

https://dspace.org/

A computer simulation for aiding the design of a DSP radio

Bassios, A. D.

2013-09-27
http://hdl.handle.net/10539/13161

Downloaded from a DSpace repository. https://dspace.org/

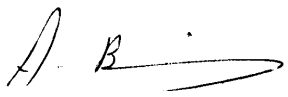
A COMPUTER SIMULATION FOR AIDING THE DESIGN OF A DSP RADIO

by : A.D. BASSIOS

A COMPUTER SIMULATION FOR AIDING THE DESIGN OF A DSP RADIO

by : A.O. BASSIOS

Presented as a four month design project in partial fulfillment of the degree of Master of Science in Engineering in the branch of Electrical Engineering



ABSTRACT

The use of digital signal processing (DSP) techniques to implement functions traditionally done by analogue methods is becoming increasingly popular amongst researchers and designers working in the telecommunications field. A good DSP design approach is to first simulate on a computer using a high level language. Problems exist with large unstructured simulations involving a number of users. This report presents a flexible and user friendly simulation structure to aid designers of radios using DSP techniques and components. The needs of the simulation users are investigated and a number of structures are evaluated. The shell of the simulation is then described. The simulation presented gives the user the ability to easily simulate the effects of different wordlengths of both fixed and floating point DSP microproces sors.

ACKNOWLEDGEMENTS

I wish to acknowledge the assistance received from Fuchs Electronics in the use of their computer facilities and I would like to thank them for the opportunity to be involved in a very interesting DSP radio development project.

In addition, I would like to thank my supervisor, Professor H.E. Hanrahan, for his helpful comments.

CONTENTS

1. INTRODUCTION 1

1.1. Possible Solutions

1.2. Structure of the Report

2. OBJECTIVE OF THE SIMULATION 10

2.1. Introduction

2.2. Needs Of The Users

2.3. Needs of the Programmers

3. SIMULATION STRUCTURE 13

3.1. Introduction

3.2. Aspects Concerning the Programmer

3.3. Aspects Concerning the User

3.4. Structures Considered for the Simulation

3.5. Structure Implemented

4. SIMULATION SHELL 27

4.1. Introduction

4.2. Programme Descriptive Language

4.3. Description of Simulation Shell Programmes

5. SIMULATION OF DIFFERENT CHIPS . . . 44

5.1. Division into Subdirectories

5.2. DSP Microprocessors

5.3. The Truncation Process

5.4. Linking versus Copying of .EXE files

5.5. Important Data Files

6. INITIALISATION OF THE SIMULATION 56

6.1. Simulation Sampling Frequency and Sample Length

6.2. Type of Chip Chosen

7. HIGH LEVEL LANGUAGE PROGRAMMING RULES AND GUIDELINES 60

8. IMPLEMENTING FUNCTIONS IN THE SIMULATION 70

8.1. Introduction

8.2. The DCL Functional Procedure

8.3. Inclusion of the FORTRAN Programmes

8.4. PDL Listings for this Chapter

9. SIMULATION FUNCTIONS 82

9.1. Utility Functions

9.2. Secondary Utility Functions

9.3. Tertiary Utility functions

10. TRANSMIT & RECEIVE PATH STEPS 93

10.1. Creating Signals

10.2. Sampling Signals

10.3. Amplification Menu

10.4. Audio Processing Menu

10.5. Modulation/Demodulation

11. EXAMPLE OF SIMULATION USAGE 96

11.1. Introduction

11.2. Explanation of Block Diagram

11.3. Example of Simulation Process

12. CONCLUSION . • • . . • • • . • . • . . . . • • . . • . 113

REFERENCES 119

LIST OF FIGURES

3.1 The Tree Structure 17

3.2 The Network Structure 18

3.3 Serial Processing in a Radio 19

3.4 The Linear-tree Structure . . . 20

3.5 The Modified Linear-Tree Structure 22

3.6 Block diagram of Simulation Structure . 24

3.7 Menu tree structure . . 25

4.1 Interaction of the main programmes 31

4.2 Transmit Path . . 36

4.3 Receive Path 38

4.4 Channel menu structure 40

4.5 Utility menu structure 43

5.1 Simulation Directory Structure 44

11.1 Block diagram of a simulation process . 97

CHAPTER 1. **INTRODUCTION**

Digital signal processing (DSP) techniques offer a number of significant advantages over conventional analogue signal processing. Much of the processing in radios has the potential to be implemented digitally. current digital signal processors (DSPs) are fast enough to do much of the baseband and IF signal processing. Future advances will make it possible to move the digital part of a radio increasingly close to the antenna, with the ultimate aim of a fully digital radio. Apart from being able to implement functions previously done by analogue methods, digital processing allows far more complex algorithms to be implemented. Thus, digital processing has applications in the radio communications field which were previously difficult or impossible to implement.

The advantages of using digital signal processing are numerous:-

1. Higher Reliability - A small number of DSP hardware components would replace many analogue components.

2. Easier Manufacture The fine tuning of analogue components would be minimised.

3. Flexibility - The modulation schemes, filters and other operations of the DSP software could be changed without altering the hardware.

4. Less Component Ageing.

5. Greater Temperature Stability.

6. Special requirements and different versions of a design may be programmed in software.

7. Built in software testing facilities are possible.

8. Techniques which are virtually unusable in analogue systems become possible in DSP. Examples are the use of new techniques of adaptive filtering to allow equalisation of the channel path [1] and complex AGC algorithms with application to HF receivers.

9. Digital designs can be accurately simulated at a high level so that algorithms can be tested before any hardware design. As a result of the above, universities and telecommunications companies the world over are investing time and resources in the research and development of DSP radios [2,3,4].

A good design approach to developing the software code to be used by the DSPs in the radio would be to first simulate on a computer using a high level language such as Pascal or FORTRAN. The advantage of this approach is that the DSP designers may experiment with, and optimise, the algorithm being developed without becoming involved with the complications of assembly level code. This report describes a simulation structure that was developed to aid in the design of a DSP radio.

A problem with large computer simulations is that the designers tend to duplicate work done by their colleagues, or even programmes or subroutines that they themselves might have done some time previously. Interfaces between the Input/Output of one programme/procedure and another can become complicated when an undisciplined, unstructured approach is adopted for the simulation.

A solution to this problem is to provide a structure whereby designers can include their contributions to the simulation without difficulty. The structure should allow the user to use other designers' contributions to the simulation both easily and effectively.

This report describes a simulation structure that was developed to aid in the design of DSP radio.

1.1. Possible Solutions

This section describes some of the options that were considered for the DSP radio development simulation.

1.1.1. Commercial Display Software

A number of general DSP display software packages are currently being offered by their vendors. Among these are packages such as ILS, DADiSP and DSPlay [5,6,7).

This type of package does not provide the optimum environment needed for the complex DSP radio simulation. The packages are oriented towards display and are very general in nature. They do not provide the specialised features or ease of use wanted for DSP radio design. The chief requirement is for a friendly DSP environment where DSP designers can write their own programmes with a view to implementation in a DSP chip at a later stage.

1.1.2. The FUN System

A DSP development environment, called FUN (functional language) was written at the University of Stellenbosch (8). FUN is a good attempt to provide a general DSP development environment. However, some of its disadvantages (with respect to DSP radio design) lie in its generality. Unlike some of the display software packages mentioned in the previous section, FUN is not menu driven (apart from the help and some other facilities). It is thus not quite as user friendly.

One of the requirements of the simulation was that it was to be run on an in-house VAX 750 computer, which does not have a Pascal compiler. Many of the functions included in FUN were written in Pascal and thus could not easily be altered to suit the DSP radio design.

It was not trivial to alter the FUN simulation environment to enable easy simulation of different DSP microprocessors with differing wordlengths.

1.1.3. A General In-Circuit-Emulator

An interesting approach to solving the problem of determining the effects of differing wordlengths on the algorithms was presented by Carter (9). He presents a general in-circuit-emulator (ICE). The ICE allows the

emulation of a number of DSPs. However, this is not an alternative to a high level simulation as the routines must be written in low-level assembly languages for each DSP chip to be investigated.

1.1.4. Developing a Simulation Specifically for DSP Radio Design

An alternative to the above methods was to develop a simulation in-house. The disadvantage of this method was that time would have to be spent on developing the structure. This was weighed up against the following factors:-

1. The simulation could be written to exactly fulfill the requirements of the DSP radio designers, and omit unnecessary and confusing functions.

2. It could incorporate the desirable features of the above mentioned options (eg. be menu based AND simulate for different DSPs).

3. It could include features to guide the inexperienced telecommunications/DSP engineer into using it correctly and facilitate understanding of the

fundamental issues of radio design, and thus help good design.

4. The DSP radio designers would be actively involved in writing the DSP software and would thus gain valuable DSP experience and expertise.

5. The programmes could be written in a way that closely simulates the actual implementation of an algorithm in a DSP chip.

6. The initial cost of buying commercial software would be avoided.

7. An in-house simulation could be written to have powerful abilities to simulate the characteristics and effects of different DSP microprocessor word configurations (eg. 24 bit fixed point vs. 22 bit floating point). It could be written so that comparisons between the effects of different chips could easily be made by the simulation user. This type of simulation is especially important in radio design where signals typically have very large dynamic ranges.

After considering the options it was decided to write an in-house simulation.

1.2. Structure of the Report

The rest of this report describes the structure and shell of the simulation developed. Much of the detail of the simulation has already been implemented, but it is beyond the scope of this report to give a description of more than a few examples of the detail. However, more information may be found on the detail and use of the simulation in the references [10,11,12].

Chapter 2 describes the needs of the user and presents the objectives of the simulation.

In chapter 3 a number of different structures are considered for the simulation.

Chapter 4 describes the shell programmes at the top of the simulation hierarchy.

Chapter 5 describes how the simulation is controlled and how the simulation of different DSP microprocessors is facilitated by subdividing the simulation's programmes into a number of subdirectories.

In chapter 6, the function of the more important programmes in the simulation are presented.

Chapter 7 gives the rules and guidelines for high level language programming in the simulation.

Chapter 8 shows how to include a function into the simulation.

Chapters 9 and 10 contain short descriptions of the functions available in the simulation.

An example of how the simulation is used is shown in chapter 11.

The report is concluded in chapter 12.

CHAPTER 2. OBJECTIVE OF THE SIMULATION

2.1. Introduction

This chapter describes the needs of the simulation users. Distinction is made between the users who use the simulation purely as a tool (users) and those actively involved with enhancing and expanding the simulation (programmers).

2.2. Needs Of The Users

1. User friendliness :- The simulation should be simple for an inexperienced user to use. It should guide the user into using it correctly.

2. Flexibility :- It should allow the more advanced user to perform unconventional sequences of functions.

3. Advanced features :- It should incorporate advanced features without complicating the simulation for the inexperienced user.

4. Different word configurations :- It should allow the easy simulation of different wordlength floating and fixed point configurations.

5. utility functions :- It should provide easy access to commonly used utility functions such as the display of signals .

2.3. Needs of the Programmers

1. Clarity :- The structure of the simulation should be clear and understandable .

2. Easy expandability :- The programmers should be able to include functions in the simulation without difficulty.

3. Flexibility :- The FORTRAN or pascal designers should write their programmes in a structured and well documented way , but there should be the minimum of restrictions imposed by the simulation.

4. Truncation methods :- The simulation should provide a method whereby it is easy for the programmers to design a programmes that simulate the effects of different word configurations.

The aim of the simulation is to attempt to fulfill as many of the above requirements as possible. The balance of this report describes the structure that was developed in an effort to meet the aims.

CHAPTER 3. SIMULATION STRUCTURE

3.1. Introduction

This chapter presents and evaluates a number of structures that could have been used for the simulation.

3.2. Aspects Concerning the Programmer

Owing to the limited time available to write the simulation, the structure should be such that engineers can write programmes/subroutines quickly and easily without worrying too much about external constraints imposed by the simulation.

The first alternative considered was to write one big simulation programme. The programme would have to be rigidly structured and divided into subroutines. The path of the signal through the simulation would be by passing parameters.

Another method considered was to write a number of shorter programmes and run them by using the DIGITAL Command Language (DCL) on the VAX. (DCL is an interpreted language that allows one to write command procedures containing VAX/VMS operating system commands. See for more details.) The processed signal would be passed from

programme to programme by writing the output in one programme to a data file and specifying this data file as the input to the next programme.

A disadvantage of this method is that it takes more time to read and write data files on the computer. However, since speed is not an essential aspect of the simulation, this is not a major problem. Unfortunately, the use of DCL confines the portability of the simulation to computers that run the VAX/VMS operating system. However, by designing the DCL command procedures using a "Programme Description Language" or POL (explained in section 4.2) rewriting the DCL code for other computers is simplified.

A major advantage of this method is its simplicity. It is easy for a number of programmers to write sub-programmes for the simulation without affecting each other. It is also much simpler for them to include their programmes into the simulation structure. The other method would require recompiling and linking every time a new simulation function was to be included.

By subdividing the simulation into smaller components, the room for error is decreased. Each programmer is able to use his own style in the programmes he writes. There would only be a few rules concerning the input and output of the

programmes that should be adhered to. Also, a record of the signal at every point in the simulation is kept, and thus these signals are readily available at any time.

DCL has some attractive file handling features and commands that can be used with great effect in the simulation.

Another advantage of this method is that programmes written in different languages (eg. FORTRAN and Pascal) can be used in the same simulation (providing that the data files have similar formats).

An advantage of using separate files rather than a big FORTRAN programme is that signals are available at each point in the simulation. This will enable the user to use a DSP simulator, programmed with machine code to process the signal data files instead of the high level language programmes. In this way, once the user has simulated his high level design, he can gradually include his DSP-coded algorithms into the design and show that it still works.

After weighing up these considerations, it was decided to use the DCL method.

3.3. Aspects Concerning the User

one of the most important goals of the simulation is to be user-friendly. It has been found that a clear menu based system can contribute to the friendliness of software such as this.

It was decided to use a menu system with easily understood instructions and questions.

3.4. STRUCTURES CONSIDERED FOR THE SIMULATION

This section proposes a number of structures for the simulation. The usefulness of each of these structures to the simulation are evaluated.

3.4.1. The Tree Structure

A diagram of a tree structure is contained in figure 3.1. This type of structure guides the user through different sub-menus, down the tree to the desired operation. This structure is rigid and useful in many applications. It restricts the number of choices and is thus simpler for the inexperienced user to use.

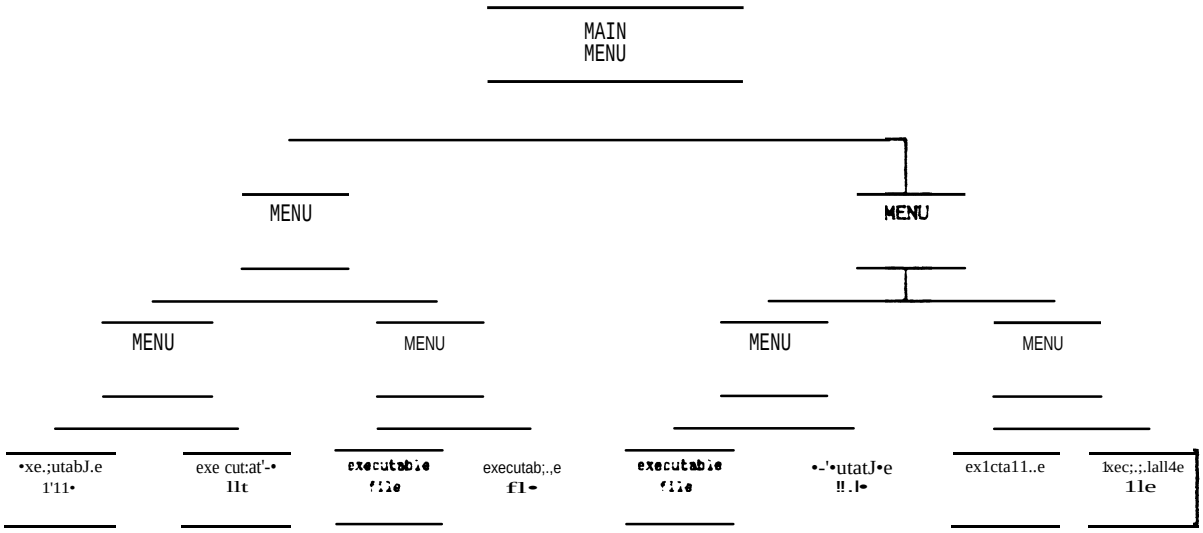


Figure 3.1 The Tree Structure

However, the structure lacks flexibility in that it is, not easy to proceed from a menu in one branch to a menu in another branch. The user would have to return back up to the tree hierarchy until a path to the required node could be found.

3.4.2. The Network Structure

A diagram of a network structure is contained in figure 3.2.

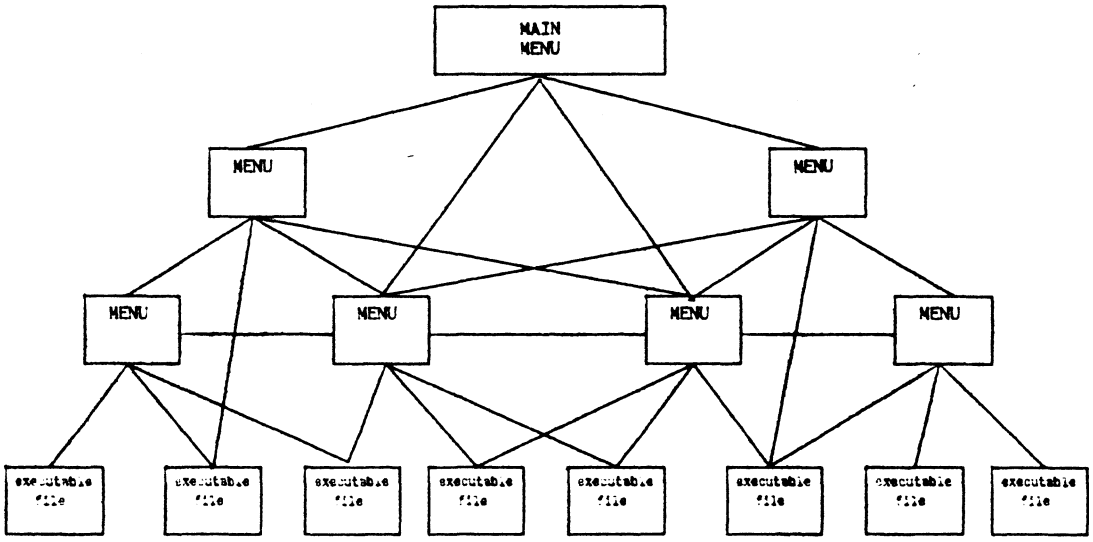


Figure 3.2 The Network Structure.

This variation of the tree structure overcomes the inflexibility of the tree by providing short-cut paths between menus. However the structure can become complex and is not as easy to implement and maintain.

3.4.3. The Linear-tree Structure

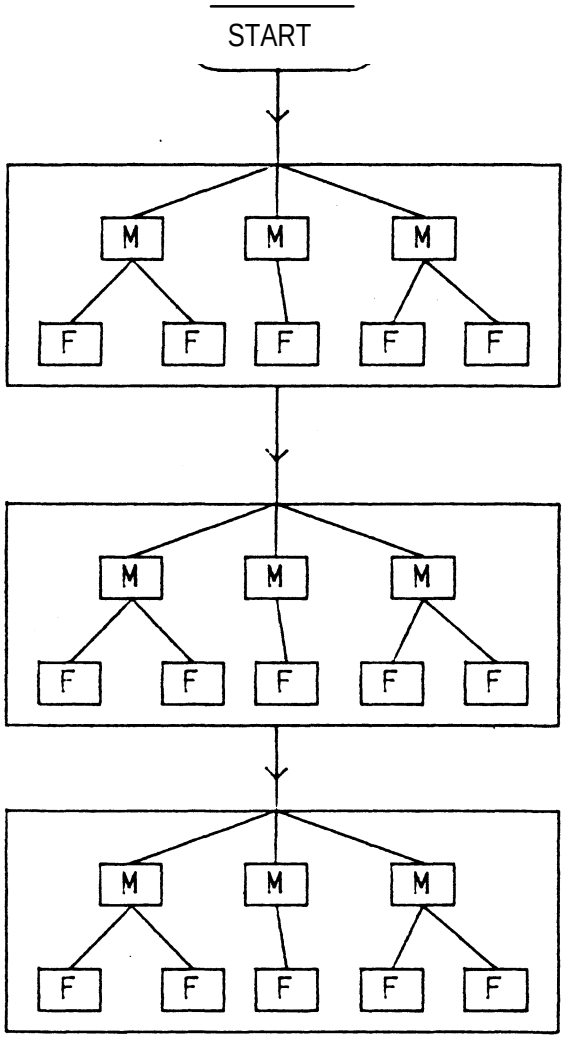
Processing in a radio can been seen to be essentially a linear, serial process

eg. :-

TX	Rx
FILTER	FILTER
AUDIO PROCESSING	AGC
MODULATION	DEMODULATION
FILTER	FILTER
AGC	AUDIO PROCESSING

Figure 3.3 Serial Processing in a Radio.

For the simulation, the different processing blocks would consist of different options to be chosen by the user. These would be implemented with a tree structure. Figure 3.4 contains a diagram demonstrating the linear-tree structure. This structure would provide the rigidity needed for the first-time user, but would be frustrating for the advanced user who might wish to take short cuts to certain parts of the simulation.

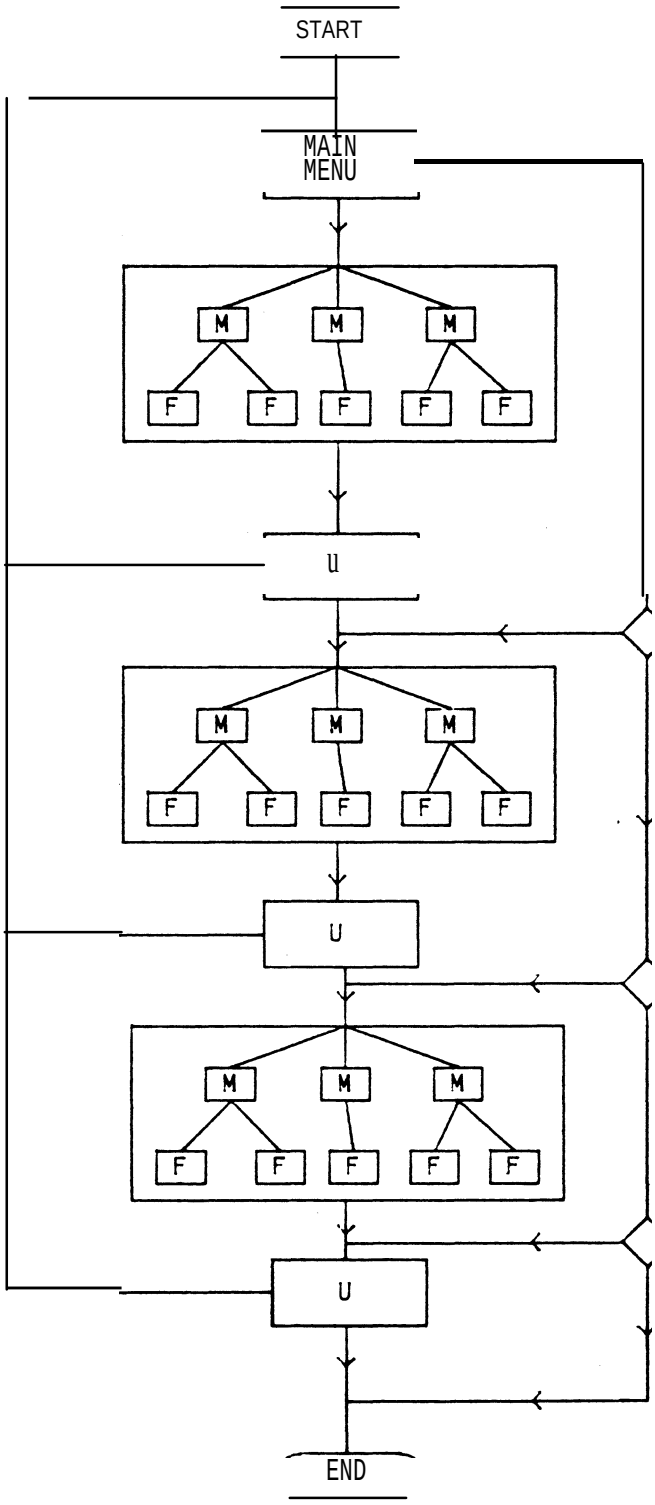


M = menu
F = functional program

Figure 3.4 Linear-tree structure

J.4.4. The Modified Linear-tree Structure

A modification to the above structure should provide the features required by an advanced user while still guiding the inexperienced user down the accepted path. This is done by providing a route back to the beginning of the path from a "Utility Menu". From the beginning of the path, any step in the path may be reached without going through the steps which normally precede it. Alternatively the user may choose the Utility menu option which leads to the next logical step in the path. Figure 3.5 contains a flowchart of the modified linear-tree structure.



M = menu
F = functional program
U = utility menu

Figure 3.5 Modified linear-tree structure

comments on modified linear-tree structure flowchart.

"U" stands for "Utility Menu". This menu would handle commonly used routines such as filter, graphics, FFT's etc. Another important function of this procedure would be to ask the user whether he would like to proceed with the recommended next step of the simulation or return to the start of the path. From the start of the path, he would be able to enter the simulation path at any point.

3.5. Structure Implemented

After considering the aspects above, it was decided to implement a combination of the above structures shown in the block diagram of figure 3.6.

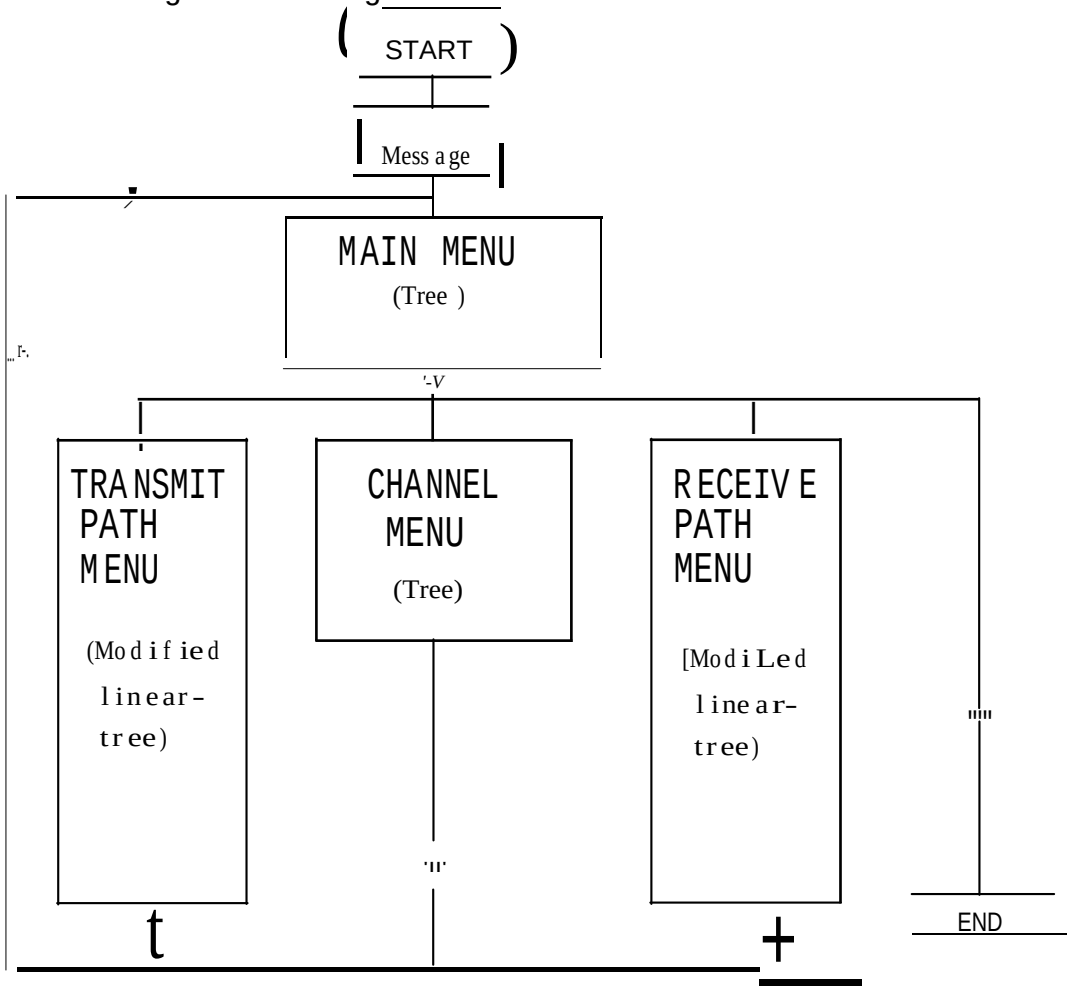


Figure 3.6 Block diagram of Simulation Structure

The modified linear-tree structure seemed to be the most desirable one for the transmit and receive paths as these paths consist of essentially serial steps. The channel menu was chosen to be implemented with a tree structure, because

it was expected that the user will not use the channels in series as a matter of course.

A Utility menu is available from the transmit and receive paths as well as the channel menu. When the Utility Menu processing is completed, programme control is returned to the calling menu.

The above menus were programmed in DCL and call sub-menus using the tree structure shown in figure 3.7.

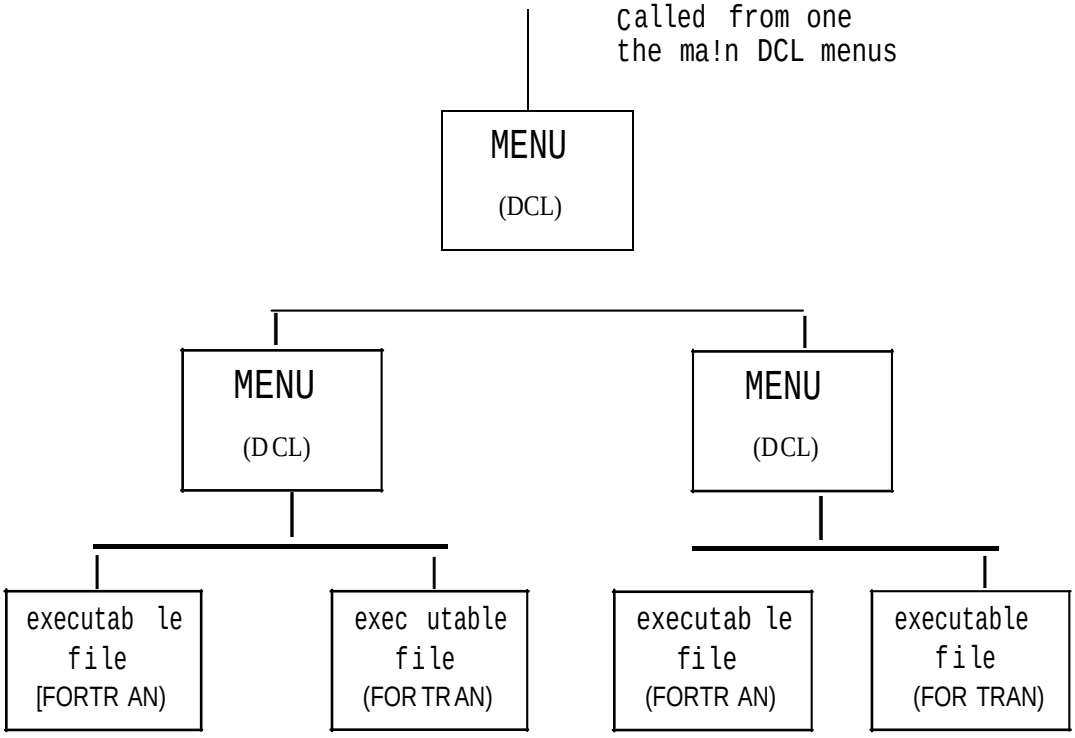


Figure 3.7 Menu tree structure

Note that the bottom of the hierarchy consists of executable files. This structure allows a high degree of freedom for the high-level language programme designer. The requirements and restrictions are explained in detail in chapter 7.

CHAPTER 4. SIMULATION SHELL

4.1. Introduction

This chapter contains a description of the programmes on the top of the simulation hierarchy. More information about the detail of the shell is given in chapters 9 and 10. The first part of the chapter concerns the "Programme Description Language" (POL) used and its application to DCL programming.

4.2. Programme Descriptive Language

The Software engineers responsibility for the maintenance of the shell should apply discipline in maintaining a structured programme. It should also be readable by people not familiar with DCL, which is an interpreted language very open to unstructured programming. A good way to achieve this would be to use a POL.

The PDL described in [14] is a good way of encouraging good design and making a programme more readable. The application of the POL to high level language programme design is well described in the above reference. However, DCL command

procedures contain operating system commands which make the specific definition of the POL not entirely applicable to OCL programme design. In spite of this fact, a POL can be of great use in encouraging good OCL procedure design. OCL is an interpreted language which is very open to unstructured programming. In addition a POL makes the programme far more readable .

It was decided to use a POL based on Walker's description in [14]. Any confusion resulting from the operating system nature of the target language could be explained by including explanatory comments in the POL listing. {In the POL used, comments are enclosed by asterisks.) The following concepts and conventions used in the simulation should enhance the understanding of the POL listings.

4.2.1. Logical names

When reading the POL listings, it is necessary to understand the VMS concept of creating logical names. A logical name may have any string of characters assigned to it. For example, the logical name **LAST** may have the name of the data file **SIGNAL.DAT** assigned to it. LAST remains defined as SIGNAL.DAT until it is reassigned or deassigned.

Logical names play an important part in the simulation and there are two of particular significance:

"LAST" always has the name of the last signal data file created assigned to it. This is a useful feature which enables the user to chain processing blocks together without remembering the names of the data files in which the processed signals are stored. (These data files are referred to as signal data files in this report)

The most important logical name in the simulation is called "MAIN DIR". This has the name of the simulation's main (or root} directory assigned to it. The name of this directory on the development computer is [ADS.MENU]. The logical name is necessary to maintain portability to other VAX/VMS computers. MAIN DIR enables programmes in other directories to access the simulation signal data files which are always stored in the main directory.

4.2.2. Storage of files

The file storage convention used in the simulation is as follows:

POL files have the extension .POL

DCL files have the extension .COM

FORTRAN files have the extension .FOR

Object files have the extension .OBJ

Executable files have the extension .EXE

Data files have the extension .DAT

Filter coefficient files have the extension .FLT

Text files have the extension .MEM

4.3. DESCRIPTION OF SIMULATION SHELL PROGRAMMES

The shell was written by writing a programme in POL form and then inserting the DCL code.

4.3.1. Shell structure

The structure of the shell programmes on the top of the simulation hierarchy was implemented according to the simulation structure block diagram presented in figure 3.6. The interaction of these shell programmes is shown in more detail in figure 4.1.

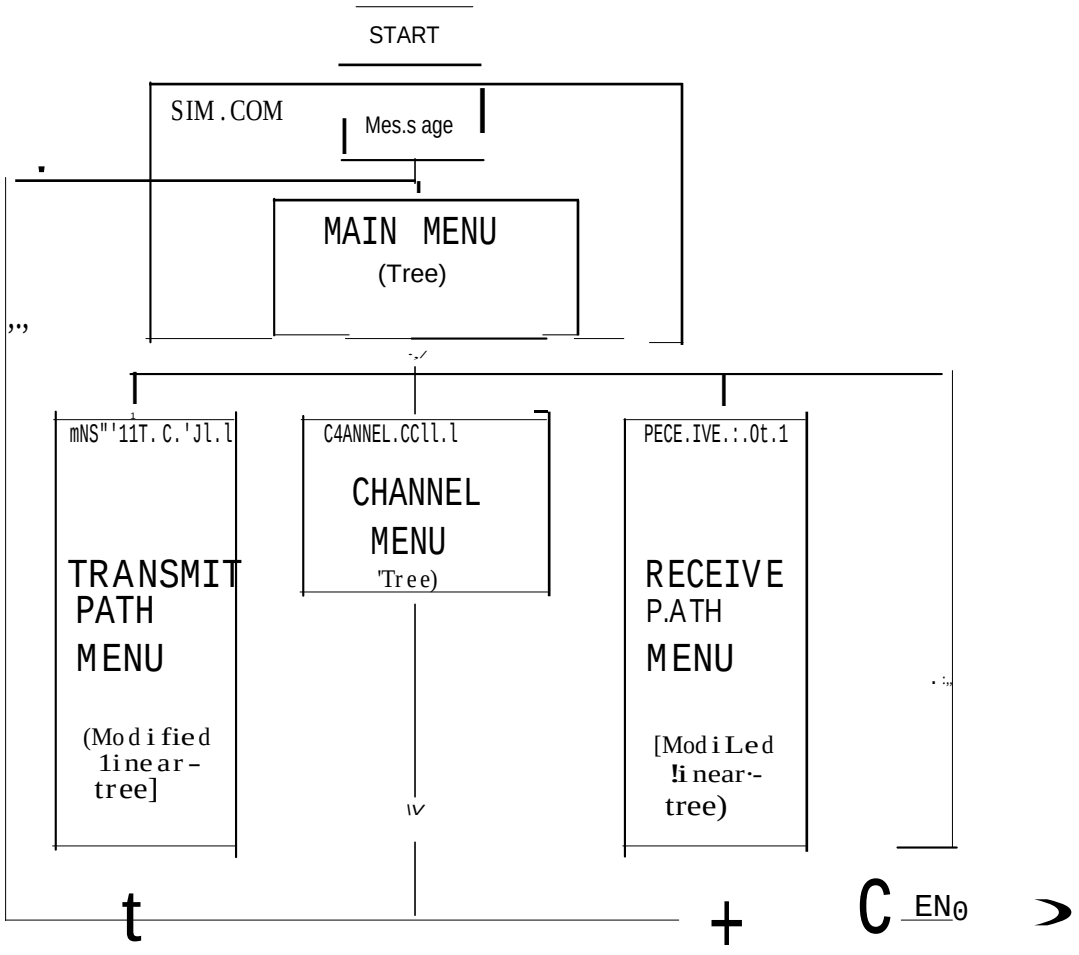


Figure 4.1 Interaction of the main programmes

4.3.2. Main Programme

SIM.COM can be seen as the main programme of the simulation. All other procedures and programmes are on a lower level of the simulation hierarchy. The function of SIM.COM is to display information and news about the simulation by displaying the news file NEWS.HEM. The MAIN MENU of the simulation is then displayed. The user has the choice of proceeding down the transmit path (stored in TRANSMIT.COM), the receive path (stored in RECEIVE.COM), the channel menu (stored in CHANNEL.COM), or terminating the simulation.

When programme control is returned by one of the DCL command procedures called, the MAIN MENU is displayed again.

The PDL of SIM.COM is shown overleaf. The programme with DCL commands included is given in the programme listings.

```
program SIM.COM

*************************************************************1
* This programme is at the top of the simulation hierarchy.    1
* It puts the simulation on either the transmit or receive path.
*************************************************************1

Variables:
   Integer:
       Single:
          Local:
            CHOICE          * input from menu *
   Integer:
       Single:
          Global:
            LAST    * LAST is an assigned value        *
                    * which has the name of the last   *
                    * signal data file assigned to it  *

            MAIN DIR  * MAIN DIR is an assigned value    *
                    * which has the name of the main    *
                    * directory (containing the         *
                    * simulation programs) assigned     *
                    * to it. ([ADS.MENU] on Fuchs VAX)  *

External procedures:
   DCL procedures:
       TRANSMIT.COM         * transmit main menu *
       RECEIVE.COM          * receive  main menu *
       CHANNEL.COM          * channel menu *
   FORTRAN programs:
   Text files:
       NEWS.MEM         * simulation information and news *

Begin:

   MAIN DIR := Name of present directory
                        * [ADS.MENU] on Fuchs VAX *
                        * DCL logical assignment *
   LAST := DUMMY.DAT    * DCL logical assignment *
   Put:
        "          DIGITAL RADIO SIMULATION"
        NEWS.MEM       * Display simulation info and news *
   End put:
   Repeat:
        Put:
        * Ask whether TRANSMIT,RECEIVE  or CHANNEL simulation *
                        *                                    is desired
             "WHICH DO YOU WISH TO DO"
             " "
             "    1   TRANSMIT "
```

```
          " "
          "    2   RECEIVE  "
          " "
          "    3   PUT SIGNAL THROUGH CHANNEL"
          " "
          *"   4   add any additional functions here"*
          " "
          "    9   END SIMULATION"
          " "
     End put:

     Get:
          CHOICE
     End get:


     Case CHOICE of:

          1   Call TRANSMIT.COM    * go to transmit path menu *
          2   Call RECEIVE.COM     * go to receive path menu *
          3   Call CHANNEL.COM     * go to channel menu *

     *    4   add new functions here *

          9
     End case:


 Until (CHOICE=9)

 Put:
      " END OF SIMULATION"
 End put:

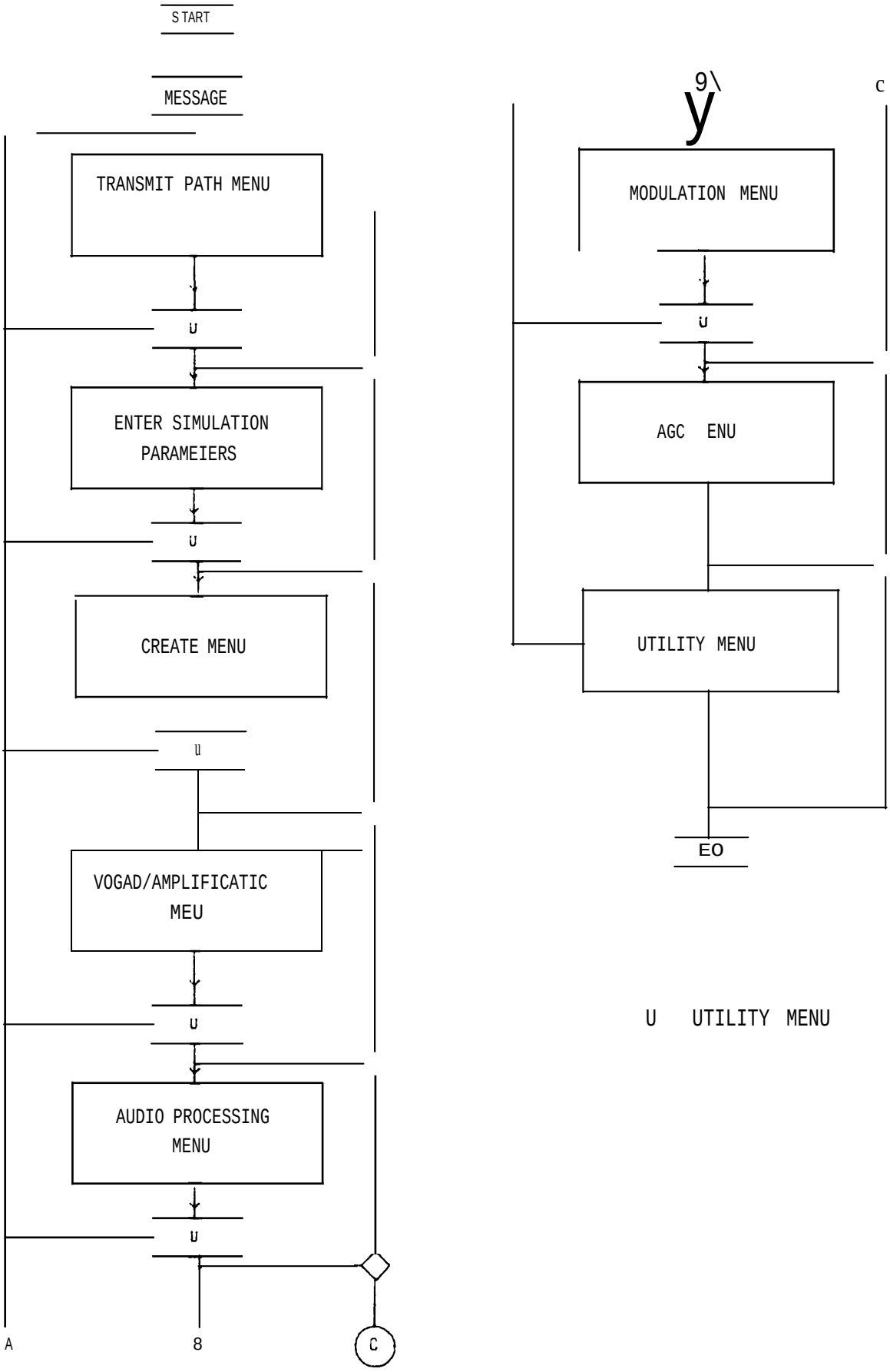
end:
End program:
```

4.3.3. Transmit Path

The TRANSMIT.COM DCL procedure is the main calling procedure for the transmit path. This procedure utilises the modified linear tree-structure explained in section 3.4.4.

The procedure first displays a message explaining its function and follows this with the TRANSMIT PATH MENU. The menu options are ordered in a logical fashion. The user may start at the beginning of the transmit path and continue along it until the programme execution is returned to the calling programme (SIM.COM). It is also possible to enter and continue along the path from any step in the path. In addition, the option of returning to the TRANSMIT PATH MENU exists after every step.

This method provides guidance for the inexperienced user and flexibility for the more experienced programme user/writer. A flowchart of TRANSMIT.COM is shown in figure 4.2. The PDL listing and DCL procedure is may be found in the programme listings .



F:i.gure 4.2 : rransmit path

4.3.4. Receive Path

The Receive Path command procedure (RECEIVE.COM) has an identical structure to the Transmit Path. The major difference between the two paths is the sequence of the steps. The steps are ordered in the logical fashion of operation in a radio. The receive path flowchart is shown in figure 4.3. The POL and DCL listings may be found in the programme listings.

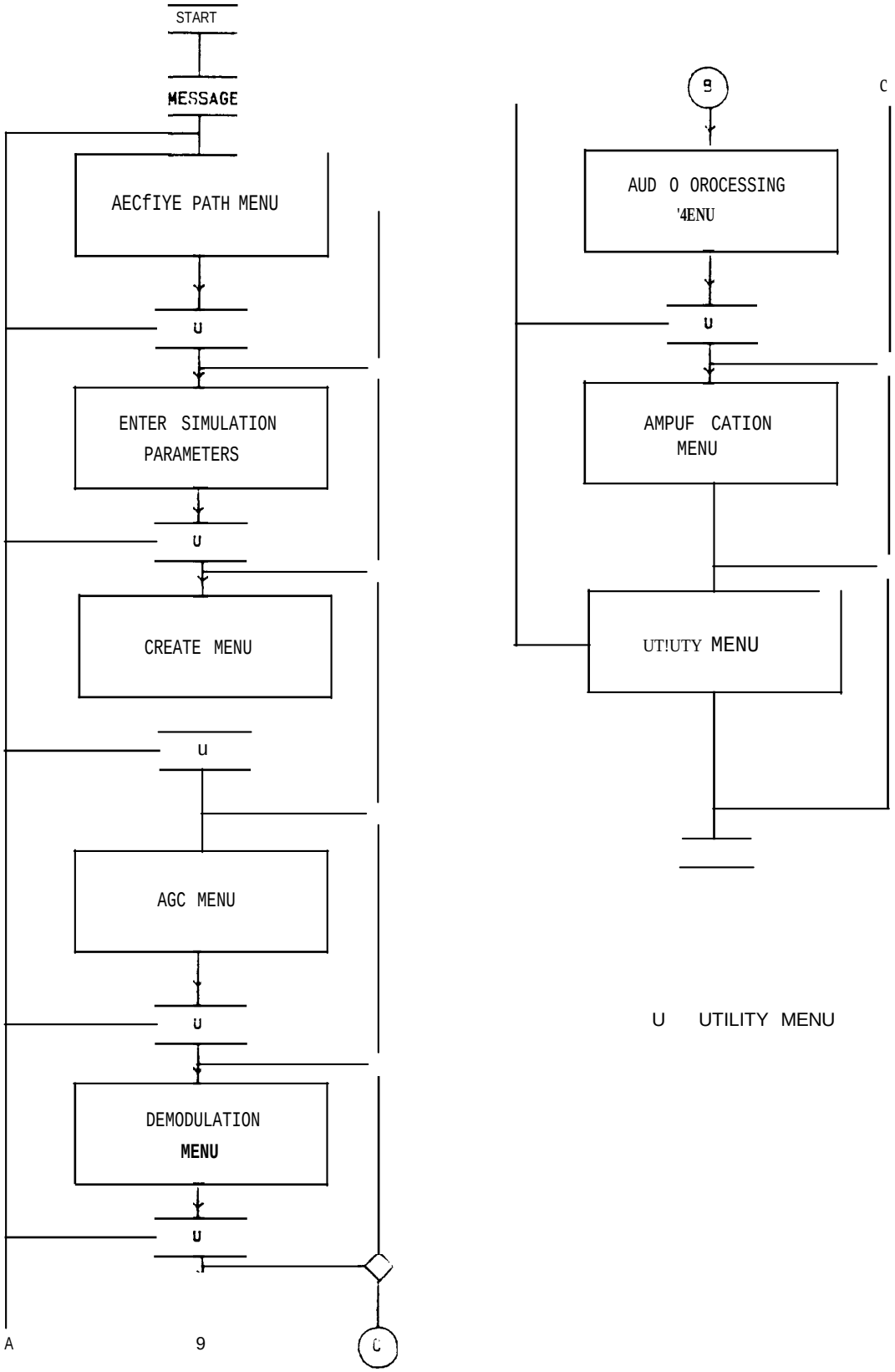


Figure 4.3 : qeceive path

4.3.5. The Channel Menu

The command procedure CHANNEL.COM contains the CHANNEL MENU. Its function is to offer the user a choice of different channels through which he may put his signal. These channels include simply adding gaussian noise and an HF channel simulation [15].

Unlike the transmit and receive paths, it is not desirable for the channel menu to use the modified linear-tree structure in its implementation. It is more logical to use the tree structure in this case, since the channels would probably not be executed in serial steps.

The channel menu presents the user with a choice of channels, the opportunity to call the UTILITY MENU, or the option of leaving from the channel menu. If the last option is not chosen, the channel menu is redisplayed on return from any of the called procedures.

An interesting feature of the CHANNEL.COM is that the user may choose to run his own channel programme which might not yet have been included in the simulation.

By looking at the PDL listing, it can be seen that the programme writers have a clear indication of where and how

to include additional channels into the procedure and thus the simulation. This helps in fulfilling the simulation's requirement of easy expansion.

A flowchart of CHANNEL.COM is shown in figure 4.4.

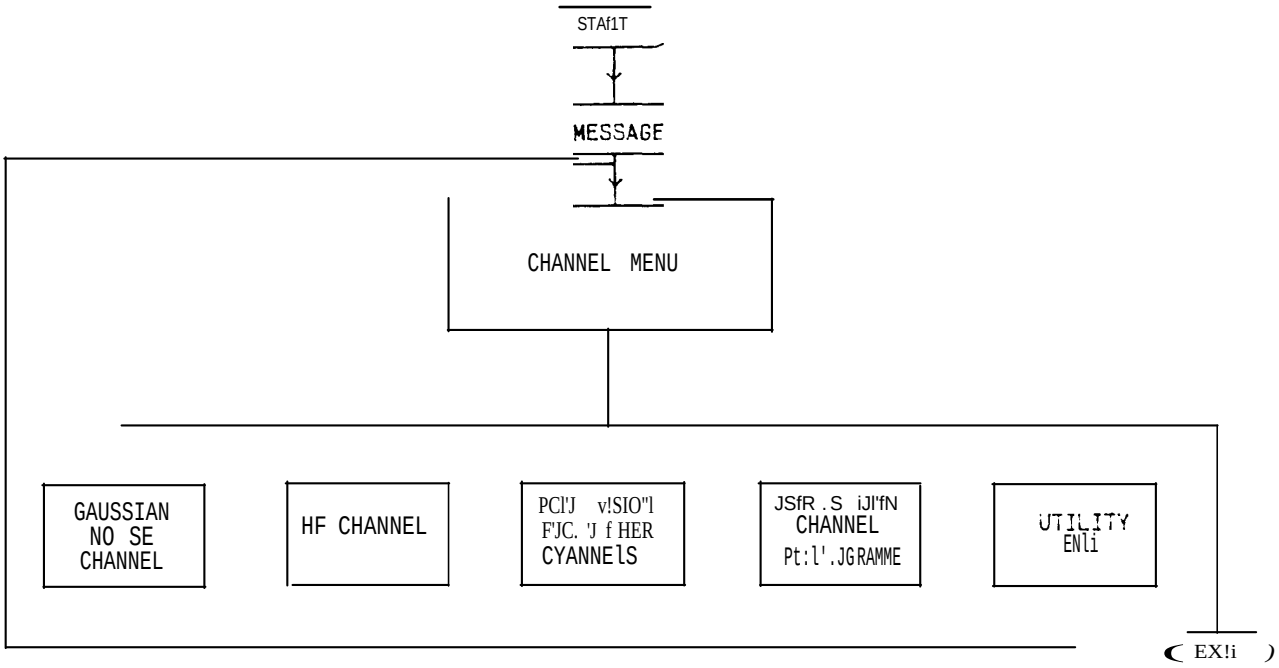


Figure 4.4 Channel menu structure

4.3.6. Utility Menu

The UTILITY.COM DCL procedure is an integral part of TRANSMIT and RECEIVE paths. One of its functions is to ask the user whether he wishes to continue along the original

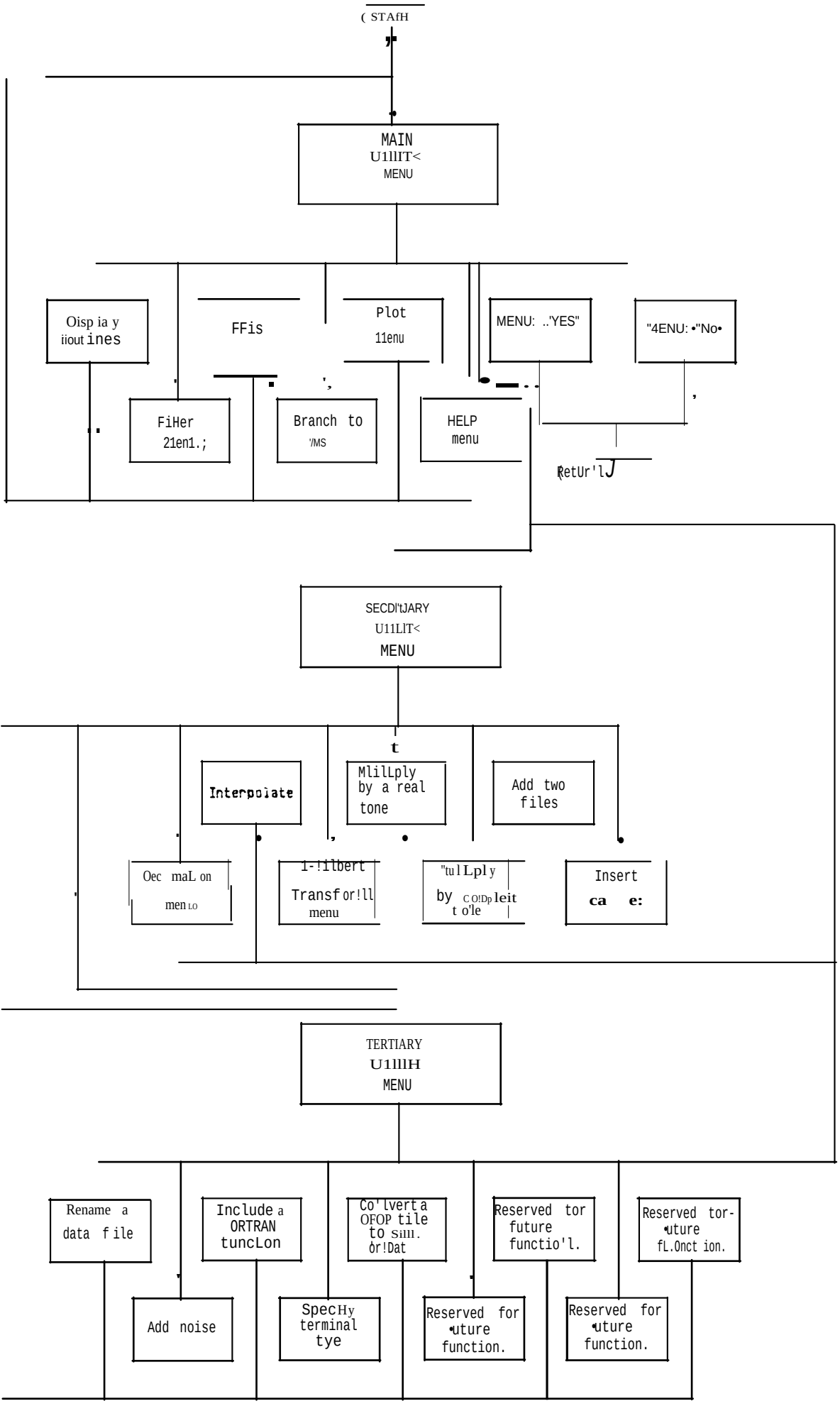
path. The alternative is to return to the start of the path where all the menu options are displayed. This information is passed to the calling procedure by altering the value of the global variable "MENU".

Other facilities accessible from UTILITY.COM include:

display routines
FFTs
hard plot routines
filters,
the facility to branch in and out of the simulation.
Noise addition
decimation
interpolation
Hilbert transformation
multiplication by a real tone
multiplication by a complex tone
signal addition
carrier insertion
renaming data files
HELP facility.
etc.

These functions were implemented in a tree structure. The functions called also use tree-structured menus. More detail on the function implementations are given in a later chapter.

The number of utility functions implemented prohibit the display of all of them on the console simultaneously. To overcome this problem, submenus of the UTILITY MENU were created and included in the tree structure as shown in figure 4.5. The submenus are contained in UTILITY2.COM and UTILITY3.COM.



i'igure 4.5 Utility menu structure

CHAPTER 5. SIMULATION OF DIFFERENT CHIPS

One of the most important aspects of the simulation is its ability to simulate the effects of different DSP micro-processors. This is complicated by the fact that not only do DSPs have differing wordlengths, but they can have either fixed or floating point configurations (or both). This chapter describes how this problem is tackled in the simulation.

5.1. Division into Subdirectories

The simulation environment consists of a main directory and four subdirectories (shown in figure 5.1).

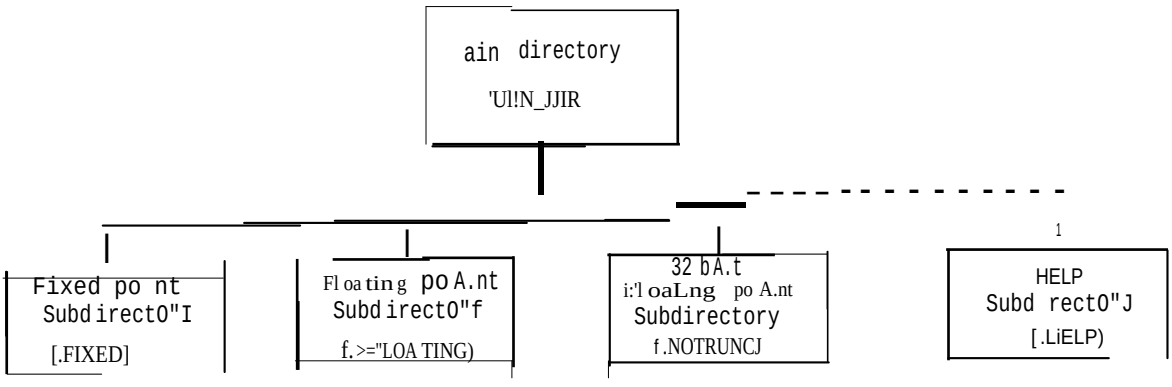


Figure 5.1 Simulation Directory Structure

The simulation's main directory stores all the programmes that are normally needed during the use of the simulation. These include DCL files, executable files, as well as data files. In addition, almost all the FORTRAN and POL listings and object files are kept in this directory. On the development system, the name of this directory was [ADB.MENU], but the simulation maintains its portability by assigning the name of the main directory to the logical name MAIN DIR.

The subdirectories [.FIXED] and [.FLOATING] contain the executable and data files which are required for the simulation of the fixed and floating point chips respectively. The subdirectory [.NOTRUNC] contains the files that are necessary for the simulation of the special case of a 32 bit floating point configuration.

[.HELP] is the subdirectory where files used for the HELP facility are stored.

The reasons for structuring the environment in the above way are given in the following sections.

5.2. DSP Microprocessors

Currently available DSP microprocessors can be divided into two broad classes:-

1. Fixed point configuration chips
2. Floating point configuration chips

Examples of fixed point chips are:-

1. TI TMS32010 and its derivatives
2. Analog Devices ADSP-2100
3. Philips PCB 5010,5011
4. Zoran VSP
5. and many others

which all have a basic 16 bit wordlength.

Another fixed point chip is the Motorola DSP 56000/1 which has a 24 bit basic wordlength (and thus a higher dynamic range).

Examples of floating point chips are:-

1. AT&T WEDSP32
2. NEC PD77230
3. TI TMS320C30 (still under development)

which have 32 floating point configurations consisting of 24 bit mantissas and eight bit exponents.

The OKI MSM6992 is a 22 bit floating point chip with a 16 bit mantissa and six bit exponent. (This chip (amongst

others) was seriously considered for DSP radio applications because of its low power consumption.)

One of the aims of the simulation was to allow the DSP radio designer to easily simulate the effects of any of the configurations and wordlengths mentioned above (or any other configuration they might care to define. The simulation is done by truncating the results of arithmetic operations to the required precision during the DSP programmes.

5.3. The Truncation Process

From the programme writers point of view, an important aspect of the simulation is that after every arithmetic operation, the subroutine TRUNCATE must be called. The parameters passed to the subroutine are:-

1. the result of the preceding operation
2. data which was read in from the file TRUNCD ATA.DAT

TRUNCDATA.DAT is a data file which contains the parameters needed by the truncation subroutine . The function of the truncation subroutine {TRUNCATE), is to truncate the value of the input real number to a value that is a possible

representation of the word length of the chip that is being simulated.

The method allows the same programme to be used for simulating any word length or configuration. The only things that alter are the data contained in TRUNCDATA.DAT and the TRUNCATE subroutine to which the programme object file is linked. Three versions of the TRUNCATE subroutine exist in the simulation. One is stored in each of the three simulation subdirectories.

The file TRUNCINT.FOR contains a TRUNCATE subroutine that does fixed point truncation. This FORTRAN file and its object file are stored in the subdirectory [.FIXED].

The file TRUNCATE.FOR contains a TRUNCATE subroutine that does floating point truncation. This FORTRAN file and its object file are stored in the subdirectory [.FLOATING].

The file TRUNCDUM.FOR contains a TRUNCATE subroutine that is a dummy routine. This FORTRAN file and its object file are stored in the subdirectory [.NOTRUNC]. The dummy routine implies that the input is returned unaltered. This is a way for simulating the effects of 32 bit floating point chips because this is the wordlength of a single precision word on the VAX.

The actual mechanics of the truncation process are explained in Appendix A.

5.4. Linking versus Copying of .EXE files

It has been shown how the same FORTRAN programmes can be made to simulate fixed point, floating point and 32 bit floating point DSP chipsby linking to the appropriate TRUNCATE subroutine. The resulting executable files (with a •EXE extension) then simulate the effects of the required chip.

A possible way to take advantage of this could have been to link all the programmes requiring truncation after the user had entered whether he wished fixed or floating point simulation. However, linking a large number of files can take an appreciable time and there would have been a long time delay while the programmes were linked.

Another method considered was to link all the programmes and permanently store all the resulting executable files in the different subdirectories.

i.e. All the executable files that were linked to TRUNCINT.OBJ would be stored in subdirectory

[.FIXED]. These files would all simulate fixed point configurations .

All the executable files that were linked to TRUNCATE.OBJ would be stored in subdirectory [.FLOATING]. These files would all simulate floating point configurations .

All the executable files that were linked to TRUNCDUM.OBJ would be stored in subdirectory [.NOTRUNC]. These files would all simulate 32 bit floating point configurations .

After the user inputs which type of chip he wishes to simulate, the executable files from the relevant directory would have to be copied to the normally active main directory (MAIN_DIR).

A disadvantage of this method was that more storage space is required to store all the .EXE files. (Storage space was not a problem on the development computer.)

Another disadvantage might have been the added complexity when including a new DSP programme into the simulation. The compiled programme would have had to be linked at least three times and the .EXE files would have had to be placed

to the correct subdirectory. This problem was overcome by writing a DCL command procedure (INCLUDE.COM) to do all the linking and placing by simply running the procedure, or by calling it from the UTILITY menu.

An advantage of this method compared to the linking method was that it takes far less time to COPY the files than to LINK them.

This method had the advantage that all the .EXE files were always available. This fact made it possible to allow the user the option of simulating for another configuration instantaneously (i.e. without copying files from one directory to another). This could be done by simply changing the default directory from the main directory (MAIN_DIR) to one of the subdirectories (where the relevant files reside).

The latter method seemed to be the better one of the two and it was the method implemented.

5.5. Important Data Files

The data and text files in the simulation can be divided into three classes:

1. Signal data files which contain sampled signal data and are stored in the unformatted way to reduce storage space needed and read/write times.

2. Auxiliary data files such as the filter coefficient data files which are stored in the "list-directed" format

3. Control data and text files which are used in the control of the simulation and are stored in the "list-directed" format to enable easy inspection of their contents. These files are small and do not take up much storage space or time.

The control data and text files are:

SIMPARAM.DAT

TRUNCDATA.DAT

CHIPEXPL.MEM

CHIPCONF.MEM

NEWS.MEM

An explanation of the contents and function of these files follows.

5.5.1. Simulation Parameters

SIMPARAM.DAT contains the simulation parameters.

i.e Sampling frequency

Length of signal data files (Sample length)

5.5.2. Truncation data

TRUNCDATA contains the information necessary for the truncation subroutine TRUNCATE to do its task. The data contained in this file is:

For the fixed point version:

The maximum positive value possible

The maximum negative value possible

The minimum value of an ordinary register (LSB)

The minimum value of accumulator (accumulator LSB)

For the floating point version:

The maximum positive value possible

The maximum negative value possible

The minimum value of a floating point word.

The range of the mantissa.

The use of this data is explained in Appendix A.

5.5.3. Chip configuration

CHIPCONF.MEM is a text file which contains the words

FIXED

FLOATING or

NOTRUNC

This file give the DSP programmes the means of determining the chip configuration active in the simulation.

5.5.4. Chip explanation

CHIPEXPL.MEM is a text file which contains an explanation of the chips parameters (i.e configuration and wordlengths). It also gives the name of a chip with these parameters (if applicable). This file is used to give the user information about the chip that is currently being simulated.

5.5.5. News and Information

NEWS.MEM is a text file containing general information and news about the simulation. It is displayed when the simulation is initiated and can be updated at any time.

The creation and alteration of these files are explained in the next chapter.

CHAPTER 6. INITIALISATION OF THE SIMULATION

This section describes how the simulation is initialised when the user choose the option:

"ENTER SIMULATION PARAMETERS"

This is always the first option in the transmit and receive paths.

6.1. Simulation Sampling Frequency and Sample Length

The first action of the path is to run the FORTRAN executable file SIMPARAM. The purpose of this programme is to define the sampling frequency and the number of samples in the signal data files.

The first action of SIMPARAM.FOR is to read the current active values of the sampling frequency and sample length by reading the data file SIMPARAM.DAT. A message is then displayed on the console explaining what the parameters are. The user is then asks if he wishes to alter these values. On a positive reply, the programme prompts for the new values and stores them in a new version of SIMPARAM.DAT. These parameters remain to be used by the simulation programmes

until changed by the above procedure, or by a decimation or interpolation process.

The POL and FORTRAN listing of SIMPARAM.FOR are included in the programme listings.

6.2. Type of Chip Chosen

After SIMPARAM has finished its execution, the procedure CHIPTYPE.COM is called.

This DCL procedure informs the user of the name and characteristics of the DSP chip which is presently active in the simulation. It does this by displaying the contents of the file CHIPEXPL.MEM on the screen. The user is asked whether he wishes to simulate for another chip. On a positive reply, it asks the user which chip configuration he wishes to use in the simulation. He may choose one of the chips commercially available from the menu displayed, or input his own configuration.

If he chooses one of the configurations from the menu, the procedure copies the applicable truncation data to the TRUNCDATA.DAT files in the main directory and the applicable subdirectory.

Text explaining the name and characteristics of the chip is then written to the text file **CHIPEXPL.MEM** in both the above mentioned directories.

The text file **CHIPCONF.MEM** is then copied from the applicable subdirectory to the main directory. This file contains the name of the subdirectory from which it was copied (i.e. either FIXED, FLOATING or NOTRUNC).

Finally, all the executable (.EXE) files are copied from the subdirectory to the main directory. This means that the simulation simulates for the desired chip unless specifically commanded by the user to simulate for another configuration. This is accomplished by temporarily changing the default from the main directory **(MAIN_DIR)** to one of the subdirectories .

If the user wishes to define a non-standard configuration, most of the above functions are performed by executing the FORTRAN programme **TRUNCDATA.** This programme asks the user for the configuration and wordlength he wishes to simulate. It then calculates the truncation parameters and stores them in the correct **TRUNCDATA.DAT** data files. (An explanation of how **TRUNCDATA.FOR** calculates the parameters is contained in Appendix A.) **TRUNCDATA.FOR** then writes the required text to

the CHIPCONF.MEM and CHIPEXPL.HEM text files. The executable files are then copied from the applicable subdirectory to the main directory by CHIPTYPE.COM.

The result of this processing by CHIPTYPE.COM is that the simulation's main (root) directory (MAIN_DIR) contains all the executable and data files required for simulating the specified chip. In addition, each of the three subdirectories have all the executable and data files that are required for performing the truncation appropriate to their most recently defined chip. This makes it easy for the simulation to offer the user the option of temporarily simulating another configuration. All the simulation has to do is to change the default directory from the main directory to one of the subdirectories.

This feature of the simulation is an extremely useful one for DSP designers who want to quickly compare the effects of different chip configurations on one of their algorithms.

CHAPTER 7. HIGH LEVEL LANGUAGE PROGRAMMING RULES AND GUIDELINES

The simulation structure offers a very flexible working environment for the programme designers, but, obviously, some rules and guidelines must be given to enable the simulation to work cohesively.

Although one of the advantages mentioned about the simulation structure was that a combination of programming languages could be used, only FORTRAN was available on the computer on which the simulation was developed. Thus the rules and guidelines were developed to optimise the use of the only language available. It is not envisaged that it will be very difficult to adapt the rules if another language such as Pascal or C becomes available. A good example of how I/O anomalies between different languages can be overcome is shown in [8].

The most important rules of the simulation apply to the data files:

1. All signal data files (i.e. data files in which the sampled signals being processed are stored} must be "unformatted". Unformatted files take up less storage

space and the data is accessed quicker than in the "list-directed" format. Unformatted files are not man-readable. In the unlikely event of the user wishing to read the data samples, he may use the FORMAT utility function to create a list-directed copy of the unformatted file. (List-directed files are man-readable.)

2. All other data files must have a "list-directed" format. This allows easy inspection of the files which are generally not long and do not require much storage space or read/write times.

3. All the data files must be stored and read from (and stored in) the main directory (even when the default directory is set to one of the subdirectories). The exceptions to this rule are some of those files which are used in the control of the simulation.

   These are:

   TRUNCDATA.DAT

   CHIPCONF.MEM and

   CHIPEXPL.MEM.

4. A message telling the user the name of the output file(s) must be displayed.

Other guidelines are:

5. The programme should only prompt the user for the name of the input signal data file and any other information it requires from him that it cannot obtain from the simulation. (eg. the sampling frequency should be obtained from SIMPARAM.DAT and not be asked for.)

6. The data needed for truncation should be read from TRUNCDATA.DAT. This data should then be passed as parameters to the TRUNCATE subroutine in the same order.

7. The TRUNCATE subroutine is called as follows:

```
CALL TRUNCATE (number,max_pos_val,max_neg_val,min_val,mantis sa_rangE
```

,where "number" is the rea1 number to be truncated and the last four parameters were read in from TRUNCDATA.DAT in that order.

In the fixed point TRUNCATE subroutine (stored in TRUNCINT.FOR), the last parameter is not used. In this case the last value read in from TRUNCDATA.DAT is the value of the accumulator LSB. This value is inserted as the second last parameter in the calling statement when

it is required to simulate the truncation effects of the accumulator.

To enable the programme to simulate the effects of the chip as closely as possible, the result of each arithmetic operation should be truncated.

8. The name of the FORTRAN programme should indicate its action. It should be preferably stored in a file with the name of the programme and a **.FOR** extension.

9. The programme's output signal data file should preferably be the same as the name of the programme with a **•DAT** extension.

10. Any special inputs required should be clearly explained in the comments. (eg. the format required for the filter and Hilbert transform coefficient data files.)

11. Comments about new programmes should be included in the "news" file NEWS.MEM and/or the HELP menu.

12. The programmes should be written in a structured way and be well documented. A good way to achieve this is to write a POL first and then insert the FORTRAN

code. An example of a POL which was converted to a FORTRAN programme is shown overleaf. This is the programme which handles IIR filtering in the simulation.

The last point on the POL method of design has particular significance when it is noted that some modern day DSP chip manufacturers are designing their chips with the aim of writing a high level language compiler for the processor. The use of POLs will make the task of converting the FORTRAN programmes to the high level OSP programme language much easier. Alternatively, it might be possible to write a POL- to-DSP machine code compiler to translate directly from the POL itself.

```
Program: IIRFILTER.FOR
*************************************************
* FORTRAN program to implement an IIR Filter    *
*********************************** *************

Input files:
            FILTERIN.DAT          * File containing input signal data
            FCOEFFS               * Char variable containing name
                                  * of file containg tap coefficients
            TRUNCDATA.DAT         * File containing truncation data
            CHIPCONF.MEM          * File containing chip configuratior
                                  * either 'FIXED   ' or 'FLOATING'

output files:
            FILTER.DAT            * File containing output signal date

Variables:
       Char:
          Single:
              Global:
                     MAIN DIR     * logical name for the simulation*
                                  * directory normally in use      *
                                  * ( (ADB.MENU] on the Fuchs VAX   *

          Single:
            Local:
                  FCOEFFS     * Char variable containing name     *
                              * of file containg tap coefficients *
                  CONFIGURATION * 'FIXED   ,or 'FLOATING' point *

       Integer:
            Single:
                   Local:
                     ORDER          * IIR filter order *
                     FILE LENGTH    * Number of data points processed
                     q              * While loop variable *
       Real:
            Single:
                   Local:
                     GAIN FACTOR    * reciprocal of passband *
                                    * attenuation            *
                     Input          * Filter input           *
                     Output         * Filter output          *
                     maxpos         * parameters needed for  *
                     maxneg         * truncation subroutine  *
                     min            *           "            *
                     mant range     *           "            *
                     ace lsb        *           "            *
                     ace min        *           "            *
                     sh             * scaling factor used in *
                                    * fixed point truncation *
            Array:
```

```
                  Local:
                    y = of size (0..100,0..2)  * nodes *
                    u = of size (100,0..2)     * nodes *
                    t = of size (100,0..2)     * nodes *
                    B = of size (100,0..2)     * forward coefficier
                    C = of size (100,0..2)     * reverse coefficier
     * See diagram at the end of this listing for a clearer descrip1
External procedures:
             LOG2     * calculates a scale factor used in fixed   *
                      * point truncation                          *
             TRUNCATE   * Truncation subroutine to simulate     *
                        * for limited fixed and floating point *
                        * chip configurations                   *
  * This program must be linked to the relevant file in the
  * subdirectories. i.e. [.FLOATING]TRUNCATE.OBJ
  *                      [.FIXED]TRUNCINT.OBJ
  *                      [.NOTRUNC]TRUNCDUM.OBJ

Begin:
    Get:
      TRUNCDATA.DAT : maxpos,maxneg,min,mant_range
    End get:
    ace lsb:=mant range    * The value for mantissa range (in *
       -* floating point trucation) and accumulator lsb value *
        * (in fixed point truncation) are stored in the same  *
        * location in file TRUNCDATA.DAT which contains the   *
        * parameters needed to call the TRUNCATE subroutine   *
        * See the notes on truncation for this simulation for *
        * more details                                        *

    Get:
     CHIPCONF.MEM: CONFIGURATION
    Endget:
    If (CONFIG='FLOATING')
       then:
           ace min:=min
       else:
           ace min:=acc lsb
    End if:
    * For handling fixed point chips where accumulator wordlength *
    * exceeds some other register wordlengths                     *

    Get:
       FCOEFFS             * Char variable containing name     *
                           * of file containg tap coefficients *
                          * eg. MAIN DIR:IIRCOEFS.DAT          *
    End get:

    Get:
      "FCOEFFS":ORDER
      "FCOEFFS":GAIN FACTOR
```

```
End get:
 ORDER=ORDER/2
CALL LOG2(GAIN FACTOR,SH)   * calculate a scale factor for TRUNCl
CALL TRUNCATE(GAIN_FACTOR,maxpos*sh,maxneg*sh,min*sh,mant range
   * Represent gain factor in limited wordlength form *  -

k:=l
While (q<=ORDER)    * Read in filter coefficients *
 do:
    Get:
       "FCOEFFS" : B(q,2),B(q,l),B(q,O),C(q,l),C(q,O)
    End get:

    * Truncate filter coefficients *

End while:

While (still input data samples remaining to be read)
 do:
 Get:
   MAIN DIR:FILTERIN.DAT:input
      -          * read new sample into first delay element *
 End get:
    * It is assumed that                                    the
      input is of limited wordlength                         *
   * If this is not so, insert a statement calling subroutine *
     * TRUNCATE here                                        *

 If (there are more samples to be read from FILTERIN.DAT)
   then:
      file_length := file_length+l

      y(0,2):=input•gain factor
      Call Truncate(y(0,2),maxpos,maxneg,acc_min,mant_range)
      q:=l
      While (q<=ORDER)
        do:                              * IIR process *
        u(q,0):=-1.0*C(q,O)*t(q,O)
           Call Truncate(u(q,O),maxpos,maxneg,acc min,mant rangE
              * High precision truncation (eg. 31-bit in TMS320J
        u(q,l):=-l.O*C(q,l)*t(q,l)
           Call Truncate(u(q,1),maxpos,maxneg,acc min,mant rangE
              * High precision truncation (eg. 31-bit in TMS320J
        u(q,l):=u{q,O)+u(q,1)
           Call Truncate(u(q,l),maxpos,maxneg,acc min,mant rangE
              * High precision truncation (eg. 31-bit in TMS320J
        u(q,2):=u(q,l)+y(q-1,2)
        t(q,2):=u{q,2)+min/2.0  •turns truncation into rounding
           Call Truncate(t(q,2),maxpos,maxneg,min,mant_range)
                * Low precision truncation (eg. 16 bit in TMS320J

        Y(q,O):=B(q,O)*t(q,O)
```

```
Call  Truncate(y(q,0),maxpos,maxneg,acc_min,mant_rangE
```

```
            * High precision truncation {eg. 31 bit in TMS320J
      y{q,l) :=B (q,l)*t{q,l)
         Call Truncate{y{q,1),maxpos,maxneg,acc min,mant rangE
            * High precision truncation {eg. 31-bit in TMS320
      y{q,l) :=y{q,O)+y{q,l)
         Call Truncate{y{q,1),maxpos,maxneg,acc min,mant rangE
      y{q,2) :=B{q,2)*t{q,2)                    -        -
         Call Truncate{y{q,2),maxpos,maxneg,acc min,mant rangE
            * High precision truncation {eg. 31-bit in TMS320J
      y{q,2) :=y{q,2)+y{q,1)
         Call Truncate{y(q,2),maxpos,maxneg,acc min,mant rangE
            * High precision truncation {eg. 31-bit in TMS320:

         t(q,O) :=t{q,1)          * z**{-1.0) process *
         t(q,l) :=t{q,2)

         q :=q+l
      End while:                 * end of  IR process *

      OUTPUT :=y{ORDER,2)+min/2.0
       Call Truncate{OUTPUT,maxpos,maxneg,min,mant range)
         * Low precision truncation {eg. 16 bit in-TMS32010) *

      Put:                      * output result *
          FILTER.DAT: OUTPUT
      End put:
      While (q<=ORDER)
         do:
           t{q,O) :=t{q,l)          * z**(-1) process *
           t{q,1):=t (q,2)

           q :=q+l
      End while:
 End If:                          * end of FILTERIN.DAT *
End while:
```

```
    Put:
     " "
     file_length
     "data samples were written to file FILTER.DAT"
     " "
     " "
    End put:

End:
End program:


*****************************************************************;
*
*
*  Diagram of the filter implemented
*
*
*   y(q-1,2)    u(q,2)          t(q,2)          y(q,2)
*        0---->---0-------->------0------>------0---->-----0
*                 |                |    B(q,2)    |
*                 |                |              |
*                                  V Z**-1
*                                  |
*           u(q,1) |               |              | y(q,1)
                    --------<-------  ------->------

*                 0               0              0
*                 |    -C(q,l)    |    B(q,l)    |
*                 |               |              |
*                                  V Z**-1
*                                  |
*                 |               |              |
*                 0--------<------0------>-------0
*           u(q,0)      -C(q,0)          B(q,0)     y(q,0)
*
*
*    The above shows a second-order subsystem of the cascade struct\
*    implemented.
*
*    The structure is identical to DOREDI structure 13 (in PROGRAMS
*        FOR DIGITAL SIGNAL PROCESSING, DSP Committee, IEEE,
*        IEEE Press, 1979)
*    Also see Oppenheim & Schafer: DIGITAL SIGNAL PROCESSING
*        page 152 Fig 4.16
*****************************************************************;
```

CHAPTER 8. IMPLEMENTING FUNCTIONS IN THE SIMULATION

8.1. Introduction

The DSP functions in the simulation are performed by the FORTRAN programmes. The user accesses these programmes by proceeding down the simulation menu structure until the specific function he requires is reached. All the menus were programmed in DCL. An example of a menu programme is SIM.COM. The POL listing of this programme was shown in chapter 3. Many more examples are contained in the programme listings.

This chapter describes how a function is included in the simulation after the menu structure leading to the function has been defined. The first section describes the DCL functional procedure on the bottom of the simulation structure hierarchy which calls the FORTRAN programme which does the actual signal processing. This is followed by a description of how the FORTRAN programme itself is included into the simulation environment.

s.2. The DCL Functional Procedure

s.2.1. Straightforward case

A PDL listing of a shell DCL functional procedure is given at the end of this chapter. The DCL coded version is included in the programme listings.

The PDL lists the input files that it uses as well as the normal declarations. Its first action is to display the name and attributes of the chip currently active and ask the user whether he wishes to use this chip for that particular function. On a negative reply, the DCL procedure CHANGDIR.COM (change directory) is called. CHANGDIR.COM displays the chip options available to the user and changes the default directory to the right subdirectory (i.e. the subdirectory which contains the version of the executable file that simulates the required chip).

Control is then returned to the calling DCL functional procedure. This procedure runs the executable file in the default (current) directory. It then assigns the name of the FORTRAN programme's output signal data file to the logical name LAST. (This negates the need for the user to remember the names of the data files when running a sequential process.)

s.2.2. More complex case

some of the simulation functions require inputs from the calling DCL procedure as well as from the normal sources (i.e. the user and the control data and text files). For example, the filter and Hilbert transform programmes need the name of the coefficient data file to be used.

The functional DCL procedure used in this case is identical to the one described above apart from the following changes:

1. It asks the user for the name of the signal data file to be processed. This data file is then copied to a specific data file. For the FILTER function the name of this file is **FILTERIN.DAT** and for the HILBERT transformer it is **HILBERTIN.DAT.** (Obviously, in this case, the FORTRAN programme does not prompt the user for any information but reads it from one of the above data files).

2. Instead of running the FORTRAN executable file, the DCL procedure calls a DCL menu procedure which may call another DCL menu in a tree structure. The DCL menu on the bottom of the hierarchy gives the user a choice of filters (or Hilbert transforms), stating the

characteristics of each one. Depending on the choice, the procedure runs either the IIR or the FIR structure included in the simulation. The filter characteristics are achieved by giving the name of the appropriate filter coefficient data file.

An example of this type of menu procedure is HILBMENU.COM. A POL listing is included at the end of this chapter. Note that the programmers are clearly shown how to include more options in the procedure. The procedure also gives the user the opportunity to specify his own Hilbert transform coefficient data file and/or run his own transform not yet included in the simulation.

8.3. Inclusion of the FORTRAN Programmes

The following is the procedure that should be undertaken when including a FORTRAN programme into the simulation environment:

The FORTRAN programme should be written according to the guidelines and rules given in the last chapter.

When it has been debugged and tested, the statement

calling the TRUNCATE subroutine should be inserted after every arithmetic operation as shown in that chapter.

The programme should then be compiled to create an object file.

The user must the run the "INCLUDE A FORTRAN FUNCTION" from the utility menu. This option runs the DCL command procedure INCLUDE.COM which links the correct TRUNCATE subroutine to the object file an stores the resulting executable file in the appropriate subdirectory. This is done three times, once for the fixed point configuration, once for the floating point configuration and once for the "no-truncation" (32 bit floating point) case.

After completion of the above procedure the FORTRAN function would have been properly included in the correct directories.

All that would remain to include this as a function in the simulation would be to write the DCL menu structure to run the programme as indicted in the previous subsection.

8.4. POL Listings for this Chapter

8.4.1. FUNCTION.COM

```
*********************************************************
* This is an example of what a DCL programme            *
* in the simulation should look like. To implement      *
* your function into the simulation, replace FUNCTION by *
* the name of your programme.                           *
*********************************************************

Procedure name: FUNCTION.COM
*********************************************************
* DCL Procedure to handle this function.                *
* Include general comments about this procedure here.   *
*********************************************************

Input files:
           CHIPEXPL.MEM        * description of default chip *

Variables:
        Char:
              Global:
                     LAST      * logical name for the last     *
                               * signal data file created      *
                     MAIN DIR  * logical name for the simulation*
                               * directory normally in use     *
                               * ( [ADS.MENU] on the Fuchs VAX  *
              Local:
                    YN         * "Y" or "N"           *
                    CONTINUE

External procedures:
                  CHANGDIR.COM      * Changes default directory  *
                  FUNCTION.FOR       * FUNCTION •s executable progrc
Begin:
      Put:
       form feed
       "DO YOU WANT THE DEFAULT CHIP TO BE USED? [Y/N]"
       "i.e"
       CHIPEXPL.MEM     * TYPEs the file CHIPEXPL.MEM which contain
                          * a description of default chip *
       " "
      End put:

      Put:
          "Y or N:"
      End put:
      Get:
       YN
```

```
      End get:

      If (YN = "N")
      then:
           Call CHANGOIR.COM    * Changes default directory  *
      End if:

      Run FUNCTION.FOR              * FUNCTION•s executable program

     * Make MAIN_DIR the default directory again *
       * MAIN_DIR is the logical name for the normal simulation *
       * directory ( [ADS.MENU] on the Fuchs VAX )              *
      Default_directory:= MAIN_DIR

      LAST := "FUNCTION.DAT"              *logical name assignment*

 * The name and characteristics of the output data file should be *
 * given by the FORTRAN programme but information or instructions *
 * be given here if required.                                     *

      Put:
       "PRESS RETURN TO CONTINUE:                "
      End put:
      Get:
       CONTINUE
      End get:

End:            * Return to calling procedure *
End procedure:
```

8.4.2. POL OF HILBERT.COM

```
Procedure name: HILBERT.COM
*********************************************************
* Top of HILBERT TRANSFORM menu hierarchy                *
*********************************************************

Input files:
           CHIPEXPL.MEM        * description of default chip *

Variables:
        Char:
              Global:
                    LAST       * logical name for the last      *
                               * signal data file created       *
                    MAIN DIR   * logical name for the simulation*
                               * directory normally in use      *
                               * ( [ADB.MENU] on the Fuchs VAX  *

              Local:
                    FILENAME         * Name of the signal data *
                                     *  file to be transformed *
                    YN         * "Y" or "N"           *
                    CONTINUE

External procedures:
                  CHANGDIR.COM      * Changes default directory  *
                  HILBMENU.COM      * HILBERT TRANSFORM MENU *
Begin:
      Put:
       "
       " "
       ' "
         WHAT IS THE NAME OF THE DATA FILE YOU WISH TO HILBERT TRAN
         eg. SIGNAL"
         "
         "
         "
         "
      End put:
      Get:
        FILENAME
      End get:

      HILBERTIN.DAT := FILENAME.DAT          * DCL COPY COMMAND *
      Put:
          "Y or N:"
      End put:
      Get:
       YN
```

```
     End get:

     If (YN = "N")
     then:
          Call CHANGDIR.COM     * Changes default directory  *
     End if:

     Call HILBMENU.COM      * HILBERT  MENU  *

     LAST := "HILBERT.DAT"              *logical name assignment*

     Put:
      "PRESS RETURN TO CONTINUE:                "
     End put:
     Get:
      CONTINUE
     End get:

End:
End procedure:
```

8.4.3. **POL of HILBMENU.COM**

```
Procedure name: HILBMENU.COM
*****************************************************
* This DCL procedure is normally called by HILBERT.COM.           *
* It allows the user to choose from the hilbert transforms already*
* included in the simulation, or to use his own hilbert transform *
* coefficients.                                                   *
*****************************************************************

Variables:
     Integer:
          Single:
               Local:
                    CHOICE

     Char:
          Single:
               Local:
                    CHOICE2
                    LOGNAME
                    FILENAME

External programs:
                HILBERT.EXE      * FORTRAN FIR program to do *
                                 *  Hilbert transform        *
                IIRHILBERT.EXE    * FORTRAN IIR program to do *
                                  * Hilbert transform         *

Begin:

     Put:
     form feed
     "HILBERT TRANSFORM MENU                                    "
     "                                                          "
     "    1 : 79 tap FIR Hilbert transform"
     "        Bandwidth  : 0.05 to 0.45 of sampling frequency"
     "        maximum error (ripple) : 0.0000010
     "                                                          "
     "    2   95 tap FIR Hilbert transform"
     "        Bandwidth  : 0.01 to 0.49 of sampling frequency"
     "        maximum error (ripple) : 0.0217910
     "                                                          "
     *"   3   Insert additional options here "*
     "                                                          "
     "    9   Your own Hilbert transformer"                     "
     "

   End put:

      Put:
         "CHOICE"
```

```
   End put:
   Get:
       CHOICE
   End get:

Case CHOICE of:

 1: Run HILBERT                * FIR program              *
    MAIN DIR:HILBERT79.FLT     * Tap coefficients are read  *
                               * from this file           *
 2: Run HILBERT                * FIR program              *
    MAIN DIR:HILBERT95.FLT     * Tap coefficients are read  *
                               * from this file           *


*   3: Add new transforms here.    *
*      eg. Run IIRHILBERT             *
*          MAIN DIR:FIRCOEFTS.FLT     *
*                                     *

*   8: etc.                          *
*                                     *

   9: Put:
          form feed
          "  Is it an IIR or  IR structure?"
          "                                "

          ""
          "       A    IIR                 "
          "                                "
          "       B    FIR                 "
          "                                "
          "       C    Another transform program "
          "                                "
          "                                "
          "                                "
          "                                "
          "                                "
          "                                "
          11                               "
          11                               "
          11                               11
      End put:

      Put:
          "CHOICE:"
      End put:
      Get:
          CHOICE2
      End get:

      Put:
```

```
        form feed
        "    What is the name of the data file    "
        "    containing the filter coefficients?  "
        "    eg. HILBERT79.FLT                    "
        "                                         "
        "    (Read the HELP MENU if you are not
             sure of the correct format.)         "
        "                                         "
        "                                         "
        "                                         "
        "                                         "
        "                                         "
        "                                         "
        "                                         "
        "                                         "
        "                                         "
        "                                         "
        "                                         "
      End put:

      Get:
        FILENAME
      End get:

      LOGNAME := MAIN DIR:FILENAME
                        * DCL logical name assignment *

      Case CHOICE2 of:

        A: Run IIRHILBERT              * IIR HILBERT TRANSFORM *
           LOGNAME               * LOGNAME:= input data file *

        B: Run HILBERT                  * FIR HILBERT TRANSFORM *
           LOGNAME

*       C: Insert another Hilbert transform program here *
*          LOGNAME                                       *

      End case:

    End case:

end:

End procedure:
```

CHAPTER 9. SIMULATION FUNCTIONS

This chapter gives short descriptions of some of the functions presently available in the simulation. For more comprehensive information on the implementation and detail of a particular function, refer to the programme listings.

9.1. Utility Functions

This section describes some of the functions accessible from the UTILITY MENU.

9.1.1. Filters

The implementation of the filter menu structure was explained in the previous chapter. Two filter programmes have been included into the simulation thus far. These are:

1. FIRFILTER.FOR, which implements the "Direct form realisation of an FIR structure" [16], and

2. IIRFILTER.FOR, which implements the "second order cascade IIR structure" [17]. This programme is a good example of

how a FORTRAN programme should be written for the simulation.

If the user requires any other structures or an implementation that introduces scaling, he has the may write his own programme and run it from the filter menu, or (preferably) he may include it into the simulation structure as explained in the last chapter (and in the filter menu procedures).

The filter programmes read their coefficients from the filter coefficient data file whose name is supplied by the calling DCL procedure. The format of the filter coefficient data file is given in the HELP MENU.

9.1.1.1. Filter design

A number of filter design programmes and techniques exist. An expert system described in [18] is a good aid in deciding which one to use. The two programmes used most commonly in the simulation thus far have been DOREDI [19] and DFDP [20].

DOREDI outputs the filter coefficients in the form required by the simulation. All that has to be done is to delete all the other unwanted documentation that gets written to the DOREDI output file.

DFDP outputs its data in a much less compatible form. The job of converting this file to the simulation format is made trivial by the FORTRAN programme DFDPCONV, which may be run from the UTILITY function: "CONVERT A DFDP FILE TO SIMULATION FORMAT".

9.1.2. Displaying time domain signals

The display DCL functional procedure is DISPLAY.COM. It runs the FORTRAN programme GRAFIX.

The programme GRAFIX.FOR makes use of in-house library routines to display graphics on a CIT-101 terminal. This terminal can be configured as a VT-100 terminal or a TEK-4014 terminal (graphics mode). The input to GRAFIX.FOR is always a signal data file. The user may specify which portion of the signal he wants to see.

This programme might have to be altered when porting the simulation to systems which have other terminals. Routines have already been written to allow the simulation to run on a PC which emulates VTlOO and TEK terminals (using an emulation software package called "SmarTerm 240" [21]). These routines may be made active by using the Utility function "SPECIFY TERMINAL TYPE (CIT-101 or ST240)". This

function must be run every time a different type of terminal is used when the simulation is run.

9.1.3. Displaying spectra

The DCL functional procedure in this case is FFT.COM. It runs the FORTRAN programme FFT.

FFT.FOR makes use of a library FFT subroutine from [22) to do a 1024 point FFT. The FFT is done on the last 1024 points of the input signal data file. This is done so that distortions in the first few samples in a data file caused by filter lag while the first samples travelling through the delay line are not processed. If the input signal data file has less than 1024 points, it is zero padded to bring the number of points to 1024.

9.1.3.1. Windows

At present, the user has the choice of the following five windows :

1. Rectangular
2. Hamming
3. Hanning
4. Blackman (beta=0.8)
5. Triangular.

It is recommended that the Blackman or Hanning windows be used to see the full dynamic range of the spectrum.

9.1.3.2. Display Output

The FFT programme determine the magnitude from the real and imaginary components returned by the subroutine. This may be displayed using either a linear or dB scale. The dB scale has a range of 100 dB.

9.1.4. Plotting signals

The DCL procedure HARDPLOT.COM displays the PLOT MENU. This menu gives the user the option of displaying time or frequency domain signals on the terminal by calling the DCL procedures mentioned in the last two sections. The only difference is that a hard copy is made of the signal displayed by calling the procedure PLOT.COM. The PLOT MENU also allows the option of displaying the last graphics display that had been shown. The HARDPLOT.COM procedure accomplishes this by simply calling PLOT.COM.

9.1.5. Branching to VMS

The simulation offers the facility to branch into the normal VMS mode. Control is returned to the simulation by typing "EXIT". This is a useful feature for the more advanced user who is familiar with VMS. It has particular application for the user who is still busy developing his simulation

function, but has not yet included it into the simulation structure.

9.1.6. HELP

The HELP facility is controlled by the DCL procedure HELP.COM. This procedure sets the default subdirectory to [.HELP], where it calls the help menu procedure HELPMENU.COM. [.HELP] is the subdirectory where all the files needed for the help facility are stored. One of the facilities available from the help menu is an explanation of the format required for the filter coefficient data files. It is envisaged that this report will be subdivided and included under different options in the help menu.

The above method of implementing the HELP facility was chosen in preference to the use of the VMS HELP "Librarian Utility" [23]. The above method is consistent with the structure of the simulation and allows easier porting to a system without a similar HELP utility.

9.2. Secondary Utility Functions

The user has the option to enter a second utility menu ("OTHER UTILITY FUNCTIONS") from the main utility menu. This

menu is stored in UTILITY2.COM. The following sections briefly explain the functions available from this menu.

9.2.1. The dec;mat;on menu

The user has the option of three decimation routines:

1. Decimation by "throwing away" the unwanted samples. This method reduces the sampling frequency and the number of samples.

2. Setting the unwantd samples to zero. This preserves the number of samples, but introduces replicas of the signal around multiples of the decimated sampling frequency.

3. Simulating a sample-and-hold circuit by setting the D-1 samples that follow every Dth sample equal to that sample. {D=decimation factor).

9.2.2. Interpol ati on

The INTERPOLATE function in the simulation inserts (1- 1) zeros between every sample. {I= interpolation multiple) . This results in the sampling frequency being increased I times. Replicas of the signal appear around the sampling frequency.

9.2.2.1. Altering the simulat;on parameters

After a decimation or interpolation process which changes the sampling frequency and length of the signal data files, the user is asked whether he wishes to alter these simulation parameters to the new values. on a positive reply, the control data file, SIMPARAM.DAT, is overwritten with the correct values.

9.2.2.2. Filtering w.r.t. sampling rate transformation

The filtering function normally associated with decimation and interpolation must be done by the user as a separate step by entering the FILTER MENU. An example of the advantage of leaving this function up to the user is that he is able to simulate SSB and DSB/SC modulation by inserting zeros and then putting the resulting signal through a bandpass filter.

9.2.3. Hilbert Transformation

The Hilbert transform (90 degree phase shifter) has particular relevance to radio design when dealing with quadrature modulation techniques.

The Hilbert transforms implemented in the simulation thus far use FIR structures [24]. Provision has been made for the use of IIR structures. Implementation is done in a similar

manner to that of FIR and IIR filters with the exceptions of different input and output signal data file names.

9.2.4. Multiplication by a real tone

This option gives the user the opportunity to multiply his signal with one of two quadrature tones. Output data is written to either COSMULT.DAT or SINMULT.DAT, depending on the option chosen.

9.2.5. Multiplication by a complex tone

This function multiplies the input signal by both quadrature tones and writes the output to COSMULT.DAT (In-phase channel) and SINMULT.DAT (Quadrature-phase channel).

9.2.6. Addition

The user has the option of adding or subtracting two signals. The output is written to SUM.DAT.

9.2.7. Adding a tone to a signal

This function enables the conversion of a DSB/SC signal to an AM signal.

9.3. Tertiary Utility functions

The user has the option to enter a third utility menu "EVEN MORE UTILITY FUNCTIONS") from the secondary utility menu. This menu is stored in UTILITY3.COM. The following sections briefly explain some of the functions available from this menu.

9.3.1. Renaming signal data files

Sometimes ambiguity might exist in the names of the signal data files. This might occur in a quadrature modulation simulation where identical functions are happening in both channels. For example, if in a quadrature modulator, both channels have to be filtered, two different FILTER.DAT signal data files would exist. The "CHANGE NAME OF DATA FILE" option provides a way for distinguishing between the two FILTER.DAT files.

In the above case the user would be presented with the option of changing the name of the last FILTER.DAT file created to either:

1. FILTER.DAT (In-phase channel)
2. QFILTER.DAT (Quadrature-phase channel)
3. any other name

9.3.2. Adding noise to signals

The NOISE menu is activated by the NOISE.COM DCL procedure. This menu gives the user a choice of a number of noise

distributions. (At this stage, only gaussian shaped noise has been implemented, but it is clearly shown how to include other distributions into the menu.)

If gaussian noise is chosen, control is passed to the procedure GAUSS.COM which runs the FORTRAN programme GAUSS which generates gaussian noise of the required amplitude and adds this to the specified input signal data file.

9.3.3. Other Utility Functions

Other utility functions such as "INCLUDE A FORTRAN PROGRAMME" and "CONVERT A DFDP FILE TO SIMULATION FORMAT" have been discussed elsewhere in the simulation.

Note that space has been left for the inclusion of even more utility functions. The programmer is clearly shown in the PDL and DCL 1istings of UTILITY3.COM, how and where to include these functions.

CHAPTER 10. TRANSMIT / RECEIVE PATH STEPS

Apart from the step which has already been described ("ENTER SIMULATION PARAMETERS"), a number of other steps exist. The steps basically consist of menus implemented in the tree or a combination of the tree and linear-tree structures (modulation and demodulation).

10.1. Creating Signals

This step gives the user the generating signals to be processed during the simulation.

The step consists of the CREATE MENU which currently has two options.

In the one option, the user has the opportunity of creating multitone signals using the sampling frequency defined in the simulation. The user chooses the wordlength of the signal to be created.

The other option allows him to simulate "analogue" signals. This is done by creating the signal using a multiple of the simulation defined sampling frequency and a 32 bit floating point precision.

10.2. Sampling Signals

This step allows the simulation of the sampling an "analogue" signal. It does this by choosing only the appropriate samples from the "analogue" signals, or by simulating the effect of a sample-and-hold circuit.

10.3. Amplification Menu

This step presents the amplification menu. The menu currently gives the user the option of simple amplification and amplification related to the input signal level (Automatic Gain Control).

10.4. Audio Processing Menu

This tree-structured menu allows the inclusion of audio processing functions such as:

Pre-emphasis De-
emphasis
Speech processing
etc.

The pre- and de-emphasis programmes have been implemented using the bilinear z-transform. Detailed information about audio processing simulation is given in (11].

10.5. Modulation/Demodulation

Many modulation schemes may be simulated by using the utility functions. However, the user may be guided along a particular modulation path by calling the required utility functions from the path. The paths constituting the different modulation methods were implemented using the linear-tree structure mentioned in section 3.4.3. An example of the use of this method is given in the next chapter. A number of different modulation schemes have been implemented in this way. Detailed information is given in [12].

CHAPTER 11. EXAMPLE OF SIMULATION USAGE

11.1. Introduction

This chapter gives an example of a few of the abilities of the simulation. The block diagram of figure 11.1 shows the sort of simulation that may be undertaken by the user.

11.2. Explanation of Block Diagram

11.2.1. Transmit Path

Processing done in the transmit path could consist of :-

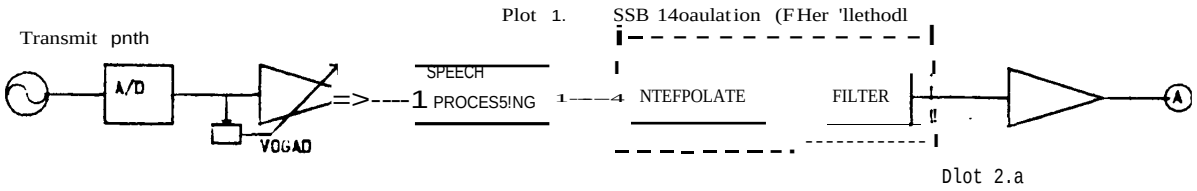
1. "Analogue" signal generation (actually a digital signal generated at a higher sampling frequency and precision).

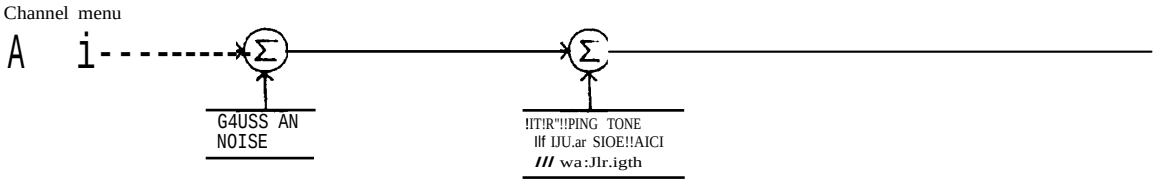
2. Simulation of A/D effects.

3. VOGAD (Voice Operated Gain Amplifying Device).

4. Speech processing.

5. SSB modulation (eg. using the filter method.)





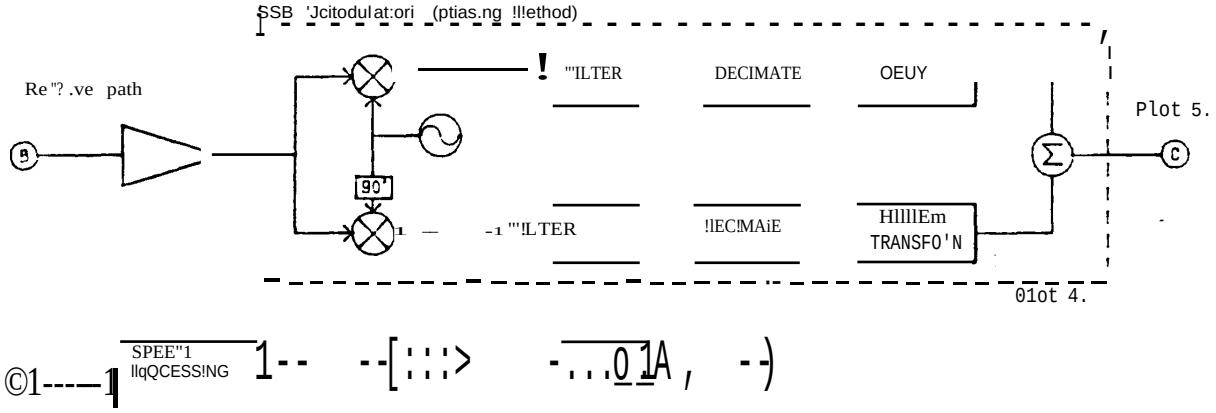


Figure 11. 1 ; 81ock diagram of an example of a s mu1ation process

11.2.2. Channel Menu

The signal could then be put through a gaussian noise channel and an interfering tone in the image sideband could be added.

11.2.3. Receive Path

The user may choose to amplify the signal, put it through an SSB demodulator eg. (using the phasing method), do some speech processing and simulate the effects of a D/A converter.

Plots of the FFTs of the signal data files observed in the above simulation are given at the end of this chapter.

11.3. Examole of Simulation Process

Space does not permit the showing of the whole simulation process here. What will be shown is how a user would implement a particular modulation process ie. SSB modulation using the filter method (indicated in Fig. 11.1). The user

would enter the modulation menu. This would give him a choice of SSB, AM, FM and Data Modulation. He could then choose the SSB modulation option which would present him with a number of SSB modulation techniques. Upon choosing a technique he is presented with a description of the technique. He is then prompted to do the correct functions to implement the techniques. He always has access to the utility menu which allows him to display the processed signal at any point.

An indication of how the user would see this process is contained in the next few pages, followed by plots of the graphics the user would see.

UTILITY MENU

1: FILTER
2: DISPLAY A SIGNAL
3: DO AN FFT ON A SIGNAL AND DISPLAY IT
4: BRANCH TO THE VMS OPERATING SYSTEM
5: OTHER UTILITY FUNCTIONS <HILBERT TRANSFORM ETC.>
6: HELF'
7: PLOT A SIGNAL ON THE PLOTTER
a: GO ON TO THE NEXT PART OF THE SIMULATION <AMPLIFY/A
9: RETURN TO MAIN CALLING MENU

CHOICE: 9

T .'.ANSMIT !='ATH

1: ENTE!=\: SIMU:_ATI0N PARAMETERS (e<j <:;amt.:1ig fre'1_uenc
1: CREATE A SIGNAL

SAMPLE
3: VOGAD/AMPLIFICATION
4: OTHER AUDIO BAND PROCESSING
S: MODULATION
6: AUTOMATIC GAIN CONTROL <AGC>
7: EXIT FROM TRANSMIT PATH
S: CALL UTILITY MENU <Fters, FFT'<.::. DisPla'::' roJues,1
9: EXIT FROM TRANSMIT MENU

CHOICE: 5

MODULATION MENU

CHOOSE ONE OF THE FOLLOWING MODULATION SCHEMES

1: SSB

2: AM

3: FM

4: DATA <MODEM>

CHOICE: 1

SSB MODULATION MENU

CHOOSE ONE OF THE FOLLOWING SSB MODULATION SCHEMES

1: Phasin9 and inter olation

2: Interpolate and filter

CHOICE: 2

This section of the simulation rom ts ou to simulate a specific modulation technique. To· tr -out more general techniques, use the UTILITY MERU functions

The method used b this section is to insert zero's a bandlimited inut,thus increasing the sampling frequency

The new si nal should then have relicas around the old sampling fequenc . If an SSB filter is placed around one of one ·Of the relicas sidebands an SSB signal should be obtained

<Note: This is a var iation of the filter method commonl used in analogue SSB radios, with the var iation that - interpolation is used instead of multiplication

Press RETURN to continue•••:
The first function required is to inter olate the signal

Do you want to:

4: INTERPOLATE A SIGNAL

5: ENTER THE UTILITY MENU

6: CONTINUE

WHICH CHOICE! 4

What is the name of the file in which ou wish to insert zeros
eg. SIGNAL

EMPHASIS

INTRPOLATION MULTIPLE 7

eg. 6 means 5 zeros will be inserted between samples

<i.e. 6 times as many samples as before)

4

The inter olation recess changes the arameters <Sampling frequency and sample length) of the simulation
The new values should be inserted into file SIMPARAM.DAT to keep the simulation running smoothly

The sampling frequenc should be changed from
14400.00 to 57600.00

The lenth of the data files should be changed from
1536 to 6144

Do ou wish this to hapen? CY/N J
y

INTERPOLATED SIGNAL IS STORED IN FILE INTPOLATE.DAT

The first function required is to interbolate the signal

Do you want to:

4!INTERPOLATE A SIGNAL

5:ENTER THE UTILITY MENU

6: CONTINUE

WHICH CHOICE: 5 .

UTILITY MENU

1: FILTER
2: DISPLAY A SIGNAL
3: DO AN FFT ON A SIGNAL AND DISPLAY IT
4: BRANCH TO THE VMS OPERATING SYSTEM
5! OTHER UTILITY FUNCTIONS <HILBERT TRANSFORM ETC.>
6: HELP
7: PLOT A SIGNAL ON THE PLOTTER
8: GO ON TO THE NEXT PART OF THE SIMULATION
9: RETURN TO MAIN CALLING MENU

CHOICE: 3
What is the narue of the inut file to be FFTed
eg. SIGNAL
LAST

WINDOW

1! RECTANGULAR
2: HAMMING
3: HANNING (RAISED COSINE>
4! BLACKMAN
S: BARTLETT <TRIANGULAR)

4

1: LINEAR
2! dB SCALE

2

FFT APPEARS ON GRAPHICS SCREEN

PRESS RETURN TO GO TO CALLING MENU:

UTILITY MENU

1: FILTER
2: DISPLAY A SIGNAL
3: DO AN FFT ON A SIGNAL AND DISPLAY IT
4! BRANCH TO THE VMS OPERATING SYSTEM
5: OTHER UTILITY FUNCTIONS <HILBERT TRANSFORM ETC.>
6: HELP
7: PLOT A SIGNAL ON THE PLOTTER
B: GO ON TO THE NEXT PART OF THE SIMULATION
9: RETURN TO MAIN CALLING MENU

CHOICE: 8
The first function reuired is to interpolate the si nal

Do ou want to:

4: INTERPOLATE A SIGNAL
5: ENTER THE UTILITY MENU
6! CONTINUE

WHICH CHOICE: 6
The nxt function reuired is to filter the si nals to remove replicas of the signal

Do ou want to!

A: FILTER A SIGNAL
B: ENTER THE UTILITY MENU
C! CONTINUE

OPTION A

WHAT IS TYE NAM OF THE DATA FILE YOU WISH TO FILTER
ej+ S GNAL.

FILENAME: INTPOlATE

DO YOU WANT THE DEFAULT CHIP TO BE USED? (Y/N)
i.e
16 bit fixed point wordlenth
eg. TMS 320, TMS320C25, ADSP-2100

Y or N: Y

FILTER MENU

1 LOW PASS
2 HIGH PASS
3 : BAND PASS <SSB TYPE>
4 BAND PASS (AM TYPE)
9 Your own filter

CHOICE: 3

1 14th order Elli tic Band ass filter Desiqned to be USB filter with nominal carrier at 8 kHz and sam lind freduenc of 40 kHz. 0.02 Passband ri ple; 46 dB st6pand sup!::'ressOn.

2 14th order Elli tic Band Pass filter Designed to be LSB filter· with nominal carrier at 16 kHz and sam lin freuenc of 40 kHz. 0.02 Dassband ripple,-46 dB stoand s1..1pPress1on.

3 10th order Elli tic Band ass filter Designed to be USB filter with nominal carrier at 10 kHz and sam lin freuenc of 40 kHz 0.02 assband ri Ple.-46 dB stoand supPression.

4 Other SSB filters

CHOICE: 4

5 10th order Elli tic Band ass filter Desi ned to be LSB filter with nominal carrier at 14.4 kHz and samDling frequenc of 57.6 kHz. 0.4 dB Passband rippl , 20 B cirrier suPDress1on, 40 dB at carrier + 400 Hz, 58 dB at· arrier 600 Hz and greater.

9 Your own filter

CHOICE: 5

6144 data sam les were written to file FILTER.DAT

LAST IS NOW DEFINED AS FILTER.DAT

ºRESS RETURN TO CONTINUE
The next function reuired is to filter the signals to remove replicas of the signal

Do o•J want to:

A: FILTER A SIGNAL

B: ENTER THE UTILITY MENU

C: CONTINUE

OPTION: C

UTILITY MENU

1! FILTER
2: DISPLAY A SIGNAL
3: DO AN FFT ON A SIGNAL AND DISPLAY IT
4: BRANCH TO THE VHS OPERATING SYSTEM
S: OTHER UTILITY FUNCTIONS <HILBERT TRANSFORM ETC >
6: HELP
7: PLOT A SIGNAL ON THE PLOTTER
B: GO ON TO THE NEXT PART OF THE SIMULATION <AMPLIFY/
9: RETURN TO MAIN CALLING MENU

CHOICE: 3
What is the naffie of the input file to be FFTed
e. SIGNAL
UST

!JINDOW

HAMMING
:t.: RECTANGULAR
3: HANNING <RAISED COSINE>
4! BLACKMAN
BARTLETT <TRIANGULAR >

4

1: LINEAR
2: dB SCALE

2

FFT APPEARS ON GRAPHICS SCREEN

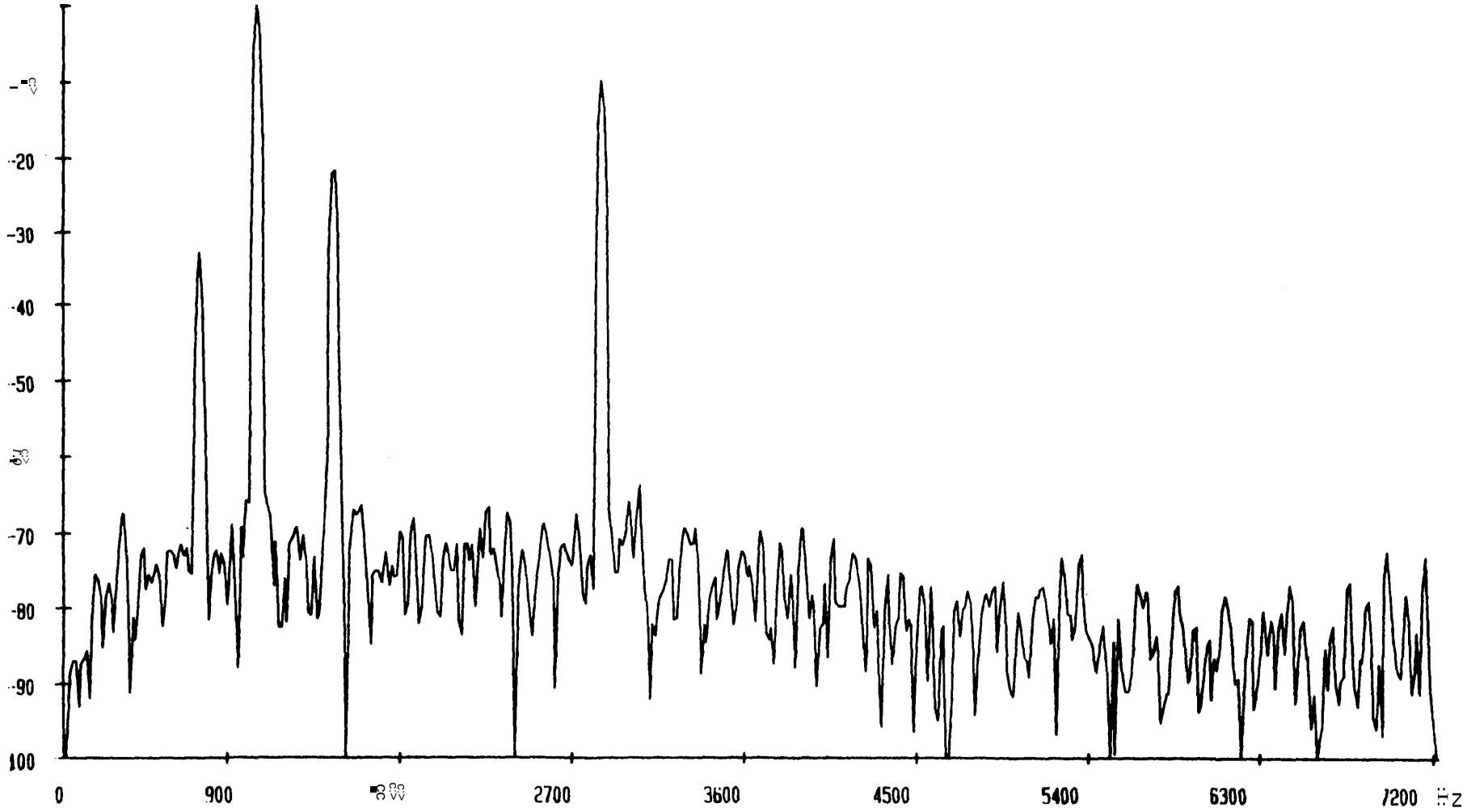
PRESS RETURN TO GO TO CALLING MENU:

:C
c:>
c:>
CCI
c:>
CT.
c:>
c:>
a..

Hz
28800
25200
21600
18000
14400
10800
7200
3600
0

Plot Z
0
3000
6000
9000
12000
15000
-6000
-3000
0
3000
6000

M
r-
a.



PLOT 4

CHAPTER 12. CONCLUSION

A structure for a computer simulation which is an effective aid for the design of a DSP radio has been presented.

A number of different structures were investigated, and the form chosen for the simulation was presented. It was explained how the structure permits the simulation of DSPs with differing wordlengths and configurations . The initialisation and control of the simulation were explained and it was shown how a function should be included into the structure. Brief descriptions of the functions already included in the simulation were given, and an example of how the structure and its functions are used was shown.

The needs of the simulation users and programmers were met by designing a user friendly, understandable, flexible and easily expandable structure. Advantage was taken of the serial nature of radio processing to do this with a structure with a well defined DCL implementation in the top part of the hierarchy, and the functional level programmed in a high level language. One of the advantages of the structure implemented is that it separates the user interface from the actual processing. (i.e. it allows the user to develop his high level language separate to the simulation and include it once it has been debugged.}

Although the simulation as presented is in a highly usable form, there are a number of improvements and alterations that could be included. Some of these are:

1. Allowing the even easier return to the last function executed. Since the simulation allows the easy simulation of different word configurations, the user might wish to return to the previous function and re-execute it with a different configuration, thus easily comparing the effects of the two configurations. (This ability already exists within the modified linear-tree structure, but it may be enhanced by introducing small changes to the structure and the utility menu.

2. Greater error tolerance. If the user inputs an illegal value into a FORTRAN programme, the programme may crash. When this happens the simulation terminates. This could be avoided by employing certain programming techniques to trap the programme in an error condition. DCL also has some functions that could be used to allow the simulation to continue after a programme crash.

3. Simulation of wordlengths greater than 32 bits. The simulation's FORTRAN programmes specifically use single precision (32 bit) real numbers. This was done purposely

to allow the easy and quick simulation of 32 bit floating point DSP chips, which are expected to become increasingly prolific. However, the disadvantage of this approach is that it limits the simulation to chips of 32 bits and less. At this time, it is felt that the approach adopted was the correct one for the current DSP scenario. However, should any chips of a higher precision become desirable to use, it would be necessary to alter the FORTRAN programmes to have double or quadruple precision real numbers.

4. The use of better I/O techniques. The use of techniques to speed up the reading and writing of data files could be investigated. When very big files are handled this could be advantageous. However, one would have to be wary of placing more than the bare minimum of restrictions and rules on the high level language programmers.

5. Use a truncation FUNCTION. It might have been neater to make the TRUNCATE subroutine a FORTRAN "Function" instead.

6. A file recording the names of the signal data files created in the order in which they were created. This would give a greater sense of continuity to the user.

7. Catalogues accessible from the help menu containing:

   a. Commonly used signal data files (eg. Audio speech signals, SSB signals, etc.)

   b. Descriptions of the filters available in the simulation.

   etc.

a. The use of commercial software as a graphics "front end" for the simulation. Although the simulation already has adequate graphics facilities, these could be enhanced by using a software display package such as "DADiSP" to display the signals with more versatility. It would be possible to include a feature such as this as a function accessible from the utility menu.

9. The inclusion of a PRBS generator and digital modulation and demodulation functions (QPSK, FSK, etc.) to enable the simulation to be used for data as well as voice transmission. The simulation structure has been specifically designed to allow the inclusion of these functions as well as other analogue modulation functions

such as frequency modulation.

None of the above points would be very difficult to implement. If I were to redo this project, I would probably include most of the above points, but the greatest difference would be that I would try to remove the simulation from the restrictive VAX environment with only DCL and FORTRAN available. I would probably do it on a PC, possibly using a more structured and friendly languages (eg. Pascal or C).

Another idea for the future might be to write a PDL-to-DSP machine code compiler. This would enable the user to finalise his algorithms at a high level and automatically compile them into DSP code.

No formal testing of the simulation software has been done, but it has been extensively used and any problems noticed have been rectified. Every programme and procedure has been used, and because of the nature of the structure, their dependence on each other is minimal.

The simulation has been used to investigate a number of sampling, audio processing, modulation and demodulation techniques, both by the author and designer of the simulation structure [10-12], as well as by a new DSP_

engineer who has recently arrived on the DSP radio project [25,26]. His reaction to the simulation has been very positive. He has found it clear and easy to use and has already begun to implement some functions into the structure .

An effective tool for aiding DSP designers in developing a DSP radio has been developed. It is expected that the effort put in designing the simulation structure will bear fruit when a DSP radio is presented in the not too distant future.

REFERENCES

1. K.-D. Kammeyer, R. Mann and W. Tobergte, "A modified FIR Equaliser for Multipath Echo Cancellation in FM Transmission", IEEE Journal on Selected Areas in Communications, volume SAC-5, number 2, pp. 226-237, February, 1987

2. D.J. Bagwell and V. Considine, "Digital Signal Processing Architectures for HF Receivers", Third International conference on HF Communication Systems and Techniques, 26-28 February 1985, pp. 86-88

3. D.T. Anderson and J.W. Whikehart, "A Digital Signal Processing HF Receiver", Third International conference on HF Communication Systems and Techniques, 26-28 February 1985, pp. 89-93.

4. A. Jongepier, "A digital FM-detector", IEEE International Conference on Consumer Electronics, 1984, pp. 180-181

5. ILS Signal Processing Software, Signal Technology Inc., 5951 Encina Road, Goleta, California, USA.

6. "DADiSP", DSP Systems, 1 Kendall Square, Cambridge, MA 02139, USA

7. "DSPlay", Airport International Industrial Park, Tucson, Arizona, USA.

8. M.W. Coetzer, "The FUN DSP System", University of Stellenbosch, Stellenbosch, 7600, South Africa.

9. A.J.A. Carter, "A General Development System for Digital Signal Processing Microprocessors", Fifth South African Symposium on Digital Signal Processing, 6 July 1987.

10. A.O. Bassios, "DIGITAL BASEBAND: Report on work done on Simulation", Fuchs Electronics, P.O. Box 57, Alberton, December, 1986

11. A.D. Bassios, "DIGITAL BASEBAND : Simulation of Audio Processing", Fuchs Electronics, Alberton, April 1987.

12. A.O. Bassios, DIGITAL BASEBAND : Report on simulation of SSB and AM Modulation, Fuchs Electronics, July 1987.

13. "Guide to Using DCL and Command Procedures on VAX/VMS ", Digital Equipment Corporation, Maynard, Massachusetts, USA, September 1984.

14. A.J. Walker, "Structured Information Processing system Design, Department of Electrical Engineering, University of the Witwatersrand, Johannesburg, June 1986.

15. C.C. Watterson, J.R. Juroshek and W.O. Bensema, "Experimental confirmation of an HF channel model", IEEE Transactions on Communications Technology, COM-18, 792-803, 1970.

16. A.V. Oppenheim and R.W. Schafer, "DIGITAL SIGNAL PROCESSING, Prentice-Hall, Englewood Cliffs, 1975, pg 156.

17. A.V. Oppenheim and R.W. Schafer, "DIGITAL SIGNAL PROCESSING", Prentice-Hall, Englewood Cliffs, pp 152.

18. A. Woermann, N.M. Bawden and H.E. Hanrahan, "Use of a Knowledge-based Expert System in the Selection of a Digital Filter Design Method", Fourth South African Symposium on Digital Signal Processing, 27 June 1986.

19. G.F. Dehner, "Program for the Design of Recursive Digital Filters", Programs for Digital Signal Processing, DSP Committee, IEEE, IEEE Press, 1979

20. "Digital Filter Design Package", Atlanta Signal Processors Inc., 770 Spring St., Atlanta, USA.

21. "Smarterm 240", Persoft Inc., 2740 Ski Lane, Madison, Wisconsin, USA, 1986.

22. G.D. Bergland and M.T. Dolan, "Fast Fourier Transform Algorithms", Programs for Digital Signal Processing, DSP Committee, IEEE Press, 1979, pp 1.2-1 to 1.2-18.

23. "VAX/VMS Librarian Reference Manual", Digital Equipment Corporation, Maynard, Massachusetts, September 1984

24. L.R. Rabiner and R.W. Schafer, "On the Behavior of Minimax FIR Digital Hilbert Transformers", The Bell Technical Journal, Vol. 53, No. 2, February 1974, pp 363-391.

25. M.W. Robson, "Digital Signal .Processing Applied to HF Radio", Fuchs Electronics, Alberton, July, 1987.

26. M.W. Robson, "DIGITAL BASEBAND Report on the simulation of AM demodulation, Fuchs Electronics, to be presented September 1987.

APPBBDIX A THE TRUBCATIOB PROCESS

The truncation process in an integral and important feature of the simulation. It enables programmers to easily simulate the effects of chips at different wordlengthe and fixed and floating point configurations.

The truncation data needed by the truncation subroutines used in the simulation is originally created by the FORTRAN programme TRUHCDATA. The limits of the truncation process are then written to the data file TRUNCDATA. DAT. The subroutines in the simulation then use the data read from TRUNCDATA. DAT to do the truncation. This appendix explains the calculation of the limits CTRUBCDATA programme) and the use of the limits to do the actual truncation <TRUNCATE subroutines> for both fixed and floating point.

FIXED POINT TRUNCATION

Initialisation

The maximum possible positive value of the 2 s compliment word is:

$$2^{1}$$

and the maximum negative value is:

$$-2^{b}$$

where b = number of bits per word

Generally, fractional arithlletic is used with fixed point configurations. This means that all values are referenced to the value 1.0.

i.e. 2^b is taken to represent the value 1.0.

Therefore, the value of the least significant bit <LSB> is:

$$1/2^{b-1}$$

$$= 2^{-(b-1)}$$

The maximum positive value a word may have is:

$$1-2^{-(b-1)}$$

and the maximum negative value is:

$$-1.0$$

The programme TRUNCDATA.FOR asks the user for the wordlength of a single precision register of the chip to be simulated as well as the wordlength of the accumulator <which generally has a higher precision>.

The maximum possible positive and negative values and the value of the LSBs are then calculated for both cases and written to the data file TRUICDATA.DAT.

The Truncation Process

The programme that wishes to do the truncation then reads the data from TRUBCDATA.DAT. Every time the programme wishes to truncate a number, it passes these parameters <along with the number to be truncated> to the truncation subroutine < TRUBCATE> .

This subroutine then checks to see whether the number falls within the limits. If not it writes an "OVERFLOW'' message to the terminal and limits the number to the maximum allowable value.

If the number falls within limits, it is truncated by dividing the number by the LSB, truncating all points to the right of the decimal point and then multiplies by the LSB again. The truncated number is then returned to the calling programme.

FLOATING POINT TRUNCATION

Initialisation

Floating point configurations do not require the use of fractional arithmetic, but they also use 2's complement arithmetic.

The Exponent

The exponent is a fixed point number. The maximum positive value it can be is:

and

the maximum negative value is:

-2^{P}

The Mantissa

As the mantissa is a fractional fixed point number similar to that explained in the previous section:

The maximum positive value is:

$1-2^{-(m-1)}$

The maximum negative value is:

-1.0

where m = no. of mantissa bits.

The exponent is automatically adjusted by the chip so that the first mantissa bit set after the decimal point is set. Therefore, apart from zero, the minimum :mantissa value is 0.5.

Therefore, the :maximum positive value of the floating point wod is:

(1-2-C:m-1>),(2P-1)

The maximum negative value is:

<-1.0).{2P-1)

The minimum positive value is:

(0,5),(2-P)

The minimum negative value is:

-(0,5),(2-P)

Where m = number at mantissa bits,

and p = number of exponent bits.

The maxima and minima are calculated by TRUBCDATA.FOR and written to TRUBCDATA.DAT, along with the value 2^{m-1} <the inverse of the value the LSB, called the mantissa_range>.

The floating point truncation subroutine then uses these parameters to check if the input value to be truncated is within bounds. If it is, the truncation process starts:

The :mantissa is separated from the exponent. If is then multiplied by the mantissa_range <2m-1), and the digits to the right of the decimal point are truncated. The process is then reversed to obtain the original number which has been truncated to a-bit mantissa format.

The truncation programmes and subroutines may be examined in the programme listing.