Fall 1980

A Real Time Microprocessor Based Digital Filter Implementation

Mark H. Shaver
University of Central Florida

A REAL TIME MICROPROCESSOR BASED DIGITAL FILTER IMPLEMENTATION

BY

MARK HAROLD SHAVER
B.S., Purdue University, 1973

RESEARCH REPORT

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Engineering
in the Graduate Studies Program of the College of Engineering
at the University of Central Florida at Orlando, Florida

Fall Quarter
1980

A REAL TIME MICROPROCESSOR BASED
DIGITAL FILTER IMPLEMENTATION

BY

MARK H. SHAVER

ABSTRACT

A major break-through in the real-time digital simulation of dynamic models has occurred with the introduction of the Intel 2920 digital signal processing chip. The problems and potentials for this new device are demonstrated by implementing an elliptic function digital low pass filter via the bilinear z-transform approach. The software implementation is presented. Debugging and software verification are accomplished via manufacturer's simulation software tools. The hardware performance is verified in the laboratory.

The results of these efforts point to much promise for wide scale applications, however, problems associated with performance indicate early version chip problems.

Fred O. Simons Jr.
Dr. Fred O. Simons
Director of Research Report

ACKNOWLEDGMENT

This paper was made possible by continued support from Dr. Herbert C. Towle in the use of the Intel Development systems. I thank Dr. Towle also for the introduction to personnel at the Naval Training Equipment Center in Orlando, Florida, who kindly gave me access to test hardware and allowed use of their development system. Finally, I owe a great debt of gratitude to Dr. Fred O. Simons Jr. for his support, enthusiasm, and outstanding example of excellence he set for me.

PREFACE

This paper attempts to determine a practical hardware digital filter implementation example using an INTEL 2920 digital signal processing chip. The emphasis here is not on the choice and design of the filter, but is on the practical steps and the use of available development aids to arrive at the software for the final hardware PROM resident on the 2920.

The paper assumes a knowledge of digital filter terminology and techniques. The software is developed with the aid of a programmable HP-29C calculator and an INTEL microprocessor development system, hence, frequently the details of the design require knowledge of these systems.

Chapter I is a demonstration of how to arrive at a set of coefficients for the digital filter. Included are techniques for testing the frequency response and predicting the response to step, pulse, sinewave, and squarewave inputs.

Chapter II is a demonstration of how to convert the coefficients to a base two representation which is optimal in the sense that it will minimize the number of steps required for a multiplication. There follows a discussion of sensitivity to coefficient variation

and a practical method of estimating the precision to which the coefficients must be specified. Precision is obtained at the expense of memory and time and therefore should not be overspecified.

Chapter III is an illustration of the development of software using development aids. It is not intended to be a comprehensive discussion of the use of the INTEL systems or assembly language, or to replace the available manuals. The purpose is to illustrate the sequence of events necessary to create a source program, debug it, and simulate the filter's response.

TABLE OF CONTENTS

PREFACE iv

LIST OF TABLES vii

LIST OF FIGURES viii

INTRODUCTION 1

CHAPTER I. DIGITAL FILTER COEFFICIENTS 4

CHAPTER II. OPTIMUM BINARY REPRESENTATION 8

CHAPTER III. SOFTWARE DEVELOPMENT AIDS 18

CHAPTER IV. HARDWARE CONSIDERATIONS AND PERFORMANCE 23

CHAPTER V. CONCLUSION 33

APPENDIX A 35

APPENDIX B 41

APPENDIX C 44

LIST OF REFERENCES 55

LIST OF TABLES

1. DIGITAL FILTER COEFFICIENTS 5

2. PREDICTED ANALOG FREQUENCY RESPONSE OF THE DIGITAL FILTER 6

3. PREDICTED STEP AND PULSE RESPONSE OF DIGITAL FILTER 7

4. APPROXIMATE REPRESENTATION FOR FILTER COEFFICIENTS 13

5. ESTIMATED SENSITIVITY AND VARIANCE OF RESPONSE TO COEFFICIENTS 14

6. CONFIDENCE FACTOR AND CORRESPONDING NUMBER OF STANDARD DEVIATIONS 15

7. SENSITIVITY DATA 16

8. LIST OF TEST EQUIPMENT 32

9. LIST OF HP-29C PROGRAM STEPS FOR FREQUENCY RESPONSE 35

10. HP-29C COEFFICIENT STORAGE LOCATIONS FOR FREQUENCY RESPONSE 36

11. HP-29C STORAGE LOCATIONS FOR COEFFICIENTS AND TEST DATA FOR SIMULATION 37

12. PROGRAM STEPS FOR SIMULATION ON HP-29C 39

LIST OF FIGURES

1. DIGITAL FILTER BLOCK DIAGRAM 3

2. GRAPH OF MINIMUM REQUIRED BITS VERSUS ΔM_{max} WITH CONFIDENCE FACTOR (y) AS A PARAMETER 17

3. SCHEMATIC DIAGRAM OF 2920 TEST BOARD 26

4. DIGITAL FILTER TEST SETUP 27

5. PREDICTED AND MEASURED FREQUENCY RESPONSE 29

6. PREDICTED AND MEASURED RESPONSE TO CHANGE OF INPUT POLARITY (1 VOLT D.C.) 30

INTRODUCTION

A complete digital filter in the sense used in this paper consists of all the hardware and software necessary to perform an analog filtering function with digital circuits rather than analog components. Figure 1 illustrates the processing of the major components in a digital filter. An analog component circuit can be replaced by an analog to digital converter, a digital processor, and a digital to analog converter. Some of the benefits which can be realized by doing this are:

1. A high speed processor could actually process a number of multiplexed signals, performing analog processing functions on a number of independent channels.
2. The processing function is permanent in software, unless deliberately changed, and will not drift with age.
3. The processing function can be changed without changing components, merely by changing software.
4. Accuracy can be made very high and can be changed merely by changing software.
5. Processing, which previously required large components such as inductors in low frequency filters, can now be performed by very small digital circuits.

The type of processor chosen to implement this example was an INTEL 2920 digital signal processing chip. It combines on a single chip the analog to digital converter, multiplexers, digital processor, digital to analog converter, read only memory for program storage, and

access memory for scratch pad calculations. The conversion processes and the multiplexing can all be controlled by software.

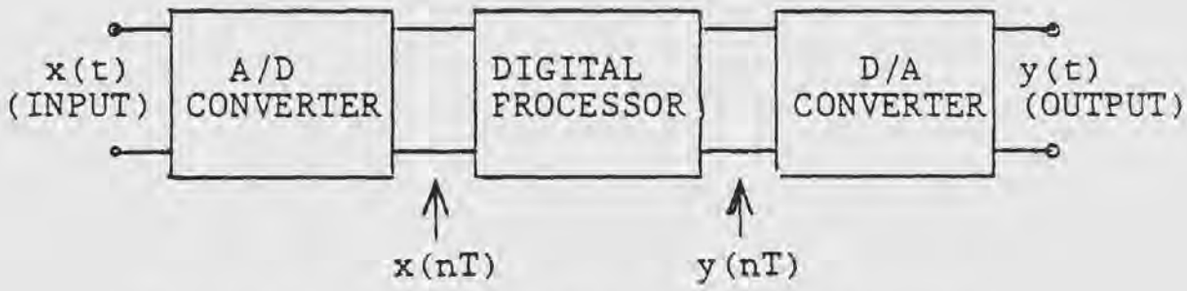

Figure 1. Digital filter block diagram.

I. DIGITAL FILTER COEFFICIENTS

In this chapter, the process for converting an analog prototype function into a set of digital filter coefficients will be domonstrated. Then the techniques for testing and predicting the response of the digital filter will be demonstrated.

This part of the implementation process can be broken down into the following major steps.

A. Determine the desired analog transfer function.

B. Determine the sample frequency.

C. Apply the bilinear Z transform.

D. Predict the response of the filter.

Determining the analog transfer function would consist of using design tables and graphs to decide on an appropriate approximation function. This function should be factored into single real poles and zeroes, and pairs of complex conjugate poles and zeroes so that it may be implemented as first and second order sections to reduce quantization errors. An elliptic function prototype was chosen to test the feasability of implementing the pure imaginary zeroes despite roundoff and quantization errors. The prototype Laplace transfer function selected was

$$H(s) = \frac{0.083974(s^2 + 17.48528)}{s^2 + 1.35715s + 1.55532}$$

and was taken from Daryanani (1).

A sample frequency of 12 KHz was chosen for the first calculation. It was decided to force the pure imaginary zeroes to correspond to a frequency of 2 KHz. The final transfer function of the digital filter second order section is of the form

$$H(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}}$$

According to the procedure given by Stanley (2), the coefficients were found and are listed in Table 1. The interesting relationship between the numerator coefficients is due to the frequency of the zeroes being chosen as exactly one third of the folding frequency and was discovered entirely by accident.

TABLE 1

DIGITAL FILTER COEFFICIENTS

Coefficient	Value
a_0	0.092001558
a_1	-0.092001558
a_2	0.092001558
b_1	-1.594733202
b_2	0.692054049

The correctness of the transformation can be verified

by testing the frequency response. This is done by replacing z in H(z) by $e^{j2\pi fT}$ where T is the reciprocal of the sampling frequency, and evaluating the magnitude of the transfer function at desired frequencies. This was done on an HP-29C calculator. The program is listed in Appendix A. The resultant frequency response is listed in Table 2. For purposes of debugging the

TABLE 2

PREDICTED ANALOG FREQUENCY RESPONSE OF THE DIGITAL FILTER

Frequency In Hertz	Gain	20Log(Gain)
0	0.94534	-0.488
400	0.99998	-0.000
800	0.56582	-4.946
1600	0.06031	-24.39
2000	1.64×10^{-10}	-195.7
6000	0.08397	-21.52

the software, it is desirable to predict the transient response to a few basic waveforms such as step and pulse waveforms. A program which will predict such responses using an HP-29C Calculator is listed in Appendix A. The responses are listed in Table 3. The primary usage of the predicted response to these two inputs was in determining the input scaling necessary to avoid saturation in the actual filter.

TABLE 3

PREDICTED STEP AND PULSE RESPONSE OF DIGITAL FILTER

Step Number	Response To Step	Response To Pulse
0	0.0	0.0
1	0.092	0.092
2	0.147	0.055
3	0.262	0.116
4	0.409	0.146
5	0.562	0.154
6	0.706	0.144
7	0.829	0.123
8	0.925	0.096
9	0.993	0.069
10	1.036	0.043
11	1.057	0.021
12	1.061	0.004
13	1.052	-0.009
14	1.035	-0.016
15	1.015	-0.020
20	0.938	0.009
25	0.934	0.0023
30	0.945	0.0016
35	0.947	-0.0002

II. OPTIMUM BINARY REPRESENTATION

In this chapter the coefficients will be expressed as sums and differences of powers of two. Using this method, one can minimize the number of steps required for a multiplication of the contents of a register by a coefficient. The coefficients can be converted with as much precision as desired, but at the expense of time and memory, hence, a method of investigating the tradeoffs is considered.

The multiplies are of the form

$$x(n) = a_0 \cdot w(n-1)$$

where x(n) and w(n-1) are register contents and a_0 is a coefficient. Standard conditional add routines are possible on the INTEL 2920 but require that every bit in a register be tested out to the required precision wasting instruction cycles on bits which are zero. A different method was used which exploits the zero bits by eliminating cycles corresponding to them and by allowing subtraction as well as addition. The result is a much faster multiply which is important in real time applications. This method is only possible if the coefficients are constant and known at the time of programming. This method is discussed in Intel's 2920 Assembly Language Manual (3). As an example, the

coefficient a_0 can be approximated as $2^{-3} - 2^{-5} - 2^{-9} + 2^{-12} - 2^{-15} - 2^{-17} - 2^{-20}$, which would evaluate to 0.092001915. This would be implemented in software as a series of addition and subtraction with shift operations. Multiplying a binary number by 2^{-n} corresponds to shifting right by n bits. A few of the steps would appear as:

Shift w(n-1) right 3 bits and store in x(n)

Shift w(n-1) right 5 bits and subtract from x(n)

Shift w(n-1) right 9 bits and subtract from x(n)

Shift w(n-1) right 12 bits and add to x(n)

Etc.

The advantage of this method is that the powers of two which are not used do not use any program steps or memory and the use of subtraction as well as addition allows the approximation to converge on the desired value in fewer steps. A program for use on an HP-29C to find the binary representation of the coefficients is listed in Appendix A. The assembly language manual for the INTEL 2920 (3) also has an algorithm which yields the same results. The approximations for the coefficients are listed in Table 4. The coefficients in Table 4 were specified to at least 2^{-20} which is less than 10^{-6}. The digital to analog conversion process consists of sending the nine most significant bits of the desired register to an analog to digital converter. The most significant bit is the sign bit, therefore, the output can be resolved to 2^{-8}

or 0.00390625. In deciding how many bits to use for a constant several factors must be considered, as even the least significant bit can affect the ninth bit of the result by a series of carries. The specification of a constant therefore involves the probability of affecting the least significant bit of the result.

The method illustrated here for determining the required coefficient precision is taken from Crochiere (4) and modified. The first step is to determine the maximum allowable shift in the amplitude response of the filter, referred to as ΔM_{max}. For example, suppose 2.6 KHz is the frequency where the least margin between the requirements and the approximation function exists. Further, suppose that the filter requirements are to have a loss of 27.5 dB at 2.6 KHz and that the actual loss is 27.53 dB. ΔM_{max} is 1.454×10^{-4} in this case.

The second step is to define the sensitivities of the response to variations in the coefficients. These sensitivities are found by taking partial derivitives of the response function with respect to the coefficients. These derivitives would be tedious to take. The sensitivities were actually estimated by allowing a coefficient to vary slightly and calculating the change in the response. The ratio of change in response to the change in the coefficient is then an estimate of the sensitivity. This relationship is

$$S_{c_i} = \left. \frac{\Delta |H(e^{j\omega T})|}{\Delta c_i} \right|_{\omega = \omega_o}$$

where S_{c_i} is the estimate of the sensitivity, and c_i is one of the coefficients. The advantage of estimating the sensitivity is that since the response was already programed it was a simple matter to let the coefficient of interest vary slightly and to calculate the resultant change in response. Once the sensitivities are found, define a variance,

$$S_T^2 = \sum_{i=1}^{m} S_{c_i}^2$$

The sensitivity estimates and variance estimates are tabulated in Table 5.

The next step is to define a confidence factor (y), as the probability that the response will not exceed the requirements. Using standard normal distribution tables find x_1 from the relationship that y is the probability that the unit normally distributed variable x is between positive and negative x_1. Some useful confidence factors and their corresponding value of x_1 are shown in Table 6.

The largest allowable quantization step can now be found from

$$q \leqq \frac{\sqrt{12}\,\Delta M_{max}}{x_1 S_T}$$

and $\quad q = 2^{-k}$

where $\quad k$ = positive integer

The coefficients must be specified to the kth bit beyond the radix. Some data calculated to investigate the tradeoffs between precision, confidence factor, and allowable shift in response are tabulated in Table 7.

A graph of the required number of bits beyond the radix versus the allowable response change is shown in Figure 2 with confidence factor as a parameter. The designer could now inspect his approximation function for the least margin, and tradeoff probability of remaining within this margin against program length. It is suggested here that this result could actually be decreased for the coefficients which have the least sensitivity at the least margin frequencies. It would not make sense to waste memory on coefficients which hardly affect the response.

TABLE 4

APPROXIMATE REPRESENTATION
FOR FILTER COEFFICIENTS

Coefficient	Binary Approximation
a_0	$2^{-3} - 2^{-5} - 2^{-9} + 2^{-12} - 2^{-15} - 2^{-17} - 2^{-20}$
a_1	$-2^{-3} + 2^{-5} + 2^{-9} - 2^{-12} + 2^{-15} + 2^{-17} + 2^{-20}$
a_2	Same as a_0
b_1	$-2^{1} + 2^{-1} - 2^{-3} + 2^{-5} - 2^{-10} - 2^{-18} - 2^{-19} - 2^{-20}$
b_2	$2^{-1} + 2^{-3} + 2^{-4} + 2^{-8} + 2^{-11} + 2^{-13} + 2^{-15} + 2^{-17} - 2^{-21}$

TABLE 5

ESTIMATED SENSITIVITY AND VARIANCE

OF RESPONSE TO COEFFICIENTS

Coefficient (C_i)	$(S_{c_i})^2$ at 2.0 KHz	$(S_{c_i})^2$ at 2.6 KHz
a_0	1.583058	0.026316
a_1	1.583058	0.611389
a_2	1.583058	0.026316
b_0	3.78×10^{-20}	0.001020
b_1	1.08×10^{-11}	0.000202
Totals (Variance)	$S_T^2 = 5.0121$	$S_T^2 = 0.66524$

TABLE 6

CONFIDENCE FACTOR AND CORRESPONDING NUMBER OF STANDARD DEVIATIONS

Confidence Factor (y)	Number of Standard Deviations (x_1)
0.950	1.96
0.980	2.33
0.997	3.00

TABLE 7

SENSITIVITY DATA

Assumed Frequency Of ΔM_{max}	Hypothetical Desired Attenuation	Attenuation Of Chosen Approximation Function	Confidence Factor (y)	Minimum Bits Beyond Radix
2.0 Khz	150 dB	Infinite	.95	26
			.98	26
			.997	26
2.0 KHz	100 dB	Infinite	.95	17
			.98	18
			.997	18
2.6 KHz	25 dB	27.53 dB	.95	6
			.98	6
			.997	6
2.6 KHz	27 dB	27.53 dB	.95	8
			.98	8
			.997	9
2.6 KHz	27.5 dB	27.53 dB	.95	14
			.98	14
			.997	15

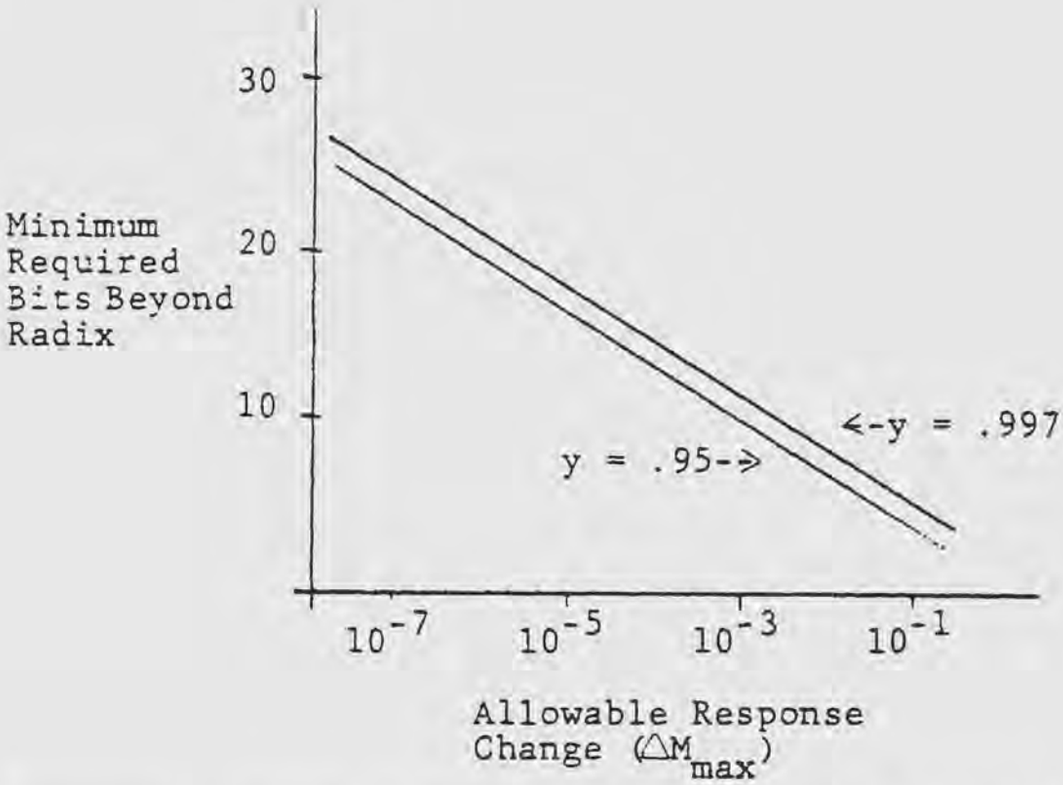

Figure 2. Graph of minimum required bits beyond radix versus ΔM_{max} with confidence factor (y) as a parameter.

III. SOFTWARE DEVELOPMENT AIDS

This chapter is devoted to a description of the use of the software development aids to produce the final version of the software which will be put into ROM for use by the digital filter microprocessor.

To cold start the development system and enter ISIS mode (5);

1. Turn on the power switches on the disk drive and the display unit.
2. Insert an ISIS disk (any disk which has all the ISIS files on it) into drive zero, (the drive on the right side) and close the door.
3. Push the reset button. After a few seconds, the display should read, "ISIS II VERSION 4.0"

To prepare a disk for use on the development system;

1. Insert a fresh disk into drive one.
2. Type, "FORMAT :F1:(name) S"

(name) can be any six letters except ISIS commands. It will take the system a few minutes to format the new disk. It will prompt for a new command when finished, by typing a hyphen.

3. Copy all 2920 software files onto the new disk.

To do this first type, "DIR 0 I"
This will copy a directory of the disk in drive zero onto the console. Any file name which deals with the 2920 must be copied to your disk. One such file is SM2920.SFT. To copy this, type, "COPY :F0:SM2920.SFT TO :F1:SM2920.SFT"
Repeat this for all 2920 files. Now obtain a directory of your disk by typing, "DIR 1 I" and verify that all 2920 files have been copied to your disk. To obtain a permanent record of your directory, type, "DIR 1 I TO :LP:" Turn on the line printer and press the select burron. The disk in drive zero can now be removed and the newly formatted disk inserted there. This is not necessary, but if done, file names do not need to be preceded with :F#:, as the computer will assume a file is located in drive zero if not preceded with a drive number. This means less typing for the operator.

To create an assembly language source file for use by the 2920:

1. Type, "EDIT (name)"
(name) is the name of the program to be created such as DIGF.SRC. The computer should respond by typing, "ISIS II TEXT EDITOR VERSION (#)" This is an interactive program which has a number of commands of its own, which allow the operator to type in new text,

make corrections, and save the new or edited program on disk. To exit back to ISIS, type, "EXIT"

2. Copy your new source program to the line printer by typing, "COPY (name) TO :LP:"

To assemble the program invoke the assembler by typing, "AS2920 (name) (controls)" (name) is your program name, and (controls) are additional commands which tell the assembler what type, format, and name of files to create during the assembly process and where to put them (3). The two end results of interest are a list file and an object file. The list file is a copy of your source program line for line, with the associated hexadecimal object code generated during assembly. It also includes error messages. Use the ISIS Copy command to copy the list file to the line printer. The hexadecimal object code file is the actual code which must be put into 2920 ROM. These two files are created automatically during assembly and stored on your disk. See (3) for an explanation of the names for these two files. If errors were made, they can be corrected by going back to the text editor or during the simulation phase. A typical assembler listing is included in Appendix B.

It is desirable to simulate the execution of the 2920 software for debugging purposes and to predict the response of the filter.

1. To load the simulator program and your object

file; type, "SM2920.SFT" This calls the simulator program (6). All commands to the simulator must be terminated with the ESC key.

2. If a record of the simulation session is desired, type, "LIST (device)" (device) is either a line printer, disk file, or the console, and the listing device may be changed any number of times. Examples, "LIST:LP:" "LIST "F1"DIGF.SIM" "LIST :CO:"

3. To load your object program type, "LOAD (name)" Examples, "LOAD DIGF.HEX" "LOAD :F1: DIGF.HEX"

4. ROM and RAM contents may be examined and modified at this point.

5. To get ready for simulation, set the input functions by typing, "IN# = (function)" Examples, "IN1 = 1" "IN1 = SIN(TPI*200*T)" The second example sets the input value at input one to $\sin(2\pi 200t)$.

6. Set program duration by typing, "TPROG = (#)" TPROG is related to clock frequency by;

$$\frac{1}{\text{TPROG}} = \frac{f_c}{4 \text{ X (PROGRAM LENGTH)}}$$

Where f_c is the clock frequency and PROGRAM LENGTH is the number of steps in the program.

7. Set the breakpoint. This tells the computer when to stop simulation. Type, "B = (CONDITION)" Examples, "B = NEVER" "B = T GT .004" (stops

when time is greater than .004 seconds.)

"B = PC = 50" (stops when the program counter reaches 50.)

8. Initialize RAM locations and DAR to any desired initial conditions.

9. Set trace. This tells the computer what data to collect. Type, "TRACE = (string)" (string) is a string of variables, expressions, RAM contents, or nearly anything imaginable which the operator wants to be recorded. Example, "TRACE = T, RAM0, IN1, RAM5, T/TINST, OUT1"

10. Set the qualifier. This tells the computer when to collect trace. Type, "Q = (condition)" Examples, "Q = ALWAYS" (Trace is collected after each program step is executed.) "Q = PC = 8 AND DAR GT .5" (Collect trace if program counter is equal to eight and DAR contents exceed 0.5.)

11. To begin simulation, Type, "S FROM 0" This resets simulation time to zero.

Or, type, "S" This starts simulation with simulation time equal to the time at which simulation was last stopped. To stop simulation, press the ESC key. A typical simulation session is listed in Appendix C.

IV. HARDWARE CONSIDERATIONS AND PERFORMANCE

In this chapter, the actual hardware and software used for the digital filter and the equipment used to test the response are discussed. There are a number of considerations to be included in the final software version which will not be found discussed in sufficient detail in most literature.

In the INTEL Component Data Catalog (1980) it is mentioned that after a CVTS instruction, the software must immediately include ADD DAR,KM2,CND6 followed by two analog NOP instructions (7). This is not mentioned in the assembly language manual.

It is necessary to include at least two analog NOP instructions after each CVT instruction. Four or five consecutive IN or OUT instructions are needed when accepting input data or sending output data.

The EOP instruction which should reset the program counter to zero after executing the following three steps would not accomplish its purpose in any location except 188, fixing overall program length at 192 steps. The signal generator used would produce a reasonably good clock signal only up to about 2.5 MHz. These two unforseen circumstances would reduce the sampling rate considerably. Four clock cycles are used per instruction.

At 2.5 MHz, and 192 steps per program pass, the sampling frequency would be about 3.255 KHz. Since the processing algorithm only occupies 60 program steps it was possible to replicate the algorithm three times in 192 steps. This raised the sample frequency by a factor of three to about 9.766 KHz. The original sample frequency used in deriving the coefficients was 12 KHz. The effect of reducing the sample frequency should be a downward shift of the passband in the frequency domain by a factor of 0.81380. This was verified by simulation. The final software used is listed in Appendix C.

To program the chip, an INTEL Universal Prom Programmer was used. The procedure is discussed in the Universal Prom Programmer Manual (8), and uses a microprocessor development system. The procedure consists of:

1. Erasing the 2920 EPROM with an ultra-violet EPROM eraser.
2. On the development system, call the PROM programming program.
3. Read the 2920 hexadecimal coded file into the development system memory.
4. Read the code from development system memory into 2920 EPROM.

To call the PROM programming program, type "UPM." The program will ask for a device number. The version of the program available did not recognize the 2920. Instead, an adapter board was used which adapts the pins and programming of the 2920 to the 2716. The user then

types, "2716". Now the user types the command, "READ FILE filename INTO 0". This reads the code from magnetic disk into development system memory. Lastly, the user types the command, "PROGRAM FROM 0 TO 47FH START 0". This reads the code from development system memory into 2920 EPROM.

The test board used to interface to the processor for response testing is shown schematically in Figure 3. The board provides an adjustable reference voltage, and a sample-hold capacitance of about 490 pF. The user must provide the board with positive five volts, negative five volts, each at about 70 mA., and a TTL compatable offset squarewave signal for the clock. The signal used was about -0.2 volts on the negative portion of the cycle, and about +2.0 volts on the postive portion of the cycle.

The setup used to test the response is illustrated in Figure 4. The list of equipment used appears in Table 8. The setup was calibrated by checking the gain or loss of the low pass filter at the measurement frequencies by comparing input and output amplitudes on the oscilloscope. The digital filter was inserted, the input and output amplitudes again compared and corrected for the low pass filter. Measured and simulated frequency responses are graphed in Figure 5.

The response to a sudden change of D.C. input

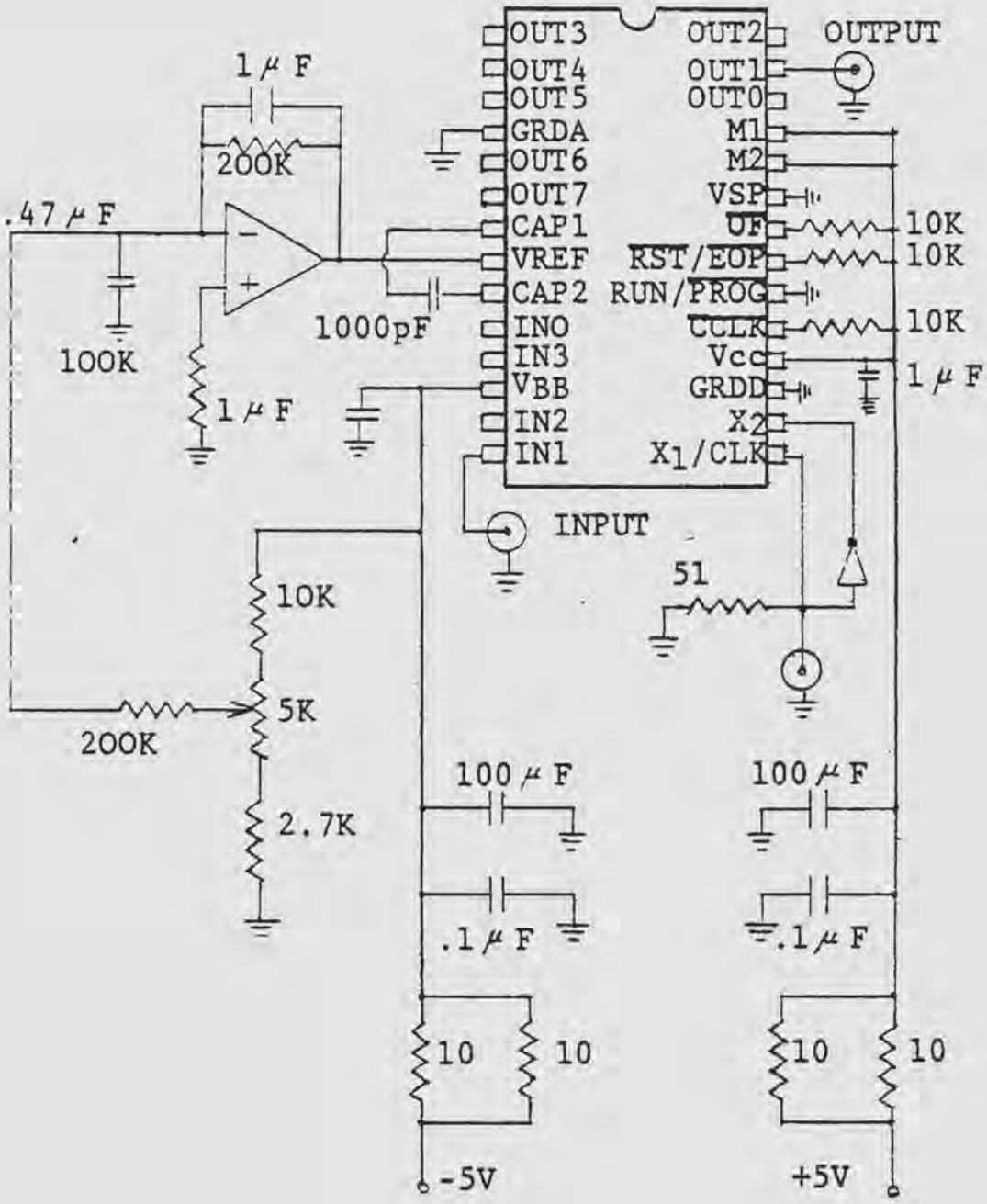

Figure 3. Schematic diagram of 2920 test board.

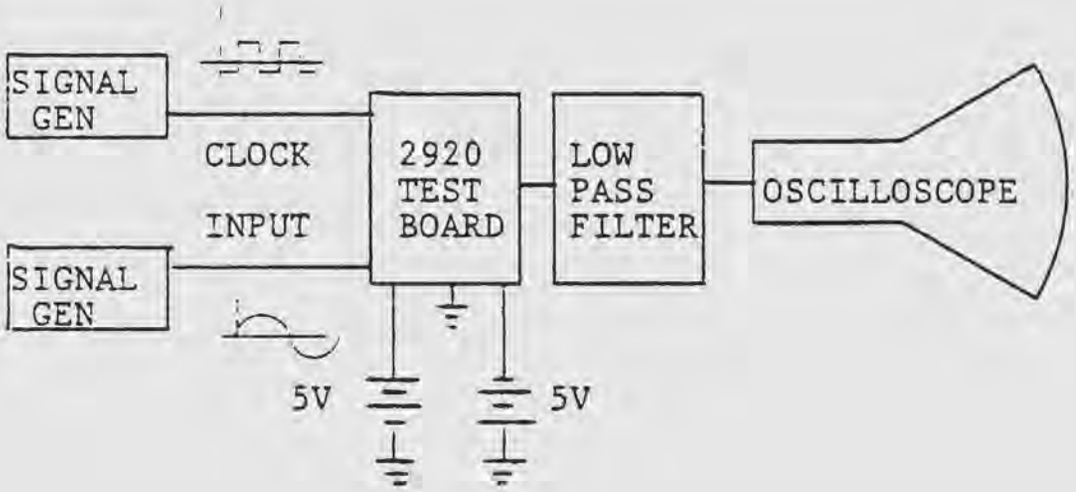

Figure 4. Digital filter test setup.

polarity was measured by using a low frequency square-wave input of 2.0 vp-p. The predicted and measured responses are graphed in Figure 6.

As seen in Figure 5, the frequency response falls off a little more rapidly than predicted, but it never falls completely to zero as it should. No null frequency or even a dip in gain occured near the zero frequency. Simulation with identical software shows that if the processor receives the correct samples and it is correctly processing them, then the output amplitude should be driven to zero at one sixth of the sampling frequency. There are either problems in the conversion processes or in the processor itself. The shape of the frequency response was improved slightly by adding additional NOP instructions after each CVT instruction, however, the gain would still not drop below the value shown in Figure 5.

For large amplitude, low frequency sinwaves, the output was a good reproduction of the input. It was noted that there were values of voltage which seemed to confuse the processor. The output would drastically overshoot the proper output and then get back on track a few samples later.

Comparing the predicted and measured responses to change of input polarity, two discrepancies are evident. The first output sample after the polarity

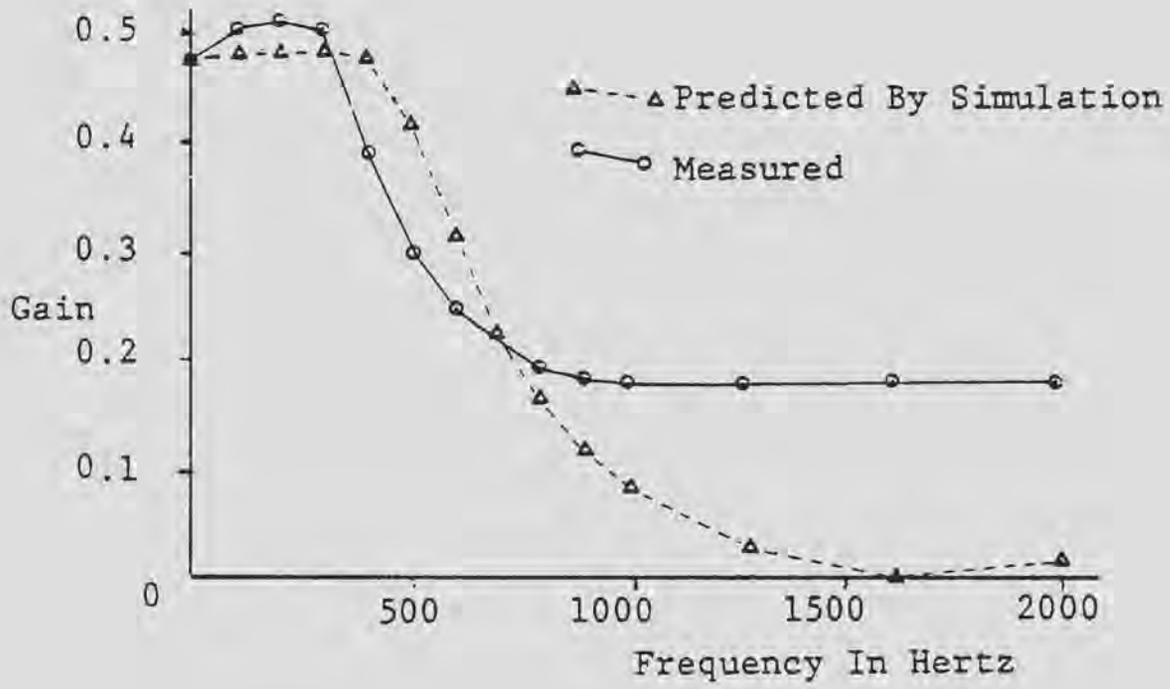

Figure 5. Predicted and measured frequency response.

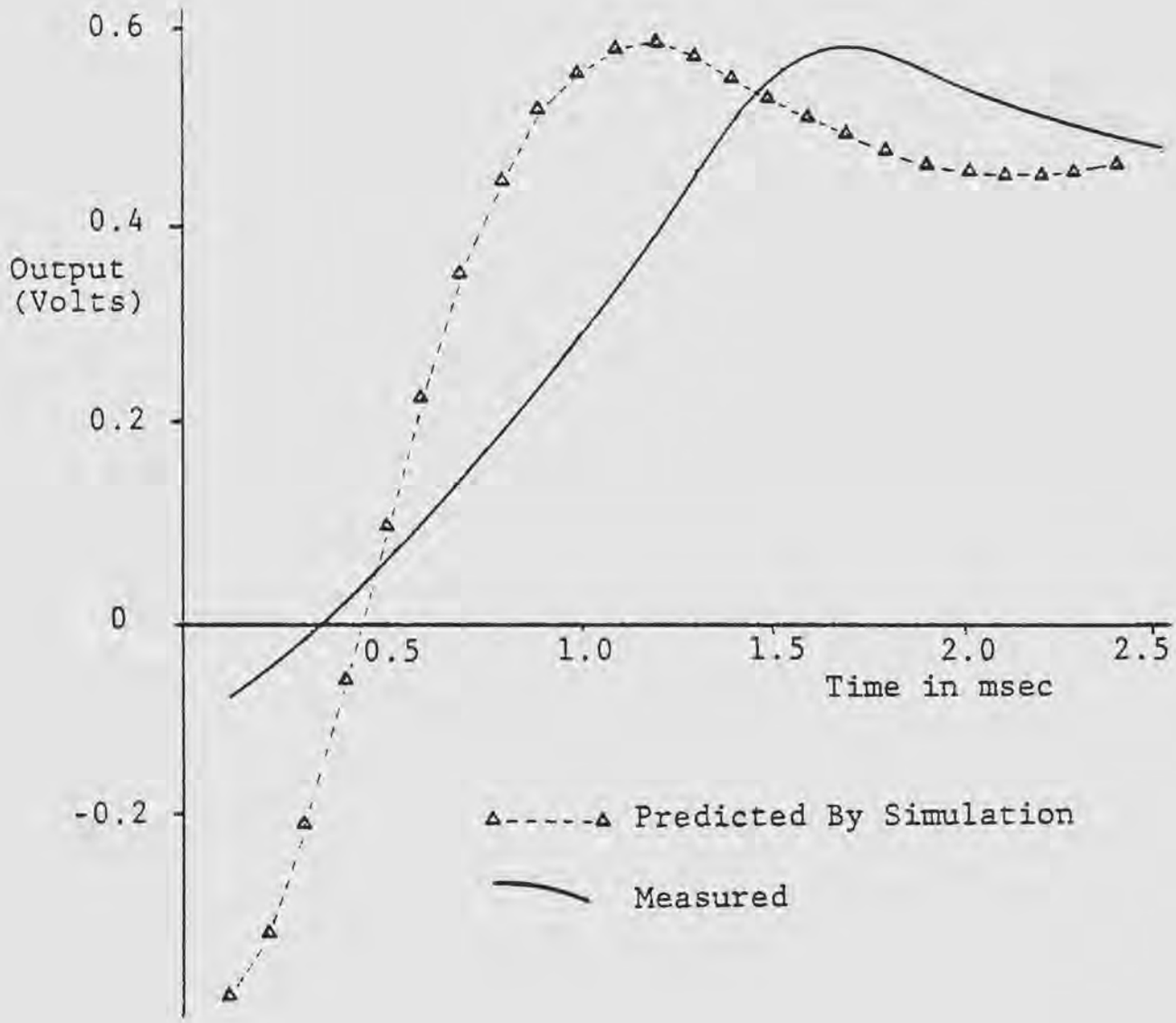

Figure 6. Predicted and measured response to change of input polarity. (1 volt D.C.)

change is much too high. Also the time constant is longer than predicted. The longer time constant and the higher passband ripple suggest that the poles of H(z) have moved toward the unit circle. The too large first sample would seem to suggest that the numerator coefficient (a_0) is dominating. This means that the output is not a properly weighted sum of the sample registers. This could also account for the apparent disappearance of the zeroes.

There are indications that both the analog and digital processing portions of the chip have some bugs which either need to be worked out or quantized so as to be accounted and corrected for when designing systems with the 2920 chip.

TABLE 8

LIST OF TEST EQUIPMENT

Clock Generator:	Exact Model 129 Function Generator 2.5 MHz Squarewave 2.2 Volts p-p 0.9 Volts D.C. Offset
Signal Generator:	Tektronix FG504 Function Generator
Low Pass Filter:	Krohn-Hite Model 3200 Filter
D.C. Supplies:	Postive And Negative 5 Volts D.C.
Oscilloscope:	Tektronix Model 561 Mainframe 3B3 Time Base 3AG Dual Trace Amplifier

V. CONCLUSION

The purpose of this paper was to illustrate the design and implementation of a real time digital filter. The coefficients were chosen based upon an approximation function and a sampling rate. The chosen sample rate could not be attained. Ordinarily, this would mean deriving a new set of coefficients. In this case the resultant frequency response was predicted and the experiment continued. The coefficients were converted to a binary approximation form which minimized required memory space and program steps. A method of estimating the required precision to which the coefficients must be specified was investigated. Software was then created for the 2920 using development aids. The software was then debugged using a simulator program. The response of the filter was predicted for a variety of input waveforms. The software was programed into the 2920 chip and the response of the chip was measured in the laboratory.

The derivation of the coefficients follows standard procedures, and presented no problems. The major item of concern is the chance of error in calculating the coefficients. Hence, there is great need to verify the frequency response before creating software.

Writing the software requires a bit of forethought in arranging the multiplication steps to avoid saturation. Also, the assembly language manual does not include several software requirements.

The simulator program is a tremendous asset, both in the initial software debugging, and in predicting the response changes due to a parameter change such as changing the sample frequency.

The 2920 chip could be used as a programable feedback compensator where gain and time response are required. With its present performance characteristics it is not suited to an application where adherance to a tight frequency response specification in the stop band is required. To use it in any application will require further investigation into how to account for the discrepancies between the predicted and actual responses in the design of software.

The digital filter will continue to invade areas which were formerly strictly analog. Chips such as the 2920 will undoubtedly become commonplace in extensive applications. As the quality and performance of these chips improves, the benefits of digital filtering will begin to manifest themselves even in real time systems, and eventually out-perform analog systems.

APPENDIX A

This section contains programs for use on an HP-29C calculator. Included are the program steps and the instructions for use.

PROGRAM FOR CALCULATING THE STEADY STATE FREQUENCY RESPONSE

This program assumes a second order section of the form

$$H(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 + b_1 z^{-1} + b_2 z^{-2}}$$

The program steps are listed in Table 9

TABLE 9

LIST OF HP29C PROGRAM STEPS FOR FREQUENCY RESPONSE

g LBL 0			
g RAD	g x^2	f SIN	+
RCL .0	STO 4	RCL .1	g x^2
X	RCL 3	X	RCL 5
g π	2	RCL 3	+
X	X	2	RCL 4
2	f COS	X	÷
X	RCL 0	f SIN	g 1/x
STO 3	X	+	f $\sqrt{\ }$
f SIN	STO 5	g x^2	
RCL 1	RCL 3	STO 5	R/S
X	f COS	RCL 3	f Log
STO 4	RCL 1	2	2
RCL 3	X	X	0
2	RCL 2	f COS	X
X	+	RCL 3	R/S
f SIN	RCL 5	f COS	GTO 0
RCL 0	+	RCL .1	
X	g x^2	X	
RCL 4	STO + 4	+	
+	RCL 3	RCL .2	

To initialize, store the coefficients in the locations given in Table 10, along with the sample time (T).

TABLE 10

HP-29C COEFFICIENT STORAGE LOCATIONS FOR FREQUENCY RESPONSE

Quantity	Storage Register
a_0	0
a_1	1
a_2	2
b_1	.1
b_2	.2
T	.0

To use, follow these steps:

1. Key in g/RTN.
2. Key in desired cyclic frequency.
3. Key in R/S.
   When execution stops, the display will be the gain at the frequency specified in step 2.
4. Key in R/S.
   When execution stops, the display will be the gain in decibels.
5. For new data, go to step 2.

PROGRAM TO CHECK THE RESPONSE TO STEP, PULSE, SQUARE-WAVE, AND SINEWAVE INPUTS

This program generates input test functions and implements the digital filter, displaying the output samples. It can be used to predict the filter's response for purposes of debugging the software.

To get the program ready for use, store the indicated data in the appropriate locations listed in Table 11.

TABLE 11

HP-29C STORAGE LOCATIONS FOR COEFFICIENTS AND TEST DATA FOR SIMULATION

Quantity	Storage Register
a_0	.0
a_1	1
a_2	2
b_1	3
b_2	4
Frequency of Test Function	.2
Amplitude of Test Function	.3

To initialize at the 0th sample, key in the following strokes:

```
0
STO 5
STO 6
STO 7
STO 8
```

To use the program follow these steps:

1. Store in register 0 the following values according to the test function desired.

   (1) if sinewave is desired

   (2) if step function is desired

   (3) if pulse is desired

   (4) if square wave is desired

2. Key in g RTN

3. Key in R/S
   When processing stops, the output sample will be displayed.

4. For next output sample, go to step 3.

5. For new test function, or new coefficients, go to the appropriate data entering sequence.

The program steps are listed in Table 12.

TABLE 12

PROGRAM STEPS FOR

SIMULATION ON HP-29C

g LBL 0	RCL 5	RCL .3	+
GSB i	STO 6	STO 5	f INT
RCL 7	RCL .0	g RTN	1
RCL 4	X	g LBL 3	CHS
CHS	STO + 8	0	x⇄y
X	g LBL 1	STO 5	f y^x
STO + 5	RCL 9	RCL 9	RCL .3
RCL 6	RCL .1	g X 0	X
RCL 3	X	g RTN	STO 5
CHS	RCL .2	RCL .3	g RTN
X	X	STO 5	
STO + 5	2	g RTN	
RCL 7	X	g LBL 4	
RCL 2	g π	RCL 9	
X	X	RCL .1	
STO 8	f SIN	X	
RCL 6	RCL .3	RCL .2	
STO 7	X	X	
RCL 1	STO 5	g FRAC	
X	g RTN	.	
STO + 8	g LBL 2	5	

Program to Calculate Binary Constant From Decimal Constant

Program steps:

```
g LBL 0
RCL 0
RCL 1
-
R/S
ENTER
2
x⇄y
f y^x
x⇄y
g x>0
GTO 2
x⇄y
STO - 1
GTO 3
g LBL 2
x⇄y
STO + 1
g LBL 3
RCL 1
R/S
GTO 0
```

To initialize:

1. Store decimal constant in register 0.
2. Initialize register 1 at zero.

To use:

1. Key in g RTN.
2. Key in R/S.
   When execution stops, the display will be the error between the binary approximation and the desired coefficient.
3. From a table of powers of two, pick an integer n such that 2^n is closest to the absolute value of the error.
4. Write down the sign of the error and 2^n.
5. Key in the value of n.
6. Key in R/S.
   When execution stops, the display will be the current value of the binary approximation.
7. Go to step 2 and check to see if the error is small enough. If not, continue.

APPENDIX B

An assembler listing is automatically created when the source program is assembled. It can be printed by typing the command, "COPY DGFLTR.LST TO :LP:" (DGFLTR.LST was the name of the list file created in the assembly process in this example.) The assember listing that resulted in this case is:

```
ISIS-II 2920 ASSEMBLER V1.0

ASSEMBLER INVOKED BY: AS2920 DGFLTR.SRC DEBUG NOPAGING


 LINE  LOC OBJECT SOURCE STATEMENT

    1            ;2920 DIGITAL FILTER PROGRAM
    2            ;THIS IS A SECOND ORDER FILTER
    3            ;IMPLEMENTATION IS DIRECT FORM 1
    4
    5            ;COEFFICIENTS ARE AS FOLLOWS
    6            ;A0=.092001558
    7            ;A1=-A0
    8            ;A2=A0
    9            ;B1=-1.594733202
   10            ;B2= .692054849
   11
   12            ;THIS GIVES AN ELLIPTIC FILTER WITH
   13            ;ZEROES AT 2KHZ
   14
   15            ;FORMAT IS AS FOLLOWS
   16            ;LABEL: OPCODE DESTINATION,SOURCE,SHIFT,ANALOG
```

```
 0 900000F OUT1
 1 900082F LDA XN,WN,R10,OUT1
 2 900029 SUB XN,WN,R25,OUT1
 3 900038A SUB XN,WN,R31,OUT1

          ;HAVE ADDED EXTRA NOP'S, IN'S,OUT'S TO ALLOW
          ;SUFFICIENT SETTLING TIME AND TO PREVENT CROSSTALK

 4 4000EF NOP
 5 4000EF NOP
 6 4000EF NOP
 7 1000EF IN1
 8 1000EF IN1
 9 1000EF IN1
10 1009CD ADD XN,WN,LB1,IN1
11 10004C ADD XN,WN,R93,IN1
12 44098F LDA TEMP,WN,R13     ;LARGE SHIFT REQUIRES TWO STEPS
13 42008C ADD XN,TEMP,R05
          ;MULTIPLY -B1 * WN AND STORE IN XN


          ;MULTIPLY -B2 * WN1 AND ADD TO XN
14 46008F LDA TEMP,WN1,R13
15 62000A SUB XN,WN1,R01,CVT5
16 EB6ED ADD DAR,KM2,R06,CND6   ;NOT MENTIONED IN
17 4000EF NOP                   ;PROGRAMMING MANUAL BUT
18 4000EF NOP                   ;REQUIRED (THESE 3 STEPS)
19 42004A SUB XN,WN1,R03
20 73006A SUB XN,WN1,R04,CVT7
21 4000EF NOP
22 4008EA SUB XN,WN1,R08
23 63004B SUB XN,WN1,R11,CVT6
24 42008B SUB XN,WN1,R13
25 42002A SUB XN,TEMP,R02       ;LARGE SHIFT


          ;MULTIPLY A2 * WN1 AND STORE IN YN
26 53104E LDA YN,WN1,R03,CVT5
27 4000EF NOP
28 42108A SUB YN,WN1,R05
29 43108B SUB YN,WN1,R09,CVT4
30 4000EF NOP
31 42106D ADD YN,WN1,R12
32 3D000F LDA TEMP,YN,R13,CVT3
33 42102A SUB YN,TEMP,R02
34 42108A SUB YN,TEMP,R05          ;TWO LARGE SHIFTS


          ;MULTIPLY A1 * WN AND ADD TO YN
35 21189C ADD YN,WN,R05,CVT2
36 4000EF NOP
```

```
 70   37 401848 SUB YN, WN, R03
 71   38 111800 ADD YN, WN, R09, CYT1
 72   39 4000EF NOP
 73   40 401368 SUB YN, WN, R12
 74   41 850089 LDA TEMP, WN, R13, CYT0
 75   42 421020 ADD YN, TEMP, R02
 76   43 421080 ADD YN, TEMP, R05          ;TWO LARGE SHIFTS
 77
 78
 79   44 402280 ADD XN, DAR, R05           ;HAVE SCALED INPUT BY 1/32 TO AVOID OVERFLOW
 80                                        ;NEW WN NOW COMPLETE AND RESIDING IN XN
 81
 82             ;MULTIPLY A0 * XN AND ADD TO YN
 83   45 401040 ADD YN, XN, R07
 84   46 401038 SUB YN, XN, R05
 85   47 401008 SUB YN, XN, R09
 86   48 401050 ADD YN, XN, R12
 87   49 440089 LDA TEMP, XN, R13
 88   50 421028 SUB YN, TEMP, R02
 89   51 421009 SUB YN, TEMP, R05
 90
 91   52 431099 LDA YN, YN, LB2, NOP       ;SCALING BY 16
 92
 93   53 4644AF LDA DAR, YN, LB2, NOP
 94
 95             ;SHIFT ALL REGISTERS
 96   54 4409FF LDA WN1, WN, R00
 97   55 4900FF LDA WN, XN, R00
 98   56 5000EF EOP
 99   57 4000EF NOP
100   58 4000EF NOP
101   59 4000EF NOP
102             END
```

SYMBOL:	VALUE:
XN	0
WN	1
TEMP	2
WN1	3
YN	4

```
ASSEMBLY COMPLETE
ERRORS     =   0
WARNINGS   =   0
RAMSIZE    =   5
ROMSIZE    =  60
```

APPENDIX C

Simulation Session

```
*;TODAY'S DATE IS 7 OCTOBER 1980
*
*
*;COMMENTS CAN BE WRITTEN IF PRECEDED BY SEMICOLON
*
*;THIS IS AN EXAMPLE OF A SIMULATION SESSION
*
*;THE SIMULATOR WAS INVOKED BY TYPING, "SM2920.SFT"
*
*;THE HEXFILE PFROGRAM CREATED DURING ASSEMBLY WILL
*;NOW BE LOADED
*
*LOAD GLITCH.HEX
*
*;THAT TOOK ABOUT 45 SECONDS
*
*;THE ROM LOCATIONS CAN BE LOOKED AT
*ROM 0 LEN 5
ROM 000 = LDA .XN.,XN,R00,NOP
ROM 001 = LDA .XN.,XN,R00,NOP
ROM 002 = LDA .XN.,XN,R00,NOP
ROM 003 = LDA .XN.,XN,R00,NOP
ROM 004 = LDA .XN.,XN,R00,OUT1
*;
*;THIS PRINTED OUT THE FIRST FIVE ROM LOCATIONS
*
*
*
*ROM 0 LEN 192   ;THIS WILL PRINT OUT ALL 192 ROM CONTENTS
ROM 000 = LDA .XN.,XN,R00,NOP
ROM 001 = LDA .XN.,XN,R00,NOP
ROM 002 = LDA .XN.,XN,R00,NOP
ROM 003 = LDA .XN.,XN,R00,NOP
ROM 004 = LDA .XN.,XN,R00,OUT1
ROM 005 = LDA .XN.,WN,R10,OUT1
ROM 006 = SUB .XN.,WN,R05,OUT1
ROM 007 = SUB .XN.,WN,R01,OUT1
ROM 008 = LDA .XN.,XN,R00,NOP
ROM 009 = LDA .XN.,XN,R00,NOP
```

```
ROM 010 = LDA . XN . XN R00, NOP
ROM 011 = LDA . XN . XN R00, IN1
ROM 012 = LDA . XN . XN R00, IN1
ROM 013 = LDA . XN . XN R00, IN1
ROM 014 = ADD . XN . WN L01, IN1
ROM 015 = ADD . XN . WN R03, IN1
ROM 016 = LDA . TEMP, . WN R13, NOP
ROM 017 = ADD . XN . TEMP, R05, NOP
ROM 018 = LDA . TEMP, . WN1 R13, NOP
ROM 019 = SUB . XN . WN1 R01, CVT5
ROM 020 = ADD DAR, KM2, R00, CND6
ROM 021 = LDA . XN . XN R00, NOP
ROM 022 = LDA . XN . XN R00, NOP
ROM 023 = SUB . XN . WN1 R03, NOP
ROM 024 = SUB . XN . WN1 R04, CVT7
ROM 025 = LDA . XN . XN R00, NOP
ROM 026 = SUB . XN . WN1 R08, NOP
ROM 027 = SUB . XN . WN1 R11, CVT6
ROM 028 = SUB . XN . WN1 R13, NOP
ROM 029 = SUB . XN . TEMP, R02, NOP
ROM 030 = LDA . YN . WN1 R03, CVT5
ROM 031 = LDA . XN . XN R00, NOP
ROM 032 = SUB . YN . WN1 R05, NOP
ROM 033 = SUB . YN . WN1 R09, CVT4
ROM 034 = LDA . XN . XN R00, NOP
ROM 035 = ADD . YN . WN1 R12, NOP
ROM 036 = LDA . TEMP, . YN R13, CVT3
ROM 037 = SUB . YN . TEMP, R02, NOP
ROM 038 = SUB . YN . TEMP, R05, NOP
ROM 039 = ADD . YN . WN R05, CVT2
ROM 040 = LDA . XN . XN R00, NOP
ROM 041 = SUB . YN . WN R03, NOP
ROM 042 = ADD . YN . WN R09, CVT1
ROM 043 = LDA . XN . XN R00, NOP
ROM 044 = SUB . YN . WN R12, NOP
ROM 045 = LDA . TEMP, . WN R13, CVT0
ROM 046 = ADD . YN . TEMP, R02, NOP
ROM 047 = ADD . YN . TEMP, R05, NOP
ROM 048 = ADD . XN DAR, R05, NOP
ROM 049 = ADD . YN . XN R03, NOP
ROM 050 = SUB . YN . XN R05, NOP
ROM 051 = SUB . YN . XN R09, NOP
ROM 052 = ADD . YN . XN R12, NOP
ROM 053 = LDA . TEMP, . XN R13, NOP
ROM 054 = SUB . YN . TEMP, R02, NOP
ROM 055 = SUB . YN . TEMP, R05, NOP
ROM 056 = LDA . YN . YN L02, NOP
ROM 057 = LDA DAR, . YN L02, NOP
ROM 058 = LDA . WN1 . WN R00, NOP
ROM 059 = LDA . WN . XN R00, NOP
```

```
ROM 060 = LDA . XN . XN R00, NOP
ROM 061 = LDA . XN . XN R00, NOP
ROM 062 = LDA . XN . XN R00, NOP
ROM 063 = LDA . XN . XN R00, NOP
ROM 064 = LDA . XN . XN R00, NOP
ROM 065 = LDA . XN . XN R00, NOP
ROM 066 = LDA . XN . XN R00, NOP
ROM 067 = LDA . XN . XN R00, NOP
ROM 068 = LDA . XN . XN R00, OUT1
ROM 069 = LDA . XN . WN R10, OUT1
ROM 070 = SUB . XN . WN R05, OUT1
ROM 071 = SUB . XN . WN R01, OUT1
ROM 072 = LDA . XN . XN R00, NOP
ROM 073 = LDA . XN . XN R00, NOP
ROM 074 = LDA . XN . XN R00, NOP
ROM 075 = LDA . XN . XN R00, IN1
ROM 076 = LDA . XN . XN R00, IN1
ROM 077 = LDA . XN . XN R00, IN1
ROM 078 = ADD . XN . WN L01, IN1
ROM 079 = ADD . XN . WN R03, IN1
ROM 080 = LDA . TEMP, . WN R13, NOP
ROM 081 = ADD . XN . TEMP, R05, NOP
ROM 082 = LDA . TEMP, . WN1, R13, NOP
ROM 083 = SUB . XN . WN1, R01, CVT5
ROM 084 = ADD DAR, KM2, R00, CND6
ROM 085 = LDA . XN . XN R00, NOP
ROM 086 = LDA . XN . XN R00, NOP
ROM 087 = SUB . XN . WN1, R03, NOP
ROM 088 = SUB . XN . WN1, R04, CVT7
ROM 089 = LDA . XN . XN R00, NOP
ROM 090 = SUB . XN . WN1, R08, NOP
ROM 091 = SUB . XN . WN1, R11, CVT6
ROM 092 = SUB . XN . WN1, R13, NOP
ROM 093 = SUB . XN . TEMP, R02, NOP
ROM 094 = LDA . YN . WN1, R03, CVT5
ROM 095 = LDA . XN . XN R00, NOP
ROM 096 = SUB . YN . WN1, R05, NOP
ROM 097 = SUB . YN . WN1, R09, CVT4
ROM 098 = LDA . XN . XN R00, NOP
ROM 099 = ADD . YN . WN1, R12, NOP
ROM 100 = LDA . TEMP, . YN R13, CVT3
ROM 101 = SUB . YN . TEMP, R02, NOP
ROM 102 = SUB . YN . TEMP, R05, NOP
ROM 103 = ADD . YN . WN R05, CVT2
ROM 104 = LDA . XN . XN R00, NOP
ROM 105 = SUB . YN . WN R03, NOP
ROM 106 = ADD . YN . WN R09, CVT1
ROM 107 = LDA . XN . XN R00, NOP
ROM 108 = SUB . YN . WN R12, NOP
ROM 109 = LDA . TEMP, . WN R13, CVT0
```

```
ROM 110 = ADD . YN . TEMP, R02, NOP
ROM 111 = ADD . YN . TEMP, R05, NOP
ROM 112 = ADD . XN DAR, R05, NOP
ROM 113 = ADD . YN . XN R03, NOP
ROM 114 = SUB . YN . XN R05, NOP
ROM 115 = SUB . YN . XN R09, NOP
ROM 116 = ADD . YN . XN R12, NOP
ROM 117 = LDA . TEMP, . XN R13, NOP
ROM 118 = SUB . YN . TEMP, R02, NOP
ROM 119 = SUB . YN . TEMP, R05, NOP
ROM 120 = LDA . YN . YN L02, NOP
ROM 121 = LDA DAR . YN L02, NOP
ROM 122 = LDA . WN1 . WN R00, NOP
ROM 123 = LDA . WN . XN R00, NOP
ROM 124 = LDA . XN . XN R00, NOP
ROM 125 = LDA . XN . XN R00, NOP
ROM 126 = LDA . XN . XN R00, NOP
ROM 127 = LDA . XN . XN R00, NOP
ROM 128 = LDA . XN . XN R00, NOP
ROM 129 = LDA . XN . XN R00, NOP
ROM 130 = LDA . XN . XN R00, NOP
ROM 131 = LDA . XN . XN R00, NOP
ROM 132 = LDA . XN . XN R00, OUT1
ROM 133 = LDA . XN . WN R10, OUT1
ROM 134 = SUB . XN . WN R05, OUT1
ROM 135 = SUB . XN . WN R01, OUT1
ROM 136 = LDA . XN . XN R00, NOP
ROM 137 = LDA . XN . XN R00, NOP
ROM 138 = LDA . XN . XN R00, NOP
ROM 139 = LDA . XN . XN R00, IN1
ROM 140 = LDA . XN . XN R00, IN1
ROM 141 = LDA . XN . XN R00, IN1
ROM 142 = ADD . XN . WN L01, IN1
ROM 143 = ADD . XN . WN R03, IN1
ROM 144 = LDA . TEMP, . WN R13, NOP
ROM 145 = ADD . XN . TEMP, R05, NOP
ROM 146 = LDA . TEMP, . WN1 R13, NOP
ROM 147 = SUB . XN . WN1 R01, CVT5
ROM 148 = ADD DAR, KM2, R00, CND6
ROM 149 = LDA . XN . XN R00, NOP
ROM 150 = LDA . XN . XN R00, NOP
ROM 151 = SUB . XN . WN1 R03, NOP
ROM 152 = SUB . XN . WN1 R04, CVT7
ROM 153 = LDA . XN . XN R00, NOP
ROM 154 = SUB . XN . WN1 R08, NOP
ROM 155 = SUB . XN . WN1 R11, CVT6
ROM 156 = SUB . XN . WN1 R13, NOP
ROM 157 = SUB . XN . TEMP, R02, NOP
ROM 158 = LDA . YN . WN1 R03, CVT5
ROM 159 = LDA . XN . XN R00, NOP
```

```
ROM 160 = SUB . YN . WN1, R05, NOP
ROM 161 = SUB . YN . WN1, R09, CVT4
ROM 162 = LDA . XN . XN, R00, NOP
ROM 163 = ADD . YN . WN1, R12, NOP
ROM 164 = LDA . TEMP, . YN, R13, CVT3
ROM 165 = SUB . YN . TEMP, R02, NOP
ROM 166 = SUB . YN . TEMP, R05, NOP
ROM 167 = ADD . YN . WN, R05, CVT2
ROM 168 = LDA . XN . XN, R00, NOP
ROM 169 = SUB . YN . WN, R03, NOP
ROM 170 = ADD . YN . WN, R09, CVT1
ROM 171 = LDA . XN . XN, R00, NOP
ROM 172 = SUB . YN . WN, R12, NOP
ROM 173 = LDA . TEMP, . WN, R13, CVT0
ROM 174 = ADD . YN . TEMP, R02, NOP
ROM 175 = ADD . YN . TEMP, R05, NOP
ROM 176 = ADD . XN, DAR, R05, NOP
ROM 177 = ADD . YN . XN, R03, NOP
ROM 178 = SUB . YN . XN, R05, NOP
ROM 179 = SUB . YN . XN, R09, NOP
ROM 180 = ADD . YN . XN, R12, NOP
ROM 181 = LDA . TEMP, . XN, R13, NOP
ROM 182 = SUB . YN . TEMP, R02, NOP
ROM 183 = SUB . YN . TEMP, R05, NOP
ROM 184 = LDA . YN . YN, L02, NOP
ROM 185 = LDA DAR . YN, L02, NOP
ROM 186 = LDA . WN1 . WN, R00, NOP
ROM 187 = LDA . WN . XN, R00, NOP
ROM 188 = LDA . XN . XN, R00, EOP
ROM 189 = LDA . XN . XN, R00, NOP
ROM 190 = LDA . XN . XN, R00, NOP
ROM 191 = LDA . XN . XN, R00, NOP
*
*
*
*;  WILL NOW SET INPUT AT  1 FOR STEP RESPONSE
*
*IN1 = 1
*
*;  NEXT THE PROGRAM DURATION WILL BE SET
*;  THIS WILL DETIRMINE THE SAMPLE FREQUENCY
*;  THE CLOCK FREQUENCY IS 2.5 MHZ
*;  192 STEPS PER PROGRAM PASS
*;  FOUR CLOCK CYCLES PER STEP
*;  PROGRAM DURATION IS 2.5 MHZ/(4 X 192)
*
*TPROG = 2500000/(4*192)
*TPROG            ;THIS WILL VERIFY
TPROG =  3.2552082E+3
*
```

```
*
*;BAD MISTAKE!  THAT WAS THE PROGRAM FREQUENCY.
*
*TPROG = 4*192/2500000
*TPROG ;VERIFY
TPROG =  0.00030720
*EVALUATE 1/TPROG ;VERIFY
 3.2552082E+3    3.2552082E+3    3.2552082E+3
*
*;ACTUAL SAMPLE FREQUENCY IS THREE TIMES AS FAST
*;BECAUSE THERE ARE THREE SAMPLES TAKEN PER PASS
*
*;EVALUATE 3/TPROG
*EVALUATE 3/TPROG
 9.7656248E+3    9.7656248E+3    9.7656248E+3
*
*;THE SAMPLE FREQUENCY IS 9.7656248 KHZ
*
*;THE BREAKPOINT WILL NOW BE SET.  THIS TELLS THE
*;COMPUTER WHEN TO STOP SIMULATING.
*
*B = NEVER  ;SIMULATION WILL NOW CONTINUE UNTIL THE USER PRESSES
*           ;THE ESCAPE KEY.
*B
BREAKPOINT = NEVER
*
*;THE INITIAL CONDITIONS MUST BE SET
*RAM 0 TO 5 = 0
*RAM 0 LEN 5    ;VERIFY INITIALIZATION AT ZERO
RAM 00 =  0.00000000
RAM 01 =  0.00000000
RAM 02 =  0.00000000
RAM 03 =  0.00000000
RAM 04 =  0.00000000
*
*DAR = 0  ;INITIALIZE DAR AT ZERO
*
*
*;SETTING TRACE TELLS THE COMPUTER WHAT DATA TO COLLECT.
*
*TRACE = T,OUT1   ;THE ONLY DATA COLLECTED AND PRINTED
*                 ;WILL BE TIME AND OUTPUT
*
*;SETTING THE QUALIFIER TELLS THE COMPUTER WHEN TO COLLECT DATA
*
*Q = PC = 7 OR PC = 71 OR PC = 135
*
*;TRACE WILL BE COLLECTED AT THE END OF EACH OUTPUT SEQUENCE
*
*;ALL READY FOR SIMULATION
```

```
*S FROM 0 ;                                STEP RESPONSE
  T                    OUT1
SIMULATION BEGUN
    0.00001120         0.00000000
    0.00011360         0.04296875
    0.00021600         0.07031250
    0.00031840         0.12890625
    0.00042080         0.20312500
    0.00052320         0.28125000
    0.00062560         0.35156250
    0.00072800         0.41406250
    0.00083040         0.46093750
    0.00093280         0.49609375
    0.00103520         0.51562500
    0.00113760         0.52734375
    0.00124000         0.52734375
    0.00134240         0.52343750
    0.00144480         0.51562500
    0.00154720         0.50781250
    0.00164960         0.49609375
    0.00175200         0.48437500
    0.00185440         0.47656250
    0.00195680         0.47265625
    0.00205920         0.46875000
    0.00216160         0.46484375
    0.00226400         0.46484375
    0.00236640         0.46484375
    0.00246880         0.46484375
    0.00257120         0.46484375
    0.00267360         0.46484375
    0.00277600         0.46875000
    0.00287840         0.46875000
    0.00298080         0.46875000
    0.00308320         0.47265625
    0.00318560         0.47265625
    0.00328800         0.47265625
    0.00339040         0.47265625
    0.00349280         0.47265625
    0.00359520         0.47265625
    0.00369760         0.47265625
    0.00380000         0.47265625
    0.00390240         0.47265625
PROCESSING ABORTED
WARN C6:NUMBER NOT BETWEEN -1 AND +1
```

```
*IN1 = -1 ;CHANGE POLARITY
*RAM 0 TO 5 = 0
*DAR = 0
*S FROM 0
   T                 OUT1
SIMULATION BEGUN
    0.00001120        0.00000000
    0.00011360       -0.04687500
    0.00021600       -0.07421875
    0.00031840       -0.13281250
    0.00042080       -0.20703125
    0.00052320       -0.28125000
    0.00062560       -0.35546875
    0.00072800       -0.41406250
    0.00083040       -0.46093750
    0.00093280       -0.49609375
    0.00103520       -0.51953125
    0.00113760       -0.52734375
    0.00124000       -0.53125000
    0.00134240       -0.52734375
    0.00144480       -0.51953125
    0.00154720       -0.50781250
    0.00164960       -0.49609375
    0.00175200       -0.48828125
    0.00185440       -0.48046875
    0.00195680       -0.47265625
    0.00205920       -0.46875000
    0.00216160       -0.46484375
    0.00226400       -0.46484375
    0.00236640       -0.46484375
    0.00246880       -0.46484375
    0.00257120       -0.46875000
    0.00267360       -0.46875000
    0.00277600       -0.46875000
    0.00287840       -0.47265625
    0.00298080       -0.47265625
    0.00308320       -0.47265625
    0.00318560       -0.47265625
    0.00328800       -0.47265625
    0.00339040       -0.47265625
    0.00349280       -0.47265625
PROCESSING ABORTED
*
*
*
*;                          SINEWAVE RESPONSE
*
*IN1 = SIN(TPI*1000*T)     ;INPUT FREQUENCY IS 1 KHZ
*
*RAM 0 TO 5 = 0
```

```
*DAR = 0
*
*;THIS WILL SHOW TRANSIENT AND STEADY STATE RESPONSE
*
*
*S FROM 0
  T                OUT1
SIMULATION BEGUN
    0.00001120      0.00000000
    0.00011360      0.00390625
    0.00021600      0.03515625
    0.00031840      0.07031250
    0.00042080      0.11718750
    0.00052320      0.16406250
    0.00062560      0.18750000
    0.00072800      0.17968750
    0.00083040      0.13671875
    0.00093280      0.06640625
    0.00103520     -0.01171875
    0.00113760     -0.07421875
    0.00124000     -0.10156250
    0.00134240     -0.08984375
    0.00144480     -0.04296875
    0.00154720      0.01171875
    0.00164960      0.05859375
    0.00175200      0.07421875
    0.00185440      0.05468750
    0.00195680      0.00781250
    0.00205920     -0.04687500
    0.00216160     -0.08593750
    0.00226400     -0.08984375
    0.00236640     -0.06250000
    0.00246880     -0.00781250
    0.00257120      0.04687500
    0.00267360      0.08593750
    0.00277600      0.08984375
    0.00287840      0.05859375
    0.00298080      0.00390625
    0.00308320     -0.05078125
    0.00318560     -0.08593750
    0.00328800     -0.08593750
    0.00339040     -0.05078125
    0.00349280      0.00390625
    0.00359520      0.05468750
    0.00369760      0.08593750
    0.00380000      0.08203125
    0.00390240      0.04296875
    0.00400480     -0.01171875
    0.00410720     -0.06250000
    0.00420960     -0.08984375
```

```
    0.00431200      -0.08203125
    0.00441440      -0.03906250
    0.00451680       0.01562500
    0.00461920       0.06640625
    0.00472160       0.08984375
    0.00482400       0.07812500
    0.00492640       0.03125000
    0.00502880      -0.02343750
    0.00513120      -0.07421875
    0.00523360      -0.08984375
    0.00533600      -0.07421875
    0.00543840      -0.02734375
    0.00554080       0.03125000
    0.00564320       0.07421875
    0.00574560       0.08984375
    0.00584800       0.06640625
    0.00595040       0.01953125
    0.00605280      -0.03906250
PROCESSING ABORTED
*
*
*
*IN1 = SIN(TPI*1627.6*T)  ;FREQUENCY OF INPUT IS 1.6276 KHZ
*                         ;THE OUTPUT SHOULD DROP TO ZERO AT THIS
*                         ;FREQUENCY.
*RAM 0 TO 5 = 0
*DAR = 0
*
*
*S FROM 0
   T                 OUT1
SIMULATION BEGUN
    0.00001120       0.00000000
    0.00011360       0.00781250
    0.00021600       0.05078125
    0.00031840       0.07031250
    0.00042080       0.08203125
    0.00052320       0.07812500
    0.00062560       0.07031250
    0.00072800       0.05468750
    0.00083040       0.04296875
    0.00093280       0.02734375
    0.00103520       0.01562500
    0.00113760       0.00390625
    0.00124000      -0.00390625
    0.00134240      -0.00781250
    0.00144480      -0.01171875
    0.00154720      -0.01171875
    0.00164960      -0.01171875
    0.00175200      -0.01171875
```

```
   0.00185440      -0.00781250
   0.00195680      -0.00781250
   0.00205920      -0.00390625
   0.00216160      -0.00390625
   0.00226400       0.00000000
   0.00236640       0.00000000
   0.00246880       0.00000000
   0.00257120       0.00000000
   0.00267360       0.00000000
   0.00277600       0.00000000
   0.00287840       0.00000000
   0.00298080       0.00000000
   0.00308320       0.00000000
   0.00318560       0.00000000
   0.00328800       0.00000000
   0.00339040       0.00000000
   0.00349280       0.00000000
   0.00359520       0.00000000
PROCESSING ABORTED
*EXIT
```

LIST OF REFERENCES

1. Daryanani, G. Principles of Active Network Synthesis and Design. New York: Wiley, 1976, 113.

2. Stanley, W. D. Digital Signal Processing. Reston, Virginia: Reston Publishing Company, 1975, 168-176.

3. 2920 Assembly Language Manual. Santa Clara, California: Intel Corporation, 1980.

4. Crochiere, R. E. "A New Statistical Approach to the Coefficient Word/Length Problem for Digital Filters." IEEE Transactions on Circuits and Systems CAS-22 (March 1975): 190-191.

5. ISIS II System User's Guide. Santa Clara, California: Intel Corporation, 1977.

6. 2920 Simulator User's Guide. Santa Clara, California: Intel Corporation, 1979.

7. Intel Component Data Catalog,(1980). Santa Clara, California: Intel Corporation, 1980.

8. Universal Prom Programer Manual. Santa Clara, California: Intel Corporation, 1980.