

30
1

**DEVELOPMENT OF EXPERIMENTS FOR THE DIGITAL SIGNAL PROCESSING
TEACHING LABORATORY**

by

KWANG-SUZ JEN

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE
in
ELECTRICAL ENGINEERING

APPROVED:



A.A. (Louis) Beer (chairman)



Ting-Chung Poon



Kai-Bor Yu

July, 1988

Blacksburg, Virginia

LD
5655
Y855
1988
J464
c.2

**To my aunt Lancy and uncle Jim
for their support**

DEVELOPMENT OF EXPERIMENTS FOR THE DIGITAL SIGNAL PROCESSING

TEACHING LABORATORY

by

KWANG-SUZ JEN

A.A. (Louis) Beex (chairman)

ELECTRICAL ENGINEERING

(ABSTRACT)

Digital Signal Processing (DSP) is a technology-driven field which develops as early as mid-1960 when computers and other digital circuitry became fast enough to process large amounts of data efficiently. Since then techniques and applications of DSP have been expanding at a tremendous rate. With the development of large-scale integration, the cost and size of digital components are reducing, and speed of digital components is increasing. Thus the range of applications of DSP techniques is growing. Almost all current discussions of speech bandwidth compression systems are directed toward digital implementation, because these are now the most practical. The importance of DSP appears to be increasing with no visible signs of saturation.

This thesis provides the description and results of designing laboratory experiments for the illustration of basic theory in the field of DSP. All experiments are written for the Texas Instruments TMS32010 digital signal processing microcomputer and based on softwares provided by Atlanta Signal Process, Inc. (ASPI). The use of the 320/pc Algorithm Development Package (ADP) and Digital Filter Design Package (DFDP) developed by ASPI is introduced. The basic concepts, such as linear convolution, Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) filter design, Fast Fourier Transform (FFT), are demonstrated. The IBM PC AT is interfaced with the TMS32010 processor. The experiments and their introductions in the thesis also serve as a manual for the DSP Laboratory; to complement the introductory signal processing course.

Acknowledgements

I thank my advisor, Dr. A.A. (Louis) Beex, for his skillful guidance and encouragement during my thesis work. I have greatly enjoyed the opportunity to work independently and learn from him.

I am grateful to Dr. Ting-Chung Poon and Dr. Kai-Bor Yu for being my committee members.

Most importantly, I would like to take this opportunity to thank my aunt Lancy C. Jen and my uncle Jim Jen for their support and encouragement, without their support this degree would not be possible. This thesis is dedicated to them. And also, I want to thank my parents, for their understanding, loving, and caring.

Recognition is extended to my friend, Dr. Richard P.E. Tymerski, who has enriched my stay in Blacksburg.

I appreciate the help from Professor Y. A. Liu in the Department of Chemical Engineering for the initial editing of this thesis.

It has been a pleasure to work with the excellent faculty, staff, and students at Virginia Polytechnic Institute and State University. I thank many of them who, in their own way, have enriched my educational experience.

Table of Contents

1. INTRODUCTION	1
1.1 Background Information	2
1.2 Preview of Material Included in this Thesis	4
2. LINEAR CONVOLUTION	6
2.1 EXPERIMENT I	10
2.1.1 Programs in the ADP	10
2.1.1.1 TI_LOAD	10
2.1.1.2 BUG	13
2.1.2 Experimental Procedure I.1	19
2.1.3 Experimental Procedure I.2	23
2.1.4 Experimental Procedure I.3	25
3. NOTCH FILTER DESIGN	26
3.1 EXPERIMENT II	29
3.1.1 Noise Rejection Using Notch Filter	31
3.1.2 Analysis and Design the Notch Filter	32

3.1.3 Implementation of the Notch Filter	34
3.1.3.1 Experimental Procedure II.1	34
3.1.3.2 Experimental Procedure II.2	36
4. ALIASING	38
4.1 EXPERIMENT III	40
4.1.1 TMS32010 MS/PC-DOS CrossWare	40
4.1.2 Link Editor	42
4.1.3 PATCH	44
4.1.3.1 Program Switch	44
4.1.3.2 Patching A Real-time Program	45
4.1.3.3 Set Sampling Rate (S)	45
4.1.3.4 Patch Message (L)	46
4.1.3.5 External File Input (X)	46
4.1.3.6 Direct Patch (#)	48
4.1.3.7 Measuring TMS32010 Efficiency	48
4.1.4 The Model 3202 Filter	49
4.1.4.1 Frequency Dials	50
4.1.4.2 Frequency Range	50
4.1.4.3 Cutoff-Frequency Calibration Accuracy	50
4.1.4.4 Bandwidth	50
4.1.4.5 Response Characteristics	51
4.1.4.6 Panel Control	52
4.1.4.7 Operation	52
4.1.4.8 Phase Response	55
4.1.5 Experimental Procedure III.1	57
4.1.6 Experimental Procedure III.2	60

5. FIR AND IIR FILTER DESIGN	62
5.1 EXPERIMENT IV	65
5.1.1 Programs in the DFDP	65
5.1.2 To Execute the Software Package	66
5.1.3 Instruction for use of the IIR Design Module	68
5.1.3.1 Executing program IIR	68
5.1.3.2 IIR Main Menu	68
5.1.3.3 The Sampling Rate	71
5.1.3.4 The Cutoff Frequencies	71
5.1.3.5 Approximation Errors	71
5.1.3.6 Selection of Approximation Type	72
5.1.3.6 Coefficient Quantization	72
5.1.3.7 Verification of the Characteristics of the Designed Filter	73
5.1.3.8 Manipulation and Plotting of Designed Filter	73
5.1.3.9 Capabilities and Limitations of the program IIR	76
5.1.4 Experimental Procedure IV.1	78
5.1.5 Instruction For Use of FIR Design Module	79
5.1.5.1 Executing the program KFIR	79
5.1.5.2 KFIR Main Menu	79
5.1.5.3 Specification of Filter Parameters	81
5.1.5.4 Coefficient Quantization	81
5.1.5.5 Verification of Characteristics of Designed Filter	81
5.1.5.6 Manipulation and Plotting of the Designed Filter	81
5.1.5.7 Capabilities and Limitations of KFIR	82
5.1.6 Experimental Procedure IV.2	83
5.1.7 Experimental Procedure IV.3	84
6. FAST FOURIER TRANSFORM	85

6.1 EXPERIMENT V	89
6.1.1 HEAR	90
6.1.1.1 Program Switches	90
6.1.1.2 File and Parameter Labels	90
6.1.1.3 A-to-D Mode - Setting Input Signal Levels	93
6.1.1.4 A-to-D Mode - Capturing A Signal	93
6.1.1.5 D-to-A Mode	95
6.1.2 Experimental Procedure V.1	96
6.1.3 Experimental Procedure V.2	97
6.1.4 Experimental Procedure V.3	98
6.1.5 Experimental Procedure V.4	99
7. DIGITAL FILTER DESIGN	101
7.1 EXPERIMENT VI	103
7.1.1 Filter Specification	103
7.1.2 Experimental Procedure VI.1	104
7.1.3 Experimental Procedure VI.2	105
7.1.4 Experimental Procedure VI.3	106
REFERENCES	107
Appendix A. Program Source Code List	109
Linear convolution	109
Appendix B. Program Source Code List	111
Exchange data between PC and the TMS32010	111
Appendix C. Program Source Code List	112

Interface with the TMS32010 112

Appendix D. Program Source Code List **113**

Fast Fourier Transform 113

VITA **116**

List of Illustrations

Figure 1. The flowchart of linear convolution program.	8
Figure 2. The format of the TI_LOAD command line.	12
Figure 3. The initial form of the command screen.	14
Figure 4. Typical display during EDIT BREAKPOINTS command.	16
Figure 5. Typical display during INV ASSEMBLE command.	18
Figure 6. The unit pulse response.	20
Figure 7. Typical example of a PATCH screen.	47
Figure 8. Normalized phase characteristics.	56
Figure 9. Characteristics of designed KFIR filter.	58
Figure 10. System diagram of digital filtering of an analog signal.	63
Figure 11. The menu of DFDP.	67
Figure 12. IIR bilinear transformation design program.	69
Figure 13. IIR bilinear transformation main menu.	70
Figure 14. Manipulation and plotting designed filter.	74
Figure 15. Kaiser-window main menu.	80
Figure 16. Diagram of four-point decimation-in-frequency FFT.	88
Figure 17. Screen display in initializing stage.	92
Figure 18. Screen display during the A-to-D stage.	94

1. INTRODUCTION

The purpose of the Digital Signal Processing (DSP) Laboratory is to help the student (a) get acquainted with the use of the IBM Personal Computer AT; (b) learn basic techniques in the design and evaluation of DSP algorithms in a real-time environment; (c) experimentally verify some of the theory he/she has learned, and (d) experience how inter-processor communication (IBM PC AT and TMS32010) can be done. The experiments in this thesis are designed toward this purpose.

1.1 Background Information

The features of the 320/PC Algorithm Development Package and Digital Filter Design Package (DFDP) from Atlanta Signal Processors Incorporated (ASPI) are described. The 320/PC Algorithm Development Package (ADP) is aimed at applications involving the TMS32010 digital signal processing (DSP) microcomputer from Texas Instruments Incorporated, and includes the 320/pc board and associated software for the IBM PC AT. The software system is developed for three types of application in conjunction with the Personal Computer/TMS32010. First, the system can be used as a program development tool for realizing DSP applications on TMS32010-based systems, using ASPI software products such as DFDP. Second, the system is useful as a signal processing workstation for interactive algorithm development. Finally, the system can be utilized directly to run programs in real time, such as one of ASPI's voice coding programs specifically written to take advantage of capabilities of the 320/PC board.

DFDP is a powerful package of interactive programs for the design of digital filters as well as the subsequent implementation of such filters on the Texas Instruments TMS32010 DSP microcomputer. DFDP has modules for the design of recursive and nonrecursive digital filters. Recursive or IIR filters are designed by the module IIR which uses the method of bilinear transformation of Butterworth, Chebyshev, and Cauer (elliptic) analog filters. Nonrecursive or FIR filters are designed using either module KFIR, which uses the Kaiser-window method [1], or the design module PMFIR, which uses the Parks-McClellan algorithm [2]. In each case, a wide variety of frequency-selective filters can be designed; in the FIR case, differentiators and Hilbert transformers can be designed also. A unique feature of the software package is module CGEN, which produces assembly language code for filters designed by modules IIR, KFIR, or PMFIR [3].

The 320/PC ADP is a powerful combination of hardware and software for developing and implementing DSP algorithms for the Texas Instruments TMS32010 DSP microcomputer. The

package is based on the 320/PC board, which contains a TMS32010 processor, an 8-kilobyte dual-port memory, a programmable clock, and very high-quality 12-bit A/D and 12-bit D/A conversion systems. Because of the unique dual-port memory architecture, the 320/PC can be used as a real-time development and debugging tool of unprecedented performance [4]. Because of its high-quality A/D and D/A systems, and its high computational capacity, the 320/PC board alone can turn a personal computer into a powerful signal processing workstation. It also can make a personal computer into a real-time speech coder, an audioband spectrum analyzer, a voice mail station, a signal generator, a real-time digital filter, a digital equalizer, or a signal conditioner.

A variety of useful software is available to facilitate the use of the 320/PC board in the ADP. These programs include: HEAR, which samples, stores, reconstructs, and edits signals using A/D and D/A subsystems; VIEW, which displays and edits a sampled waveform from data files stored on the system disk; BUG, a multi-breakpoint debugger for programs being executed in the TMS32010; PATCH, a real-time program for interactively patching a program while executing on the TMS32010 processor and for monitoring the efficiency of the TMS32010 in a real-time operation; and TI_LOAD, a utility program for loading programs into the TMS32010's programming memory and controlling the state of the 320/PC board [5].

1.2 Preview of Material Included in this Thesis

This thesis serves as a manual for the DSP Laboratory course accompanying the course "Introduction to Signal Processing". The experiments are written in the order of the material covered in "Digital Filters and Signal Processing" by Leland B. Jackson [1]. Since most of the experiments must be done in a fixed period of time, new software, laboratory instruments and techniques are introduced in every experiment except the last one, experiment VII. Some software such as the TMS32010 MS/PC-DOS CrossWare and link editor will not be explained until Chapter 4, Section III.1 and III.2. The following paragraphs summarize the contents of this thesis.

Chapter 2 presents the use of some software associated with the ADP. The linear convolution of an impulse response with an input from a specified data memory location in the TMS32010 processor is implemented and verified. This TMS32010 assembly program can serve as a FIR filter when the digital filter is represented by its impulse response. A simple notch filter is designed and implemented, and subsequently run in real time.

Chapter 3 discusses how inter-processor communications (IBM PC AT and the TMS32010) can be done. Digital filters (FIR and IIR) are designed to demonstrate and enhance the theory that students are learning.

Chapter 4 shows the use of the TMS32010 MS/PC-DOS CrossWare and link editor, and Model 3202 solid-state variable-frequency filters. Aliasing is demonstrated with an input from the external waveform generator. Next we show how aliasing can be reduced with the aid of Model 3202 filters. The phase response of the filter is measured with an oscilloscope. The utilization of the TMS32010 processor, under real-time operating conditions is measured.

Chapter 5 describes the procedure of using the DFDP as well as a discussion of the limitation of the DFDP.

Chapter 6 discusses the Fast Fourier Transform (FFT) (which is implemented in both FORTRAN and the TMS32010 assembly language) of an input signal which is created from a FORTRAN program or obtained directly from the A/D converter. The purpose of zero padding is demonstrated. The FFT result is provided by the FORTRAN program or the TMS32010.

Chapter 7 gives an opportunity to design digital filters according to specifications. Infinite Impulse Response (IIR) filters are designed by using a bilinear transformation of analog filters; Finite Impulse Response (FIR) filters are designed using different windowing methods.

The experiments are intended to be completed within a specific time period, but several require reading/preparation prior to performing the experiment. It is important that each experiment be performed with care and thought, not just to obtain data and write a report, but rather to become familiar with the principle demonstrated and the techniques employed.

2. LINEAR CONVOLUTION

Convolution is one of the most frequently used signal processing operations. Linear convolution of sequences, one of which is of finite length, can be implemented as an FIR filter with an N -sequence unit pulse response, driven by the other sequence as an input. In spectral estimation, the basic operation of autocorrelation is simply the convolution of the signal with a reversed-time version of itself.

LCONV.ASM is a TMS32010 assembly program that implements linear convolution defined as follows

$$y(n) = \sum_{m=0}^{N-1} x(n-m)h(m). \quad (2-1)$$

where $y(n)$ is the result of the linear convolution of the unit pulse response $h(m)$ and an input sequence $x(n)$. N is the length of the unit pulse response. The input sequence x_n is convolved with the permanent sequence h_n . In this program, the latter is an 8-point sequence with the first four points equal to one and the remainder equal to zero. The input is read sequentially from port 6 (A/D), and the output is written to port 6 (D/A). The program will load the unit pulse response at location 0009 through 000F and initialize the input function (that is, $x_{n-1}, x_{n-2}, \dots, x_{-1}$) to zero

at location 0001 through 0008. Students are asked to check both locations to ensure that the unit pulse response has been loaded and that the input function has been initialized. The program is then activated into its first loop, where a new data point is received from port 6 (A/D). In experimental procedures I.1 and I.2, students will input data points of a specific function to location 0001, one at a time, to simulate time domain sampling. Finally, the program goes into the second loop where the main computation takes place.

The main computation in the convolution is done using one of the most powerful instruction pairs, LTD and MPY, in the TMS32010 assembly language, which multiplies the sequences point by point and shifts the input in preparation for the next input data point. Students will have an opportunity to see exactly how the linear convolution works. Experimental procedure I.1 serves this purpose. The result will be at location 0011 and it will output to port 6 (D/A). The real-time signal processing result can be viewed on an oscilloscope connected to port 6 (D/A). Experimental procedure I.3 demonstrates this real-time signal processing. After presenting the result, the program goes back to the first loop to get the next input data point. The flowchart of this program is shown in Figure 1.

Since LCONV.LOD (an object file generated by the TMS32010 MS/PC-DOS CrossWare and link editor, which will be discussed in Chapter 4) is downloaded with three switches (flags, parameters) S, G and B (TI_LOAD program switches will be discussed later in this chapter).

BUG is the TMS32010 debugger utility supplied with the 320/pc Algorithm Development Package (ADP). Because of the unique architectural features of the 320/pc board, BUG can perform many functions not normally available in a debugging package. For example, programs and data can be loaded and extracted from the dual-port memory, while the TMS32010 is executing. There is, however, a problem associated with multiple breakpoints; that is, after a breakpoint is reached several times within a program loop, the data-page pointer changes from 0 to 1. (Usually the second page of data memory contains infrequently accessed system variables, such as those used by the interrupt routine). Once the data-page pointer is changed, BUG is no

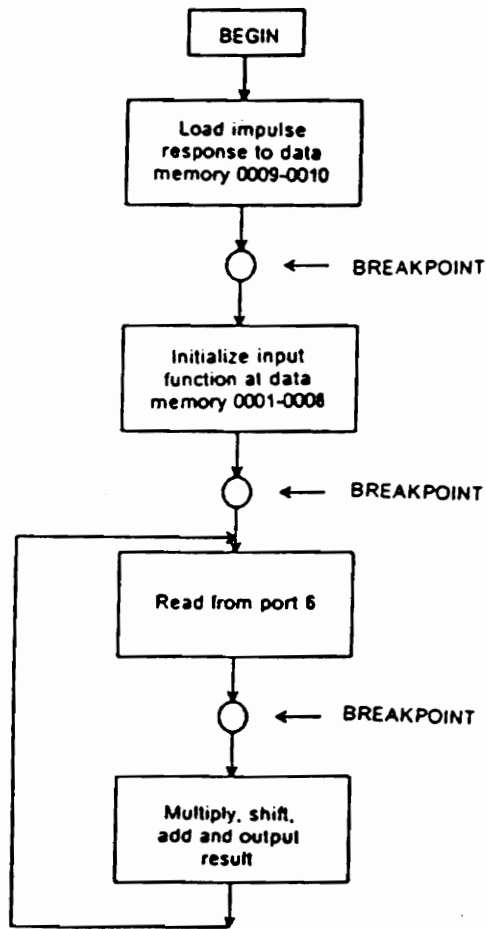
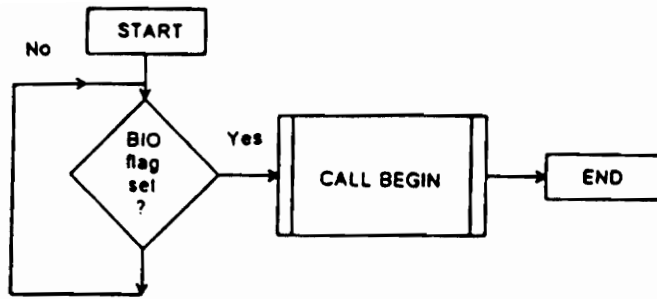


Figure 1. The flowchart of linear convolution program.

longer looking at the same program as before. Near the end of LCONV.ASM, there is a statement LDPK to set the data-memory page pointer register to 0, which is not needed in this program if there is no breakpoint set within the loop.

The TMS32010 has a 32-bit ALU/accumulator and a 16-bit instruction/data word. A Q15 format [10] for number representation is assumed for all data and coefficient values. The format involves one sign bit plus 15 fractional bits and the absolute value of all represented numbers is less than two. It simplifies calculations in a fixed-point machine such as the TMS32010 digital signal processor. In other words, all the input data which have been left-shifted for 14 bits must be within ± 2 in decimal.

The LCONV.ASM program can also function as an FIR filter if the permanent sequence represents the unit pulse response of a filter. In experimental procedure I.2, students will find the unit pulse response of a notch filter and its hexadecimal representation. The hexadecimal value will be found from the absolute value discussed above. The unit pulse response of the notch filter is loaded into location 0009 through 0010 by using the function key for editing the data memory. This experiment shows students how to use an FIR filter for linear convolution. The filtering is demonstrated by entering data point by point at location 0001 and recording the result of equation (2-1) from 0011. The characteristic of a notch filter is also demonstrated by using two sinusoidal signals with different frequencies as input. One of these frequencies is the notch frequency. In experimental procedure I.3, this notch filter is demonstrated in real-time.

2.1 EXPERIMENT I

The purpose of this experiment is to acquaint the student with some of the software, principles, and pitfalls associated with the Algorithm Development Package (ADP). Programs `TI_LOAD`, and `BUG` are studied. The linear convolution of a finite length unit pulse response with an input from a specified data memory location, is implemented and verified.

2.1.1 Programs in the ADP

- `TI_LOAD`: To down-load the unit pulse response program (`LCONV.LOD`) to the TMS32010.
- `PATCH` : To measure the utilization of the TMS32010 processor in real time, and to install TMS32010 program patches, such as for changing the sampling rate.
- `BUG` : To start the program in a specific location, in order to demonstrate the linear convolution.

2.1.1.1 TI_LOAD

The purpose of `TI_LOAD` is to load the TMS32010 program, which has been assembled and linked by the TMS32010 Assembler and Linker, onto the 320/PC board for execution.

`TI_LOAD` is the 320/PC utility program, downloading programs and data into the program memory of the TMS32010 processor. It can also set the state of the 320/PC processor and the rate of the sampling clock. If the program name "`TI_LOAD`" is typed without arguments, then the

program will type as Figure 2 to remind the user of the format of the command line. The first letter of the switch names is used by TI_LOAD. Note that all programs in this manual are under the control of the BIO flag.

To down-load the program, type the following command line:

TI_LOAD[/S]/G[/I]/B[/C] INPUTFILE SAMPLING-RATE

For example:

TI_LOAD/S/G/B LCON 8000. (2 – 2)

This TI_LOAD command contains switches (i.e. /S), and file labels (i.e. LCON and 8000).

I.1.1.1 Program Switches

/Stop (Abbreviation /S as in 2-2). Stop TMS32010 before loading. This is accomplished by holding the TMS32010 processor in a reset condition.

/Go (Abbreviation /G as in 2-2). Start TMS32010 after loading. This is accomplished by releasing the TMS32010 processor from a reset condition.

/Interrupt Enable external interrupts (XINT). This has the effect of closing the XINT switch so that the TMS32010 interrupt input (INT) is connected to the external source.

/BIO (Abbreviation /B as in 2-2). Enable the external flag (XBIO). This has the effect of closing the XBIO switch so that the TMS32010 I/O flag input (BIO) is connected to the external source.

320/PC DOWNLOADING UTILITY
(C) COPYRIGHT 1985: ATLANTA SIGNAL PROCESSORS INC -- VERSION 1.0

TI_LOAD[/STOP[/GO[/INTERRUPT[/BIO] FILENAME [SAMPLING-RATE]

STOP: Stop (reset) TMS32010 processor before load
GO: Start (relebase reset) TMS32010 processor after load
INTERRUPT: Enable external interrupt (XINT)
BIO: Enable external I/O flags (XBIO)
FILENAME: Filename (.LOD) of program to be loaded
SAMPLING-RATE: Sampling rate in Hertz

Figure 2. The format of the TI_LOAD command line.

/Clock Enable interrupts from the programmable clock (XCLK). This has the effect of closing the XCLK switch so that the TMS32010 interrupt input (INT) is connected to the output of the programmable clock.

2.1.1.2 File Labels

INPUTFILE This is the file to be loaded into TMS32010 program memory for execution; if no extension is given, the .LOD extension is assumed.

SAMPLING-RATE A numerical value for the sampling rate in Hertz. The maximum sampling rate is 30 kiloHertz.

2.1.1.2 *BUG*

BUG is a highly interactive debugging tool which allows the user to effectively, and dynamically, access all the features of the 320/PC. The debugger features include a line-by-line symbolic assembler, an inverse assembler, multiple breakpoints, and a single-step operation feature. The user also can directly edit the registers, data memory, and program memory of the TMS32010 processor.

To execute the program, type **BUG**. Figure 3 shows the command screen that will appear as soon as **BUG** is initiated. The screen is divided into three sub-screens: command menu, command-data area and status line.

Only eight of the Bug commands are used in this experiment, and these commands are described below.

EDIT REGISTER (F1): The purpose of the **EDIT REGISTER** command is to edit an internal register of the TMS32010 processor. Since the intent is to change the current TMS32010 register

F1	EDIT REGISTERS	F6	ASSEMBLE	S1	CHANGE XBIO	S6	GO
F2	EDIT DATA MEMORY	F7	INV ASSEMBLE	S2	CHANGE BIO	S7	STOP
F3	EDIT PROGRAM MEMORY	F8	LOD IN	S3	CHANGE XINT	S8	RESET
F4	EDIT BREAKPOINTS	F9	BIN IN	S4	CHANGE XCLK	S9	SINGLE STEP
F5	EDIT CLOCK/PROCESSOR	F10	BIN OUT	S5	CHANGE INT	S10	EXIT

001 BRKEN RESET XBIO CLR BIO CLR XINT CLR XCLK CLR INT CLR

Figure 3. The initial form of the command screen.

values, this command only operates when the processor is STOPPED or RESET. If this command is requested while the processor is RUNNING, an error message will appear.

EDIT DATA MEMORY (F2): The purpose of the EDIT DATA MEMORY command is to display and modify the data memory of the TMS32010 processor. Since the data memory is internal to the TMS32010 processor chip, the processor must either be STOPPED or RESET to use this command. If the processor is RUNNING when this command is initiated, an error message will be displayed.

The command begins by requesting a data-memory address in hexadecimal. Once that number has been entered by the user, a total of 64 data-memory locations is displayed, beginning at the specified address. The left- most column of the numbers is the address of the first memory location in the row.

The user modifies the contents of the data-memory location by editing the numbers shown on the display. To change an entry, first move the cursor to the number to be changed. The fields are changed by the "Up-Arrow" and "Down-Arrow" keys, while the position within the number is changed by the "Left-Arrow" and "Right-Arrow" keys. Once the number has been correctly modified, the "Enter" key must be pressed. (The register value will not change until the "Enter" key has been pressed).

The user can exit from the EDIT DATA MEMORY command either by exiting from the end of the register display or by pressing a function key to initiate another command.

EDIT BREAKPOINTS (F4): The purpose of the EDIT BREAKPOINTS command is to display and modify the list of breakpoints. There can be a maximum of ten breakpoints in the program at any time. Since 0000 denotes no breakpoint, a breakpoint is set by entering the desired location for a breakpoint, and pressing the Enter key. An example breakpoint is shown in Figure 4.

F1	EDIT REGISTERS	F6	ASSEMBLE	S1	CHANGE XBIO	S6	GO
F2	EDIT DATA MEMORY	F7	INV ASSEMBLE	S2	CHANGE BIO	S7	STOP
F3	EDIT PROGRAM MEMORY	F8	LOD IN	S3	CHANGE XINT	S8	RESET
F4	EDIT BREAKPOINTS	F9	BIN IN	S4	CHANGE XCLK	S9	SINGLE STEP
F5	EDIT CLOCK/PROCESSOR	F10	BIN OUT	S5	CHANGE INT	S10	EXIT

MODIFY BREAKPOINTS

BREAKPOINT 1: 0090
BREAKPOINT 2: 0A10
BREAKPOINT 3: 0020
BREAKPOINT 4: 0000
BREAKPOINT 5: 0000
BREAKPOINT 6: 0000
BREAKPOINT 7: 0000
BREAKPOINT 8: 0000

001 BRKEN RESET XBIO CLR BIO CLR XINT CLR XCLK CLR INT CLR

Figure 4. Typical display during EDIT BREAKPOINTS command.

GO (S6) (SHIFT F6): The GO command serves to start the TMS32010 processor. If the processor is in the STOPPED or RESET state prior to issuance of the GO command, the TMS32010 processor is started and goes into the RUNNING condition, the GO command does nothing but check for a BROKEN condition caused by a breakpoint. If a breakpoint has occurred, the TMS32010 is STOPPED, and the EDIT REGISTER command is entered automatically.

INV ASSEMBLE (F7): The INV ASSEMBLE initiates the inverse assembler. After this command is initiated, it prompts the user for a starting address in hexadecimal. The starting address is preset to the current value of the program counter. Once the starting address is chosen, a total of 48 inverse assembled instructions is display in three columns. An example is shown in Figure 5.

STOP (S7) (SHIFT F7): The STOP command terminates TMS32010 processing. It has no effect if the processor is in the RESET mode. If the processor is RUNNING, it is forced into the STOPPED mode.

SINGLE STEP (S9) (SHIFT F9): The SINGLE STEP command is to execute a single instruction on the TMS32010 processor. The address of the instruction being executed is found in the program counter register. To operate the SINGLE STEP command, the processor must be in either the STOPPED or RESET mode. When the SINGLE STEP instruction has been completed, BUG automatically executes an EDIT REGISTER command. From this command, the user can easily see the results of the single-step operation, change any register, and initiate another SINGLE STEP operation. Note that memory locations 0 and 128 are set to zero during the single-step operation.

EXIT (S10) (SHIFT F10): The EXIT command terminates BUG and returns control to the operating system.

F1	EDIT REGISTERS	F6	ASSEMBLE	S1	CHANGE XBIO	S6	GO
F2	EDIT DATA MEMORY	F7	INV ASSEMBLE	S2	CHANGE BIO	S7	STOP
F3	EDIT PROGRAM MEMORY	F8	LOD IN	S3	CHANGE XINT	S8	RESET
F4	EDIT BREAKPOINTS	F9	BIN IN	S4	CHANGE XCLK	S9	SINGLE STEP
F5	EDIT CLOCK/PROCESSOR	F10	BIN OUT	S5	CHANGE INT	S10	EXIT

0000:	MPYK 07F8	0010:	IN *+,2	0020:	???? (F8F0)
0001:	SACL *+,7,0	0011:	LTA *+*-	0021:	SUB *+,C,1
0002:	LAR 0,68	0012:	MPYK 07C4	0022:	MPYK 0A0C
0003:	MPYK 1E73	0013:	???? (F303)	0023:	LARK 2,75
0004:	???? (D7EC)	0014:	MPYK 08C0	0024:	???? (F904)
0005:	SUBS 13	0015:	???? (FD31)	0025:	MPYK 13AB
0006:	SACH *-,7	0016:	SAR 4,1D	0026:	SUB 44,0
0007:	MPYK 11F3	0017:	SAR 7,32	0027:	???? (E38B)
0008:	LT *-	0018:	OUT *,1,0	0028:	???? (FAAF)
0009:	SACL *+*-,4,1	0019:	MPYK 136A	0029:	ADD *+*-,3,1
000A:	???? (01FC)	001A:	MPYK 1EEE	002A:	MPYK 0A40
000B:	LARK 5,AB	001B:	LAR 4,7F	002B:	BGEZ 60EF
000C:	???? (F7B3)	001C:	???? (03BD)	002D:	SAR 6,31
000D:	SUB *+*-,1,0	001D:	ADD *,C	002E:	???? (DFD6)
000E:	MPYK 13DC	001E:	???? (F38E)	002F:	SACL 72,5
000F:	ADDS 57	001F:	OR 02	0030:	BGEZ A41F

001 BRKEN RESET XBIO CLR BIO CLR XINT CLR XCLK CLR INT CLR

Figure 5. Typical display during INV ASSEMBLE command.

2.1.2 Experimental Procedure I.1

1. LCONV.ASM is a TMS320 assembly program for an 8-point-length linear convolution. The unit pulse response is depicted in Figure 6. Copy LCONV.ASM to EXPI.ASM by typing "COPY LCONV.ASM EXPI.ASM".
2. Assemble and link EXPI by typing "XASM3 EXPI;" and "LINKER EXPI;", respectively. Print EXPI.LST, a listing file. Load EXPI.LOD to the 320/PC board with a specified sampling rate by using TI_LOAD.
3. Change the directory to BUG and execute the BUG program. Use INV ASSEMBLE (F7) to find the starting location with the aid of the EXPI.LST listing file that you printed.
4. Use EDIT REGISTER (F1) to set the program counter at the beginning of the linear convolution and use SINGLE STEP (S9) to that location.
5. Set breakpoints at locations 00A9, 00AE, and 00B0. Use the GO (S6) command to run the program. Note that the GO (S6) command can be used only after EDIT REGISTER (F1) is used.
6. Check the data memory at locations 0009 through 000F by using EDIT DATA MEMORY (F2) after the break at 00A9. These locations should contain the impulse response. If not, restart with the beginning of the linear convolution as the starting position. Note that the values of the unit pulse response have been left-shifted 14 bits.
7. Use GO (S6) to run the program and check data-memory locations 0001 through 0008 by using EDIT DATA MEMORY (F2) after the break at 00AE. The data in location 0001 through 0008 should initialize the input function $x_{n-1}, x_{n-2}, \dots, x_{-1}$ to zero. If not, use GO (S6) again starting with location 00A9.

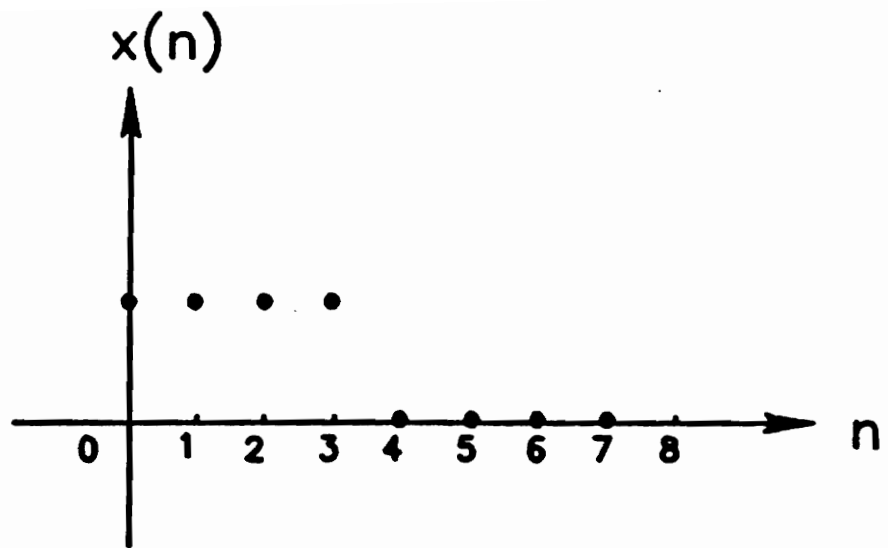


Figure 6. The unit pulse response.

8. Use GO (S6) with starting location 00AE; a break at 00B0 should occur. Use EDIT DATA MEMORY (F2) to enter the first point of the input function into location 0001. Make sure that the input point data is always in location 0001 and Enter one data point at a time. The input function is as follows:

$$x(n) = 1 \quad 0 \leq n \leq 3$$

$$x(n) = 0 \quad 4 \leq n \leq 7$$

Contents of the data memory must be in hexadecimal. The TMS32010 has a 32-bit ALU/accumulator and a 16-bit instruction/data word. Data can only be output from either the upper part, the most significant 16 bits, or the lower part, the least significant 16 bits. In order to maximize the usage of the 32-bit ALU/accumulator, and to ensure data output from the accumulator, the data words have been left shifted 14 bits. The input data and the result will be between -2 and 2. For example: decimal 1 can be represented by 4000 in hexadecimal.

9. Use EDIT REGISTER (F1) to ensure that the input data are entered, then use GO (S6) to run the program.
10. After the program breaks at 00B0, use EDIT DATA MEMORY (F2) starting at location 0001, to check where the previous data are stored, and to enter the second data point into location 0001. Record the result y_0 from location 0011. Note that the result in location 0011 may not be the value you "expect"; there is a (computable/known) scaling factor.
11. Repeat step 10 until the last data point is entered and use GO (S6) to get the last result y_7 . Draw the resulting wave form. Is this what you expect? Explain.
12. Enter the new input function

$$x(n) = 1 \quad 0 \leq n \leq 1$$

$$x(n) = 0 \quad 2 \leq n \leq 7$$

point-by-point as in steps 7 through 9. Record the result and draw the resulting wave form. Is it different from the result obtained from the first input function? What is the significance of the present result?

13. Use EXIT (S10) to terminate BUG.

2.1.3 Experimental Procedure I.2

1. EXPI.ASM can be used as a digital filter if the fixed sequence represents a unit pulse response.

Suppose that we have a notch filter with a unit pulse response

$$h(n) = \delta(n) - 2 \cos \theta \delta(n-1) + \delta(n-2), \quad \theta = 45^\circ.$$

2. The input function is as follows:

$$x(n) = \sin \frac{2}{3} \omega_o n T \quad 0 \leq n \leq 12$$

where $\omega_o = \pi$ and $T = \frac{1}{4}$.

3. In order to implement the linear convolution with the unit pulse response $h(n)$, both the unit pulse response $h(n)$ and the input function $x(n)$ should be converted to hexadecimal form. Note that for good results, six digits after the decimal point should be carried. (5 decimal digits equal to 16 bits approximately.)
4. The hexadecimal form can be obtained from either a calculator or computer program.
 - a. Under the DOS operating system, type BASICA. Load the TEST.BAS program; list the program and run it. Input $h(n)$ and $x(n)$ in decimal, and record the result (which must be in hexadecimal).
 - b. Return to the DOS operating system.
5. Change the directory to BUG, and execute BUG.

6. Find the beginning of the linear convolution as in experimental procedure I.1, and use SINGLE STEP (S9) to start the program in that location.
7. Use EDIT BREAKPOINTS (F4) to set a break point at location 00B0, and use GO (S6) to run the program.
8. A break should occur at location 00B0. Use EDIT DATA MEMORY (F2) to input the unit pulse response in hexadecimal into locations 0009 through 0010, and enter the first data point of x_n into location 0001.
9. Use GO (S6) to run the program again. A break should occur at location 00B0. Record the result from location 0011 and enter the next input data point into location 0001.
10. Repeat the above step until all input-data points have been entered and the results have been recorded. What do you observe from the result?
11. Set the program counter at the beginning of the linear convolution. Repeat steps 4 through 11 with the following input function:

$$x(n) = \sin \omega_o n T \quad 0 \leq n \leq 8$$

where $\omega_o = \pi$ and $T = \frac{1}{4}$. Explain any difference between the present and the former result.

12. Use EXIT (S10) to terminate BUG.

2.1.4 Experimental Procedure I.3

1. Connect the waveform generator to the input of the 320/pc board (lower BNC connector) and one channel of an oscilloscope, and connect the output of the 320/pc board (upper BNC connector) to the other channel of the oscilloscope.
2. Use EDLIN or DVED to edit EXPI.ASM. Note that DVED.COM has an on-line manual.
3. Change the unit pulse response to that for a notch filter, that is, input the coefficients of a notch filter into the unit pulse-response section in the EXPI.ASM, and keep the same unit pulse-response length as before. Take off the command star (*) in front of the instruction "CALL OUT" at the end of the program.
4. Assemble and link as in experimental procedure I.1. Down-load to the TMS32010 with 16 kHz sampling rate.
5. Turn on both the oscilloscope and the waveform generator.
6. Vary the frequency in the range from 1 to 20 kHz and sketch the magnitude response of the filter. Explain.

3. NOTCH FILTER DESIGN

Finite-Impulse-Response (FIR) and Infinite-Impulse-Response (IIR) filters form two classes of digital filter. The unit pulse response of the FIR filter has nonzero values only for a finite duration. Usually, FIR filters are implemented nonrecursively. In contrast with FIR filters, IIR filters are always implemented recursively, i.e. with feedback and their impulse responses have nonzero values for an infinite duration.

The characteristics of both FIR and IIR filters can be demonstrated with two different transfer functions, each representing a second order filter.

$$H_1(z) = 1 - 2 \cos \theta z^{-1} + z^{-2} \quad (3 - 1)$$

and

$$H_2(z) = \frac{H_1(z)}{1 - 2r \cos \theta z^{-1} + r^2 z^{-2}} \quad (3 - 2)$$

where the pole radius, r , is slightly less than unity. $H_1(z)$ and $H_2(z)$ are system functions of FIR and IIR filters, respectively.

In the first part of experiment II, students investigate the relationship between the input and output of a digital filter, including the unit pulse response, the system function, and the difference equation. Students are required to modify the FIR and/or IIR filter(s) so that the output is closer to the desired signal. From the viewpoint of system stability, all the poles of a system function must be inside the unit circle in the z -plane, i.e. $0 < r < 1$. Students can modify the IIR notch filter by simply varying the value of r and comparing the magnitude and phase characteristics. When $r > 1$, the system is unstable. In experimental procedure II.2, this fact is demonstrated.

D CONV.ASM is a TMS32010 assembly language program which implements the FIR notch filter. D CONV.ASM is the same as L CONV.ASM in experiment II.2, except that data points are read from port 7 (PC dual-port memory) instead of from port 6 (A/D). The results will be in an individual file which is stored on the hard disk. Real-time filtering can also be demonstrated by changing the input data from port 7 to port 6 (A/D). Students are asked to find notch frequencies for both the FIR and the IIR filter running in real-time. DIIR.ASM is another TMS32010 assembly language program which implements the IIR filter. An IIR filter (recursive filter) can be described as

$$y(n) = \sum_{m=0}^M b_m x(n-m) - \sum_{k=1}^N a_k y(n-k) \quad (3-3)$$

where the b_m and a_k are constant coefficients. The expression shows that the present output value $y(n)$ can be computed from the present and M past input values and N past output values. D CONV.ASM implements an FIR filter described by

$$y(n) = \sum_{m=0}^N h(n-m)x(m) \quad (3-4)$$

This expression is the same as the first part of the definition of an IIR filter. The second part of an IIR filter is simply the same as the first part with b_k being substituted for a_k and $x(n-k)$ being

substituted for $y(n - k)$, and with the index number starting at 1. DIIR.ASM is a modified version of DCONV.ASM.

The second part of the experiment demonstrates how inter-processor (between IBM PC AT and TMS32010) communication can be achieved. The heart of the 320/PC board is the dual-port memory. The dual port memory can be accessed by both processors simultaneously without causing wait states with both processors running. This makes the dual-port memory a uniquely powerful structure for debugging real-time TMS32010 implementations as well as an extremely flexible medium for inter-processor communication.

Experimental procedure II.1 is about FIR filtering. It shows students how the TMS32010 processor can implement an FIR filter, and how it can be run in both real-time and as a FORTRAN program. IIR filters with different poles inside or outside the unit circle in the z-plane, are demonstrated in experimental procedure II.2. It can be seen on an oscilloscope when the IIR notch filter becomes unstable.

3.1 EXPERIMENT II

The purpose of this experiment is for students to work with digital filters, to demonstrate and enhance the theory they have learned, and to familiarize them further with programs such as TI_LOAD, and BUG, which they have used already during experiment I.

There are two parts to this experiment. The first part requires the student to be familiar with the IBM PC (or other personal computer). The second part of this experiment requires the student to understand the TMS32010 assembly program DCONV.ASM and the FORTRAN program TEST.FOR, which are used to demonstrate how inter-processor (between IBM PC AT and TMS32010) communications can be achieved.

From the perspective of the TMS32010 processor, there are two methods which may be used for transferring data to and from the host processor. The first method is most appropriate for arrays and it uses table-read (TBLR) and table-write (TBLW) operations, done directly to the dual-port memory. This method is not used in this experiment. The second method is to use the IN instruction to channel 5 or 7 (in this case, it is channel 7), and the OUT instructions from channel 4, 5, or 6 (in this case, it is channel 6). This method is essentially equivalent to operating an ordinary I/O channel to the host computer, and it makes the host computer act like a standard I/O device (A/D converter, D/A converter, etc.) to the TMS32010. The advantage of this method is that it uses only IN and OUT instructions, which require but two cycles (400 nsec); and it allows for the generation of programs, which can use either the host computer or an actual I/O device as a data source. The disadvantage is that it is less flexible than TBLR and TBLW operations for arrays, and that it requires close coordination with the host processor for data transfers.

From the perspective of the host processor, all communication with the TMS32010 processor is performed through the dual-port memory. Thus, all memory operations available on the host

processor can be used for communications with the TMS32010. The most efficient one of these is the block-move instruction available on the 8086 family of processors.

DCONV.ASM, shown in Appendix A, is a TMS32010 assembly program that implements an 8-point-length linear convolution. It reads data from port 7 and writes to port 6 of the TMS32010.

INTER.ASM, listed in Appendix B, is a 8086 assembly program. This program is used to communicate with the TMS32010 under flag (BIO) control. It transfers the data point from a FORTRAN program to port 7 of the TMS32010 and returns the corresponding data from port 6 of the TMS32010 to the FORTRAN program.

TEST.FOR, listed in Appendix C, is a FORTRAN program that creates the data to be transferred to the TMS32010. The interface is done in a subroutine called OUT320, which contains the data to be transferred to the TMS32010 and returns the corresponding sample.

The entire inter-processor demonstration can be implemented using the COM.BAT file.

3.1.1 Noise Rejection Using Notch Filter

Suppose that we have a signal

$$s_n = \cos \omega_o nT - \frac{1}{3} \cos 3\omega_o nT \quad (3-5)$$

with $\omega_o = \pi$ rad/sec and sampling frequency $\omega_s = 30\pi$ rad/sec. This signal is subject to a disturbance

$$d_n = \frac{1}{2} \sin 2\omega_o nT, \quad (3-6)$$

so that we actually measure $x_n = s_n + d_n$, which forms an input to a digital filter

$$H_1(z) = 1 - 2 \cos \theta z^{-1} + z^{-2} \quad (3-7)$$

or alternatively to a filter

$$H_2(z) = \frac{H_1(z)}{1 - 2r \cos \theta z^{-1} + r^2 z^{-2}} \quad (3-8)$$

where $r = 0.95$.

If filters are designed correctly, the disturbance will be removed and the output y_n will be a "cleaned-up" version of the signal component in the input. The designed filter $H(z)$ defines what the term "cleaned-up" means.

3.1.2 Analysis and Design the Notch Filter

1. Find the unit-pulse response $h_{1,n}$ and $h_{2,n}$ by taking inverse Z-transforms of $H_1(z)$ and $H_2(z)$, which are causal filters.
2. Determine $H_1(z)$ and $H_2(z)$, that is, find θ and use it. Plot pole/zero diagrams and sketch the frequency response, in order to clarify the significance of θ .
3. Write a computer program that implements the filter in a difference-equation form.
4. Plot $H_1(\omega)$ and $H_2(\omega)$ in terms of both magnitude and phase. In case that you do not have a graphing program, there is one BASIC graphing programs available in the DSP Lab. VID.BAS has an on-line manual. It requires the input file to contain two arrays.
5. Drive the filters with x_n , and plot s_n , x_n and y_n .
6. Determine the coefficients of the impulse response of the FIR filter and convert them to hexadecimal as in experiment I. (Use either a calculator or the BASIC program TEST.BAS).
7. Use the following relationship

$$H_2(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{m=0}^M b_m z^{-m}}{\sum_{k=0}^N a_k z^{-k}} = \frac{H_1(z)}{1 - 2r \cos \theta z^{-1} + r^2 z^{-2}} \quad (3-9)$$

to determine b_m and a_k , and then convert them to hexadecimal.

8. How could you modify $H_1(z)$ and/or $H_2(z)$ so that the output y_n is closer to the desired signal s_n . Give your arguments. Are they based on the magnitude and/or the phase characteristic?
9. Determine the coefficients of the new unit pulse response of the FIR filter and/or the new b_m and a_k of the IIR filter. Convert them to hexadecimal.
10. Plot the new $H_1(\omega)$ and/or $H_2(\omega)$ in terms of both magnitude and phase. Did you get the effects you intended; or additional undesired ones as well?
11. Drive the new filters with x_n , and plot the new y_n . Compare to the results obtained under 5. Did things improve or deteriorate? What do you think you lost/gained in its place?

3.1.3 Implementation of the Notch Filter

3.1.3.1 Experimental Procedure II.1

1. Copy DCONV.ASM to EXPIL.ASM by typing "COPY DCONV.ASM EXPIL.ASM".
2. Use EDLIN or DVED to edit EXPIL.ASM. Note that DVED.COM has an on-line manual.
3. Input the coefficients of the unmodified unit pulse response of the FIR filter in hexadecimal to the unit pulse response section of the EXPIL.ASM. Change all the parameters according to the order of the filter, that is, change the data-memory address and loop counters in the linear-convolution section of EXPIL.ASM.
4. Assemble and link the program by typing "XASM3 EXPIL;" and "LINKER EXPIL;", respectively.
5. Down-load the program with a "proper" sampling rate. Explain how you determined. What is "proper"?
6. Copy TEST.FOR to EXPIL.FOR as in step 1.
7. Change the input signal and sampling rate in EXPIL.FOR, that is, enter the input function $x_n = d_n + s_n$ and the sampling frequency.
8. To run the filter, type COM EXPIL.
9. Plot x_n , s_n and y_n . The data files for plotting are ifile, sfile, and ofile, respectively. Are they what you expect? Compare with corresponding results from part I.

10. Connect the waveform generator to the input of the 320/pc board (lower BNC connector) and an oscilloscope, and connect the output of the 320/pc board (upper BNC connector) to the other channel of the oscilloscope.
11. Change DIN from port 7 to port 6 in the beginning of the EXPIL.ASM. Assemble and link the program. Down-load the program with 15 kHz sampling rate.
12. Vary frequencies from 1 to 20 kHz. Find the actual notch frequency and also sketch the magnitude response of the filter. Does this filter meet the specification? Which are satisfied/violated? Why or why not?
13. Is this filter stable? Give your arguments!

3.1.3.2 Experimental Procedure II.2

1. Copy DIIR.ASM to EXPIL.ASM.
2. Input the unmodified coefficients of the IIR filter to both "coefficient Bm section" and "coefficient Am section" in hexadecimal.
3. Assemble, link and down-load the program. Since this program reads data from port 7, run the filter with the FORTRAN program EXPIL.FOR.
4. Plot x_n , s_n and y_n . The data files for plotting are ifile, sfile, and ofile, respectively. Are they what you expect? Compare with results in part I. Is the output y_n the same as that from the FIR filter? Explain why or why not?
5. Input the modified coefficients of the IIR filter to the "coefficient Am section" in hexadecimal of the EXPIL.ASM.
6. Run the filter, plot x_n , s_n and y_n as in step 4. Compare with the results from step 4.
7. Connect the waveform generator to the input of the 320/pc board (lower BNC connector) and an oscilloscope, and connect the output of the 320/pc board (upper BNC connector) to the other channel of the oscilloscope.
8. Change DIN from port 7 to port 6 in the beginning of the EXPIL.ASM. Assemble and link the program. Down-load the program with a sampling rate of 15 kHz.
9. Vary the frequency from 1 to 20 kHz. Find the actual notch frequency and also sketch the magnitude response of the filter. Does this filter meet the specification? Where is it violated/satisfied? Why?

10. Let $r = 1.3$. Calculate a_k in equation (3-9) and convert to hexadecimal.
11. Change to the BUG directory, and execute the BUG program. Find the starting location for the calculation of the IIR filter and set the program counter at this location with the aid of EXPIL.LST.
12. Set the breakpoint at location 007B and run the program by using GO (S6).
13. After a break at location 007B, use EDIT DATA MEMORY (F2) to input the new a_m in hexadecimal from step 10 into locations 0019 to 0020.
14. Use EDIT REGISTER (F1) to ensure the input and run the implementation by using GO (S6).
15. Vary the frequency from the waveform generator from 1 to 20 kHz.
16. What do you observe from the output? Is this filter stable? Explain why or why not? Give complete arguments!

4. ALIASING

Aliasing is unavoidable in signal processing where an input signal is not a bandlimited continuous signal. In experiment III, aliasing is demonstrated with an input from an external waveform generator. The reduction of aliasing with the aid of the Model 3202 analog filter, as anti-aliasing filter, is also shown.

The TMS32010 MS/PC-DOS CrossWare, the PATCH program from the Algorithm Development Package (ADP), and the Model 3202 solid-state variable filters are introduced. There are two unique functions of the PATCH program. It can be used: (a) to measure the utilization of the TMS32010 processor in a real-time operation; (b) to change the sampling rate without having to down-load the program again; and (c) to turn the algorithm, which has been down-loaded to the TMS32010, on or off. These functions are all demonstrated in experimental procedure III.1. The TMS32010 processor is utilized in a real-time operation. The "theoretically highest" order of a filter that can be implemented in the TMS32010 processor is calculated. The aliasing phenomenon is demonstrated in experimental procedure III.1 by varying the frequency of an input signal generated by a waveform generator, and by varying the sampling rate of a sinusoidal signal which has a fixed frequency.

In order to prevent aliasing, the continuous-time signal must be bandlimited before sampling, although real-world signals are not strictly bandlimited. Usually, an analog prefilter is used before the analog-to-digital converter. Likewise, the output of a digital-to-analog converter is not band-limited. An analog postfilter is used to attenuate high-frequency images corresponding to sampled signals. In experimental procedure III.2, students will gain experience with prefilters and postfilters. Also, they will measure the phase response of the designed filter by measuring on an oscilloscope the period of the output signal which is from the postfilter (analog filter). The output signal of the implemented digital filter is connected to a low-pass filter (postfilter) with an attenuation rate of 48 dB per octave. The phase response of the postfilter is given, and therefore, the phase response of a Kaiser window design "linear phase" FIR is measurable, and thus demonstrated.

4.1 EXPERIMENT III

The purpose of this experiment is to acquaint the student with the use of the TMS32010 MS/PC-DOS CrossWare and Link Editor; and to become familiar with the PATCH program, from the Algorithm Development Package (ADP), and the Model 3202 solid-state, variable-frequency electronic filter. Aliasing is demonstrated with an input from the external waveform generator. Reduction of aliasing is achieved with the aid of the Model 3202 filter. The oscilloscope is used to view both input and output of the digital filter hardware.

4.1.1 TMS32010 MS/PC-DOS CrossWare

To execute the Macro Assembler, enter:

```
XASM3
```

The command-line parser prompts for the source, listing, and object-file names:

Source File Enter the source-file name (if the source file does not have an extension, then type the file name with an explicit '.').

Listing File Enter the output-listing file name.

Object File Enter the output-object file name.

MS/PC DOS creates defaults for the listing and object files and/or their extensions. The default extensions are:

- Source file - .ASM
- Listing file -.LST
- Object file -.MPO

A source-file name can be followed by a semicolon, either on the command line or in response to a prompt; this will cause the Macro Assembler to generate the default file without displaying further prompts.

Example:

```
XASM3 <filename>;
```

By using <filename> with a default extension .ASM, the Macro Assembler will generate defaults for the listing and object files as indicated above.

```
XASM3 <filename>, <newname>;
```

By using <filename> with a default extension .ASM, the Macro Assembler will generate the listing file <newname> .LST and the object file <newname> .MPO.

```
XASM3 <filename>, <newname>
```

By using <filename> with a default extension .ASM, the Macro Assembler will generate the listing file <newname> .LST and prompt for the object-file name [6].

4.1.2 Link Editor

To execute the link editor, enter:

LINKER

The command-line parser will prompt for the control-link map, and load-file names.

Control File Enter the control-file name with an extension (if the control file does not have an extension, then type the file name with an explicit '.').

Map File Enter the link map file name with extension.

Load File Enter the load-module file name with an extension.

MS/PC DOS creates defaults for the listing and object files and/or their extensions. The default extensions are:

- Control file - .CTL
- Linkmap file -.MAP
- Listing file -.LOD

A source-file name can be followed by a semicolon, either on the command line or in response to a prompt; this will cause the Macro Assembler to generate the default file without displaying further prompts.

Example:

```
LINKER <filename>;
```

By using <filename> with default extension .CTL, the Macro Assembler will generate defaults for the link map and load files as indicated above.

```
LINKER <filename>, <newname>;
```

By using <filename> with default extension .CTL, the Macro Assembler will generate the link map file <newname>.MAP and the load file <newname>.LOD.

```
XASM3 <filename>, <newname>
```

By using <filename> with default extension .CTL, the Macro Assembler will generate the link map file <newname>.MAP and prompt for the load file name.

The link-control file is an input file that controls the operation of the Link Editor. This file contains a set of link-control commands which define the modules to be linked and how they are to be linked. The Link Editor links the object modules in the order specified by the linker commands.

The link-control file must be created ahead of time. Entering a pathname instructs the editor to look for a file containing the necessary control commands [7]. Since all the control files have been created already, details of how to write a control file are not discussed here.

4.1.3 PATCH

The PATCH program has two basic purposes. The first is to install TMS32010 program patches under user control. The second is to measure the utilization of the TMS32010 processor real-time operating conditions.

To execute the program, type the following command line:

```
PATCH[/S][/G][/I][/B][/C][/P] PATCHFILE [SCREEN-MESSAGE]
```

For example:

```
PATCH/S/G/P LCON.PCH (4 - 1)
```

4.1.3.1 Program Switch

/Stop (Abbreviation /S as in 4-1) stop TMS32010 before the patch is loaded. This is accomplished by holding the TMS32010 processor in a reset condition.

/Go (Abbreviation /G as in 4-1) start TMS32010 after loading. This is accomplished by releasing the TMS32010 processor from a reset condition.

/Interrupt Enable the external interrupt (XINT). This has the effect of closing the XINT switch so that the TMS32010 interrupt input (INT) is connected to the external source.

/BIO Enable the external flags (XBIO). This has the effect of closing the XBIO switch so that the TMS32010 I/O flag input (BIO) is connected to the external source.

/Clock Enable interrupts from the programmable clock (XCLK). This has the effect of closing the XCLK switch so that the TMS32010 interrupt input (INT) is connected to the output of the programmable clock.

/Patch (Abbreviation /P as in 4-1) skip patch-file name in message.

4.1.3.2 Patching A Real-time Program

In order to patch a TMS32010 program, the user must provide a patch file. The patch file is in ASCII format and is normally created with a text editor.

The format of a line in a patch file is #Cxxxxxx..., where # is the number of the patch to which the line belongs (1-8), C is a one-letter command, and the rest of the line (xxxxxx...) depends on the command. PATCH recognizes four commands: S, L, X, and a hexadecimal number (0-9, A-F).

4.1.3.3 Set Sampling Rate (S)

The S command sets the sampling rate (in Hertz) of the programmable clock of the 320/PC board. In the example below,

1S8000 (4-2)

3S16000

patch 1 sets the sampling clock to 8000 Hertz, and patch 3 sets the sampling clock to 16000 Hertz. Patches 2, 4, 5, 6, 7 and 8 (if they exist) do not change the sampling rate in (4-2).

4.1.3.4 Patch Message (L)

The L command is used to associate a screen message with each of the patches. These are the messages which are displayed by the name of the function key for each patch. For example:

```
1L Sampling Rate = 100
2L Sampling Rate = 500
3L Sampling Rate = 1000
4L Sampling Rate = 2000
5L Sampling Rate = 4000
6L Sampling Rate = 8000
7L Sampling Rate = 16000
81 Sampling Rate = 30000
```

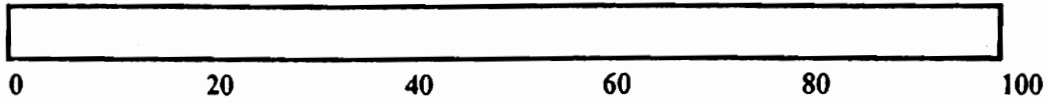
these lead to the display of Figure 7.

4.1.3.5 External File Input (X)

The X command is used to input a patch which has been assembled by an assembly program. The argument for this command is the name of the file to be loaded. If no extension is given, the .LOD extension is assumed. An example of using this external file command is shown below

```
1X FFT
2X LCONV.MPO
1X DFT.LOD
```

ATLANTA SIGNAL PROCESSORS INC.



PERCENT TMS32010 UTILIZATION

F1	SAMPLING RATE=	100	F2	SAMPLING RATE=	500
F3	SAMPLING RATE=	1000	F4	SAMPLING RATE=	2000
F5	SAMPLING RATE=	4000	F6	SAMPLING RATE=	8000
F7	SAMPLING RATE=	16000	F8	SAMPLING RATE=	30000

F9: Toggle Algorithm On/Off Condition	F10: Exit
---------------------------------------	-----------

Algorithm Active

Figure 7. Typical example of a PATCH screen.

The file FFT.LOD is loaded into patch number 1 and LCONV.MPO is loaded into patch number 2. The file DFT.LOD is added to patch number 1.

4.1.3.6 Direct Patch (#)

If the command field is a legal hexadecimal number (0-9, A-F), then the line is a direct patch. The form of a direct patch is #XXXXYYYY, where # is the patch number which is from 1 to 8, XXXX is a four-digit hexadecimal number for the patch (word) address in the dual-port memory, and YYYY is a four digit hexadecimal number for the data to be patched. The direct patch is not used in this experiment.

4.1.3.7 Measuring TMS32010 Efficiency

One of the most powerful features of the 320/PC board is its dual-port memory. Because this memory can be simultaneously accessed from both the TMS32010 processor and the host processor, and because the dual-port memory requires no wait states from either processor, a running TMS32010 program can be patched (that is, to change the sampling rate under user control through a host processor) and measured in real-time.

The second function of PATCH is to measure the efficiency of the TMS32010 in real-time operation. PATCH computes the percent utilization by sampling the value of location 8 in the memory window, and taking averages of the results. The average utilization is displayed on the screen in real-time.

The final function of PATCH is to signal the TMS32010 program to deactivate the algorithm. Like the previous function, the TMS32010 program must be written specifically to utilize this

function. PATCH signals the TMS32010 processor that an algorithm should be active by placing a 1 in location 10 of the TMS32010 assembly program.

4.1.4 The Model 3202 Filter

The Model 3202 filter is a solid-state, variable-frequency, electronic filter with cutoff frequencies continuously adjustable over a range from 20 Hz to 2 MHz [8]. The pass-band gain is unity (0 dB), with an attenuation rate of 24 dB per octave outside the pass-band. The maximum attenuation is greater than 80 dB and the output hum and noise are less than 100 microVolts.

The Model 3202 filter can function as either a High-Pass or a Low-Pass Filter. In the High-Pass mode of operation, the maximum input signal is 3 Volts rms and the upper 3 dB point occurs at approximately 10 MHz. In the Low-Pass mode, the Filter is direct-coupled and the combined AC plus DC input signal should not exceed 4.2 Volts peak-to-peak. The Model 3202 Filter can also function as either a Band-Reject Filter with cutoff frequency limits from 20 Hz to 2 MHz, or to provide a sharp null at any frequency between 40 Hz and 800 kHz when the two channels are connected in parallel. When these two filter channels are switched to the same mode of operation and connected in series with both dials set to the same cutoff frequency, the Model 3202 Filter will function as a High-Pass or Low-Pass Filter with an attenuation rate of 48 dB per octave. If the two channels are connected in series, and one channel is operated in the Low- Pass mode and the other in the High-Pass mode , the Model 3202 Filter will function as a Band-Pass Filter with an attenuation rate of 24 dB per octave outside the pass-band.

This filter has a maximally flat or Butterworth characteristic when the RESPONSE switch(s) located on the rear of the chassis, is in the MAX FLAT position. For pulse-type wave-forms, this switch should be in the SAMPLE RC position, optimum for transient-free filtering.

4.1.4.1 Frequency Dials

Each channel has a single-decade, frequency dial (calibrated from 19 to 210) and an associated high-pass/low-pass band switch that provides five multiplier ranges for each mode.

4.1.4.2 Frequency Range

High-Pass and Low-Pass cutoff frequencies continuously adjustable from 20 Hz to 2 MHz in five bands. If frequency dials is set at 20, and frequency range is set at 100 Hz, the cutoff frequency is at 2000 Hz.

4.1.4.3 Cutoff-Frequency Calibration Accuracy

The cutoff-frequency calibration accuracy is about $\pm 5\%$ for bands one to four, and $\pm 10\%$ for band five, with Response Switch in Max. Flat (Butterworth) position; less accurate in R-C position. Relative to mid-band level, the filter output is down 3 dB at cutoff in Max. Flat position, and approximately 13 dB in the R-C position.

4.1.4.4 Bandwidth

Low-Pass Mode: Frequency response from DC to the cutoff frequency set within the range from 20 Hz to 2 MHz.

High-Pass Mode: Continuously adjustable between 20 Hz and 2MHz with upper 3 dB point at approximately 10 MHz.

Band-Pass Mode: Continuously variable within cutoff-frequency limits of 20 Hz to 2 MHz. For minimum bandwidth, the high-pass and low-pass cutoff frequencies are set equal. This produces an insertion loss of 6 dB, with the -3 dB points at 0.8 and 1.25 times the midband frequency.

Band-Pass Mode: Continuously variable within cutoff-frequency limits of 20 Hz and 2 MHz or a sharp null at any frequency between 40 Hz and 800 kHz. The low-pass band extends to DC. The high-pass band has its upper 3 dB point at approximately 10 MHz. The null is sharper than that of a balanced "parallel T" filter, and is obtained by setting the high-pass cutoff at approximately twice the desired null frequency, and the low-pass cutoff at approximately one-half the desired null frequency.

4.1.4.5 Response Characteristics

Butterworth Each channel exhibits a maximally flat, fourth-order, Butterworth response for a optimum performance in the frequency domain.

Simple RC Fourth order RC response for transient-free time-domain performance. Higher-order characteristics may be obtained by cascading individual channels.

Attenuation slope: Nominal 24 dB per octave per channel in high-pass or low-pass modes.

Maximum attenuation: Greater than 80 dB.

4.1.4.6 Panel Control

The front panel of the Model 3200 or each channel of the Model 3202 includes a frequency dial, a band multiplier/function switch, two BNC coaxial connectors for INPUT and OUTPUT signals, and a screwdriver control for the adjustment of the output DC level. A POWER-ON switch and an indicator light are used in both models.

The rear chassis of each channel of the Model 3202 Filter has two switches; one for selecting the filter response of either the Butterworth type (Maximal flatness) or simple RC (Transient-free), and one for disconnecting the signal ground from the chassis ground.

4.1.4.7 Operation

1. Make appropriate connections to the INPUT and OUTPUT connectors of the filter. The rms INPUT voltage should not exceed 4.2 Volts peak-to-peak in the Low-Pass mode. The filter can sustain a combined AC and DC INPUT voltage of up to 200 Volts peak without causing permanent damage. In the event of an overload, the output waveform will appear distorted.
2. Set the mode of operation and cutoff frequency by means of the band multiplier switch(es) and the frequency dial(s).
3. Turn the power switch to ON. After a sufficient warm-up time, check the output DC level. If necessary, adjust DC LEVEL potentiometer(s) for zero Volts on the output(s).
4. For normal filter operation, the floating/chassis ground switch(s), located on the rear of the chassis, should be in the chassis position. If the filter is used in a system where ground loops make ungrounded operation essential, this switch(s) should be in the floating position.

CAUTION: In FLOATING operation, the signal ground should be connected to system ground to prevent excessive hum and noise.

5. When the filtering operation consists principally of separating frequency components of a signal, the RESPONSE switch(s) located on the rear of the chassis should be in the MAX-FLAT position. If the filter is used to separate pulse-type signals from noise (in the time domain), this switch should be in the RC position.
6. BNC coaxial connectors are provided on the front panel and on the rear of the chassis for both INPUT and OUTPUT connections.

High-Pass or Low-Pass operation with 48 dB per octave attenuation

1. Link together the two channels in series by connecting the output of the left channel to the input of the right channel.
2. Select identical mode of operation and multiplier position for both channels.
3. Set both dials to the same cutoff frequency. Note that when two channels are in series and set to the same mode of operation with an identical cutoff frequency, the gain at the cutoff frequency will be down 6 dB from the pass-band gain with the two RESPONSE switches in the MAX- FLAT (Butterworth) position. In the simple R-C position, the gain at the cutoff frequency will be down approximately 26 dB.

Band-Pass operation with 24 dB per octave attenuation

1. Connect the two channels in series.
2. Set the left channel to the High-Pass mode (this will control the Low-Cutoff frequency). Set the right channel to the Low-Pass mode (this will control the High-Cutoff frequency).

Band-Pass operation could also be obtained by setting the left channel to the Low-Pass mode and the right channel to the High-Pass mode. This method has the advantage that the Low Cutoff Frequency (High-Pass mode) is on the right, which is a logical arrangement since it coincides with our customary graphical representation of a Band-Pass filter. This may be disadvantageous, since the output is DC-coupled when the Low-Pass channel is on the right. If this method is used, the output is AC-coupled, which is desirable in some applications where no DC fluctuations on the output can be tolerated.

3. The minimum Pass-Band is obtained by setting the high cutoff frequency equal to the low cutoff frequency. In this condition, the insertion loss is 6 dB, and the -3 dB cutoff frequencies occur at 0.8 and 1.25 times the mid-band frequency.

Band-Reject or Notch filter operation

1. Arrange together the two channels in parallel by connecting the input signal to the BNC INPUT connector of both channels simultaneously. The OUTPUT from both channels should be added through two equal external resistors in series with each output. The junction of these resistors becomes the output of the filter. It is recommended that the resistors be approximately 1,000 Ohms and of the carbon or metal-film type if the filter is used at high frequencies. If the two resistors are not equal, the gain on one side of the notch will be different than the gain on the other side. The smaller the adding resistors, the greater the loss will be through the filter in the Pass-Band region, because of the loading effect of the filter output impedance of 50 Ohm.
2. The first channel should be set for Low-Pass operation. The second channel should be set for High-Pass operation.
3. It should be noted that the output impedance in the band-Reject mode will not be 50 Ohm, but approximately one half the resistance of one adding resistor. The maximum input should

not exceed 3 Volts rms, and the maximum output voltage in this mode will be 1.5 Volts rms open circuit.

4.1.4.8 Phase Response

The phase response of each channel of the Model 3202 filter can be obtained from Figure 4-2 which gives the normalized phase response characteristic for either mode of operation in degrees lead (+) or lag (-) as a function of ratio of the operating frequency f to the low cutoff frequency f_L (High-Pass mode) or high cutoff frequency f_H (Low-Pass mode). The solid curve is for the MAXIMALLY FLAT or Butterworth mode, and the dotted curve is for the transient-free or simple R-C mode.

EXAMPLE:

Determine the phase shift of the filter in the MAXIMALLY FLAT or Butterworth mode, with the function switch set to the High-Pass mode at the X1 position, and the cutoff frequency (f_L) set to 100 Hz and an input frequency (f) of 300Hz.

Since $\frac{f}{f_L} = \frac{300}{100} = 3$ the output of the filter leads the input by 50 degrees, from Figure 8.

The phase response of the Model 3202 filter could be obtained in the same manner by taking the algebraic sum of the phase response of each channel.

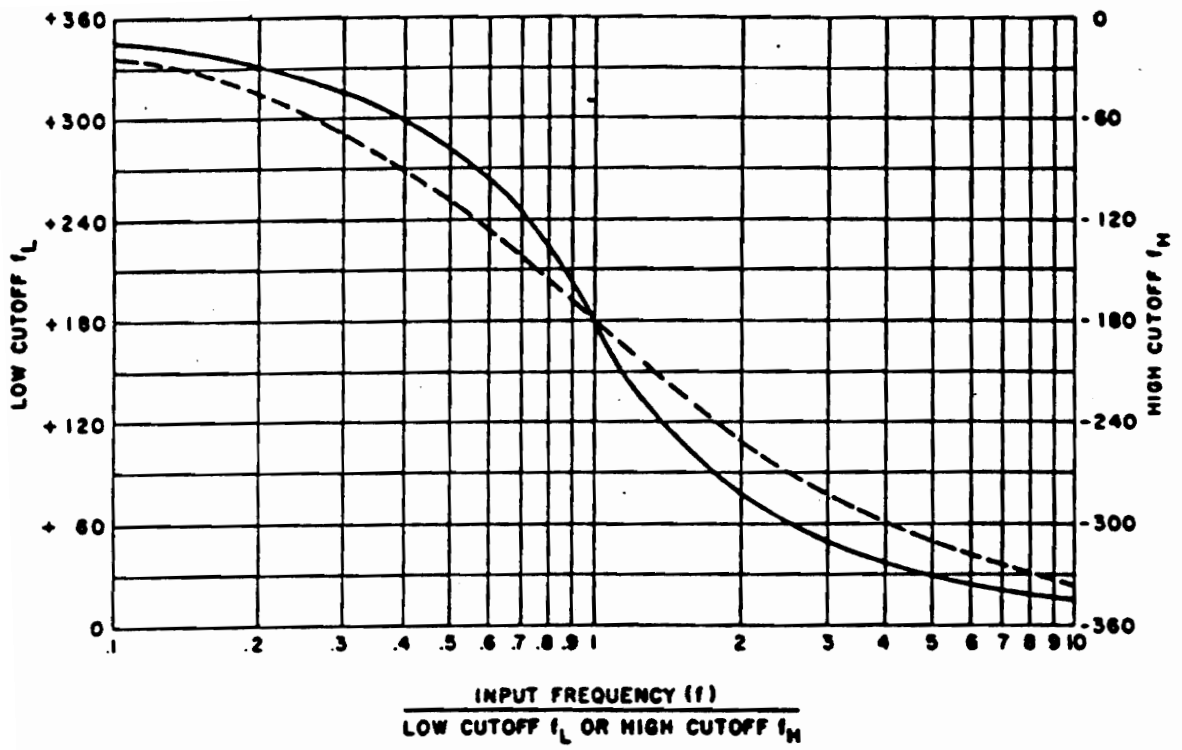


Figure 8. Normalized phase characteristics.

4.1.5 Experimental Procedure III.1

1. Connect the waveform generator to the input of the 320/PC board (lower BNC connector) and an oscilloscope, and connect the output of the 320/PC board (upper BNC connector) to the other channel of the oscilloscope.
2. KFIR.ASM is a TMS32010 assembly program. It implements a nonrecursive lowpass filter designed by using the Kaiser window method. Its specification is listed in Figure 9. Use TMS32010 MS/PC-DOS CrossWare to assemble KFIR.ASM.
3. KFIR.CTL is a link-control file which has been written already. Use the linker to produce an object file with .LOD as the extension.
4. Down-load KFIR.LOD to the 320/PC board, with a 10 kHz sampling rate.
5. Measure the utilization of the TMS32010 processor in real-time operation, by using the PATCH program.
6. The TMS32010 executes instructions at a speed of 200 ns per cycle. It takes 2 cycles to complete one multiplier and add; and it takes about 25 cycles to read data from the A/D, write to the D/A and some other functions. If the filter you down-load to the TMS32010 processor has an order of 32, what is the "theoretical" utilization of the TMS32010 processor for a 10 kHz sampling rate?
7. How do 5 and 6 compare with the implemented filter?
8. If the maximum sampling rate is used to sample the input, what is the highest order filter that can be implemented with the TMS32010 processor?

.....

FINITE IMPULSE RESPONSE (FIR)
 LINEAR-PHASE DIGITAL FILTER DESIGN
 KAISER-WINDOW ALGORITHM

BANDPASS FILTER

FILTER LENGTH = 32
 SAMPLING FREQUENCY = 10.00 KILOHERTZ
 DESIRED RIPPLE = 26.021 (DB)
 KAISER WINDOW PARAMETER, ALPHA = 2.146

***** IMPULSE RESPONSE *****

12-BIT QUANTIZED COEFFICIENTS

H(1) = .112305E-01 = H(32)
 H(2) = .842285E-02 = H(31)
 H(3) = -.610352E-03 = H(30)
 H(4) = .111084E-01 = H(29)
 H(5) = .817871E-02 = H(28)
 H(6) = -.325928E-01 = H(27)
 H(7) = -.386963E-01 = H(26)
 H(8) = .137939E-01 = H(25)
 H(9) = .269775E-01 = H(24)
 H(10) = -.207520E-02 = H(23)
 H(11) = .443115E-01 = H(22)
 H(12) = .913086E-01 = H(21)
 H(13) = -.538330E-01 = H(20)
 H(14) = -.230591E-00 = H(19)
 H(15) = -.845947E-01 = H(18)
 H(16) = .240234E-00 = H(17)

*** CHARACTERISTICS OF DESIGNED FILTER ***

	BAND 1	BAND 2	BAND 3
LOWER BAND EDGE	.0000	1.5000	3.0000
UPPER BAND EDGE	1.0000	2.5000	5.0000
NOMINAL GAIN	.0000	1.0000	.0000
NOMINAL RIPPLE	.0500	.0500	.0500
MAXIMUM RIPPLE	.0402	.0402	.0278
RIPPLE IN DB	-27.9258	.3609	-31.1267

.....

Figure 9. Characteristics of designed KFIR filter.

9. Vary the frequency of the input signal over the range of 1 kHz to 20 kHz. Record the 3 dB cutoff frequencies from observation with the oscilloscope. Does this filter meet the specifications listed in Figure 4-3? Why or why not?

10. Use PATCH to vary the sampling rate over the range of 100 Hz to 30 kHz. What do you expect the output to be? Over what frequency range, does the output not have aliasing theoretically and experimentally? How did you determine this? Give complete arguments.

11. Use the deactivate algorithm key (F9). What do you expect the output to be and why? What is actually happening?

12. Use the exit key (F10) to exit from the PATCH program.

4.1.6 Experimental Procedure III.2

1. Connect the waveform generator to the input of the 320/PC board (lower BNC connector) and an oscilloscope, and connect the output of the 320/PC board (upper BNC connector) to the other channel of the oscilloscope as in experimental procedure I.3.
2. Down-load KFIR.LOD to the TMS32010 board, with a sampling rate of 10 kHz.
3. Slowly increase the frequency of the input signal until aliasing occurs. Adjust to the lowest frequency where the aliased signal has maximum amplitude. What is the input frequency from the waveform generator? What is the output frequency seen from the oscilloscope? Explain how they are related to the sampling rate.
4. Connect the waveform generator to the input of the Model 3202 filter, and the output of Model 3202 filter to the oscilloscope. Find the range of Low-Pass filter cutoff frequencies for which the designed passband is maintained.
5. Proceed to connect the output of the Model 3202 filter to the input of 320 board. Adjust the Low-Pass filter cutoff frequency until the aliasing gain is down by 20 dB. Record the Low-Pass filter cutoff frequency.
6. Connect the Low-Pass filters in series for an attenuation rate of 48 dB per octave to achieve an aliasing gain down by 40 dB.
7. Explain how the Model 3202 filter can function as an anti-aliasing filter. Provide graphs to clarify your answer.

8. Connect the output of the TMS32010 to the input of the Model 3202 Low-Pass filter with an attenuation rate of 48 dB per octave, and connect the output of the Model 3202 to the oscilloscope.
9. The Low-Pass filter functions as a postfilter to "smooth out" the output of the TMS32010. Sketch the phase response of the KFIR filter.
 - a. Set both channel 1 and channel 2 to overlap.
 - b. The easiest way to measure the phase response is to vary the frequency until the phase shift is a multiple of 90° . Slowly increase the frequency from the lower band edge until the input and output signals are on top of each other, that is, the phase difference is 0° .
 - c. Record the frequency, which is not read from the waveform generator dial but measured from the oscilloscope.
 - d. Increase the frequency again until the difference between the input and output increases by 90° and record the frequency.
 - e. Repeat (d) until the frequency has increased to the upper band edge.

Note that the phase response you measured is not the phase response of the KFIR filter. What is the relationship between the phase response of the KFIR filter, the postfilter, and the phase response you measured? Give arguments!

10. What do you observe from the phase response of the KFIR filter and which property do you expect? Justify your answer.

5. FIR AND IIR FILTER DESIGN

Filtering is one of the most useful and important operations in signal processing. In the context of analog signals and systems, the concept of filtering comes about due to the fact that Fourier transforms of the input, $X_a(\omega)$, and of the output, $Y_a(\omega)$, of a linear time-invariant system are related as follows:

$$Y_a(\omega) = H_a(\omega)X_a(\omega) \quad (5 - 1)$$

where $H_a(\omega)$ is the transfer function of the system. Analog filters can be implemented in a variety of ways; but, for many reasons, it may be of interest to filter an analog signal using digital methods. A general block diagram of a system for digital filtering of analog signals is shown in Figure 10. There are five basic components: a prefilter, an analog-to-digital (A/D) converter, a numerical processor or a digital filter (e.g., the TMS32010 microcomputer), a digital-to-analog (D/A) converter, and a postfilter. It is important to understand the issues that bear upon the use of the filter design/implementation package.

The Digital Filter Design Package (DFDP) allows students to design recursive (IIR), Kaiser-Window Nonrecursive (KFIR), Parks-McClellan and Equiripple Nonrecursive (PMFIR) filters. The package also generates the TMS32010 assembly code for the designed filter. Once the

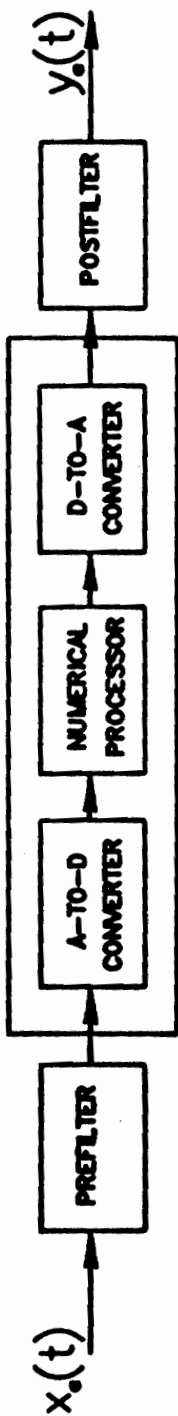


Figure 10. System diagram of digital filtering of an analog signal.

designed filter is down-loaded to the TMS32010, the whole system can act as a digital filter. The instructions for use of both the IIR and FIR modules to design a filter are discussed step-by-step in experiment IV. Capabilities and limitations of both IIR and FIR filters are described. Different procedures, such as bilinear transformation of Butterworth, Chebyshev and Cauer (elliptic) filters can be used to design an IIR filter. The characteristics of each type of filter are presented to the designer in the following plots of linear magnitude, log magnitude in dB, phase, pole and zero location in the z-plane, and unit pulse response. FIR filters using the Kaiser-Window method (KFIR), or the Parks-McClellan Equiripple method (PMFIR) can be designed as well. Students are asked to compare KFIR and PMFIR filters with the help of plots of magnitude, log magnitude, unit-sample response and frequency error.

5.1 EXPERIMENT IV

The purpose of this experiment is to acquaint the students with the software, principles, and problems associated with the Digital Filter Design Package (DFDP). It allows the student to design recursive (IIR), Kaiser-Window Nonrecursive (FIR), and Parks-McClellan Equiripple Nonrecursive (FIR) Filters. It also generates TMS32010 assembly code for the designed filter. In this experiment, two modules are demonstrated; for IIR, and for FIR designs. Details of the execution of these programs are covered in this experiment.

5.1.1 Programs in the DFDP

IIR To design recursive filters by the method of bilinear transformation of Butterworth, Chebyshev, or elliptic prototypes.

KFIR To design nonrecursive filters by the method of windowing using a Kaiser window.

PMFIR To design optimal nonrecursive filters by the Remez exchange algorithm as developed by Parks and McClellan.

CGEN To generate assembly language programs for the Texas Instruments TMS32010 microcomputer to implement both recursive and nonrecursive digital filters as designed by the above design modules.

INSTALL This program allows the user to specify where each program module will reside. INSTALL writes a file named DFDP.PAR which contains the location of each of the program modules, and whether or not there is a graphics printer. In this and all other experiments, the INSTALL program has been run.

5.1.2 To Execute the Software Package

After the INSTALL Program has been run, the user may type DFDP to execute the software package. The menu will appear on the screen as shown in Figure 11, after carriage return.

The user may select one of the design modules by entering either 1, 2, 3, or 4. Also, each of the design modules, IIR, KFIR, PMFIR and CGEN, can be executed by simply typing the design module name in response to the DOS prompt. For example, C: KFIR (carriage return). This may be the most efficient method of operation when the user is only interested in one filter type.

*** Digital Filter Design Package ***

(C) COPYRIGHT, 1984: ATLANTA SIGNAL PROCESSORS INC, VERSION 2.01

BN: IBM20134

PROGRAM SELECTION MENU

ENTER THE NUMBER CORRESPONDING TO THE DESIGN TECHNIQUE DESIRED

1. RECURSIVE (IIR) FILTER DESIGN
2. KAISER WINDOW NONRECURSIVE (FIR) FILTER DESIGN
3. PARKS-McCLELLAN EQUIRIPPLE (FIR) FILTER DESIGN
4. TMS320 CODE GENERATOR
5. QUIT

OPTION DESIRED =

Figure 11. The menu of DFDP.

5.1.3 Instruction for use of the IIR Design Module

5.1.3.1 Executing program IIR

The design module IIR uses the method of bilinear transformation of an analog filter of Butterworth, Chebyshev, or Cauer (elliptic) type.

Once IIR is typed, the program begins with the message shown in Figure 12.

The note (CT TO CONT) is used in several places in this and all other design modules to prompt the user to enter a carriage return to cause the program to proceed to its next step.

5.1.3.2 IIR Main Menu

After entering a carriage return, the menu will appear on the screen as shown in Figure 13.

If a filter is to be designed, the first step is to decide on the type of filter, that is, lowpass, highpass, bandpass, or bandstop.

If READ SAVED FILE action is selected, the program will ask for the name of a file containing a filter previously designed by typing

```
ENTER FILENAME (IIR.FLT): ELLIPT.FLT
```

The file name of a previously saved filter can be entered, or if a carriage return is entered without the file name, the program will attempt to read the file which will contain the most recently designed filter that was saved. In the example above, it will be 'IIR.FLT'.

**** Digital Filter Design Package ****

IIR BILINEAR TRANSFORM DESIGN PROGRAM

(C) COPYRIGHT, 1984: ATLANTA SIGNAL PROCESSORS INC., VERSION 1.02

SN: IBM20134

**THIS FILTER DESIGN PROGRAM DESIGN RECURSIVE DIGITAL
FILTER FROM BUTTERWORTH, CHEBYSHEV, AND ELLIPTIC
ANALOG PROTOTYES.**

CR TO CONT

Figure 12. IIR bilinear transformation design program.

***** IIR BILINEAR TRANSFORM MAIN MENU *****

ENTER THE NUMBER CORESPONDING TO THE FILTER DESIGNED

- 1. LOWPASS**
- 2. HIGHPASS**
- 3. BANDPASS**
- 4. BANDSTOP**

OR TAKE THE FOLLOWING ACTION

- 5. READ SAVED FILE**
- 6. RETURN TO PROGRAM SELECTION MENU**
- 7. QUIT (RETURN TO DOS)**

OPTION DESIRED =

Figure 13. IIR bilinear transformation main menu.

This option is very useful when it is desired to display the response characteristics of a previously designed filter, or to quantize the coefficients of a previously designed, but unquantized filter. If the name of a file containing an FIR filter is given, an error message will be typed, and no file will be read.

It is possible to return to the main DFDP menu or to the DOS operating system by entering 6 or 7, respectively.

5.1.3.3 The Sampling Rate

All frequencies are assumed to be in kHz. These units are used in labeling subsequent plots of response functions. All frequencies are entered in the same units as the sampling frequency. This is because of the scaling property of sampling (i.e., $\Omega = \omega T$).

5.1.3.4 The Cutoff Frequencies

For lowpass or highpass filters, only a single passband or a single stopband frequency is requested. In any case, all of the passband and stopband frequencies must be less than one-half of the specified sampling frequency.

5.1.3.5 Approximation Errors

These errors are referred to as passband ripple and stopband ripple. Note that the gain in the passband is always normalized to one, and the magnitude of the frequency response of the designed filter will vary around one in the passband, and between zero and one in the stopband. Although the passband and stopband ripples can be different, the ripple must be the same in the two

stopbands. This is also true of the two passbands of a bandstop filter, except in a Parks-McClellan design.

5.1.3.6 Selection of Approximation Type

The entry of the approximation-error limits completes the specification of the filter. The program uses the specified cutoff frequencies and approximation-error limits to compute the filter order required to meet the specifications. The design equation for any of the approximation types will generally yield a non-integer filter order. If the order were rounded down, the specifications would probably not be met. Good filter orders to work with are the smallest higher, or even bigger, integer.

The Cauer (elliptic) approximation always meets the specifications with the lowest order. The Butterworth and Chebyshev filters may have more desirable phase characteristics. The desired approximation type can be selected by entering the appropriate number. If results of the order calculation are not satisfactory, the user can return to the beginning of the IIR program to enter a new set of specifications by entering 5. It is possible to return to the DFDP menu or to the DOS operating system by entering 6 or 7, respectively.

If one of the approximation types is selected, the program proceeds to compute the coefficients for the filter. While the computation is in progress, no key should be pressed (since that key will be interpreted as the answer to the next question asked by the program).

5.1.3.6 Coefficient Quantization

After the choice of approximation method is made and the filter coefficients have been computed, the following question appears:

DO YOU WISH TO QUANTIZE COEFFICIENTS FOR THE TMS320 ? (Y OR N) N

The unquantized coefficients are represented with the full 32-bit floating-point representation of the IBM PC. If the user wishes to quantize the coefficients to 16-bits for use in a TMS32010 program as produced by the design module CGEN or for other fixed point hardware implementations, enter Y followed by a carriage return. But quantization can be avoided by simply entering a carriage return. Note that no choice is given here for the number of bits of quantization. Because of all the issues associated with quantization in a recursive filter, and because of the limited scaling capabilities of the TMS32010, it was decided to work with 16-bit quantization.

5.1.3.7 Verification of the Characteristics of the Designed Filter

After responding appropriately to the quantization question, the program measures the approximation errors of the filter. Since this measurement requires the evaluation of the frequency response on a dense set of frequencies, it can require a significant amount of time. A summary of the characteristics of the designed filter is then printed on the screen. Note that the program asks

DO YOU WANT RESULTS SENT TO THE LINE PRINTER ? (Y OR N) N

If Y is entered, all the information concerning the characteristics of the designed filter is printed.

If the filter specifications are not met (the measured passband or stopband ripple is too large), a warning message will appear.

5.1.3.8 Manipulation and Plotting of Designed Filter

After the filter has been designed and after the line-printer option has been selected, a message will be displayed as shown in Figure 14.

ENTER CORRESPONDING NUMBER FOR
INSTRUCTION DESIRED

1. AUTOMATICALLY INCREMENT FILTER ORDER
2. PLOT RESPONSES
3. DISPLAY FILTER COEFFICIENTS
4. OUTPUT FILTER COEFFICIENTS
5. QUANTIZE COEFFICIENTS

6. RETURN TO IIR BILINEAR TRANSFORM MAIN MENU
7. RETURN TO PROGRAM SELECTION MENU
8. QUIT (RETURN TO DOS)

OPTION DESIRED =

Figure 14. Manipulation and plotting designed filter.

At this point, the user has several options for further manipulation and analysis of the designed filter.

1. Automatic Incrementing of the Filter Order

When the filter does not meet specifications due to coefficient rounding, it may only fail by a small amount and the design may be acceptable otherwise. By using this option, the filter order can be increased systematically after quantization, until the specifications are met. Since the filter order is incremented by 1, this option may need to be selected repeatedly.

2. Plotting of Response Functions

When the plot response option is selected, a plot menu will be displayed. Entering the appropriate selection will cause the program to plot linear magnitude, log-magnitude in dB, phase, group delay, pole and zero locations in the z-plane, and unit pulse response, or all of the above in sequence. For all the plots except zero location in the z-plane and unit pulse response, the program has the following message at the top of the plot:

DO YOU WANT AN EXPLODED VIEW ? (Y OR N) N

If N is entered, the program returns to the menu to permit another plot selection. If Y is entered, the program requests the upper and lower limits of the range of frequencies (in kHz) for which the characteristic is desired to be plotted. This feature is extremely useful for examining the frequency response at the edge of the stopband or passband when the filter does not meet specifications. If N is entered, the program returns to the plot menu.

3. Display of Filter Coefficients

The heading summarizes the approximation type, and also specifies whether or not the coefficients have been quantized.

4. Saving Designed Filter on Disk

It is possible to output the filter coefficients to a disk file. It is essential to save the filter coefficients if the automatic TMS32010 code generator CGEN is to be used. When using CGEN, it requests the name of the file in which the coefficients are stored. If no file name is given before pressing the enter key, the information is written to the default file IIR.FLT.

5. Quantization of Filter Coefficients

If the coefficients were already previously quantized, this request for quantization will have no effect. The program proceeds to measure the response characteristic of the quantized filter, and then proceeds as if a new quantized design has just been completed.

6. Terminating the IIR Design program

The IIR design can be terminated in three ways: (a) to return to the beginning of the IIR design module to permit entry of a new set of specifications; (b) to return to the main DFDP menu to permit the selection of one of the FIR design modules; or (c) to return to the DOS operating system. Options (a), (b) or (c) are implemented by entering 6, 7, or 8, respectively.

5.1.3.9 Capabilities and Limitations of the program IIR

The program IIR contains numerous checks to insure that the user enters a consistent set of specifications. In order to detect all conditions which might cause numerical problems in designing the filter, it would be necessary to greatly restrict the flexibility available to the user. The program will allow the user to ask for filter specifications that may be impossible to obtain using the 32-bit floating-point arithmetic of the PC, or that may be impossible to implement using 16-bit fixed-point arithmetic on the TMS32010.

Some factors which may cause difficulties in the design of recursive filters are the following:

1. Unreasonably low approximation-error specifications. It generally does not make sense to specify either passband or stopband approximation errors less than 0.001 (0.0087 dB passband or -60 dB stopband ripple) for unity-gain, frequency-selective filters.
2. Unreasonably narrow transition regions between passbands and stopbands. Since all frequencies are relative to the sampling frequency, a narrow transition region is one which is a small fraction of the sampling frequency. A transition region as small as 1 Hz might be unreasonably small if the sampling rate is 10 kHz, but if the sampling rate were 10 Hz, it would not be difficult to achieve a 1 Hz transition region. This holds for unreasonably narrow passband or stopband specifications too.

No computer program promises to perform the impossible; the user must always verify the results. The program IIR permits the user to specify filters which cannot be designed by using the arithmetic capabilities of the PC.

5.1.4 Experimental Procedure IV.1

1. Design an IIR filter with the following specifications: Highpass, cutoff frequency $0.35\omega_c$, passband ripple 0.1, transition width $0.5\omega_c$, stopband attenuation > 60 dB. Let $\omega_c = 10$ kHz.
2. Use the method of bilinear transformation of a Butterworth, Chebyshev, or Cauer (elliptic) functions.
3. Print characteristics for each of the above types of filter. Compare the order of each filter.
4. Plot linear and log-magnitude (in dB), phase, pole and zero location in the z-plane and unit pulse response for each type of filter. Note that the plots can be obtained by using GRAPHICS in response to the DOS operating system.
5. What do you observe from the plots above? Give a brief description of each type of filter.

5.1.5 Instruction For Use of FIR Design Module

This program is capable of designing nonrecursive or finite impulse-response digital filters of the frequency-selective type, as well as differentiators (FM detection) and Hilbert transformers (SSB generation). All filters designed by this program are generated to have an exact linear phase characteristics. This linear phase corresponds to a time delay through the filter of exactly $M/2$ samples. The differentiator and the Hilbert transformer exhibit exactly 90-degree phase shifts in addition to a linear phase caused by the delay of $M/2$ samples. The design is based on the windowing approach, using the family of Kaiser windows.

5.1.5.1 Executing the program KFIR

The design module KFIR can be executed either by selecting it from the DFDP menu or by typing KFIR in response to the DOS operating system. Once this has been done, a similar message will appear on the screen as in the program IIR.

5.1.5.2 KFIR Main Menu

After entering a carriage return while the previous screen is displaying, the menu will appear on the screen as shown in Figure 15.

As can be seen, the user can select the type of filter, that is, lowpass, highpass, bandpass, bandstop, multiband, differentiator, Hilbert transformer and pulse-shaping filter (lowpass filters with raised cosine responses). Since multiband, differentiator, Hilbert transformer, and pulse shaping filters are not used in the experiments in the DSP Teaching Laboratory, they are not discussed here. Other options are the same as in the program IIR discussed before.

***** KAISER-WINDOW MAIN MENU *****

ENTER THE NUMBER CORESPONDING TO THE FILTER DESIGNED

1. **LOWPASS**
2. **HIGHPASS**
3. **BANDPASS**
4. **BANDSTOP**
5. **MULTIBAND**
6. **DIFFERENTIATOR**
7. **HILBERT SHAPING FILTER**

OR TAKE THE FOLLOWING ACTION

8. **READ SAVED FILE**
9. **RETURN TO PROGRAM SELECTION MENU**
10. **QUIT (RETURN TO DOS)**

OPTION DESIRED =

Figure 15. Kaiser-window main menu.

5.1.5.3 Specification of Filter Parameters

The sampling rate, cutoff frequencies, and approximation errors are specified; the program computes an estimate of the required length of the unit pulse response of the filter. The inherent linear phase of the filters imposes the constraint that the magnitude of the frequency response of a frequency-selective filter be zero at half the sampling frequency when the filter length is even. Problem 5-5 in [1]. Therefore, only odd lengths should be used when designing full-band highpass or bandstop filters.

5.1.5.4 Coefficient Quantization

The coefficient quantization is the same as in program IIR.

5.1.5.5 Verification of Characteristics of Designed Filter

After the coefficients have been quantified, the program evaluates the frequency response on a dense grid and at the specified band edges. After that, the approximation errors are measured, and a summary of the desired and measured frequency response characteristics is displayed on the screen as in the IIR program. A warning message will appear when specifications are not met.

5.1.5.6 Manipulation and Plotting of the Designed Filter

1. Changing Filter Length

This option is the same as changing the filter length in the program IIR. It can also be used

to find the smallest-length filter that will meet the specifications when the length predicted by the design formula is higher than necessary.

2. Plotting of Response Functions

The plot menu contains options as follows: linear magnitude, log-magnitude, unit sample response, and frequency error. Note that since the filter always has a precisely piecewise linear phase, there is no need for plots of phase or group delay.

The procedure of using this plot menu and the other option, such as display of filter coefficients and saving designed coefficients, are discussed in detail under the IIR in the previous section.

5.1.5.7 Capabilities and Limitations of KFIR

The program KFIR contains numerous checks to insure that specifications are entered consistently by the user. In order to detect all conditions which might cause numerical problems in designing the filter, it would be necessary to greatly restrict the flexibility available to the user.

Factors which may cause difficulties in the design of nonrecursive filters are the same as in the program IIR. Note that the instructions for use of the PMFIR design module are very similar to those for the KFIR design module.

5.1.6 Experimental Procedure IV.2

1. Design a KFIR filter with the same specifications as those for the IIR filter.
2. Print filter coefficients and plot linear magnitude, log-magnitude, unit sample response and frequency error.
3. What are the differences and similarities between IIR and KFIR filters? Explain!

5.1.7 Experimental Procedure IV.3

1. Design a PMFIR filter with the same specifications as those for the IIR filter.
2. The instructions for using the PMFIR module are the same as those for the KFIR module.
3. Print filter coefficients and plot linear magnitude, log-magnitude, unit sample response and frequency error.
4. Compare KFIR and PMFIR filters.

6. FAST FOURIER TRANSFORM

The Fourier transform has been a principal analytical tool in such diverse fields as linear systems, optics, probability theory, quantum physics, antennas, and signal analysis. Historically, as use of computers increased, the overlap in the realms of applications of discrete- and continuous-time techniques grew, and this provided a natural connection between the two methodologies that heretofore had developed essentially independently. Analysis of many discrete-time systems requires the calculation of Fourier transforms, which traditionally presented a prohibitive computational burden. Nevertheless, the possibilities that were opened up by the digital computer were sufficiently tempting that active work began on the investigation of digital voice encoders, digital spectrum analyzers, and other all-digital systems, with the hope that eventually such systems would be practical. The Fast Fourier Transform (FFT) is an algorithm that proved to be perfectly suited for efficient digital implementation, as it reduced the computation time for discrete Fourier transforms by orders of magnitude. With this tool many interesting but previously impractical ideas suddenly became practical, and the development of discrete-time signal and system analysis techniques moved forward at an accelerated pace.

In experiment V, the FFT of a sinusoidal signal, obtained from a signal generator, is evaluated and displayed. The basic radix-2 Cooley-Tukey decimation-in-frequency FFT algorithm [9] is

implemented in both FORTRAN and the TMS32010 assembly language. The Discrete Fourier Transform (DFT) is defined as

$$X(n) = \sum_{k=0}^{N-1} x_o(k)W^{nk} \tag{6-1}$$

where $W = e^{-j2\pi/N}$ and $n = 0, 1, \dots, N-1$. For $N = 4$, k and n can be represent as 2-bit binary numbers.

$$k = 0, 1, 2, 3 \quad \text{or} \quad k = (k_1, k_o) = 00, 01, 10, 11$$

$$n = 0, 1, 2, 3 \quad \text{or} \quad n = (n_1, n_o) = 00, 01, 10, 11$$

A compact method of writing k and n is:

$$k = 2k_1 + k_o \tag{6-2}$$

$$n = 2n_1 + n_o \tag{6-3}$$

Using the representation (2) and (3); for $N = 4$, (1) can be rewritten as

$$X(n_1, n_o) = \sum_{k_o=0}^1 \sum_{k_1=0}^1 x_o(k_1, k_o)W^{(2n_1 + n_o)(2k_1 + k_o)} \tag{6-4}$$

The basis of the FFT algorithm is shown as following:

$$X(n_1, n_2) = \sum_{k_o=0}^1 \sum_{k_1=0}^1 x_o(k_1, k_o)W^{2n_o k_1} \tag{6-5}$$

The basic radix-2 Cooley-Tukey FFT algorithm

$$X(n) = \sum_{k=0}^{N-1} x_o(k) W^{nk} \quad (6-6)$$

$$x_2(n_o, n_1) = \sum_{k_o=0}^1 x_1(n_o, k_o) W^{(2n_1 + n_o)k_o} \quad (6-7)$$

$$X(n) = x_2(n_o, n_1)$$

Equations (6-6) and (6-7) can be illustrated graphically as shown in Fig. 16.

In experimental procedure V.1, students are asked to compare the results from both the FORTRAN program and the TMS32010 processor. Due to the different word lengths used by the FORTRAN program and the TMS32010 processor the results may differ somewhat. The results for the phase responses deviate from each other to a greater extent than those for the magnitudes, because phase determination is more sensitive to word length. The Fourier Transform of a cosine waveform, with and without a truncation interval equal to a multiple of its period, is demonstrated in experimental procedure V.2. The use of zero padding to obtain more frequency interpolation is demonstrated in experimental procedure V.3. Finally, the magnitude and phase of the FFT of an input signal, obtained from the A/D, is displayed on the screen. The laboratory station now serves as a spectrum analyzer.

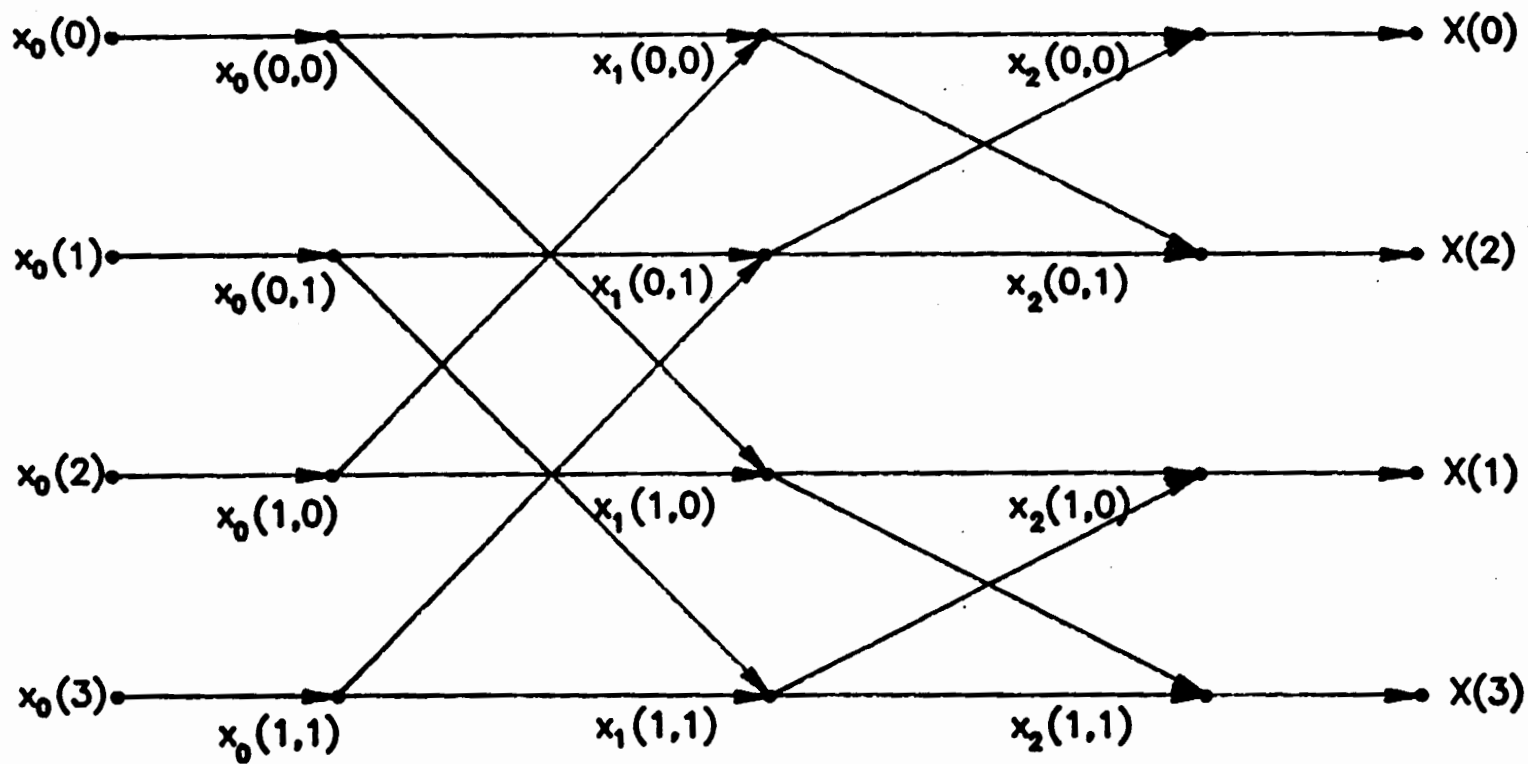


Figure 16. Digram of four-point decimation-in-frequency FFT.

6.1 EXPERIMENT V

The purpose of this experiment is to acquaint the student with the use of the Algorithm Development Package (ADP) subprograms, such as HEAR and TI_LOAD, the TMS32010 MS/PC-DOS CrossWare, and the link editor. The Fast Fourier Transform (FFT) of an input signal, which is generated by a FORTRAN program or measured through the Analog-to-Digital (A/D) converter, is demonstrated.

The basic radix-2 Cooley-Tukey decimation-in-frequency FFT algorithm is implemented in both FORTRAN and the TMS32010 assembly language. Both programs are very similar. The complex input data resides in arrays X and Y, and the DFT is calculated in place. In other words, the output is written back into the X and Y arrays over the input data, which are destroyed.

In the TMS32010 implementation of a single butterfly radix-2 Cooley-Tukey FFT, all data are in an external data memory (I/O, i.e., an address is initially set to a data-instruction address counter and then the data word read from or written to the memory). Because the real and imaginary parts of the complex input data are in sequential locations, not in separate arrays, data index I has a value twice that in the corresponding FORTRAN routine.

A TMS32010 normally supports eight input instructions (IN to channels 0-7), and eight output instructions as well. One of the most useful I/O instructions is IN XX,PA7 that inputs data from dual-port memory location 007 and clears the SINT and SBIO flags. In the FFT.ASM program, there is no checking for the SBIO flag to start the program. When FFT.LOD resides in the TMS32010 with only two switches, "S" and "G", the program will stop after the IN instruction, and wait for BFFT.FOR (listed in Appendix D) to interface with it. After the 128 point FFT is completed, the TMS32010 processor will stop.

6.1.1 HEAR

Program HEAR samples an analog signal, stores the samples in a circular buffer in PC memory, and writes the data to a disk file [5].

The program is executed by typing the following:

```
HEAR[/D]/N/A[OUTPUTFILE/O][INPUTFILE/I][#/S][Text/M]
```

6.1.1.1 Program Switches

/DELETE-OUTPUT Delete output file if it already exists.

/NO-LOAD Do not load TMS32010 sampling program (CIRCLE.LOD)

/ASSUME-DEFAULT Use program defaults (8000 Hz sample rate and no message).

6.1.1.2 File and Parameter Labels

/OUTPUT Output file name.

/INPUT Input file name.

/SAMPLING-RATE Sampling rate in Hertz (less than 24000)

/LIMIT Limit on the number of blocks used, each with 2048 samples.

/MESSAGE Text of message to be included in the file header.

/REPEAT Repeat count for playback (default is infinite).

/PROCESSOR Processor number (for multiprocessor configuration).

HEAR has three modes of operation which are referred to as the initializing mode, the A-to-D mode, and the D-to-A mode. In the initializing mode, the user specifies any parameters not specified by the common line information. In the A-to-D mode, the program takes samples from the A/D converter and writes them into a circular buffer, i.e. when the end of the buffer is reached, the pointer returns to the beginning of the buffer. In the D-to-A mode, the contents of the circular buffer is read out sequentially to the D/A converter. If the command line information is not sufficient to completely specify the operation of the A-to-D stage of the program, then the program Enters the initializing stage as displayed in Figure 17.

Note that when the initializing mode is first Entered, the values specified for the output file, input file, sampling rate, and the message for the file header, will be displayed in the appropriate space. These values (output file name, input file name, sampling rate and message) can be accepted by pressing the Enter key. To change the file name, simply type the new name followed by the Enter key. The default value for the sampling rate is 8000 Hz. The maximum number of characters for the message is 32.

After the text is Entered, the program proceeds to either the A-to-D mode or the D-to-A mode. If an input file was specified, HEAR Enters the D-to-A mode. Otherwise, Hear Enters the A-to-D mode. When all the information is desired, press the Enter key to past the "Message" line, and the program will proceed to the appropriate mode.

320/PC SIGNAL SAMPLING SYSTEM

(C) COPYRIGHT 1985: ATLANTA SIGNAL PROCESSORS INC -- VERSION 1.01

Output Filename:
 Input filename:
 Sampling Rate(Hertz): 008000
 Message:
 F1 to Exit from Program

HEAR[/DELETE-OUTPUT][/NO-LOAD][/ASSUME-DEFAULT] [#1/OUTPUT] [#2/INPUT]
 [#3/SAMPING-RATE] [#4/LIMIT]
 [#5/MESSAGE] [#6/REPEAT]
 [#7/PROCESSOR]

DELETE-OUTPUT: Delete output file if it already exists
 NO-LOAD: Do not load TMS32010 sampling program (CIRCLE.LOD)
 ASSUME-DEFAULT: Use defaults (Rate= 8000 Hertz and no message)
 OUTPUT: Output file name
 INPUT: Input file name
 SAMPLING-RATE: Sampling rate (less than 24000 Hertz)
 LIMIT: Limit on number of 2048 sample block used
 MESSAGE: Message for output file header
 REPEAT: Repeat for playback (default is infinite)
 PROCESSOR: Processor number for multiprocessor configurations

Figure 17. Screen display in initializing stage.

6.1.1.3 A-to-D Mode - Setting Input Signal Levels

When HEAR enters the A-to-D mode, a display will appear as in Figure 18. The horizontal bars in the top half of the display are "meters" that display the periodically updated peak signal level, energy, and position in the circular buffer. If an active signal source is connected to the A/D terminal of the ASPI 320/PC board, the solid horizontal bars in the upper two meters show the instantaneous value of peak signal level and energy in dB. The small vertical bar in the third meter shows the position of the circular buffer pointer at that time. The bars will be continuously moving in actual operation.

The Peak Signal Level meter measures in bits. In order to get the A/D converter accurate, it is necessary to adjust the gain on the signal source until the Peak Signal Level exceeds 10 bits while not exceeding 11.

The energy meter displays the rms value of the signal in dB. The averaging is over 1024 samples, and the display is up-dated every 1024 samples.

6.1.1.4 A-to-D Mode - Capturing A Signal

There are two ways to use the circular buffer of HEAR to capture a signal. Pressing the F1 key when HEAR is in the A-to-D mode, causes the buffer to be filled exactly once followed by a shift to the D-to-A mode. Thus, the signal that occurs after pressing F1 is digitized and stored in the buffer. The amount of signal that is captured depends on the length of the buffer.

Pressing F2 terminates the filling of the buffer immediately, so that the signal that occurred prior to pressing F2 is captured.

ATLANTA SIGNAL PROCESSORS INC.

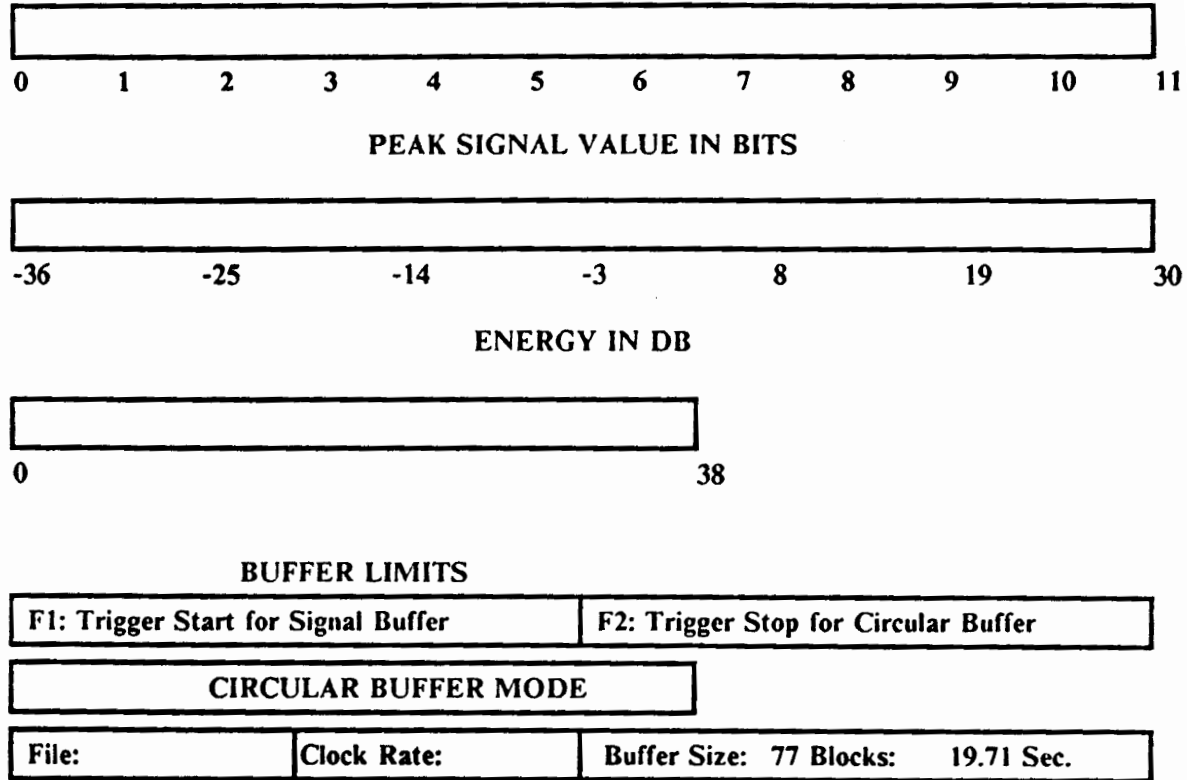


Figure 18. Screen display during the A-to-D stage.

The length of the circular buffer is automatically set by HEAR to use all the memory space not used by the program itself. The third meter shows the buffer limits. It is calibrated in blocks (one block is 2048 samples). The entry in the bottom line shows the file name to be written, the sampling rate, and the buffer size in blocks and in seconds. The latter value is simply 2048 times the number of blocks in the buffer, divided by the sampling rate in samples/sec.

6.1.1.5 D-to-A Mode

If an input file is specified in the initializing mode, the program Enters the D-to-A mode. In this mode, the captured signal can be played back repeatedly to determine if it is acceptable.

The captured signal is repeatedly played back to the D/A converter. The function keys F1-F4 can be used to move the playback limits to the left or to the right. Pressing once causes this limit to move one block of 2048 samples.

Pressing F5 causes the program to leave the D-to-A mode and to return to DOS without writing the captured data to the specified disk file.

Pressing F6 causes the program to write the captured data, delimited by the playback limit marks, into the specified disk file.

Pressing F7 causes the program to exit the D-to-A mode and to return to the A-to-D mode. This can be used when the signal segment captured was not satisfactory and the user wishes to go back and try again to capture a suitable result to save in the specified output file.

6.1.2 Experimental Procedure V.1

1. Copy AFFT.FOR and BFFT.FOR to AEXPV.FOR and BEXPV.FOR.
2. Down-load FFT.LOD to the TMS32010 with two switches, "Stop" and "Go".
3. Use Microsoft FORTRAN to compile, and link to execute AEXPV.FOR.
4. Use VID.BAS to plot the input signal, and the magnitude and phase of the FFT of the input signal.
5. Run BEXPV.FOR by typing "COM BEXPV".
6. Use VID.BAS to plot the input signal, and the magnitude and phase of the FFT of the input signal.
7. Compare the two sets of plots and explain similarities and differences.

6.1.3 Experimental Procedure V.2

1. Suppose we have an input signal

$$f(t) = \cos(2\pi f_o t)$$

where $f_o = \frac{1}{64}$, with sampling period $T = 1.0$ and $N = 128$ samples. Note that if the truncation interval is chosen equal to a multiple of the period, the frequency domain sampling function (i.e. $H(f) = \frac{1}{T} \sum_{n=-\infty}^{n=\infty} \delta(f - \frac{n}{T})$) is coincident with the zeros of the $\sin(f)/f$ function. As a result, the side-lobe characteristics of the $\sin(f)/f$ function do not alter the discrete Fourier Transform results [9].

2. Change the input signal in BEXPV.FOR to $f(t) = \cos(2\pi f_o t)$.
3. Plot $f(t)$, and the magnitude and phase of its FFT.
4. Suppose that there is another signal $f_1(t)$ with $f_1 = \frac{1}{36.572}$. Run the implementation again.
5. Plot $f_1(t)$, and the magnitude and phase of its FFT.
6. Compare the two magnitude responses. Is it what you expect? Why or why not?
7. What do you suggest to improve the situation? Provide graphs to show that, and indicate what supports your arguments.

6.1.4 Experimental Procedure V.3

1. F16T.ASM performs a 16 point FFT. Down-load F16T.LOD to the TMS32010 as you did in experimental procedure II.1.
2. Let $f(t) = \cos(2\pi f_1 t) + \cos(2\pi f_2 t)$ where $f_1 = \frac{1}{4}$ and $f_2 = \frac{1}{8}$, with $N = 16$ samples and $T = 1.0$.
3. CFFT.FOR is a FORTRAN program which will communicate with F16T.ASM. It is the same as BFFT.FOR, except that it is for a 16-point input signal. Copy CFFT.FOR to CEXPV.FOR. Input the above function to CFFT.FOR and run the implementation.
4. Plot the magnitude of the 16-point FFT of the input signal.
5. Down-load FFT.LOD to the TMS32010.
6. Input the above function with zero padding of 112 points.
7. Run the implementation. Plot the magnitude of the 128 point FFT of the input signal.
8. Compare with the result from step 4. Explain if, and why, the two plots are the same; and in which way.

6.1.5 Experimental Procedure V.4

1. Connect the signal generator to the Analog-to-Digital (A/D) converter and set the frequency at 1000Hz.
2. Use HEAR to capture the signal from the A/D converter, by using the default sampling frequency. Note that every block represents 2048 bits, and try to make the signal file as small as possible.
3. The signal file which has been generated with HEAR is in machine code and the first 128 bytes are used for heading, which indicates the sampling rate, the name of the file and so on. The data of the signal file is stored after the heading. APPEND.PAS strips off the first 128 bytes from a signal file which has been generated with HEAR. Upon running the program the student is prompted to either (1) strip off a header or (2) append a file header. Both options prompt the student for an input and output file name. In this experiment, the first option is used, and the output file name must be OUT4.DAT. To strip off the heading, type "append" with option 1.
4. Link the FORTRAN program and plotting subroutine, and execute the program by typing:
go expv4.
5. The plot of the input signal will be displayed on the screen, and the magnitude and phase of the FFT of that input signal will be displayed upon hitting the Enter key.
6. Use the printScreen key to plot these three waveforms.
7. Is it what you expected? Why or why not?

8. Use zero padding of 64 points and 96 points to increase frequency interpolation. Plot input signal, magnitude and phase. Compare the three sets of waveforms.
9. After finishing all of the above, please delete the signal file OUT4.DAT.

7. DIGITAL FILTER DESIGN

In the previous chapters and experiments, the use of software associated with the Algorithm Development Package (ADP), the TMS32010 MS/PC-DOS CrossWare and the Link Editor, the Model 3202 solid-state, variable-frequency filters, and the Digital Filter Design Package (DFDP) have been introduced and discussed. In experiment VI, students are given an opportunity to design some digital filters based on the knowledge and experience they have gained in class, and from previous experiments.

In experimental procedure VI.1, students will design four different types of IIR filter by using the bilinear transformation of analog Butterworth, Chebyshev I, Chebyshev II, and elliptic designs. The results are supported by graphs from DFDP. In experimental procedure VI.2, FIR filters are designed by using the Kaiser window method and the Parks-McClellan method. If the specification of the filter allows $\pm 10\%$ of error in both stopband and passband, the number of bits for quantization can be determined by choosing the number of quantization bits in the DFDP. The differences between the Kaiser window method and the Parks-McClellan method are demonstrated, and students are asked to draw conclusions there from. At the end, students are asked to design the smallest order FIR and IIR filters for given specifications, both with 12 bits of quantization, and to write them to an output file. The output file is used to generate the TMS32010 assembly code by using CGEN, which is another module in DFDP. The programs are assembled, linked

and loaded to the TMS32010 processor. A system for running both filters in real-time is designed. In order to improve the aliasing and imaging attenuation by 20 dB, the Model 3202 variable-frequency, electronic filter is used. Students will have an opportunity to design and then implement this system, for real-time operation and verification.

7.1 EXPERIMENT VI

The purpose of this experiment is to give the student an opportunity to design some digital filters with the use of all the software associated with the Algorithm Development Package (ADP), the Digital Filter Design Package (DFDP), the TMS32010 MS/PC-DOS CrossWare, and the Link Editor, and the Model 3202 solid-state variable-frequency, electronic filters.

7.1.1 Filter Specification

Lowpass, cutoff frequency = $0.25 \omega_c$, passband ripple = 0.01, transition width = $0.05 \omega_c$, stopband attenuation ≥ 60 dB. $\omega_c = 1$ kHz.

7.1.2 Experimental Procedure VI.1

1. Design IIR filters that satisfy the specifications above and in addition have respectively the following characteristics:
 - a. The magnitude response is a smooth function of frequency. The filter has N zeros at $z = -1$ ($\omega = \pi/T$), where N is the order of the denominator.
 - b. The magnitude response exhibits equiripple behavior in the passband and decreases monotonically to zero in the stopband. The filter has N zeros at $z = -1$.
 - c. The magnitude response exhibits equiripple behavior in the passband and monotone decreasing behavior in the stopband. The filter has N poles and N zeros, with the zeros being positioned on the unit circle.
 - d. The magnitude response exhibits equiripple approximations in both passband and stopband. The filter has N poles and N zeros, with the zeros being placed on the unit circle.
2. Define each filter type, and plot the magnitude and phase response of each filter.
3. Provide graphs to support your answers, by indicating where the specifications are met.

7.1.3 Experimental Procedure VI.2

1. Design FIR filters that satisfy the specifications above using the Kaiser window method and the Parks-McClellan method.
2. Plot the magnitude and phase response of the filters.
3. How many bits do you need for quantized coefficients in order to satisfy the specification to within errors of $\pm 10\%$? Provide graphs to support your answer.
4. What is the difference between the Kaiser window method and the Parks-McClellan Method? Explain why.

7.1.4 Experimental Procedure VI.3

1. Choose the smallest order IIR filter and FIR filter that you designed in VI.1 and VI.2. Quantize the coefficients to 12 bits for both filters.
2. Use CGEN to generate the TMS32010 assembly language program to implement the above digital filters.
3. Connect the waveform generator, oscilloscope, and TMS32010 board as in Experiment III. Use the Model 3202 electronic filter as needed.
4. Assemble, link, and down-load the programs.
5. Plot the magnitude and phase response of each filter from observations with the oscilloscope.
6. Suppose that we want to improve the stopband attenuation by 20 dB without changing the order of the filters. Design and implement the system.
7. What are the cutoff frequencies of the Model 3201 filter, for the IIR filter, and for the FIR filter? Do the observations from the oscilloscope corroborate the specification? Plot the magnitude response of each filter to support your answer.

REFERENCES

1. Leland B. Jackson, DIGITAL FILTERS AND SIGNAL PROCESSING, Kluwer Academic Publishers, Hingham, MA., 1986, pp. 103-142.
2. T.W. Parks and J.H. McClellan, "Chebyshev Approximation for Nonrecursive Digital Filters with Linear Phase," IEEE Trans. Circuit Theory, Vol. CT-19, March 1972, pp. 189-194.
3. Digital Filter Design Package Manual, Atlanta Signal Processors Incorporated, Version 2.01, 1985.
4. TMS32010 User's Guide, Texas Instruments, Houston, TX., 1983.
5. The Algorithm Development Package Manual, Atlanta Signal Processors Incorporated, Version 1.01, 1985.
6. TMS32010 CrossWare Installation Guide, Texas Instruments, Houston, TX., 1985.
7. Link Editor User's Guide, Texas Instruments, Houston, TX., 1985.
8. Model 3200 (R) & Model 3202 (R) Operating and Maintenance Manual, Avon, MA., Krohn-Hite Corporation, 1984.

9. E. Oran Brigham, THE FAST FOURIER TRANSFORM, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1974, pp. 172-178.
10. C. S. Burrus and T. W. Parks, DFT/FFT CONVOLUTION ALGORITHMS: THEORY AND IMPLEMENTATION John Wiley & Sons, 1985, pp. 145-221

Appendix A. Program Source Code List

Linear convolution

```

      IDT 'LCONV'
*
*
*   A general routine for a length-N linear convlution.
*   For this particular implementation, n = 8.
*
*
XNEW EQU 1           * NEWEST INPUT SAMPLE
DOUT EQU 6           * OUTPUT TO PORT 6 (D/A)
DIN  EQU 7           * INPUT FROM PORT 7 (A/D)
X    EQU 8           * END OF DATA POINTS X (N)
H    EQU 16          * END OF IMPULSE RESPONSE SEQUENCE (2N)
YOUT EQU 17          * OUTPUT LOCATION (2N+1)
ONE  EQU 18          * CONTAINS THE VALUE 1 (2N+2)
*
      AORG 0           * BRANCH TO THE BEGINNING OF THE PROGRAM
      B    LAST
*
*   IMPULSE RESPONE TERMS.
*
      AORG 50
H1   DATA >4000
H2   DATA >AA58
H3   DATA >4000
H4   DATA >0000
H5   DATA >0000
H6   DATA >0000
H7   DATA >0000
H8   DATA >0000
*
*   MAIN PROGRAM
*
      AORG 100
LAST BIOZ GET        * CHECK FOR BIO FLAG SET
      B    LAST      * IF BIO IS NOT SET, CONTINUE TO WAIT
*
*   BEGIN OF LINEAR CONVOLUTION
*
GET   LDPK 0
      LACK 1
      SACL ONE      * ONE = 1
*
```



```

LARK ARO,H      * ARO ADDRESSES DATA LOCATIONS
LARK AR1,7     * AR1 IS USED AS A LOOP COUNTER
LOADH          *
LARP ARO       * LOAD THE IMPULSE RESPONSE
TBLR *- ,AR1
SUB ONE
BANZ LOADH
*
LARK AR1,X     * AR1 USED TO ADDRESS DARA AND AS A COUNTER
LOADX          *
ZAC            * INITIALIZE FILTER
SACL *
BANZ LOADX
*
TEST          *
BIOZ PUT
B TEST
PUT          *
OUT YOUT,DOUT
LARP ARO
IN XNEW,DIN * GET NEXT INPUT SAMPLE
*
LARK ARO,X     * ARO POINTS TO THE INPUT SEQUENCE
LARK AR1,H     * AR1 POINTS TO THE IMPULSE RESPONSE
*
ZAC
LT *- ,AR1
MPY *- ,ARO
LOOP          *
LTD *- ,AR1    * LOAD INPUT SEQUENCE, ACCMULATE RESULT
MPY *- ,ARO    * MULTIPLY IMPULSE RESPONSE
BANZ LOOP      * LOOP N TIMES
*
APAC          * ACCUMULATE LAST MUTIPLY
SACH YOUT,1   * ACC*2-1=DMA
*
B TEST        * GET THE NEXT INPUT SAMPLE
END

```

Appendix B. Program Source Code List

Exchange data between PC and the TMS32010

```
; THIS PROGRAM IS TO COMMUNICATE WITH TMS32010
; PROCESSOR UNDER (BIO) CONTROL. AX CONTAINS
; THE DATA RECEIVED FROM THE TMS32010 BX CONTAINS
; THE DATA TO BE TRANSMITTED TO THE TMS32010 AND
; ES THE SEGMENT POINTER TO 320/PC MEMORY WINDOW.
; THE CORRESPONDING ON THE HOST PROCESSOR IS
; GIVEN IN THE FORM OF SUBROUTINE CALL OUT320.
;
DATA SEGMENT PUBLIC 'DATA'
DATA ENDS
DGROUP GROUP DATA
CODE SEGMENT 'CODE'
ASSUME CS:CODE,DS:DGROUP, SS:DGROUP
;
PUBLIC OUT320
OUT320 PROC FAR
PUSH BP ; SAVE FRAMEPOINTER ON STACK
MOV BP,SP ; SET BP = TO STACK POINTER
PUSH DI
PUSH SI
LES BX,DWORD PTR [BP+6] ; GET PARAMETER ADDR IN BX
MOV AX,ES:[BX] ; GET PARAMETER VALUE IN AX
MOV BX,AX
;
MOV AX,40960 ; PREPARE TO SETUP ES
MOV ES,AX ; ES PTS TO SEG. START AT 6
;
BIO320: MOV AL,2 ; SBIO STATUS BIT IS 2
LP1: TEST AL,ES:2000H ; CHECK FOR 1 IN STATUS BIT
JZ LP1 ; 0 MEANS PRE. DATA NOT USE
MOV AX,ES:12 ; GET DATA FROM ADDR 12
MOV ES:14,BX ; PUT DATA IN ADDR 14
PUSH AX ; SAVE RESULT FOR LATE
MOV AL,2 ; PUT A 2 IN AX FOR SBIO CTRL
MOV ES:2000H,AL ; OUTPUT CTRL BYTE TO SET BIO
POP AX ; RETRIEVE THE ANSWER
;
EXIT: POP SI ; RETURN TO CALLING PROGRAM
POP DI
MOV SP,BP
POP BP
RET 4 ; NUM := NO. PARAMETER * SIZE
;
OUT320 ENDP
CODE ENDS
END
```

Appendix C. Program Source Code List

Interface with the TMS32010

```
$LARGE
C
C   THIS PROGRAM CREATES AN INPUT SIGNAL Y WHICH
C   WILL COMMUNICATE WITH THE TMS32010 PROCESSOR.
C   SINCE THE MAXIMUM BITS OF ACCUMULATE IS USED
C   (LEFT SHIFTED FOR 14 BITS), THE VALUE OF Y
C   WILL HAVE TO BE GREATER THAN -2 AND LESS THAN
C   2. OUT320 IS A SUBROUTINE WHICH WILL INPUT
C   THE DATA TO THE TMS32010 PROCESSOR AND RETURN
C   THE CORRESPONDING
C   sample is in INTER.ASM
C
C   INTEGER*2 OUT320,K
C   OPEN (UNIT = 10, FILE = 'SFILE', STATUS = 'NEW')
C   OPEN (UNIT = 20, FILE = 'DFILE', STATUS = 'NEW')
C   OPEN (UNIT = 30, FILE = 'OFILE', STATUS = 'NEW')
C   OPEN (UNIT = 40, FILE = 'IFILE', STATUS = 'NEW')
C
C   Implement a sine wave with sampling frequency 1/T
C
C   T=1./3.
C   PI=3.14157
C   DO 20 I=1,100
C       X=SIN(PI*T*FLOAT(I))
C
C   Make the maximum value of input in accumulate
C
C       K=INT(X*1024.*16.)
C
C   Call OUT320
C
C       Y=FLOAT(OUT320(K))/(1024.*16.)
C
C   Create output data file
C
C       WRITE(10,12) S,FLOAT(I)
C       WRITE(20,12) D,FLOAT(I)
C       WRITE(30,12) Y,FLOAT(I)
C       WRITE(40,12) X,FLOAT(I)
12      FORMAT(2F20.9)
20      CONTINUE
      STOP
      END
```

Appendix D. Program Source Code List

Fast Fourier Transform

```
#LARGE
C
C   This program creates an input signal with
C   real part XI and imaginary part YI, and
C   it communicates with the TMS32010
C   processor under control of the 8086
C   assembly program INTER.ASM. All the
C   value in this program must be within -7
C   to 7 as discussed in experiment II.
C
REAL XI(640),YI(640),XO(640),YO(640),OUT(640),
+T(640),P(640)
OPEN (UNIT = 10, FILE = 'INFO', STATUS = 'NEW')
OPEN (UNIT = 20, FILE = 'INFN', STATUS = 'NEW')
OPEN (UNIT = 30, FILE = 'OTFN', STATUS = 'NEW')
C
C   Implement an input signal for 128 points
C
XI(0)=0.00
YI(0)=0.0
DO 20 I=1,16
  YI(I)=0.0
  IF (I .LE. 8) THEN
    XI(I)=0.01+XI(I-1)
  ELSE
    XI(I)=XI(I-1)-0.01
  ENDIF
  WRITE(10,12) FLOAT(I), XI(I)
20 CONTINUE
C
C   Call Subroutine Section
C
CALL FFT(XI,YI,XO,YO)
C
C   Comput Magnitude and Phase Response
C
DO 40 K=1,16
  OUT(K)=SQRT(XO(K)**2+YO(K)**2)
  IF (( -0.001 .LT. XO(K) .AND.
+XO(K) .LT. 0.001 ) .AND. ( -0.001 .LT.
+YO(K) .AND. YO(K) .LT. 0.001 )) THEN
    XO(K)=0.0
    YO(K)=0.0
```

```

        ELSE IF ( XO(K) .EQ. 0.0 .AND.
+YO(K) .EQ. 0.0 ) THEN
            P(K)=0.
        ELSE IF ( XO(K) .EQ. 0.0 .AND.
+YO(K) .GT. 0.0 ) THEN
            P(K)=90.
        ELSE IF ( XO(K) .EQ. 0.0 .AND.
+YO(K) .LT. 0.0 ) THEN
            P(K)=-90.
        ELSE
            P(K)=ATAN2(YO(K),XO(K))*180./3.141593
        ENDIF
12      WRITE(20,12) FLOAT(K), P(K)
        FORMAT(2F15.6)
40     WRITE(30,12) FLOAT(K), OUT(K)
        CONTINUE
        STOP
        END

```

C
C
C
C
C
C
C
C
C
C
C

```

        SUBROUTINE SECTION
        SUBROUTINE FFT(XI,YI,XO,YO)

```

This subroutine communicates with TMS32010 processor when TMS32010 processor is running a FFT program. Since the maximim bits of accumulate is used, the value of XI and YI have to be greater than -7 and less than 7. The subroutine computes stage by stage, the end results are still in XI and YI. XO and YO which are the results of FFT are the results after digital reverse.

```

        INTEGER*2 OUT320,KI(640),JI(640),IA,KL(640),
+JL(640),II,LI,IO,LO
        REAL XI(640),YI(640),XO(640),YO(640),T(640)
        J=OUT320(IA)

```

C
C
C

First Stage

```

        DO 25 I=1,8
            IA=0000
            II=INT(OUT320(IA)/2+1)
            JI(II)=INT(XI(II)*1024.*16.)
            KI(II)=INT(YI(II)*1024.*16.)
            IA=OUT320(JI(II))
            LI=INT(OUT320(KI(II))/2+1)
            JI(LI)=INT(XI(LI)*1024.*16.)
            KI(LI)=INT(YI(LI)*1024.*16.)
            IA=OUT320(JI(LI))
            IA=OUT320(KI(LI))
            T(IO)=OUT320(IA)
            IO=INT(OUT320(IA)/2+1)
            JI(IO)=OUT320(IA)
            KI(IO)=OUT320(IA)
            XO(IO)=FLOAT(JI(IO))/(1024.*16.)
            YO(IO)=FLOAT(KI(IO))/(1024.*16.)
            LO=INT(OUT320(IA)/2+1)
            JI(LO)=OUT320(IA)
            KI(LO)=OUT320(IA)
            XO(LO)=FLOAT(JI(LO))/(1024.*16.)
            YO(LO)=FLOAT(KI(LO))/(1024.*16.)
25     CONTINUE

```

```

C
C
C      Comput Rest of Stage
DO 30 L=1,3

      DO 101 M=1,8
      IA=0000
      II=INT(OUT320(IA)/2+1)
      IA=OUT320(JI(II))
      LI=INT(OUT320(KI(II))/2+1)
      IA=OUT320(JI(LI))
      IA=OUT320(KI(LI))
      T(IO)=OUT320(IA)
      IO=INT(OUT320(IA)/2+1)
      JI(IO)=OUT320(IA)
      KI(IO)=OUT320(IA)
      XO(IO)=FLOAT(JI(IO))/(1024.*16.)
      YO(IO)=FLOAT(KI(IO))/(1024.*16.)
      LO=INT(OUT320(IA)/2+1)
      JI(LO)=OUT320(IA)
      KI(LO)=OUT320(IA)
      XO(LO)=FLOAT(JI(LO))/(1024.*16.)
      YO(LO)=FLOAT(KI(LO))/(1024.*16.)
101  CONTINUE
30   CONTINUE

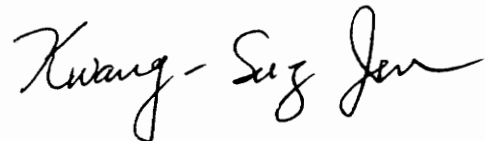
C
C
C      Digital Reverse Counter
J=1
N1=15
DO 50 I=1,N1
  IF (I .GE. J) GOTO 60
  XT=XO(J)
  XO(J)=XO(I)
  XO(I)=XT
  XT=YO(J)
  YO(J)=YO(I)
  YO(I)=XT
60   K=16/2
70   IF (K .GE. J) GOTO 80
      J=J-K
      K=K/2
      GOTO 70
80   J=J+K
50  CONTINUE
RETURN
END

```

VITA

The author was born in Shanghai, China on February 26, 1961. She studied in textile engineering at the Shanghai Textile Engineering Institute in 1979. After her sophomore year at the Shanghai Textile Engineering Institute, she came to the United States. She graduated from Savannah State College in Georgia with a degree of Bachelor of Science in Electronics Engineering Technology in March 1985. During her studies at Savannah State, she worked as a tutor in mathematics. She received the Sarah Mills Hodge Memory Scholarship 1984-1985.

She began her graduate studies at Virginia Polytechnic Institute and State University (VPI&SU) in September 1985. She taught Electronics Devices, Electronic Circuits, Circuit Analysis and associated laboratory courses at New River Community College.

A handwritten signature in black ink, reading "Kwang-Suz Jen". The signature is written in a cursive style with a long horizontal stroke at the end.