IF (S4.NE.L4+1) C4 = '*'
      IF (S5.NE.L5+1) C5 = '*'
      IF (S6.NE.L6+1) C6 = '*'
538      L1 = ARRAY(I1,3)
      L2 = ARRAY(I2,3)
      L3 = ARRAY(I3,3)
      L4 = ARRAY(I4,3)
      L5 = ARRAY(I5,3)
      L6 = ARRAY(I6,3)
***      NEED TO DECREMENT THE POINTERS BY ONE BECAUSE
***      THE DATA STARTS AT CODE = 0
      I1 = I1-1
      I2 = I2-1
      I3 = I3-1
      I4 = I4-1
      I5 = I5-1
      I6 = I6-1
      WRITE(11,540) I1,N1,S1,L1,C1,I2,N2,S2,L2,C2,I3,N3,S3,L3,C3,
&      I4,N4,S4,L4,C4,I5,N5,S5,L5,C5,I6,N6,S6,L6,C6
540      FORMAT (' ',6(I3,I3,2X,I4,1X,I4,A1,4X))
550      CONTINUE
***      ALL DONE
***      DISPLAY SOME DASHES TO HELP DIFFERENTIATE THE COLUMNS
      WRITE(11,560)
560      FORMAT (' ',6(' ----- '))
      WRITE(11,565)
565      FORMAT (' * INDICATES NON-MONOTONICITY ')
      GOTO 100

*****
C      THIS IS THE ERROR SECTION

750      DISPLAY(9,760,LINE=22) IERR
760      FORMAT (' ERROR NUMBER ',Z8,' OCCURRED IN CALL TO AOW')
      GOTO 800

770      DISPLAY(9,780,LINE=22) IERR
780      FORMAT (' ERROR NUMBER ',Z8,' OCCURRED IN CALL TO "DASCL"')
      GOTO 800

*****
C      NOW, STOP THE PROGRAM
800      DISPLAY(9,810,LINE=23,POSITION=1)
810      FORMAT (' WAITING FOR <RETURN> TO TERMINATE ')
      ACCEPT(9,820,LINE=23,POSITION=36) RETURN
820      FORMAT (A2)

      STOP
      END

*****
***      SOME SUBROUTINES

***      SUBROUTINE THAT OUTPUTS THE VALUE TO THE DAC SYSTEM
      SUBROUTINE DOCONV(IVAL,BYTE)

```

```

        DIMENSION  ADDRI(2,1), ADDRO(2,1)

        INTEGER*2  ANALOG, STARTC, ENDCON
        INTEGER*2  ADDRI, ADDRO, IVAL, I, IERR, LOOPCNT, MAXLOOP, BYTE
        INTEGER*2  J, SUBBYTE, TOTBYTE
        REAL        RVAL

        COMMON      ANALOG, STARTC, ENDCON, LOOPCNT, MAXLOOP

***  SOME SETUP
        TOTBYTE = 0
        ADDRI(1,1) = >40 + >0400
        ADDRO(1,1) = >60 + >0400

        ADDRO(2,1) = ANALOG
***  OUTPUT THE ANALOG VOLTAGE
        CALL AOW (1, ADDRO, IVAL, IERR)

***  WANT TO DO THIS SECTION 10 TIMES SO AS TO AVERAGE OUT THE NOISE
        DO 1300, J=1, 10

***  NOW OUTPUT THE START CONVERSION LEVEL
***  NEEDS TO BE A ZERO TO ONE LOGIC TRANSITION
        CALL DASCL (3,2,4,9, IVAL, IERR)
        ADDRO(2,1) = STARTC
        CALL AOW (1, ADDRO, IVAL, IERR)

***  STAY IN THE LOOP LOOKING FOR THE END-OF-CONVERSION LEVEL
***  WHICH IS HIGH TO LOW LOGIC TRANSITION
        LOOPCNT = 0
        ADDRI(2,1) = ENDCON
1100    LOOPCNT = LOOPCNT + 1
        CALL AIRDW(1, ADDRI, IVAL, IERR)
        CALL ADSCL (3,2, IVAL, RVAL, IERR)
***  CHECK AND SEE IF WE HAVE TIMED OUT
        IF (LOOPCNT.GT. MAXLOOP) GOTO 1200
***  A LOGIC LEVEL OF LOW INDICATES THE CONVERSION HAS COMPLETED
        IF (RVAL.GT. 1.0) GOTO 1100
***  RETURN THE ADC SYSTEM TO THE RESET STATE
***  WHICH IS A LOW ON THE START CONVERSION LINE
***  ALSO, ENABLES THE OUTPUTS ON THE ADC UNDER TEST
1200    ADDRO(2,1) = STARTC
        CALL DASCL (3,2,0,1, IVAL, IERR)
        CALL AOW (1, ADDRO, IVAL, IERR)

***  READ THE TI ADC SYSTEM AND ASSEMBLE THE 8 VALUES INTO A BYTE
        CALL RDBYTE (0, SUBBYTE)
        TOTBYTE = TOTBYTE + SUBBYTE
1300    CONTINUE
***  NEED TO DIVIDE OUT THE 10 SAMPLES TO GET AVERAGE
        BYTE = TOTBYTE/10

        RETURN
        END

***  START OFF WITH A CLEAN SCREEN AND A HEADER
SUBROUTINE CLRCRT
2000  DISPLAY (9,2010, LINE=1, POSITION=20, ERASE)
2010  FORMAT (' ADC CHARACTERIZATION PROFILE')
        RETURN

```

END

```
*** ROUTINE THAT CLEARS SCREEN AND THEN DISPLAYS A
*** PLEASE WAIT MESSAGE
SUBROUTINE PLWAIT
2100  CALL CLRCRT
      DISPLAY (9,2110,LINE=10,POSITION=20)
2110  FORMAT (' PLEASE WAIT ....')
      RETURN
      END
```

SUBROUTINE RDBYTE (START, IVALUE)

*** SUBROUTINE THAT READS 8 VALUES FROM THE ADC SYSTEM STARTING
 *** AT THE CHANNEL NUMBER "START", AND TREATS THE RESULTS AS
 *** AS TTL LOGIC LEVELS. THESE LEVELS ARE THEN CONVERTED TO
 *** AN 8 BIT VALUE THAT IS RETURNED TO THE PROGRAM IN "IVALUE"
 *** AS AN INTEGER VALUE.

DIMENSION ADDR(2,8), SAMPLE(8)
 DIMENSION RVALUE(8)

INTEGER*2 START, IVALUE, I, ADDR, SAMPLE, CNT, IERR
 REAL RVALUE

IVALUE = 0
 CNT = START

*** NOW, SET UP FOR THE SUBROUTINE CALL THAT WILL READ
 *** THE CHANNELS SPECIFIED
 DO 100, I=1,8

ADDR(1,I) = >40 + >0400
 ADDR(2,I) = CNT
 CNT = CNT+1
 IF (CNT.EQ.4) CNT = 8

100 CONTINUE

*** PERFORM THE ADC READ
 CALL AIRDW(8, ADDR, SAMPLE, IERR)

*** NOW, CHANGE THE VALUES OBTAINED INTO REAL VALUES DEPENDING
 *** ON THE MODES THAT THE ADC SYSTEM IS SET FOR
 DO 200, I=1,8

CALL ADSCL(3,2, SAMPLE(I), RVALUE(I), IERR).

200 CONTINUE

*** NOW, CONVERT THE REAL VALUES AS LOGIC LEVELS, INTO AN
 *** INTEGER REPRESENTATION
 DO 300, I=1,8

IF (RVALUE(I).GT.1.0) IVALUE = IVALUE + (2**(I-1))

300 CONTINUE

*** COMPLETED
 RETURN
 END

APPENDIX E

FAST FOURIER TRANSFORM PROGRAM

```

***      TRANSF
***      18-SEP-85
***      R. SCHONHOFF

***      ROUTINE THAT READS DATA FROM THE ADC SYSTEM AND OUTPUTS THE
***      SAMPLED DATA THROUGH THE DAC SYSTEM TO A SCOPE FOR DISPLAY.
***      ALTERNATELY, THE DATA SAMPLED CAN HAVE A FAST FOURIER TRANSFORM
***      PERFORMED ON IT AND ALSO A FILTERING OPERATION.
***      AFTER THE FILTERING OPERATION, THE INVERSE FFT CAN BE TAKEN

***      PROGRAM EXPECTS THAT ADC CHANNEL 1 IS THE ANALOG INPUT CHANNEL
***      ATTACH THE DAC CHANNELS AS FOLLOWS:
***          1 - NOT USED
***          2 - BLANKING ON OSCILLOSCOPE
***          3 - SWEEP ON OSCILLOSCOPE
***          4 - MAGNITUDE ON OSCILLOSCOPE

***      ARRAYS
***          DIMENSION SAMPLE(256), ADDR1(2,256), SCOPE(1024), ADDR2(2,1024)
***          DIMENSION RBUFF(256), CBUFF(256)
***      TYPE OF ALL VARIABLES
***          COMPLEX      CBUFF
***          REAL         RBUFF, TEMPR, TEMPR1, TEMPR2, MAX
***          INTEGER*2    SAMPLE, ADDR1, SCOPE, ADDR2, NUMSPT, ICHAR, TEMP1
***          INTEGER*2    IERR, ISTAT, TYPE, ORDER, CUTOFF, LN, PLOTTYPE
***          CHARACTER    RETURN*1, SELECT*1, HEADER*30

***      FIRST, CLEAR THE SCREEN
50      CALL CLRCRT
***      ASK USER SOME QUESTIONS
***          DISPLAY (9,60,LINE=5)
60      FORMAT (' NUMBER OF SAMPLE (POWER OF 2)          ')
***          ACCEPT (9,65,LINE=5,POSITION=31) NUMSPT
65      FORMAT (I4)
***          IF (NUMSPT.EQ.0) GOTO 50
***          IF (NUMSPT.GT.256) GOTO 50

***      NEED TO SETUP SOME TABLES
***          CALL PLWAIT
***          DO 70, I=1,NUMSPT
***              ADDR1(1,I) = >40 + >0400
***              ADDR1(2,I) = 1
C          ADDR1(2,I) = ADCCHNL
70      CONTINUE
***          PLOTTYPE = 1
***      NOW, GO DIRECTLY TO THE SAMPLE PART OF THE PROGRAM
***          GOTO 200

*****
***      MAIN LOOP
***      THIS SECTION DISPLAYS THE DATA SAMPLED ONTO THE SCOPE
***      AND WAITS FOR INPUT FROM THE USER TO PERFORM ANOTHER
***      TASK
100     CALL CLRCRT
***          DISPLAY (9,110,LINE=5,POSITION=1)
110     FORMAT (' (S)AMPLE, (T)RANSFORM, (I)NVERSE FFT, (F)ILTER',
& ' ',(P)RINT,(R)ESTART,(Q)UIT          ')
***          ICHAR = 1
***          CALL BUFIN (10,1,SELECT, ICHAR)

```

```

*** DISPLAY THE SAMPLED DATA ON THE SCOPE
    TEMPI = NUMSPT*4
150  CALL AOW (TEMPI, ADDR2, SCOPE, IERR)
    IF (IERR.NE.1) GOTO 750
*** CHECK THE CHANNEL NUMBER FOR USER INPUT
    ISTAT = IUNIT(10)
    IF (ISTAT.EQ.1) GOTO 150

    IF (SELECT.EQ.'S') GOTO 200
    IF (SELECT.EQ.'T') GOTO 600
    IF (SELECT.EQ.'I') GOTO 650
    IF (SELECT.EQ.'F') GOTO 300
    IF (SELECT.EQ.'P') GOTO 500
    IF (SELECT.EQ.'R') GOTO 50
    IF (SELECT.EQ.'Q') GOTO 800
    GOTO 100

*****
*** SAMPLE
*** THIS SECTION SAMPLES THE ADC SYSTEM AND CONVERTS THE
*** SAMPLED DATA INTO REAL NUMBERS
200  CALL PLWAIT
    CALL AIRDW (NUMSPT, ADDR1, SAMPLE, IERR)
    IF (IERR.NE.1) GOTO 700
*** CONVERT THE INTEGER SAMPLES INTO REAL NUMBERS.
*** OF COURSE, THE REAL NUMBER DEPENDS ON THE SETUP OF THE
*** ADC SYSTEM.
    DO 210, I=1, NUMSPT
        CALL ADSCL (3, 2, SAMPLE(I), TEMPR, IERR)
        IF (IERR.NE.1) GOTO 720
        RBUFF(I) = TEMPR
210  CONTINUE

*** NOW, PLACE THE DATA INTO A FORM TO BE DISPLAYED
    CALL PUTDAC (NUMSPT, RBUFF, SCOPE, ADDR2, 4, 8, 2)
*** RETURN TO THE MENU SELECTION AND DISPLAY PART OF PROGRAM
    GOTO 100

*****
*** FILTER
*** THIS SECTION APPLIES FILTERS TO THE DATA IN THE "F" ARRAY
300  CALL CLRCRT
    DISPLAY (9, 305, LINE=10)
305  FORMAT (' 0=IDEAL HI/1=IDEAL LO/2=BUTTERW HI/3=BUTTERW LO',
    & '/4=EXP HI/5=EXP LO')
    DISPLAY (9, 307, LINE=12)
307  FORMAT (' FILTER TYPE : ')
    ACCEPT (9, 310, LINE=12, POSITION=15) TYPE
310  FORMAT (I3)
    DISPLAY (9, 315, LINE=14)
315  FORMAT (' FILTER ORDER ')
    ACCEPT (9, 320, LINE=14, POSITION=15) ORDER
320  FORMAT (I2)
    DISPLAY (9, 325, LINE=16)
325  FORMAT (' CUTOFF FREQUENCY ')
    ACCEPT (9, 330, LINE=16, POSITION=19) CUTOFF
330  FORMAT (I4)

    CALL FILTER (TYPE, CUTOFF, ORDER, CBUFF, NUMSPT)

```

```

*** NOW, PLACE THE COMPLEX ARRAY INTO THE REAL ARRAY
GOTO 618

*****
*** PRINT
***
*** THIS SECTION OUTPUTS DATA TO THE PRINTER. EITHER A PLOT
*** OF THE WAVEFORM OR A TABLE OF THE ACTUAL VALUES CAN BE PRINTED
500 CALL CLRCRT
    DISPLAY (9,510,LINE=8)
510 FORMAT (' (P)LOT OR (T)ABLE ? ')
    ACCEPT (9,515,LINE=8,POSITION=22) SELECT
515 FORMAT (A1)
*** BRANCH TO DIFFERENT TYPES OF PRINTOUTS
    IF (SELECT.EQ.'P') GOTO 530
*** HERE, THEN USER WANTS A TABLE
*** DOES USER WANT OUTPUT OF TIME OR FREQUENCY VALUES
    DISPLAY (9,520,LINE=10)
520 FORMAT (' (T)IME OR (F)REQUENCY WAVEFORM ')
    ACCEPT (9,525,LINE=10,POSITION=33) SEL2
525 FORMAT (A1)
    DISPLAY (9,530,LINE=12)
530 FORMAT (' HEADER FOR TABLE ')
    ACCEPT (9,535,LINE=12,POSITION=19) HEADER
535 FORMAT (A30)
    IF (SEL2.EQ.'T') CALL PRTRBUF(11,RBUFF,NUMSPT,HEADER)
    IF (SEL2.EQ.'F') CALL PRFCBUF(11,CBUFF,NUMSPT,HEADER)
    GOTO 100

*** HERE, THEN USER WANTS A PLOT
550 DISPLAY (9,560,LINE=12)
560 FORMAT (' HEADER FOR PLOT ')
    ACCEPT (9,565,LINE=12,POSITION=18) HEADER
565 FORMAT (A30)
    CALL PLWAIT
    DISPLAY (9,570,LINE=14)
570 FORMAT (' PLOT OF WAVEFORM CURRENTLY DISPLAYED ON SCOPE')
    CALL PRTPLOT (11,RBUFF,NUMSPT,HEADER,PLOTTYPE)
    GOTO 100

*****
*** FASTFT
***
*** THIS IS THE SECTION THAT CALLS THE EXTERNAL SUBROUTINE "FASTFT"
*** TO PERFORM A FAST FOURIER TRANSFORM ON THE SAMPLED DATA

*** FIRST, PLACE DATA IN THE COMPLEX ARRAY
600 DO 610 I=1,NUMSPT
    CBUFF(I) = CMPLX(RBUFF(I),0.0)
610 CONTINUE
    DISPLAY (9,615,LINE=16)
615 FORMAT (' BEGIN FFT SUBPROGRAM ')
    LN = NINT( (ALOG(REAL(NUMSPT)))/(ALOG(REAL(2))) )
    DISPLAY (9,616,LINE=20) LN
616 FORMAT (I5)
    CALL FASTFT (CBUFF,LN,MAX)
*** NOW, PLACE THE MAGNITUDE OF THE RESULTS BACK IN THE ARRAY
618 DO 620, I=1,NUMSPT
    TEMPR1 = REALX(CBUFF(I))
    TEMPR2 = AIMAG(CBUFF(I))
    TEMPR = SQRT((TEMPR1*TEMPR1) + (TEMPR2*TEMPR2) )

```

```

        RBUFF(I) = (TEMPR/MAX) * 5.0
620    CONTINUE
***    PLACE THE VALUES IN RBUFF INTO THE SCOPE BUFFER FOR DISPLAY
        CALL PUTDAC (NUMSPT, RBUFF, SCOPE, ADDR2, 4, 8, 2)
***    LASTLY, SET A FLAG THAT WILL DETERMINE THE PLOT TYPE
***    FROM WHETHER WE NEED A FREQUENCY OR TIME PLOT
        PLOTTYPE = 2
        GOTO 100

*****
***    INVERSE FFT
***
***    THIS SECTION TAKES THE INVERSE TRANSFORM
***    FIRST, TAKE THE COMPLEX CONJUGATE OF F(I)
650    DO 660, I=1, NUMSPT
        CBUFF(I) = CONJG(CBUFF(I))
660    CONTINUE
***    NOW CALL THE FFT ROUTINE
        CALL FASTFT(CBUFF, LN, MAX)
***    NOW STORE THE REAL PART BACK IN THE BUFF ARRAY
***    REMEMBER, MUST MULTIPLY BY NUMBER OF SAMPLES
        DO 670, I=1, NUMSPT
            RBUFF(I) = NUMSPT*REALCX(CBUFF(I))
670    CONTINUE
***    PLACE INTO THE SCOPE BUFFER FOR DISPLAY
        CALL PUTDAC (NUMSPT, RBUFF, SCOPE, ADDR2, 4, 8, 2)
***    SET A FLAG THAT DETERMINES WHETHER A TIME OR FREQUENCY
***    PLOT WILL BE PRINTED WHEN ASKED FOR.
        PLOTTYPE = 1
        GOTO 100

*****
C    THIS IS THE ERROR SECTION
700    DISPLAY (9, 710, LINE=22) IERR
710    FORMAT (' ERROR NUMBER ', Z8, ' OCCURRED IN CALL TO AIRDW')
        GOTO 800

720    DISPLAY (9, 730, LINE=22) IERR
730    FORMAT (' ERROR NUMBER ', Z8, ' OCCURRED IN CALL TO "ADSCL"')
        GOTO 800

750    DISPLAY (9, 760, LINE=22) IERR
760    FORMAT (' ERROR NUMBER ', Z8, ' OCCURRED IN CALL TO AOW')
        GOTO 800

770    DISPLAY (9, 780, LINE=22) IERR
780    FORMAT (' ERROR NUMBER ', Z8, ' OCCURRED IN CALL TO "DASCL"')
        GOTO 800

*****
C    NOW, STOP THE PROGRAM
800    DISPLAY (9, 810, LINE=23, POSITION=1)
810    FORMAT (' WAITING FOR <RETURN> TO TERMINATE ')
        ACCEPT (9, 820, LINE=23, POSITION=36) RETURN
820    FORMAT (A2)

        STOP
        END

```

```
***   START OFF WITH A CLEAN SCREEN AND A HEADER
SUBROUTINE CLRCRT
2000   DISPLAY (9,2010,LINE=1,POSITION=20,ERASE)
2010   FORMAT ( ' FAST FOURIER TRANSFORM ' )
      RETURN
      END
```

```
***   ROUTINE THAT CLEARS SCREEN AND THEN DISPLAYS A
***   PLEASE WAIT MESSAGE
SUBROUTINE PLWAIT
2100   CALL CLRCRT
      DISPLAY (9,2110,LINE=10,POSITION=20)
2110   FORMAT ( ' PLEASE WAIT .... ' )
      RETURN
      END
```

SUBROUTINE FILTER (TYPE, DO, ORDER, F, N)

*** SUBROUTINE THAT IMPLEMENTS FILTERS IN THE FREQUENCY DOMAIN

*** VARIABLES ARE AS FOLLOWS:

*** TYPE: 0=IDEAL HI PASS
*** 1=IDEAL LO PASS
*** 2=BUTTERWORTH HI PASS
*** 3=BUTTERWORTH LO PASS
*** 4=EXPONENTIAL HI PASS
*** 5=EXPONENTIAL LO PASS

*** DO: (0-ZERO) CUTOFF FREQUENCY FOR ABOVE FILTERS

*** ORDER: SOME OF THE ABOVE FILTERS CAN BE IMPLEMENTED
*** FOR DIFFERENT ORDERS

*** F: THE ARRAY OF REAL AND IMAGINARY PARTS

*** N: THE NUMBER OF SAMPLES IN ARRAY

DIMENSION F(256)
COMPLEX F, TEMPC
INTEGER*2 TYPE, DO, ORDER, N, I, TEMPI
REAL MULT

*** FIRST, BRANCH TO APPLICABLE SECTION OF CODE

TEMPI = TYPE+1
GOTO (100, 150, 200, 250, 300, 350), TEMPI
RETURN

*** IDEAL HI PASS FILTER

100 DO 110, I=1, N
IF (I.LE.DO) F(I) = (0.0, 0.0)
110 CONTINUE
RETURN

*** IDEAL LO PASS FILTER

150 DO 160, I=1, N
IF (I.GT.DO) F(I) = (0.0, 0.0)
160 CONTINUE
RETURN

*** BUTTERWORTH HI PASS FILTER

200 DO 210, I=1, N
MULT = 1.0/(1.0+((REAL(DO) / REAL(I))**(2*ORDER)))
F(I) = F(I) * MULT
210 CONTINUE
RETURN

*** BUTTERWORTH LO PASS FILTER

250 DO 260, I=1, N
MULT = 1.0/(1.0+((REAL(I)/REAL(DO))**(2*ORDER)))
F(I) = F(I) * MULT
260 CONTINUE
RETURN

*** EXPONENTIAL HI PASS FILTER

300 RETURN

*** EXPONENTIAL LO PASS FILTER

350 RETURN

END

APPENDIX F
COMMON SUBROUTINES

SUBROUTINE ADSCL (RANGE, CODE, INVALUE, NUMBER, IERR)

C SUBROUTINE THAT SCALES THE INCOMING INTEGER VALUES
C FROM THE ADC SUBSYSTEM "INVALUE", AND RETURNS A FLOATING POINT
C REPRESENTATION "NUMBER" TO THE CALLING PROGRAM.

C RANGE : 1 = 0 TO 10.24 VOLTS
C 2 = 0 TO 5.12 VOLTS
C 3 = -10.24 TO 10.24 VOLTS
C 4 = -5.12 TO 5.12 VOLTS
C CODE : 1 = OFFSET BINARY (BIPOLAR) OR
C STRAIGHT BINARY (UNIPOLAR)
C 2 = TWO'S COMPLEMENT

INTEGER*2 RANGE, CODE, INVALUE, IERR, TEMPI
REAL NUMBER, RES, FS

C FIRST, ASSUME THE CALL WILL FAIL

IERR = 0
GOTO (10, 20, 30, 40), RANGE
5 RETURN

10 RES = .0025
IF (CODE.EQ.2) RETURN
GOTO 100

20 RES = .00125
IF (CODE.EQ.2) RETURN
GOTO 100

30 RES = .005
FS = 10.235
GOTO 200

40 RES = .0025
FS = 5.1175
GOTO 200

C THIS SECTION IS FOR STRAIGHT BINARY CONVERSION FOR UNIPOLAR INPUTS

100 NUMBER = RES*INVALUE
GOTO 1000

C THIS SECTION IS FOR BIPOLAR. FIRST DETERMINE WHICH CODING

200 GOTO (230, 250), CODE
RETURN

230 NUMBER = (RES*INVALUE) - FS
GOTO 1000

250 TEMPI = INVALUE
IF (IBTEST(TEMPI, 11)) TEMPI = IOR(TEMPI, 42F000)
NUMBER = RES*TEMPI
GOTO 1000

```
*****  
C  COMES HERE FOR NO ERRORS RETURN  
  
1000  IERR = 1  
      RETURN  
  
      END
```

SUBROUTINE DASCL (RANGE, CODE, INUMBER, VALUE, IERR)

C SUBROUTINE THAT SCALES THE INCOMING FLOATING POINT
C NUMBER "INUMBER", AND CHANGES IT TO AN INTEGER REPRESENTATION
C "VALUE" THAT IS APPROPRIATE FOR THE RANGE AND CODE THAT
C THE DIGITAL TO ANALOG CONVERTER IS SET FOR.

C RANGE : 1 = 0 TO 10.24 VOLTS
C 2 = 0 TO 5.12 VOLTS
C 3 = -10.24 TO 10.24 VOLTS
C 4 = -5.12 TO 5.12 VOLTS
C CODE : 1 = OFFSET BINARY (BIPOLAR) OR
C STRAIGHT BINARY (UNIPOLAR)
C 2 = TWO'S COMPLEMENT

INTEGER*2 RANGE, CODE, VALUE, IERR, TEMPI
REAL INUMBER, RES, FS

C FIRST, ASSUME THE CALL WILL FAIL
IERR = 0

GOTO (10, 20, 30, 40), RANGE

5 RETURN

10 RES = .0025
IF (CODE.EQ.2) RETURN
IF (INUMBER.GT.10.235) INUMBER=10.235
IF (INUMBER.LT.0) INUMBER = 0
GOTO 100

20 RES = .00125
IF (CODE.EQ.2) RETURN
IF (INUMBER.GT.5.118) INUMBER=5.118
IF (INUMBER.LT.0) INUMBER = 0
GOTO 100

30 RES = .005
FS = 10.235
IF (INUMBER.GT.10.235) INUMBER=10.235
IF (INUMBER.LT.-10.235) INUMBER=-10.235
GOTO 200

40 RES = .0025
FS = 5.1175
IF (INUMBER.GT.5.118) INUMBER=5.118
IF (INUMBER.LT.-5.118) INUMBER=-5.118
GOTO 200

C THIS SECTION IS FOR STRAIGHT BINARY CONVERSION FOR UNIPOLAR INPUTS

100 VALUE = INUMBER/RES
GOTO 1000

C THIS SECTION IS FOR BIPOLAR. FIRST DETERMINE WHICH CODING

200 GOTO (230, 250), CODE
RETURN

```
230      VALUE = (INUMBER+FS)/RES  
      GOTO 1000
```

```
250      VALUE = INUMBER/RES  
      GOTO 1000
```

```
*****
```

```
C      COMES HERE FOR NO ERRORS RETURN
```

```
1000     IERR = 1  
      RETURN
```

```
      END
```

SUBROUTINE FASTFT (F, LN, MAX)

 *** COOLEEY, LEWIS, WELCH
 *** PROGRAM IS AN ADAPTATION OF THE ROUTINE FROM COOLEY, LEWIS
 *** AND WELCH. OBTAINED FROM "ONEE DIMENSIONAL DIGITAL SIGNAL
 *** PROCESSING" BY CHI-TSONG CHEN. MARCEL DEKKER, NEW YORK, PAGE 102
 *** 1979.

*** ROUTINE THAT PERFORMS A FAST FOURIER TRANSFORM
 *** THE ARRAY "F" IS A COMPLEX ARRAY WITH THE IMAGINARY
 *** PART SET EQUAL TO ZERO ON INPUT. "LN" IS THE POWER
 *** OF TWO THAT THE SAMPLE SET IS GIVEN FOR.

DIMENSION F(1024)
 COMPLEX F, U, W, T
 REAL MAX, TEMPR, TEMP1, TEMP2, PI
 INTEGER*2 NV2, NM1, I, J, K, L, LN, N, LE, LE1, IP

*** INITIALIZE SOME CONSTANTS
 PI = 3.14159
 N = 2**LN
 NV2 = N/2
 NM1 = N-1
 J=1

*** FIRST THE DATA IN THE ARRAY NEEDS TO BE REORDERED
 *** INTO A PATTERN THAT IS NEEDED BY THE FFT ROUTINE

```

DO 4 I=1, NM1
  IF (I.GE. J) GOTO 1
  T=F(J)
  F(J) = F(I)
  F(I) = T
  K=NV2
  IF (K.GE. J) GOTO 3
  J=J-K
  K=K/2
  GOTO 2
3  J=J+K
4  CONTINUE
  
```

*** NOW, BEGIN THE FFT USING THE SUCESSIVE DOUBLING ALGORITHM

```

DO 5 L=1, LN
  LE = 2**L
  LE1 = LE/2
  U = (1.0, 0.0)
  W=CMPLX( COS(PI/LE1), -SIN(PI/LE1) )
  DO 7 J=1, LE1
    DO 7 I=J, N, LE
      IP = I+LE1
      T = F(IP)*U
      F(IP) = F(I) - T
      F(I) = F(I) + T
7    CONTINUE
    U=U*W
5  CONTINUE
  
```

*** NOW DIVIDE THE RESULT BY "N"
 *** ALSO, DETERMINE THE MAXIMUM VALUE
 MAX = 0.0

```

DO 6 I=1,N
  F(I) = F(I)/FLOAT(N)
  TEMP1 = REALX(F(I))
  TEMP2 = AIMAG(F(I))
  TEMPR = SQRT((TEMP1*TEMP1)+(TEMP2*TEMP2))
  IF (MAX.LT. TEMPR) MAX = TEMPR
6  CONTINUE
*** ALL DONE
  RETURN
  END

```

SUBROUTINE PRTCBUF (UNIT,CBUFF,N,HEADER)

*** ROUTINE THAT OUTPUTS THE COMPLEX BUFFER CBUFF TO
*** THE PRINTER IN TABULAR FORM

DIMENSION CBUFF(256)
INTEGER*2 UNIT,N,I,IERR,FOURTH
INTEGER*2 T2,T3,T4
COMPLEX CBUFF
REAL VALR1,VALR2,VALR3,VALR4
REAL VALI1,VALI2,VALI3,VALI4
CHARACTER HEADER*30

*** FIRST OUPUT THE HEADER

WRITE (UNIT,100) HEADER
100 FORMAT ('1',T30,A30)
WRITE (UNIT,110)
110 FORMAT (' SAMPLE VALUE',T18,'SAMPLE VALUE',T38,'SAMPLE VALUE'
& T58,'SAMPLE VALUE')

*** NOW CYCLE THROUGH THE BUFFER AND OUTPUT THE VALUES

FOURTH = N/4
DO 300, I=1,FOURTH
T2 = I + FOURTH
T3 = I + (2*FOURTH)
T4 = I + (3*FOURTH)
VALR1 = REALCX(CBUFF(I))
VALI1 = AIMAG(CBUFF(I))
VALR2 = REALCX(CBUFF(T2))
VALI2 = AIMAG(CBUFF(T2))
VALR3 = REALCX(CBUFF(T3))
VALI3 = AIMAG(CBUFF(T3))
VALR4 = REALCX(CBUFF(T4))
VALI4 = AIMAG(CBUFF(T4))

*** FIRST, WRITE THE REAL PART

WRITE (UNIT,200) I,VALR1,T2,VALR2,T3,VALR3,T4,VALR4
200 FORMAT (T4,I3,' ',F8.4,T20,I3,' ',F8.4,T40,I3,' ',F8.4,
& T60,I3,' ',F8.4)

*** NOW, WRITE THE IMAGINARY PART

WRITE (UNIT,250) VALI1,VALI2,VALI3,VALI4
250 FORMAT (T9,F8.4,T25,F8.4,T45,F8.4,T65,F8.4)

300 CONTINUE

RETURN

END

```

SUBROUTINE PRTPLOT (UNIT,ARRAY,N,HEADER,TYPE)

*** SUBROUTINE THAT REPRESENTS ON THE PRINTER THE OUTPUT
*** SHOWN ON THE SCOPE ( WAVEFORMS)

    DIMENSION ARRAY(256)
    REAL ARRAY, YDELTA, MAX, MIN, CURRENTY, LASTY
    INTEGER*2 N, UNIT, I, X, Y, IERR, XMAX, YMAX, COUNT
    INTEGER*2 TYPE
    CHARACTER HEADER*30, LINE*120, BLANKS*120, DASHES*120
    CHARACTER SEMICOLON*1

*** FIRST, SOME SETUP.  NEED TO FILL BLANKS WITH SPACES
    CALL REPEAT (BLANKS,120,' ',1,120,IERR)
    CALL REPEAT (DASHES,120,'-',1,120,IERR)
    SEMICOLON = ':'

*** NOW, DETERMINE THE MAXIMUM AND MINIMUM VALUES IN ARRAY
*** NEED TO SET THE MINIMUM TO LARGE NUMBER AND THE MAXIMUM
*** TO A SMALL NUMBER
    MAX = -15.0
    MIN = 15.0
    DO 100, I=1,N
        IF (ARRAY(I).GT.MAX) MAX=ARRAY(I)
        IF (ARRAY(I).LT.MIN) MIN=ARRAY(I)
100    CONTINUE

*** NOW, DETERMINE SOME CONSTANTS
    YDELTA = (MAX-MIN)/50
    XMAX = 120
    IF (N.LT.XMAX) XMAX=N
    COUNT = 4

*** PRINT THE BANNER
    WRITE (UNIT,120) HEADER
120    FORMAT ('1',T49,A30)
    WRITE (UNIT,130) BLANKS
130    FORMAT (A30)

*** THE OUTER LOOP IS EXECUTED ONE STEP FOR EACH LINE IN THE PRINTOUT
    YMAX = 51
    CURRENTY = MAX+YDELTA
    DO 300, Y=1,YMAX
        LASTY = CURRENTY
        CURRENTY = (MAX-(YDELTA*(Y-1)))
*** THIS INSERT DRAWS SQUARES ON THE PLOT FOR EASY READING
        COUNT = COUNT + 1
        LINE = BLANKS
        IF (COUNT.EQ.5) LINE = DASHES
        IF (COUNT.EQ.5) COUNT = 0
        DO 150, I=10,120,10
            CALL SUBSTR(':',1,1,LINE,I,1,IERR)
150        CONTINUE
*** NOW, CALCULATE WHERE TO PLACE THE STARS
        DO 200, X=1,XMAX
            IF (TYPE.EQ.1) GOTO 170
            IF ( ARRAY(X).GE.CURRENTY )
            & CALL SUBSTR('*',1,1,LINE,X,1,IERR)
            GOTO 200
170            IF ((ARRAY(X).GE.CURRENTY).AND.(ARRAY(X).LT.LASTY))
            & CALL SUBSTR('*',1,1,LINE,X,1,IERR)
200        CONTINUE

```

```

      WRITE (UNIT,210) CURRENTY, SEMICOLON, LINE
210      FORMAT (F8.4,2X,A1,A120)
300      CONTINUE

***  NOW LABEL THE BOTTOM OF THE AXIS
      WRITE (UNIT,410)
410      FORMAT (10X,'          10          20          30          40',
& '          50          60          70          80          90',
& '          100         110         120')
      RETURN
C      WRITE (UNIT,410)
C410      FORMAT (' ')
C      DO 430, I=1,XMAX,10
C      WRITE (UNIT,420) LINE
C420      FORMAT (10X,A120)
      END

```

SUBROUTINE PRTRBUF (UNIT,RBUFF,N,HEADER)

*** ROUTINE THAT OUTPUTS THE REAL BUFFER THAT IS USED IN
*** THE DAC OUTPUT ROUTINE AOW.

DIMENSION RBUFF(256)
INTEGER*2 UNIT,N,I,IERR,FOURTH
INTEGER*2 T2,T3,T4
REAL RBUFF
CHARACTER HEADER*30

*** FIRST OUPUT THE HEADER

WRITE (UNIT,100) HEADER

100 FORMAT ('1',T30,A30)

WRITE (UNIT,110)

110 FORMAT (' SAMPLE VALUE',T18,'SAMPLE VALUE',T38,'SAMPLE VALUE'
& T58,'SAMPLE VALUE')

*** NOW CYCLE THROUGH THE BUFFER AND OUTPUT THE VALUES

FOURTH = N/4

DO 300, I=1,FOURTH

T2 = I + FOURTH

T3 = I + (2*FOURTH)

T4 = I + (3*FOURTH)

WRITE (UNIT,200) I,RBUFF(I),T2,RBUFF(T2),T3,RBUFF(T3),

& T4,RBUFF(T4)

200 FORMAT (T4,I3,' ':',F8.4,T20,I3,' ':',F8.4,T40,I3,' ':',F8.4,

& T60,I3,' ':',F8.4)

300 CONTINUE

RETURN

END

SUBROUTINE PUTDAC(N, INPUT, OUT1, OUT2, SWEEP, MAG, BLANK)

*** SUBROUTINE THAT IS USED TO PLACE THE VALUES IN THE INPUT ARRAY
*** "INPUT" INTO THE ARRAY "OUT1" AND "OUT2" SO THAT THEY MAY
*** BE USED DIRECTLY AS INPUTS TO THE DAC ROUTINE "AQW" TO DISPLAY
*** ON THE OSCILLOSCOPE.

DIMENSION INPUT(256), OUT1(1024), OUT2(2, 1024)
REAL INPUT, STEPR, SPOTR, TEMPR
INTEGER*2 N, OUT1, OUT2, SWEEP, MAG, BLANK, FOURI, IERR, TEMPI
INTEGER*2 MINUS5, ZERO

STEPR=4.5/N
SPOTR = -2.5

CALL DASCL (3, 2, -5.0, MINUS5, IERR)
CALL DASCL (3, 2, 0.0, ZERO, IERR)

DO 200, I=1, N
 FOURI = I*4
 OUT2(1, FOURI-3) = >60 + >0400
 OUT2(2, FOURI-3) = SWEEP
 SPOTR = SPOTR + STEPR
 CALL DASCL (3, 2, SPOTR, TEMPI, IERR)
 OUT1(FOURI-3) = TEMPI

 OUT2(1, FOURI-2) = >60 + >0400
 OUT2(2, FOURI-2) = MAG
 TEMPR = INPUT(I)
 CALL DASCL (3, 2, TEMPR, TEMPI, IERR)
 OUT1(FOURI-2) = TEMPI

 OUT2(1, FOURI-1) = >60 + >0400
 OUT2(2, FOURI-1) = BLANK
 OUT1(FOURI-1) = ZERO

 OUT2(1, FOURI) = >60 + >0400
 OUT2(2, FOURI) = BLANK
 OUT1(FOURI) = MINUS5

200 CONTINUE

RETURN
END

APPENDIX G

SCHEMATIC OF EIGHT-BIT ADC

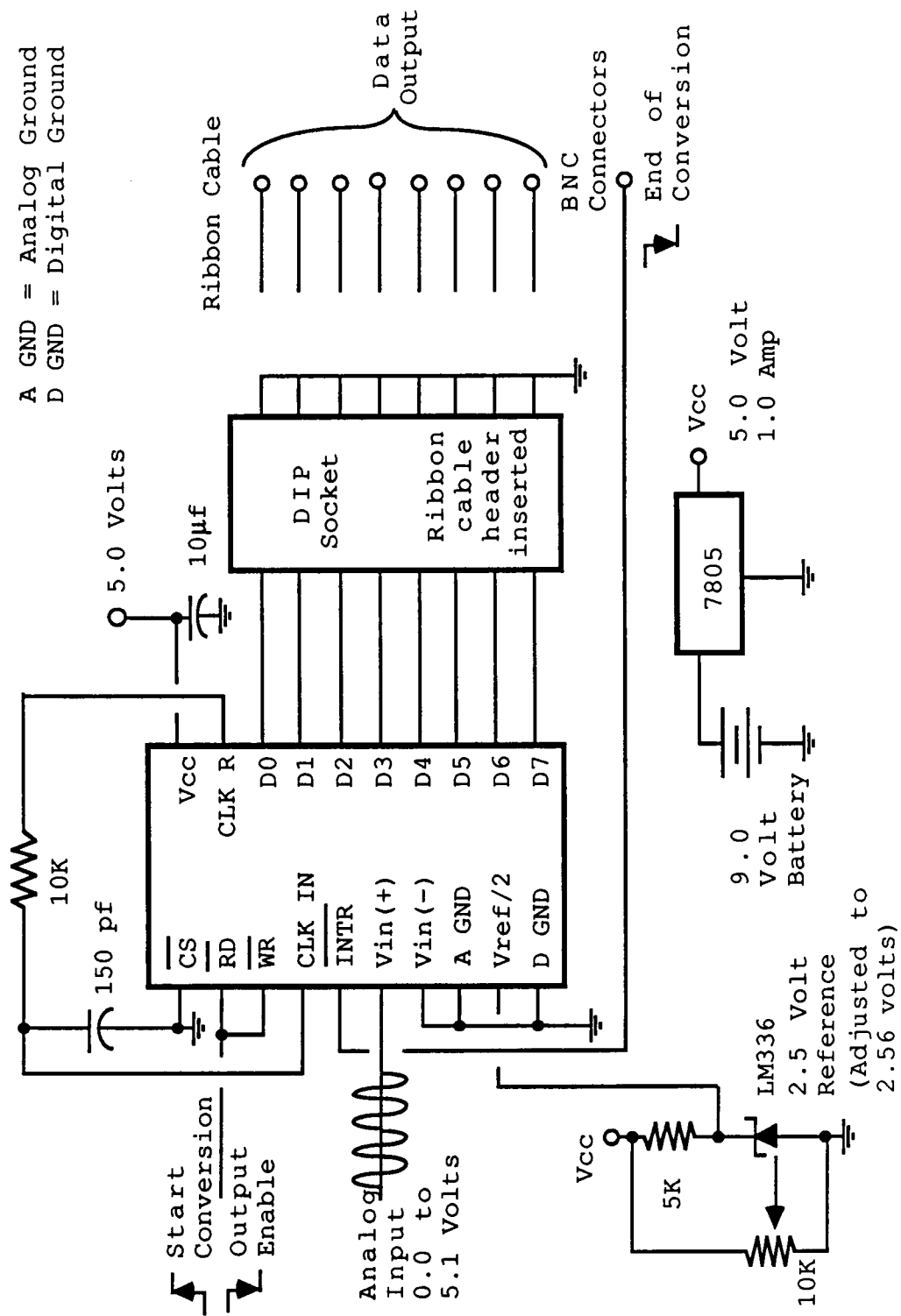


FIGURE G.1. Schematic of Eight-bit 0804 ADC System.

VITA

Randall Thomas Schonhoff was born in Nashville, Tennessee in 1957. As the son of a Mechanical Engineer, it was not unexpected when he entered the University of Tennessee at Knoxville to study engineering; it was unexpected that the particular specialty was electrical. He graduated in 1981 and accepted a position at a local business as a software engineer.

In 1984, he entered the graduate program at the University of Tennessee at Knoxville and received his advanced degree in the fall of 1985. He worked at Oak Ridge National Laboratories during his graduate program and accepted a position in Florida as an engineer after graduation.