1

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service

Services des thèses canadiennes

Ottawa, Canada
K1A 0N4

CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

**THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED**

**LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE**

National Library
of Canada

Bibliothèque nationale
du Canada

CANADIAN THESES
ON MICROFICHE

THÈSES CANADIENNES
SUR MICROFICHE

0-315-22235-2

NAME OF AUTHOR/NOM DE L'AUTEUR SEAN BROPHY
TITLE OF THESIS/TITRE DE LA THÈSE SYNCHRONIZATION AND ITS EFFECT ON
ADAPTIVE ECHO CANCELLER PERFORMANCE IN THE
DIGITAL SUBSCRIBER LOOP ENVIRONMENT
UNIVERSITY/UNIVERSITÉ CARLETON
DEGREE FOR WHICH THESIS WAS PRESENTED/
GRADE POUR LEQUEL CETTE THÈSE FUT PRÉSENTÉE Master of ENGINEERING
YEAR THIS DEGREE CONFERRED/ANNÉE D'OBTENTION DE CE GRADE 1985
NAME OF SUPERVISOR/NOM DU DIRECTEUR DE THÈSE DR. DAVID FALCONER

Permission is hereby granted to the NATIONAL LIBRARY OF
CANADA to microfilm this thesis and to lend or sell copies
of the film.

The author reserves other publication rights, and neither the
thesis nor extensive extracts from it may be printed or other-
wise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈ-
QUE NATIONALE DU CANADA de microfilmer cette thèse et
de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la
thèse ni de longs extraits de celle-ci ne doivent être imprimés
ou autrement reproduits sans l'autorisation écrite de l'auteur.

DATED/DATE 29 Jan 85 SIGNED/SIGNÉ Sean Brophy

PERMANENT ADDRESS/RÉSIDENCE FIXE 32 WALTER WALTON COURT
OTTAWA, ONT
K1V 9T1

**SYNCHRONIZATION AND ITS EFFECT ON
ADAPTIVE ECHO CANCELLER PERFORMANCE IN THE DIGITAL
SUBSCRIBER LOOP ENVIRONMENT**

by

Sean G. Brophy, B.Eng.

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfilment of the requirements
for the degree of
Master of Engineering

Faculty of Engineering
Department of Systems and Computer Engineering
Carleton University

January 1985

© copyright

1985, Sean G. Brophy

The undersigned hereby recommends to the Faculty of
Graduate Studies and Research, acceptance of the thesis,

**"Synchronization and Its Effect on Adaptive Echo
Canceller Performance in the Digital Subscriber Loop
Environment"**

submitted by Sean G. Brophy, in ~~partial~~ fulfilment of the
requirements for the degree of Master of Engineering.

Thesis Supervisor

Chairman,
Department of Systems and
Computer Engineering

Department of Systems and Computer Engineering

Faculty of Engineering

Carleton University

January 1985

ABSTRACT

It is known that the performance of an adaptive echo canceller in the digital subscriber loop environment is sensitive to timing jitter. Tradeoffs in the design of the common timing recovery subsystem employing a prefilter, a nonlinear element and a phase locked loop are investigated through simulation, with respect to the echo canceller performance.

The scope of the investigation includes the effects of prefilter shape, phase locked loop bandwidth and equalization over various loop configurations varying in length and number of bridged taps.

It is found that judicious design of the timing recovery subsystem will lead to satisfactory echo canceller performance for worst case loops. The primary factors include use of a narrow bandwidth phase locked loop, use of fairly narrow bandwidth prefilter accurately tuned to half the baud rate and use of a fixed equalizer matched to the worst case loops.

ACKNOWLEDGMENTS

I would like to express a very special thanks to Dr. David Falconer. I am very grateful for having the opportunity to work with him and his inspiration will not be forgotten.

I would like to thank Vilas Joshi and Pat Kavanaugh for their advice and encouragement. Also I would like to thank C. Eastwood for her patience and confidence. Not to be forgotten are my fellow graduate students too numerous to mention individually. Many thanks are extended to Diane Dodds for her generous advice and cooperation.

The author is indebted to D.G. Messerschmitt of University of California, Berkeley, for his assistance and advice with BLOSIM. I would also like to extend my thanks to Dr. Mahshad Koohgoli for his interest and Bell-Northern Research Ltd. (BNR) for their support. This research was funded in part by BNR.

TABLE OF CONTENTS

ABSTRACT		1
ACKNOWLEDGMENTS		11
LIST OF FIGURES		v
LIST OF TABLES		ix
GLOSSARY OF TERMS		xi
 CHAPTER 1	 INTRODUCTION	 1
1.1	Introduction	1
1.2	The Loop Plant	2
1.3	Two-Wire Duplex Data Transmission Methods	8
1.4	Synchronous Versus Non-Synchronous Operation	9
1.5	Line Code	11
1.6	Summary	12
1.7	Outline of the Report	13
 CHAPTER 2	 ECHO CANCELLATION TECHNIQUES	 17
2.1	Introduction	17
2.2	Transmission Path and Echo Path Model of the DSL	20
 CHAPTER 3	 TIMING RECOVERY	 35
3.1	Background	35
3.2	Equalizer	42
3.3	The Prefilter	45
3.4	The Nonlinear Device	45
3.5	The Post Filter	46
3.6	The Phase-Locked Loop	46
 CHAPTER 4	 BLOSIM	 51
4.1	BLQSIM	51
4.2	The BLOSIM Experience	59
4.3	Conclusion	61

CHAPTER 5	SIMULATION AND PERFORMANCE EVALUATION	62
5.1	Background	62
5.2	Near- and Far-End Transmitters	62
5.3	The DSL Channel	65
5.4	Echo Canceller	67
5.5	Timing Recovery System	67
5.6	Performance Evaluation	70
CHAPTER 6	SIMULATION RESULTS	74
6.1	Scope of the Investigation	74
6.2	Simulation Details	75
6.3	Effect of Prefilter Bandwidth	80
6.4	Effect of Prefilter Mistuning	87
6.5	The Effect of Bridged Taps	87
6.6	Equalizer Considerations	90
6.7	Choice of Nonlinear Function	98
6.8	The Effect of PLL Natural Frequency	101
6.9	Summary	101
CHAPTER 7	CONCLUSIONS	109
7.1	Conclusions	109
7.2	Suggestions for Further Work	110
REFERENCES		112
APPENDIX A	ABCD Parameters for Bridged Taps	A1
APPENDIX B	Cable Characteristics Data	B1
APPENDIX C	Analog to Digital Transformations	C1
APPENDIX D	IIR Band Pass Filter Frequency Responses	D1
APPENDIX F	Computer Simulation Program Listings	F1
APPENDIX G	Jitter Spectrum Estimation Program	G1

LIST OF FIGURES

Figure

1.1	Illustration of a simple telephone connection	3
1.2	A typical subscriber line	5
1.3	A typical DSL arrangement showing sources of echo due to imperfect isolation in the hybrid and open ended bridge of tap reflection	6
1.4	Bipolar or AMI code frequency spectrum and coding state diagram	14
1.5	Worst case DSL relative signal levels	16
2.1	Basic echo canceller arrangement for two-wire full duplex transmission	18
2.2	Echo canceller structure for $M=2$ phases	23
2.3	Illustration of zero reference adaptation algorithm	25
2.4	Zero reference echo canceller arrangement for $M=4$ phases	27
2.5	General full duplex two-wire echo canceller configuration	28
2.6	Illustration of exaggerated timing jitter	30
3.1	Timing recovery placement in DSL receiver	36
3.2	Block diagram of timing recovery elements showing frequency response at several points	39
3.3	Required symmetry conditions for reduced timing jitter	41
3.4	General DSL timing recovery configuration employing a PLL	43
3.5	Attenuation versus frequency for typical cable gauges	44
3.6	Phase locked loop components	48

3.7	Basic analog PLL model	48
3.8	Linearized analog PLL model	48
4.1	Illustration of a simple example simulation structure (i.e., the UNIVERSE)	54
4.2	BLOSIM code for a simple example STAR	55
4.3	BLOSIM code for topology definition of a simple example GALAXY	56
4.4	BLOSIM code for topology definition of the complete example universe	57
4.5	Narrative mode output for example universe	58
5.1	DSL simulation block diagram	63
5.2	Transmitter simulation structure	64
5.3	General timing recovery configuration	64
5.4	Complete detailed DSL simulation block diagram without performance evaluation	69
5.5	Complete detailed DSL simulation block diagram with performance evaluation	71
6.1	An impedance-balancing hybrid structure	78
6.2	Fixed equalizer structure	78
6.3	Illustration of the use of a bandpass limiter	79
6.4	Jitter spectrum illustrating advantage of using appropriate bandwidth prefilter	81
6.5	R versus prefilter bandwidth for various values of PLL natural frequency	83
6.6	R versus prefilter bandwidth for PLL $f_n=1.1$ for equalizer and no equalizer on 3.5 mile line with 2 bridge taps	84
6.7	R versus prefilter bandwidth for PLL $f_n=.77$ for equalizer and no equalizer on 3.5 mile line with 2 bridge taps	85
6.8	R versus prefilter bandwidth for PLL $f_n=.23$ for equalizer and no equalizer on 3.5 mile line with 2 bridge taps	86

6.9	Jitter spectrum for 2.5 mile for no bridge tap and bridged tap cases	95
6.10	Jitter spectrum for equalizer constants $K = .7, .8, .9$ on 1.5 mile line with one bridged tap using a 40 KHz BW prefilter	96
6.11	Jitter spectrum for equalizer constants $K = .7, .8, .9$ on 3.5 mile line with 2 bridged taps	97
6.12	Jitter spectrum for the squaring and absolute value nonlinearities on a 2.5 mile line with PLL $f_n = .50$ KHz, and equalization	100
6.13	Jitter spectrum for various values of PLL natural frequency f_n on a 3.5 mile line with 2 bridged taps using a 40 KHz BW prefilter, an equalizer and squarer	103
6.14	Jitter spectrum for various values of PLL natural frequency f_n on a 3.5 mile line with 2 bridged taps using a 10 KHz BW prefilter, and absolute value	104
6.15	Illustration showing the near-end far-end and residual echo powers for short (1.5 mile), medium (2.5 mile) and long (3.5 mile) lines	106
6.16	Plot of the derivative of the impulse response of the echo-path channel for 1.5, 2.5 and 3.5 mile lines with one bridged tap	108
A.1	ABCD parameters	A3
A.2	Use of the ABCD parameter matrix	A3
A.3	Illustration of a line with a bridged tap and use of ABCD matrix	A3
B.1	Cable characteristics for 19 AWG PIC cable at 70°F	B2
B2.	Cable characteristics for 26 AWG PIC cable at 70°F	B3
D.1	Amplitude response of IIR prefilter centered at 72KHz with bandwidth 10 KHz	D2

D.2	Amplitude response of IIR prefilter centered at 72KHz with bandwidth 20 KHz	D3
D.3	Amplitude response of IIR prefilter centered at 72KHz with bandwidth 40 KHz	D4
D.4	Amplitude response of IIR prefilter centered at 72KHz with bandwidth 80 KHz	D5
D.5	Amplitude response of IIR prefilter centered at 72 KHz with bandwidth 144 KHz	D6
D.6	Amplitude response of IIR prefilters centered at 72 KHz and 74 KHz with bandwidth 20 KHz (i.e., mistuned)	D7
E.1	Impulse response of 1.5 mile line transmission channel with and without a bridged tap. (Plot shows impulse response over 20 symbol times)	E3
E.2	Impulse response of 2.5 mile line transmission channel with and without a bridged tap. (Plot shows impulse response over 20 symbol times)	E4
E.3	Impulse response of a 3.5 mile line transmission channel with and without a bridged tap. (Plot shows impulse response over 20 symbol times)	E5
E.4	Impulse response of 1.5 mile line echo channel with and without bridged tap. (Plot is for 20 symbol times)	E6
E.5	Impulse response of 2.5 mile line echo channel with and without bridged tap. (Plot is for 20 symbol times)	E7
E.6	Impulse response of 3.5 mile line echo channel with and without bridged tap. (Plot is for 20 symbol times)	E8
F.1	UNIVERSE.T Listing	F2
F.2	PLL_GAL.T Listing	F5
F.3	Listing of STAR's	F6
G.1	Spectral Estimation Program Listing	G2

LIST OF TABLES

Table

6.1	PLL parameters for a second order PLL	77
6.2	Effect of mistuning prefilter (R , R_{margin}) for mistuned by 2 KHz to 74 KHz center frequency and 20 KHz BW on a 3.5 mile line with 2 bridged taps	88
6.3	Effect of mistuning prefilter on EC (R , R_{margin}) for 2 KHz mistuning, 20 KHz BW on a 2.5 mile line with and without bridged taps	89
6.4	Effect of mistuning prefilter on EC (R , R_{margin}) for 2 KHz mistuning, 20 KHz BW, on 1.5 mile line with one bridged tap, no equalizer	89
6.5	Effect of presence of bridged taps on line versus prefilter BW giving (R , R_{margin}) PLL $f_n = .50$, on 2.5 mile line, no equalizer	91
6.6	Effect of presence of bridged taps on line versus prefilter BW giving (R , R_{margin}) PLL $f_n = .03$, on 2.5 mile line, no equalizer	92
6.7	Effect of equalizer constant k values on EC performance (R , R_{margin}) for 1.5 mile line with 1 bridged tap PLL $f_n = .50$, nonlinearity = absolute value for various prefilter bandwidths	93
6.8	Effect of varying the equalizer constant k (.7, .8, .9) on EC performance (R) for various prefilter BW on a 3.5 mile line with 2 bridged taps, using an equalizer and PLL $f_n = .50$, nonlinearity was absolute value	94
6.9	Effect of which nonlinearity (i.e., () ² or 11) on the EC performance (R , R_{margin}) for various prefilter BWs and PLL natural frequencies on a 3.5 mile line with 2 bridged taps	99

- 6.10 Effect of a mistuned prefilter (2 KHz) on the EC performance with and without equalization for a 20 KHz BW prefilter on a 3.5 mile line with 2 bridged taps 102
- B.1 Cable characteristics for 19 AWG PIC cable at 70°F B2
- B.2 Cable characteristics for 26 AWG PIC cable at 70°F B3

GLOSSARY OF TERMS

γ	step size parameter
Γ	propagation constant
τ_1	PLL time constant(s)
τ_2	PLL time constant(s)
ω_n	PLL natural frequency (radians)
$\Delta(n)$	jitter sample(s)
$\delta(n)$	difference in jitter samples (s)
ABCD	hybrid parameters
A/D	analog to digital
APLL	analog phase locked loop
AREC	adaptive reference echo canceller
BW	bandwidth
dB	decibels (log measurement)
DPLL	digital phase locked loop
DSL	digital subscriber loop
EC	echo canceller
f_c	center frequency of a filter
f_n	phase locked loop natural frequency (HZ)
FDM	frequency division multiplexing
FIFO	first-in-first-out buffer
FIR	finite impulse response
IIR	infinite impulse response
ISI	intersymbol interference
K	PLL loop gain parameter
LMS	least mean squares

MSE	mean square error
MSEJ	mean square error due to jitter
PLL	phase-locked loop
Q	narrowness factor of a filter
R	echo canceller performance factor
R_{margin}	performance margin = $R - R_{\text{req}}$
R_{req}	required R factor for good performance
$S(f)$	timing jitter power spectrum
T	one symbol time (s).
TCM	time compression multiplexing
VCO	voltage controlled oscillator
WDM	wave difference method
ZREC	zero reference echo canceller
$(\cdot)^2$	squaring function
$ \cdot $	absolute value function

CHAPTER 1

INTRODUCTION

1.1 Introduction

We are currently witnessing the evolution of the existing analog telephone network into an all digital one, called the integrated services digital network (ISDN). The transformation towards digital transmission and switching is not due to the recent discovery of the advantage of digital implementation, but instead due "primarily to the advances in solid state electronics" [1]. With the advent of such a network, numerous applications of digital data transmission in addition to simultaneous transmission of digitized speech will arise. Services for the home and office such as fire alarms, burglar alarms, remote utility reading, teletext, database communication capability etc. are anticipated [2]. Many of these services including digitized speech, raise the need for digital transmission to the subscribers premises [3].

Large scale application of digital transmission for full duplex transmission of parallel data and digitized speech depends on an efficient line transmission principle that can utilize the existing two-wire loop plant. This is due to the fact that a large part of the investment of the existing telephone system is in the

subscriber loops, which are twisted pairs of wires that connect a subscriber to the local telephone office.

1.2 The Loop Plant

The loop plant refers to the copper wire that connects subscribers to the local telephone switch (usually a two wire connection). With the enormous investment involved, the existing loop plant must be taken advantage of. We find, however, that it suffers from a number of imperfections including cable loss, crosstalk¹, bridged taps², impulse noise and assorted nonlinearities.

Figure 1.1 illustrates a simplified telephone connection. The two-wire sections at the ends are usually referred to as the subscriber loop where both directions of transmission are carried on a single wire pair. The middle section of the connection is a four-wire link, in which the two directions of transmission are transmitted on physically separate facilities.

A circuit known as a "hybrid" performs the two-wire to four-wire conversion. It presents a high loss from four-wire port to four-wire port, preventing oscillation and echos and a low loss from two-wire port to four-wire port thus reducing attenuation on the transmission path [1].

1 Crosstalk: is a form of interference between the signals of different cable pairs, which arises from electromagnetic coupling.

2 To be explained more completely later.

Figure 1.1: Illustration of a simple telephone connection

Figure 1.5: Worst case DSL relative signal levels

Figure 1.2: A typical subscriber line

Figure 1.3: A typical DSL arrangement showing sources of echo due to imperfect isolation in the hybrid and open ended bridged tap reflection

Note: Dashed lines are echo signal paths, solid lines are the intended direction of transmission

hybrids and the effects of bridged taps. Figure 1.3 illustrates the origin of different echo components found on DSL's. We will denote the subscriber's end of the DSL as the "near-end" and the central office end as the "far-end". This terminology will be assumed henceforth.

Bridged taps produce echos that degrade the performance of the system. The effect of the bridged tap is not necessarily related to the length of the loop it sits off, but more the length of the bridged tap itself. That is, a bridged tap of a critical length and appropriate gauge placed anywhere along the line may severely degrade its performance [4]. Ahamed also reported that with increasing loop length, performance deteriorated, since the far end signal gets increasingly attenuated and the reflected signal retains its strength [4]. The expected worst case length of a bridged tap is an odd multiple of one quarter of a wavelength (i.e., for 144 KHz it equals .4-.5 miles) [5].

The hybrid's capability of isolation is dependent on the accuracy of the balance network to the line. For practical implementation a single compromise balance network is employed that best matches the greatest number of loops.

1.3 Two-Wire Duplex Data Transmission Methods

Several transmission schemes for the DSL environment have been proposed. In general they can be divided into three main classes:

- (1) Frequency division multiplexing (FDM)
- (2) Time compression multiplexing (TCM) (also called ping pong or burst mode)
- (3) Echo cancellation (EC) (also called hybrid method)

The practical choice comes down to TCM or echo cancellation. FDM at the rates that we are considering (144Kbit/s), is not satisfactory. The high frequency channel is severely attenuated, thus limiting transmission range and the complexity of the modulation and demodulation schemes is considerable.

The transmission modes employing the TCM or EC approach both isolate the data received from the far-end, from the data transmitted at the near-end. The EC method achieves isolation by way of echo cancellation during full-duplex transmission. The TCM approach simplifies the separation of the two directions of transmission by isolating in time intervals for transmission and reception. That is each station alternately transmits and receives. Therefore the TCM approach requires a line rate on the DSL of at least twice the data rate. In fact the amount that it exceeds twice the data rate is dependent on:

- (1) Length of the block that is transmitted
- (2) The time for the pulse to travel along the line
- (3) The time during which transmission and reception are suspended to allow the line transients to subside [6]

The line rate for the EC approach is equal to the data rate. This is possible via continuous echo cancellation of the received signal, removing the contamination caused by reflections of the transmitted data.

Therefore the echo cancellation method requires the least bandwidth for a given data rate. This provides a significant advantage with respect to immunity from crosstalk and cable attenuation, which both increase as a function of frequency [7]. The benefit gained in bandwidth efficiency is at the penalty of greater complexity at the receiver since a dependable EC system hinges on the requirement for a high quality echo canceller circuit.

It is the echo cancellation approach that will be considered henceforth, with more indepth analysis being presented in Chapter 2.

1.4 Synchronous Versus Non-Synchronous Operation

The Echo Cancellation approach has been established as the method of duplex transmission as far as we are concerned. It is assumed that the far-end (i.e., Central office) transmitter receives its clock information from

the stable network clock, but the near-end or subscriber transmitter can generate its own clock or lock onto the far-end clock. This choice has major implications on the complexity and performance of the DSL.

The case of the near-end clock being slaved to the far-end clock (i.e., synchronous) has the advantages of a simple far-end interface to the network, simple clock recovery at the far-end, and recovery of the data directly from the echo cancelled signals. These benefits are not without cost. Because of the very close interaction between the clock recovery and echo cancellation (i.e., if there is not sufficient echo cancellation, clock recovery is impossible), a startup procedure will be necessary. In addition to this, independent of how clock recovery is performed, the recovered clock will contain some jitter, which will in turn affect the echo canceller [8].

If non-synchronous operation were employed, neither of these problems would exist. There would be no jitter to affect the echo canceller and the timing recovery circuit would be less complex. With this approach, however, the far-end connection to the network is more complex, and full time clock recovery is necessary at both ends and a clock generating circuit is required at the near-end [8].

Synchronous operation with the near-end slaved to the far-end clock will be assumed. In Chapter 3 we investigate the interaction between timing jitter from the

timing recovery subsystem at the near-end with the echo canceller.

1.5 Line Code

In the DSL environment, the characteristics of the transmission channel and the transmission equipment impose a number of restrictions on the signals that can be successfully transmitted. In general some kind of encoding is necessary to shape the spectrum of the transmitted signal in order to be compatible with the channel frequency response. The choice of line code employed should consider the following requirements:

- (1) high immunity to additive disturbances
- (2) small bandwidth frequency spectrum situated at low frequencies to minimize adverse influence of crosstalk, thermal noise, radio interference and cable loss.
- (3) No dc component: to allow minimum distortion through line transformers.
- (4) Sufficient timing information for clock recovery at the receiver.
- (5) A linear relationship between the coded and uncoded signals in case adaptive filtering is utilized [7].
- (6) Transparency, that is the line code must not impose any restrictions on the content of the transmitted messages.

The bipolar or alternate mark inversion (AMI) code meets

Chapter 2

ECHO CANCELLATION TECHNIQUES

2.1 Introduction

Echo cancellation has received a great deal of attention in the last few years as a viable technique in the Integrated Services Digital Network (ISDN) environment [9], [11], [12], [13], [14], [15], [16] and [17]. Using this method, there is no interruption in transmission or expansion of the required transmission bandwidth and bi-directional information transmission proceeds with little or no echo [18]. The basic echo canceller arrangement for two wire full duplex transmission is illustrated in Figure 2.1.

Adaptive echo cancellation is facilitated by automatically synthesizing a copy of the echo path impulse response, which is then convolved with the near-end signal creating an echo estimate which is subtracted from the received signal [18].

Two different techniques of implementing the echo canceller that have gained wide acceptance are the classical finite impulse response (FIR) or transversal filter structure [13], and the memory compensation principle [16]. The advantage of the memory compensation or table lookup canceller is that it can mitigate the effects of nonlinear distortion in the echo path and its

worst case condition the near-end echo signal may be 30-40 dB higher in level than the far-end data signal, assuming equivalent transmission levels at both ends. If a 20 dB signal to interference ratio is required for reliable data transmission, the echo canceller may be required to attenuate the residual echo signal entering the receiver by as much as 60-70 dB relative to its original level. Figure 1.5 illustrates the worst case relative signal levels.

The echo canceller's design is obviously critical, if this degree of performance is required. Also seeming small imperfections in such things as timing recovery could have very drastic effects in the echo canceller performance.

The specific problem under investigation here is, what are the requirements of the timing recovery subsystem and what is the effect of timing jitter from the timing recovery on the echo canceller's performance.

1.7 Outline of the Report

With the problem statement mentioned in the previous section in mind, we proceed with the study. Chapter 2 introduces echo cancellation techniques and provides detailed insight into the effect of timing jitter on the echo canceller.

Chapter 3 introduces timing recovery techniques and then discusses in detail the composite elements of the method that we are interested in.

Figure 1.4: Bipolar or AMI code frequency spectrum and coding state diagram [10]

Chapter 4 presents the simulation package that was used for this work. A number of comments regarding this package and simulation environments in general are then provided.

Chapter 5 discusses the specific details of the simulation of the digital subscriber loop. Different models are introduced that are used to simulate certain components of the system.

Chapter 6 presents the results of the simulations and discusses those results.

Chapter 7 gives some concluding remarks and suggestions for further work.

Figure 1.5: Worst case DSL relative signal levels

Chapter 2

ECHO CANCELLATION TECHNIQUES

2.1 Introduction

Echo cancellation has received a great deal of attention in the last few years as a viable technique in the Integrated Services Digital Network (ISDN) environment [9], [11], [12], [13], [14], [15], [16] and [17]. Using this method, there is no interruption in transmission or expansion of the required transmission bandwidth and bi-directional information transmission proceeds with little or no echo [18]. The basic echo canceller arrangement for two wire full duplex transmission is illustrated in Figure 2.1.

Adaptive echo cancellation is facilitated by automatically synthesizing a copy of the echo path impulse response, which is then convolved with the near-end signal creating an echo estimate which is subtracted from the received signal [18].

Two different techniques of implementing the echo canceller that have gained wide acceptance are the classical finite impulse response (FIR) or transversal filter structure [13], and the memory compensation principle [16]. The advantage of the memory compensation or table lookup canceller is that it can mitigate the effects of nonlinear distortion in the echo path and its

Figure 2.1: Basic echo canceller arrangement for two-wire full duplex transmission

ease of implementation using today's VLSI technology. Nonlinearities can arise from such things as asymmetry in the transmitted pulse, hybrid transformer saturation, nonlinearities in digital to analog conversion and vice versa or the effect of oxidation on the loop [17]. Basically, an independent output is assigned to each possible combination of N transmitted bits. Thus there is complete generality in the nature of the nonlinearities that it can compensate for. The problem with this method is that for large N , a very large memory is required, and its adaptation is slow [17].

Transversal filter-based echo cancellers have the advantage of faster coefficient adaptation and only a linear increase in complexity with increased filter length. Henceforth we will only consider the transversal filter echo cancellation method.

In order to investigate the influence of timing jitter on the echo canceller we present the following model of the DSL environment. Due to the stringent requirements placed upon the echo canceller and the potential overpowering of the far-end signal by the near-end signal, it is considered advantageous to have the signal fed to the echo canceller as a sampled analog waveform [12]. Digitization of the waveform would require very long A/D wordlengths. Therefore sampled analog waveforms will be assumed in the echo canceller signals.

2.2 Transmission Path and Echo Path Model of the DSL

The signal $r(t)$ is the received waveform at the output of a hybrid. It can be modelled as

$$r(t) = \sum_{k=-\infty}^{\infty} a(k) F(t-kT) + \sum_{k=-\infty}^{\infty} b(k) G(t-kT) + v(t) \quad (2.1)$$

where

$a(k) = \pm 1$: far-end binary data symbols

$b(k) = \pm 1$: near-end data symbols

$F(t)$: impulse response of the transmission path (from the far-end to the near-end) including coding

$G(t)$: impulse response of the echo path including coding

$v(t)$: additive noise of which two primary components include near-end crosstalk and impulse noise

The first term of (2.1) represents the far-end signal component, the second term represents the near-end signal and $v(t)$ is noise. For analytical reasons, the echo path impulse response is defined as including the effect of reflections from the far-end, bridged taps and line discontinuities.

When the received signal, $r(t)$ is sampled an integer M times per period, that is every T/M seconds, it is represented by

$$r(nT + \frac{mT}{M}) = \sum_{k=-\infty}^{\infty} a(k) F_{n-k}^{(m)} + \sum_{k=-\infty}^{\infty} b(k) G_{n-k}^{(m)} + v(n)^{(m)} \quad (2.2)$$

where $m = 0 \dots M-1$, representing the sampling phase and

$$F_{n-k}^{(m)} = F(nT + \frac{mT}{M} - kT)$$

$$G_{n-k}^{(m)} = G(nT + \frac{mT}{M} - kT)$$

$$v_n^{(m)} = v(nT + \frac{mT}{M})$$

Now for both analytical and practical reasons we assume that the transmission path impulse response $F(t)$ and the echo path impulse response $G(t)$ have finite durations, N_a and N_b symbol times respectively, that is $N_a T$ seconds and $N_b T$ seconds respectively given T is the symbol period in seconds. Thus the following vectors are defined, where the superscript "T" denotes transpose.

$$a(n)^T = (a(n), a(n-1), \dots, a(n-N_a+1)) \quad (2.3)$$

$$b(n)^T = (b(n), b(n-1), \dots, b(n-N_b+1)) \quad (2.4)$$

$$F^{(m)T} = (F_0^{(m)}, F_1^{(m)}, \dots, F_{N_a-1}^{(m)}) \quad (2.5)$$

$$G^{(m)T} = (G_0^{(m)}, G_1^{(m)}, \dots, G_{N_b-1}^{(m)}) \quad (2.6)$$

Rewriting (2.2) gives

$$r(nT + \frac{mT}{M}) = a(n)^T F^{(m)} + b(n)^T G^{(m)} + v(n)^{(m)} \quad (2.7)$$

It is the echo cancellers' function to remove the echo path component, (i.e., second term of (2.7)). Consider the echo canceller configuration depicted in

Figure 2.2 for $M=2$ phases. The echo canceller for each phase is called a subcanceller. Each subcanceller contains a transversal filter whose N_b long coefficient vector $\mathbf{g}(n)^{(m)}$ contains an estimate of the corresponding $\mathbf{Q}(n)^{(m)}$. The coefficient vector $\mathbf{g}(n)^{(m)}$ is adaptively updated using the error signal to a more accurate replica of $\mathbf{Q}(n)^{(m)}$.

The m^{th} subcanceller generates an output $b(n)^T \mathbf{g}(n)^{(m)}$ which is subtracted from the received signal $r(nT + \frac{mT}{M})$. The resultant sample $y(n)^{(m)}$ is

$$y(n)^{(m)} = r(nT + \frac{mT}{M}) - b(n)^T \mathbf{g}(n)^{(m)} \quad (2.8)$$

is fed to the receiver.

The most widely used practical algorithm for the coefficient adaptation of an echo canceller is the stochastic gradient algorithm, commonly referred to as Least Mean Squares (LMS) algorithm [9]. For this algorithm coefficient adaptation is governed by (2.9).

$$\mathbf{g}(n+1)^{(m)} = \mathbf{g}(n)^{(m)} + \gamma e(n)^{(m)} b(n) \quad (2.9)$$

where γ is a step-size parameter used to control the convergence of the algorithm. In general, making γ larger speeds convergence, while making γ smaller reduces the asymptotic error [9]. $e(n)$ is the "error" or difference between the desired output and the actual output.

Figure 2.2: Echo canceller structure for $M=2$ phases

Adaptive echo cancellers employing the above mentioned LMS adaptation algorithm may be subclassified into two categories. The classification is based on the choice of reference signal against which the cancelled signal output is compared for adaptation purposes. They are zero-reference schemes and nonzero-reference schemes.

Zero reference adaptation is based on the premise that if there is no far-end signal present, then the received signal, after it has been cancelled ($y(n)^{(m)}$ in Figure 2.2) should be zero. Therefore, the effect of having a far-end signal present that is uncorrelated with the near end signal is to act as noise for the adaptation algorithm [15]. In this case $e(n)$ in Figure 2.2 is simply equal to $y(n)^{(m)}$. Echo cancellers employing this method can be referred to as "zero-reference echo cancellers", (ZREC). Figure 2.3 illustrates a zero-reference adaptation unit for one tap of a subcanceller. In view of the fact that far-end signal presence degrades the adaption performance, γ must be kept small and thus speed of convergence is compromised. A complete ZREC for $M=4$ is presented in Figure 2.4.

Nonzero reference schemes, such as Adaptive Reference Echo Cancellation (AREC) proposed in [15] make use of receiver decisions on the far-end data to establish a reference signal. The estimate of the far-end signal is subtracted from the cancelled signal $y(n)^{(m)}$, to generate an error signal. This algorithm converges more rapidly than ZREC.

$$g(n+1)^{(m)} = g(n)^{(m)} + \gamma y(n)^{(m)} b(n)$$

$$\text{Echo estimate} = \sum g(n)^{(m)} b(n)$$

Figure 2.3: Illustration of zero reference adaptation algorithm

$i(n)$ represents the difference in jittered sampling instants, 1 samples apart. It is assumed that the jitter in the received far-end signal is negligible.

In order to evaluate the effect of timing jitter on the performance of the echo canceller let us define a vector $U(n)^{(m)}$ which we call the coefficient error vector. This vector represents the difference between impulse response of echo channel and the echo canceller's estimate of its impulse response

$$U(n)^{(m)} = Q(n)^{(m)} - g(n)^{(m)} \quad (2.14)$$

We define Residual Echo, as that component of the signal that is presented to the local receiver, after echo cancellation, that is due to a non-ideal echo canceller.

$$b(n)^T U(n)^{(m)} = y(n)^{(m)} - a(n)^T F(n)^{(m)} - v(n)^{(m)} \quad (2.15)$$

if the echo canceller length equals or exceeds the duration of the channel's impulse response.

It can be seen intuitively that under worst case conditions, where the near-end signal is as much as 40 dB stronger than the far-end signal, even very small differences between $g(n)^{(m)}$ and $Q(n)^{(m)}$, (i.e., $U(n)^{(m)}$) may produce significant degradation in receiver performance. Consider this very rough analysis [20] in which 60 dB is required for "degree of echo cancellation" where

Figure 2.4: Zero reference echo canceller arrangement for $M=4$ phases

Figure 2.5: General full duplex 2-wire echo canceller configuration at the subscriber end

$m=0 \dots M-1$ is

$$t_k^{(m)} = kT + \frac{mT}{M} + \Delta(k)$$

Figure 2.6 illustrates an exaggerated effect of timing jitter. As this clock is used to launch the near end data, the echo signal $S_e(t)$ is given as

$$S_e(t) = \sum_{k=-\infty}^{\infty} b_k G(t - kT - \Delta(k))$$

Using the same clock to sample the echo signal at the n^{th} sample of the m^{th} phase, i.e.,

$$t_n^{(m)} = nT + \frac{mT}{M} + \Delta(n)$$

we get

$$S_e(t_n^{(m)}) = \sum_{k=-\infty}^{\infty} b_k G((n-k)T + \frac{mT}{M} + \Delta(n) - \Delta(k))$$

Although differences in adjacent $\Delta(n)$'s are quite small, $\Delta(n)$'s separated by a number of samples in time may have much larger differences (i.e., $\Delta(n) - \Delta(k)$).

Assuming a stable far end clock the received far-end signal $S_f(t)$ is

$$S_f(t) = \sum_{k=-\infty}^{\infty} a(k) F(t - kT)$$

Again, using the local or near-end jittered clock to sample the far end signal at the n^{th} sample and m^{th} phase gives

Jittered sampling times $t_n^{(m)} = nT + \frac{nT}{M} + \Delta_n$

Figure 2.6: Exaggerated timing jitter example

$$S_f(t^{(m)}) = \sum_{k=-\infty}^{\infty} a_k F((n-k)T + \frac{mT}{M} + \Delta(n))$$

In this case, the effect of using a jittered clock to sample a signal generated by a stable clock is equivalent to using a stable sampling clock and having a channel whose impulse response has been shifted in time by $\Delta(n)$. If $\Delta(n) \ll T$ there will be a minimal effect on the recovered far-end signal.

If (2.7) is rewritten accounting for timing jitter, then the received waveform may be expressed as

$$r(t_n^{(m)}) = r(n)^{(m)} = a(n)^T F(n)^{(m)} + b(n)^T G(n)^{(m)} + y(n)^{(m)} \quad (2.10)$$

where $a(n)^T$ and $b(n)^T$ are defined as in (2.3) and (2.4) respectively. $F(n)^{(m)}$ and $G(n)^{(m)}$, however, are now different. The jittered or time varying impulse responses are defined as

$$F(n)^{(m)T} = [F(\frac{mT}{M} + \Delta(n)), F(T + \frac{mT}{M} + \Delta(n)), \dots, F((N_a-1)T + \frac{mT}{M} + \Delta(n))] \quad (2.11)$$

and

$$G(n)^{(m)T} = [G(\frac{mT}{M} + \delta_0(n)), G(T + \frac{mT}{M} + \delta_1(n)), \dots, G((N_b-1)T + \frac{mT}{M} + \delta_{N_b-1}(n))] \quad (2.12)$$

where $\delta_1(n) = \Delta(n) - \Delta(n-1)$

$$v(n)^{(m)} = v(nT + \frac{mT}{M} + \Delta(n))$$

$i(n)$ represents the difference in jittered sampling instants, 1 samples apart. It is assumed that the jitter in the received far-end signal is negligible.

In order to evaluate the effect of timing jitter on the performance of the echo canceller let us define a vector $\underline{U}(n)^{(m)}$ which we call the coefficient error vector. This vector represents the difference between impulse response of echo channel and the echo canceller's estimate of its impulse response

$$\underline{U}(n)^{(m)} = \underline{Q}(n)^{(m)} - \underline{g}(n)^{(m)} \quad (2.14)$$

We define Residual Echo, as that component of the signal that is presented to the local receiver, after echo cancellation, that is due to a non-ideal echo canceller.

$$\underline{b}(n)^T \underline{U}(n)^{(m)} = y(n)^{(m)} - \underline{a}(n)^T \underline{F}(n)^{(m)} - v(n)^{(m)} \quad (2.15)$$

if the echo canceller length equals or exceeds the duration of the channel's impulse response.

It can be seen intuitively that under worst case conditions, where the near-end signal is as much as 40 dB stronger than the far-end signal, even very small differences between $\underline{g}(n)^{(m)}$ and $\underline{Q}(n)^{(m)}$, (i.e., $\underline{U}(n)^{(m)}$) may produce significant degradation in receiver performance. Consider this very rough analysis [20] in which 60 dB is required for "degree of echo cancellation" where

$$\text{degree of echo cancellation} = \frac{|\text{echo before cancellation}|}{|\text{echo after cancellation}|}$$

if we can make the approximation that [20]

$$\text{degree of echo cancellation} = \frac{|\delta_n|^{-1}}{T}$$

where $\delta_n = \Delta_n - \Delta_{n-1}$, then for a 60 dB requirement, δ_n should be less than $10^{-3}T$, i.e.,

$$\text{for } T = \frac{1}{144 \text{ KHz}}, \delta_n < 6.94 \times 10^{-9} \text{ seconds} \quad [20]$$

The effect of timing jitter on the performance of the echo canceller has been evaluated analytically in [20]. The residual steady state mean squared error for the m th sampling phase for the ZREC is

$$\text{MSE}(m) = \frac{1}{2-\gamma N_b} A_G^{(m)} + \frac{\gamma N_b (\sigma^2 + |F^{(m)}|^2)}{2 - \gamma N_b} \quad (2.16)$$

where

$$A_G^{(m)} = \int_{-1/2T}^{1/2T} A(f) B(f)^{(m)} S_{\Delta}(f) df$$

$$A(f) = \frac{8(2-\gamma) \sin^2 \pi f T}{\gamma^2 + 4(1-\gamma) \sin^2 \pi f T}$$

$$B(f)^{(m)} = 2T^2 \sum_{i=1}^{N_b-1} G'(iT + \frac{mT}{M})^2 \sin^2 \pi i f T$$

where $G'(t)$ is derivative of $G(t)$ with respect to time

$S_{\Delta}(f)$ is the power spectral density of the jitter sequence $\{\Delta(n)\}$

$$\text{defined as } S_{\Delta}(f) = \sum_k \langle \Delta_n \Delta_{n+k} \rangle \exp(-j2\pi f k T)$$

The first term of (2.16) is the contribution of timing jitter to the residual mean squared error. The second term is the contribution due to the effect of noise and the far end signal on the ZREC coefficient adaptation algorithm. We are particularly interested in the first term, that due to jitter

$$\text{MSEJ}(m) = \frac{1}{2 - \gamma N_b} A_G^{(m)}$$

Evaluation of this result in [19] indicates that MSEJ is "especially sensitive to the high-frequency part of the jitter spectrum". This observation pointed to preferable use of at least a second order phase-locked loop.

Chapter 3

TIMING RECOVERY

3.1 Background

Timing recovery is one of the most critical receiver tasks in the digital subscriber loop environment, where the subscriber or near-end transmitter clock is slaved to the central office or far end clock. If the echo canceller method is used, the performance of the timing recovery system, in terms of timing jitter will have significant influence on the performance of the echo canceller, as is shown in Chapter 2.

In recent years timing recovery techniques for the DSL environment have received increased attention [21], [19], [5], [8]. The received waveform at the output of the echo canceller is passed through an analog interpolation filter. This analog waveform is fed to the timing recovery subsystem as shown in Figure 3.1. We are interested in the level of complexity and the required parameter accuracy for each element of this subsystem, such that an overall satisfactory performance is realized in terms of certain criteria. These criteria may include the degree of echo cancellation achieved, acceptable timing jitter spectral content or bit error rate at the receiver output. This study is primarily interested in the effect of various timing recovery schemes on the

Figure 3.1: Timing recovery placement in DSL receiver.

amount of timing jitter present at their output and its corresponding effect on the echo canceller. Therefore a closer inspection of each element of the timing recovery subsystem in terms of its functionality and implementation would be required.

The timing recovery problem associated with the DSL involves the optimization of the sampling instants of the received data signal by continually adjusting the phase and frequency of the receiver clock to compensate for frequency drifts between the near-end clock and the far end clock [21].

"Data-derived" or "self-timed" synchronization techniques are of prime interest in the DSL environment for bandwidth efficiency reasons. Here the timing frequency and phase are derived directly from the information bearing signal instead of using a separate sub-channel to send synchronization information [22]. These techniques operate by presenting the received data signal to a timing circuit that generates a "timing wave". This timing wave should contain some periodic attribute, such as uniformly spaced zero crossings from which optimized sampling instants may be determined [23]. There are two broad classes that these techniques may be divided into, namely continuous time and discrete time techniques. The more common continuous time methods include the Spectral line [24], Threshold Crossing [25], Sampled Derivative [25], and the Maximum Likelihood Estimation [25] methods. The discrete time class include such

methods as Wave Difference Method (WDM) [26] and Baud-Rate Sampling [21], [5].

The commonly used Spectral line technique is of particular interest. A nonlinear device is employed to generate a discrete clock component from the received signal. This discrete component is then selected using a band pass filter or phase-locked loop (PLL) [9].

Figure 3.2 gives a block diagram of a data-derived timing recovery circuit [28]. The received signal $r(t)$ is first filtered by the prefilter to shape the received data pulses to improve the timing circuit performance. The output of the prefilter is then fed to a nonlinear device which generates a discrete frequency component at $f=1/T$ where T is the symbol duration. The nonlinear device could be squarer, full wave rectifier (absolute value) or delay and multiplication function for instance. The output of the nonlinear device is presented to the band pass filter or PLL which selects the discrete component at $f=1/T$ and attempts to reject all other frequency components (i.e., data related continuous spectrum components) [28]. These functions are illustrated in Figure 3.2 with their associated spectrums. It is the residual continuous spectrum which passes in the bandwidth of the band pass filter or PLL that is the primary source of clock timing jitter [28].

Figure 3.2: Frequency spectrum of signal at various elements of timing recovery [28]

squarer versus an absolute value device.

3.5 The Post Filter

The nature of this filter is to reduce the noise bandwidth before the signal is fed to the PLL. The higher order harmonics generated by the nonlinearity are also filtered out. The effect of the presence of this filter and its bandwidth and Q with respect to timing jitter requires investigation.

3.6 The Phase-Locked Loop

It is a function of the PLL to generate a timing wave locked to the best estimate of the transition times [32].

A PLL consists of three basic elements (see Figure 3.6):

- (1) A Phase Detector
- (2) A Loop Filter
- (3) A Voltage Controlled Oscillator (VCO)

The phase detector compares the phases of the input signal and the output of the VCO, and generates a signal that is proportional to the phase difference between its inputs. The loop filter filters the phase detector output and presents a control voltage to the VCO. This control voltage changes the frequency of the VCO output in a direction such that the phase difference between the phase detector inputs is reduced [32]. The PLL is said to be in "lock" when the frequency of the VCO output is equal to the average frequency of the input signal [33].

Figure 3.3: Required symmetry conditions for reduced timing jitter

In practice, ideal symmetry conditions cannot be met and some degree of self-noise will be present. Self-noise is classified as a cyclo-stationary process. A random process is cyclo-stationary if the mean and autocorrelation are both periodic in time [23]. If self-noise is resolved into its inphase and quadrature components, it is the spectrum of the quadrature component that contributes to the phase jitter if the steady-state phase offset is zero. The quadrature component is dependent upon the shape of the signal entering the nonlinearity and the kind of nonlinearity used.

Figure 3.4 illustrates a general DSL timing recovery configuration. In the most general sense the timing recovery subsystem consists of an equalizer, a prefilter, a nonlinear device, a post band pass filter and a PLL.

3.2 The Equalizer

The signal that is presented to the equalizer is a signal created by the far-end transmitter that has passed through a nonideal channel. The channel introduces intersymbol interference (ISI) in the waveform. It is the function of the equalizer to remove the ISI component of the received waveform. The attenuation versus frequency for some common gauges of paired cable are shown in Figure 3.5 [30]. For this application we are interested in keeping the complexity to a minimum.

Figure 3.4: General DSL timing recovery configuration employing a PLL [19]

Figure 3.5: Attenuation versus frequency for typical cable gauges

3.3 The Prefilter

The importance of prefiltering for symmetry reasons was outlined earlier. The ideal prefilter should be such that the frequency response at its output is symmetric about $1/2T$. In practice, such a filter is difficult to design and the detrimental effects of the following prefilter characteristics and combinations of, should be evaluated.

- (1) The tradeoffs associated with using prefilters of various Q 's. The higher the Q of the filter, the more complex the design of the filter and VLSI implementation and the more sensitive is the system to mistuning.
- (2) The effect of various degrees of mistuning from the ideal $f=1/2T$ center frequency.
- (3) The effect of asymmetry in the prefilter design itself.

For a bit rate of 144 kbps, the prefilter should be centered at 72 KHz.

3.4 The Nonlinear Device

A number of nonlinear functions exist that are easily implementable. For example a full wave rectifier (absolute value), a squarer $((\cdot)^2)$ or a fourth power function $((\cdot)^4)$ would suffice. It has been reported that some advantage may be gained by using a full wave rectifier or absolute value device when the PLL noise bandwidth is large [28]. Therefore, the nonlinear device parameter that should be investigated is the effect of a

squarer versus an absolute value device.

3.5 The Post Filter

The nature of this filter is to reduce the noise bandwidth before the signal is fed to the PLL. The higher order harmonics generated by the nonlinearity are also filtered out. The effect of the presence of this filter and its bandwidth and Q with respect to timing jitter requires investigation.

3.6 The Phase-Locked Loop

It is a function of the PLL to generate a timing wave locked to the best estimate of the transition times [32]. A PLL consists of three basic elements (see Figure 3.6):

- (1) A Phase Detector
- (2) A Loop Filter
- (3) A Voltage Controlled Oscillator (VCO)

The phase detector compares the phases of the input signal and the output of the VCO, and generates a signal that is proportional to the phase difference between its inputs. The loop filter filters the phase detector output and presents a control voltage to the VCO. This control voltage changes the frequency of the VCO output in a direction such that the phase difference between the phase detector inputs is reduced [32]. The PLL is said to be in "lock" when the frequency of the VCO output is equal to the average frequency of the input signal [33].

There are a number of different kinds of PLL's such as digital-analog or hybrid PLL, discrete PLL, Digital PLL or DPLL and the analog PLL (APLL). Considering the frequencies of operation that we require and the emphasis on small timing jitter we will only consider the analog PLL.

Figure 3.7 illustrates the basic analog PLL. Phase detectors are often modelled as multipliers, both because of analytical ease and many practical phase detectors make good approximations to multipliers [32]. Neglecting noise in the input signal, the APLL of Figure 3.7 can be modelled as in Figure 3.8. When the phase error $\theta_1(t) - \theta_0(t)$ is small, then the approximation

$$\sin(\theta_1(t) - \theta_0(t)) \approx \theta_1(t) - \theta_0(t)$$

can be made.

Assuming that the PLL is in lock, with the input phase equal to $\theta_1(t)$ and the VCO output signal phase $\theta_0(t)$, then the output of the phase detector is

$$v_d = k_d (\theta_1 - \theta_0)$$

where k_d = phase detector gain factor (volts/radian). The loop filter with transfer function $F(s)$ removes high frequency components and noise from the phase error voltage v_d . The output from the loop filter is the VCO control voltage v_c . Letting k_o be the "VCO gain factor"

Figure 3.6: Phase locked loop components

Figure 3.7: Basic analog PLL model

Figure 3.8: Linearized analog PLL model [32]

```

/*****
SUMMER
*****/
*
*
*      purpose: this star sums the samples from two
*               input fifos and outputs the result to the
*               output fifo.
*
*      input  0  x  samples
*      input  1  y  samples
*      output 0  z  samples
*
*      model :   z = x + y
*
*      paramaters : none
*****/

no_input_fifos 2
no_output_fifos 1

declarations
    SAMPLE x,y,z ;

end

main_code
    while( length_input_fifo(0) > 0 &&
           length_input_fifo(1) > 0 &&
           length_output_fifo(0) < maxlength_output_fifo(0) ){
        get(0,&x) ;
        get(1,&y) ;
        z = x + y ;
        put( 0,z ) ;
    }
    return(0) ;
end

```

Figure 4.2: BLOSIM code for a simple example star

In practice, PLL's of second order are much more common [32]. They can track both frequency and phase errors.

For the purposes of this study, a first order passive lag-lead filter is chosen as a loop filter. The transfer function $F(s)$ is

$$F(s) = \frac{s\tau_2 + 1}{s\tau_1 + 1}$$

where τ_1 and τ_2 are time constants ($\tau_1 > \tau_2$) [32].

The characteristics for the PLL employing this loop filter are

$$\text{Open loop transfer function } G(s) = \frac{k_o k_d \tau_2}{1 + s\tau_1} \left(1 + \frac{1}{s\tau_2}\right) \quad (4.1)$$

$$\text{Loop gain } k = \frac{k_o k_d \tau_2}{\tau_1} \quad (4.2)$$

$$\text{Natural frequency } \omega_n = \sqrt{\frac{k}{\tau_2}} \quad (4.3)$$

$$\text{Damping factor } \delta = \frac{1}{2} \sqrt{k \tau_2} \left(1 + \frac{k}{\tau_1}\right) \quad (4.4)$$

[32]

This PLL may be characterized by three parameters τ_1 , τ_2 and k or by two parameters ω_n and δ . PLL's of second order are often compared in terms of the two parameters, natural frequency and damping factor.

A commonly used value of the damping factor is $\delta = .707$ which is in the region $0.25 < \delta < 1.0$ where the noise bandwidth of the PLL is within 1 dB of the minimum. Therefore, by fixing $\delta = .707$, and adjusting τ_1 , τ_2 and k , it is possible to acquire a PLL for a particular desired value of natural frequency ω_n .

CHAPTER 4

BLOSIM

4.1 Blosim

The simulations carried out for this work were run on a VAX 750 under the UNIX operating system. The vehicle for them, is a simulation package called "BLOSIM" [34]. This package was written at the University of California, Berkeley by Dr. D. Messerschmitt. "BLOSIM" is derived from "BLOCK SIMULATOR".

BLOSIM is a time driven simulator that can support different sampling rates within the same simulation. As the name implies, this package is for simulation of a modular system. BLOSIM itself does not include blocks of any particular function. The blocks are provided by the user, thus maintaining complete generality.

An important criterion behind the BLOSIM design is portability of user blocks among other users. That is, it is designed to encourage the user to build a library of routines that can be used by other users. The "block" aspect creates an inherently structured environment for programming simulations. Module or block portability is encouraged by enforcing a single carefully defined interface between blocks.

One of the most important design features of BLOSIM is this common interface by which modules of the

simulation pass data samples to other blocks. The data structure provided for this task is "first-in, first-out" buffers or FIFO's as they will be called from now on. BLOSIM provides complete support and management of the FIFO's and provides two input/output routines for user blocks use, `get( )` and `put( )`, respectively.

To maintain module interconnection independence BLOSIM enforces the design procedure whereby the user provides the user blocks and then in a separate routine, a specification of the topology of interconnection of the blocks. This topology file specifies the connection of input and output FIFO's between blocks and the expected input parameters for the blocks.

BLOSIM supports a hierarchical definition of blocks. The "STAR" is the most elementary block. Next, a "GALAXY" is a block that may be composed of STAR's and GALAXY's. The "UNIVERSE" is the complete simulation and consists of an interconnection of STAR's and GALAXY's. The importance in having GALAXY's is two-fold. The first is for clarity in the overall topology in the UNIVERSE since details of certain blocks may be hidden in a GALAXY definition. The second purpose is for optimization, when feedback is required.

BLOSIM supports multiple instances of the same block, where specialization can take place via the input parameters. In association with this feature, BLOSIM supports the concept of state variables. State variables

are those variables that maintain their value between calls to the block in which they are defined. The state variable data structure in BLOSIM has been designed so that multiple instances of the same block using state variables is no problem.

To provide more insight into BLOSIM, consider the simulation of a simple structure as in Figure 4.1. This example has no practical value, but, it will shed some light on BLOSIM. Figure 4.1 specifies the block diagram of a UNIVERSE. Each of the elementary blocks (DATA-GENERATOR, SUMMER, FORK and PRINT-BLOCK) are considered STARS on their own. We can define a GALAXY indicated by the contents of the dotted box of the same figure (i.e., SUMMER, FORK). The code for a sample block such as SUMMER which adds the two samples appearing on the two input FIFOs is given in Figure 4.2. The GALAXY called "SUM-KEEP" is defined in Figure 4.3 which specifies its topology. In general, for efficiency reasons modules that are part of a feedback system should be part of a GALAXY. The complete UNIVERSE describing Figure 4.1 is given in Figure 4.4.

For debugging purposes, BLOSIM provides a "NARRATIVE" mode which interactively indicates what BLOSIM is doing as the simulation progresses. Figure 4.5 is the NARRATIVE mode output for this example.

Figure 4.1: Illustration of a simple example simulation structure (i.e., the UNIVERSE)

It seems that this is not sufficient. It is thought that a more interactive debugging tool should be available with a feature such as FIFO inspection where the contents of a selected FIFO may be examined.

BLOSIM makes use of a preprocessor that parses a user's block to generate an internal BLOSIM format. This preprocessor, for example, will generate useless code leading to simulation termination if it detects one of a number of undocumented key words such as "end" in a comment statement. This type of error is very difficult to isolate in BLOSIM and its effects are catastrophic. Better documentation and tighter more critically defined parsing is required in this case.

BLOSIM restricts the data type of the samples that are passed between blocks to the "float" type. In many cases, either for accuracy reasons or generality such data types as "double" and a user defined complex data structure would be beneficial.

A major problem in BLOSIM arose as the size of the simulation increased. A totally unpredictable problem occurred that manifested itself in the form of "bus errors" which lead to program termination. Upon substantial investigation the cause remained unknown, but a temporary solution was found. A call to a UNIX routine that dynamically allocates storage (i.e., CALLOC) was changed to allocate blocks of a large array that was requested at initialization time of the simulation. The


```

# GALAXY sum_keep
#
#
star    summer                summer.c
#
star    fork                  fork.c
#
param   int    1
star    delay                delay.c
#
#
connect  input    0  summer    0    1
connect  summer   0  fork      0    1
connect  fork     0  output    0    1
connect  fork     1  delay     0    1
connect  delay    0  summer    1    1
#

```

Figure 4.3: BLOSIM code for topology definition of a simple example Galaxy

```

## EXAMPLE UNIVERSE
# -data samples are transmitted to GALAXY keep_sum then to
#   print_block to print current sum.
#
param   int      5
star    data_gen  data_gen.c
#
galaxy   keep_sum  sum_keep.t
#
param   int      1
star    print_block  print_lf.c
#
#
connect data_gen 0 keep_sum 0 3
connect keep_sum 0 print_block 0 3
#

```

Figure 4.4: BLOSIM code for topology definition
of a complete UNIVERSE

```

Created: Star data_gen
Created: Galaxy keep_sum
Created: Star print_block
Created: Star keep_sum/summer
Created: Star keep_sum/fork
Created: Star keep_sum/delay
Connection: keep_sum/fork #0 to print_block #0
Connection: keep_sum/delay #0 to keep_sum/summer #1
Connection: keep_sum/fork #1 to keep_sum/delay #0
Connection: keep_sum/summer #0 to keep_sum/fork #0
Connection: data_gen #0 to keep_sum/summer #0
Scheduled: Star data_gen
Scheduled: Galaxy keep_sum
Scheduled: Star keep_sum/delay (forced)
Scheduled: Star keep_sum/summer
Scheduled: Star keep_sum/fork
Scheduled: Star print_block
Execution: Star data_gen
Execution: Galaxy keep_sum
Execution: Star keep_sum/delay
Execution: Star keep_sum/summer
Execution: Star keep_sum/fork
Execution: Star keep_sum/delay
Execution: Star keep_sum/summer
Execution: Star keep_sum/fork
Execution: Star keep_sum/delay
Execution: Star keep_sum/summer
Execution: Star keep_sum/fork
Execution: Star keep_sum/delay
Execution: Star keep_sum/summer
Execution: Star keep_sum/fork
Execution: Star keep_sum/delay
Execution: Star keep_sum/summer
Execution: Star keep_sum/fork
Execution: Star print_block
0.000000
0.000000
0.000000
Execution: Star data_gen
Execution: Galaxy keep_sum
Execution: Star keep_sum/delay
Execution: Star keep_sum/summer
Execution: Star keep_sum/fork
Execution: Star keep_sum/delay
Execution: Star keep_sum/summer
Execution: Star keep_sum/fork
Execution: Star print_block
0.000000
1.000000
1.000000
Execution: Star data_gen
Execution: Galaxy keep_sum
Execution: Star keep_sum/delay
Execution: Star keep_sum/summer
Execution: Star keep_sum/fork
Execution: Star print_block

```

Figure 4.8: Narrative mode output for example universe

4.2. The BLOSIM Experience

In many respects BLOSIM provides a good simulation environment. Modularity, which is encouraged, leads to a simulation that is partitioned in a manner similar to its practical implementation. Further to this, smaller, more functionally specific modules tend to be more reusable thus making a library of routines more realizable.

BLOSIM is written in C, and user blocks are written using the same. Measurements of BLOSIM system overhead during simulation indicate that it is very small. Memory allocation for FIFO's and state variables within blocks is allocated dynamically, therefore storage is not wasted. Because parameters are set and changed in the topology file UNIVERSE.T, it is no effort to change the parameters and rerun the new simulation.

BLOSIM, despite its well thought-out design, is not free of short comings. The startup time for a BLOSIM neophyte, that is the time before sufficient understanding of the system is gained for the user to be able to do useful work seems excessively long. The reasons for this are two-fold: unsatisfactory documentation with problem scenarios, and a fault tolerance threshold, before system crash that is too low. Obviously during a user's initial orientation a number of mistakes will be made. At this time the user looks towards the available debugging tools. The only debug tools available with BLOSIM are the aforementioned NARRATIVE mode and a TRACE mode that prints the sample values being passed between selected blocks.

It seems that this is not sufficient. It is thought that a more interactive debugging tool should be available with a feature such as FIFO inspection where the contents of a selected FIFO may be examined.

BLOSIM makes use of a preprocessor that parses a user's block to generate an internal BLOSIM format. This preprocessor, for example, will generate useless code leading to simulation termination if it detects one of a number of undocumented key words such as "end" in a comment statement. This type of error is very difficult to isolate in BLOSIM and its effects are catastrophic. Better documentation and tighter more critically defined parsing is required in this case.

BLOSIM restricts the data type of the samples that are passed between blocks to the "float" type. In many cases, either for accuracy reasons or generality such data types as "double" and a user defined complex data structure would be beneficial.

A major problem in BLOSIM arose as the size of the simulation increased. A totally unpredictable problem occurred that manifested itself in the form of "bus errors" which lead to program termination. Upon substantial investigation the cause remained unknown, but a temporary solution was found. A call to a UNIX routine that dynamically allocates storage (i.e., CALLOC) was changed to allocate blocks of a large array that was requested at initialization time of the simulation. The

problem cause remains a mystery but it seems that it has been circumvented. The use of dynamic allocation in the UNIX system is suspect.

BLOSIM provides a facility called "STATE" that enables a block to have access to a number of previous samples on the input and output FIFO's. This has use for such applications as transversal filters, where the user block need not store a certain number of past samples but need only make a call to a BLOSIM routine. User experience indicates that this facility has some major limitations and design flaws. It was decided not to use this "feature" and the affected blocks were rewritten with the added necessary complexity.

The control of scheduling of the order of block execution is in the hands of BLOSIM. In general the optimal order is chosen, however in a closed loop feedback configuration, a block is virtually chosen at random. A more deterministic start-up procedure is useful in these circumstances, where a selected block may somehow be flagged as the one to start execution.

4.3 Conclusion

The experience with BLOSIM indicates the complexity of the design of a simulation environment. The approach taken in BLOSIM is superior to the standard Fortran simulation alternative, yet there is still room for improvement. We anticipate that the next iteration of BLOSIM will more than account for its predecessor's inadequacies.

CHAPTER 5

SIMULATION AND PERFORMANCE EVALUATION

5.1 Background

The simulations of the Digital Subscriber Loop were performed using the simulation environment (i.e., BLOSIM) described in the previous chapter. The configuration is illustrated in Figure 5.1. In these simulations there is no interest in the received data with respect to a bit error rate, but instead on the quality of the timing recovery in terms of the jitter spectrum and its effect on the echo canceller. The system was simulated entirely in the time domain assuming a bit rate of 144Kbps (equals also the baud rate) and four samples per baud.

5.2 Near and Far End Transmitters

The signal at point X in Figure 5.1 is the signal that is presented to the near-end receiver at the receiver side of the hybrid. The blocks in Figure 5.1 can be subdivided. Figure 5.2 indicates the structure of both the near and far-end transmitters. The difference between the far-end and near-end is the use of different seeds in the data generators (i.e., for uncorrelated data) and the presence of a fork in the near-end transmitter so that the raw binary data bits can be fed to the echo canceller for coefficient adaptation. The BIPOLAR-CODER block codes the

Circuitry Present
At Central Office

Represents the DSL Channel (i.e., Effects
of Hybrid, Bridged Taps, etc.)

Circuitry Present In Near-End Receiver

Figure 5.1: DSL simulation block diagram

Figure 5.2: Transmitter simulation structure

Figure 5.3: General timing recovery configuration

incoming random data bits into a bipolar code. The coded samples are then fed to the TRANSMIT_FILTER block where four samples are generated (i.e., four phases) for each incoming bit and the requested duty cycle (50% for most cases) is given to the output data.

5.3 The DSL Channel

Simulation of the transmission channel and echo channel make use of impulse responses of lines of interest that are calculated using a program developed on the CP6 Honeywell mainframe computer. This program makes use of the theory presented in Appendix A to simulate subscriber loops that are composed of sections of cable varying in gauge, temperature and length, and including the effects of bridged taps, the hybrid and its balancing network parameters. The characteristics of the cable types used (i.e., type PIC 19 gauge @ 70°F and type PIC 26 gauge at 70°F), which are needed for the subscriber line simulation are given in Appendix B. Using the above mentioned theory and data, the program generates the impulse response and derivative of the impulse response for the transmission and echo paths of the desired line configuration.

Examples of some impulse responses for the transmit channel and the echo channel are shown in Appendix E.

The TRANSMIT_CHANNEL was simulated with an eighty tap transversal filter, equivalent to twenty symbol periods with four samples per symbol time. This channel has no effects of jitter simulated on it.

The ECHO_CHANNEL simulation is more complex in order to account for the effect of a jittered clock launching the near-end data bits which are then sampled later using the same clock. This channel response, is truncated using an eighty tap transversal filter also. The tap coefficients, or samples of the impulse response, however, are derived by equation (5.1) per sample time.

$$Y(k) = \sum_{n=0}^{N_b-1} [X[n] * (\text{imp-resp}[n] + (\Delta(k) - \Delta(n)) * \text{deriv-imp-resp}[n])] \quad (5.1)$$

where

$$N_b = 80$$

$X[n]$ = near-end binary data bits

$Y(k)$ = channel output

$\text{imp-resp}[n]$ = nth sample of impulse response of echo channel, it is a vector of length 80

$\text{deriv-imp-resp}[n]$ = nth sample of time-derivative of impulse response of echo channel also a vector of length 80

$\Delta(n)$ = vector composed of last eighty jitter samples coming from the timing recovery. These times are the result of phase (radians) to time (sec) transformations of the phase modifications that take place in the VCO of the PLL.

The NOISE block generates white Gaussian noise samples of zero mean and desired variance. This is simulated using the sum of twelve uniformly distributed random variables.

This method approximates the Gaussian distribution well, except at the extreme ends.

5.4 Echo Cancellor

The echo canceller that is simulated is of the form of Figure 2.4. This four phase configuration employs four parallel transversal filters, twenty taps long. The tap coefficient adaptation uses the ZREC algorithm given by equation (2.9).

In general, the results reported in this work were obtained by establishing a "good" set of coefficients first and then using these converged values. In essence this is a simulated startup procedure. These coefficients were obtained by removing the interfering far-end signal, noise and jitter components and increasing γ for fast convergence. Using the results of [14] for the ZREC case, for the above conditions it can be shown that for $\gamma < \frac{1}{\text{Number of Taps}}$, a very low steady state mean squared error is achieved with as few as one thousand iterations. Once converged coefficients were generated then the complete simulation could be performed.

5.5 Timing Recovery System

The timing recovery system in the most general configuration is illustrated in Figure 5.3. The PLL has been described in detail in Chapter 3. The transformation of the analog VCO and the loop filter to digital form is

shown in Appendix C.

The nonlinear block was limited to the two most common functions, the squarer and absolute value.

The equalizer employed in these simulations is a simple fixed equalizer given by:

$$y(n+1) = x(n) - \text{eta} \cdot x(n-1)$$

where:

$X(\cdot)$: samples of signal to be equalized

$Y(\cdot)$: equalized signal

eta: constant = 0.8 for these simulations

The bandpass filters used in these simulations are of the Infinite Impulse Response (IIR) variety. The advantage of IIR filters over FIR filters from the simulation point of view is the reduced execution time. The model for the filter used is given by:

$$y(n) = y(n-1)2r\cos(2\pi f_0 T) + x(n) - r^2 y(n-2)$$

where:

r: determines the Q of the filter

f_0 : center frequency

$x(n)$: input to be filtered

$y(n)$: filtered output

These filters were cascaded in pairs for superior response characteristics. Appendix D indicates plots of these filters with varying r factors.

Figure 5.4: Complete DSL simulation block diagram without showing performance evaluation blocks

5.6 Performance Evaluation

The objective is to determine the effect of jitter from the timing recovery on the echo canceller. By altering the parameters of the elements of the timing recovery system, insight into causes and remedies of the jitter problem can be gained. The following is a list of possible variables that can be changed.

- (1) Fixed equalizer vs. no equalizer
- (2) Prefilter vs. no prefilter
- (3) If prefilter: vary bandwidth
- (4) : vary center frequency (mistuned)
- (5) Nonlinearity $(\cdot)^2$ versus $\cdot$
- (6) Post filter versus no post filter
- (7) Vary PLL parameters, f_n

A number of blocks must be added to the simulation configuration for data acquisition. Figures 5.4 and 5.5 illustrate the simulation block diagrams without and with performance evaluation blocks respectively.

The following are provided for evaluation of performance:

- (1) Residual echo calculation with results being printed to a file.
- (2) Jitter samples ($\Delta(n)$'s) are printed to a file permitting evaluation of the spectral density of the jitter $S_{\Delta}(f)$.
- (3) Near-end, far-end and noise power calculations; from which the desired performance can be generated.

Figure 6.4: Jitter spectrum illustrating advantage of using appropriate bandwidth prefilter for 3.5 mile line with 2 bridged taps using an equalizer and absolute value nonlinearity

to generate the spectral densities is found in Appendix G. In general, we will want to know what the timing jitter power spectrum is, at the output of the PLL.

CHAPTER 6

SIMULATION RESULTS

6.1 Scope of the Investigation

There have been a number of studies of the effect of timing jitter on the echo canceller and the effect of various timing recovery configurations on the amount of timing jitter [19], [20], [5] and [28].

There are a number of differences between the published work and the following work. Messerschmitt [5] compared the spectral line technique and the Wave Difference method for only one bridged tap and only a 2.0 mile line. The study by Falconer [19], [20] concentrated on a 2.5 mile line with no bridged taps, the squaring nonlinear device and a PLL with a natural frequency of .1 KHz. His work was constrained to linear analysis. The work of Abdulsatar and Maral [28] did not consider different prefilter characteristics other than the near optimal prefilter for their case, nor did they investigate different PLL natural frequencies.

Our investigation covers the effects of loops of variable length and numbers of bridged taps. Two nonlinearities, the squaring and absolute value functions are considered. The effects of several values of the PLL natural frequency are also examined.

Thus the results cover a broad spectrum of timing

recovery subsystem variables and different line configurations. Some details on the simulation follow and then the results are presented.

6.2 Simulation Details

The simulation results for the digital subscriber loop environment described previously are reported here. A number of assumptions and details should be pointed out before presentation of the results.

The simulations were carried out for cases of subscriber loops varying in length from 1.5 to 3.5 miles. Bridged taps when employed are given the almost worst case length of 0.5 miles. The loops simulated all have 26 AWG cable for the main line sections and 19 AWG cable for the bridged taps. All bridged taps are assumed open ended. In the channel simulations, the impulse responses are truncated to twenty symbol times.¹ Simulation of the lines employed the theory outlined in Appendices 1 and 2. The transformer transfer function that is used is given by

$$T(f) = \frac{jf}{a+jf} \quad a = 5000 \text{ Hz}$$

The hybrid that is used is shown in Figure 6.1.

The equalizer that is used is a fixed equalizer,²

-
- 1 Truncation allows concentration on jitter effects without having to worry about excess error due to too short a canceller.
 - 2 A fixed equalizer matched to the longest possible line is desirable from the viewpoint of implementation simplicity.

where the structure for simulation purposes is shown in Figure 6.2. The assumed value of the equalizer constant, "k", is 0.8 unless otherwise stated.

In all cases where a mistuned prefilter is employed, it has a center frequency $f_c = 74$ KHz (i.e., 2 KHz above best value).

The simulations were carried out using two stages as described before, to save simulation time and permit focussing on jitter effects.

(1) To acquire good EC coefficients

- $\gamma = 0.1$
- No far-end signal
- No noise
- No jitter

(2) Using coefficients in EC from (1)

- $\gamma = .000001$

Each stage was run for 40,000 samples.

The phase locked loop that is used has a damping coefficient close to .707, a common choice. The VCO's own clock was used with a $10^{-4} \times 144000 = 14.4$ Hz offset. The PLL is simulated for varying natural frequency by way of the PLL parameters described in Table 6.1 that were calculated using the previously described equations.

To ensure that the PLL bandwidth and performance were independent of the input signal level a band pass limiter was used prior to the PLL (i.e., see Figure 6.3). This limiter simply assigned a one or minus one dependent on

PLL Parameters

	τ_1 (s)	τ_2 (s)	loop gain k	δ_n (k-radians)	f_n (KHz)
1	1.0e-6	.993e-6	15000	.702	1.69
2	5.0e-7	4.987e-7	10000	.692	1.148
3	8.0e-6	7.785e-7	7000	.702	.772
4	5.0e-5	4.500e-5	5000	.711	.503
5	7.0e-6	6.1950e-6	2000	.695	.227
6	9.0e-5	8.880e-5	300	.702	.033

Table 6.1: PLL parameters for a second order PLL

Figure 6.1: An Impedance-balancing hybrid structure [35]

Figure 6.2: Fixed equalizer structure

Figure 6.3: Illustration of the use of a bandpass limiter

the sign of the input sample, these one's and minus one's were then filtered using an appropriate filter centered at $1/T$. Therefore the signals entering the PLL for different simulations all have approximately the same magnitude.

The level of Gaussian noise was not varied. Thus for the lines simulated this had the following effect. For long lines (i.e., 3.5 miles) the noise level was 25 dB below the far-end signal, but on short lines (i.e., 1.5 miles) the noise level was as much as 50 dB below the far-end signal.

The jitter spectrum for the different runs are calculated using a periodogram approach as mentioned earlier.

When echo canceller performance is mentioned, generally it means the degree of echo cancellation achieved which is quantified by the factor R , previously described.

6.3 Effect of Prefilter Bandwidth

It has been found that the bandwidth of the prefilter in the timing recovery subsystem has a significant effect on the performance of the echo canceller [19]. Figure 6.4 illustrates the importance to the timing jitter of using a "proper" bandwidth prefilter for the case of a PLL with a large natural frequency, f_n , with respect to the timing jitter spectrum. More specifically, too narrow a prefilter or no prefilter at all, produce larger jitter spectra than a 20 KHz wide prefilter.

Figure 6.4: Jitter spectrum illustrating advantage of using appropriate bandwidth prefilter for 3.5 mile line with 2 bridged taps using an equalizer and absolute value nonlinearity

22
OF / DE

Figure 6.6: R versus prefilter bandwidth for PLL $f_n=1.1$ for equalizer and no equalizer on 3.5 mile line with 2 bridge taps

Figure 6.5: R versus prefilter bandwidth for various values of PLL natural frequency and prefilter bandwidth, for a 2.5 mile line with no bridged taps and no equalizer

Figure 6.6: R versus prefilter bandwidth for PLL $f_n=1.1$ for equalizer and no equalizer on 3.5 mile line with 2 bridge taps

Figure 6.7: R versus prefilter bandwidth for PLL $f = .77$ for equalizer and no equalizer on 3.5 mileⁿ line with 2 bridge taps

Figure 6.8: R versus prefilter bandwidth for PLL $f_n = .23$ for equalizer and no equalizer on 3.5 mileⁿ with 2 bridged taps

6.4 Effect of Prefilter Mistuning

The effect of mistuning in the prefilter from its ideal, center frequency of 72 KHz to 74 KHz can have drastic effects on the performance of the EC. Table 6.2 shows for varying PLL natural frequency, nonlinearity, and equalizer the effect of mistuning on the EC performance. Table 6.3 indicates the small effect on R_{margin} of a bridged tap on a 2.5 mile line under mistuned prefilter conditions. It is clear that there is a distinct degradation in performance caused by a mistuned prefilter. The level of degradation diminishes with smaller PLL f_n values. Therefore, there exists room for compromise in the design of the prefilter as long as the PLL f_n is not too large. Results for a 1.5 mile line show a negligible effect of a mistuned prefilter (see Table 6.4). Thus longer loops seem more susceptible to adverse effects brought on by prefilter mistuning.

6.5 The Effect of Bridged Taps

The simulations detailed in Tables 6.5 and 6.6 point out that the presence of a bridged tap in the loop does not degrade the echo canceller's performances in the presence of jitter. In fact, from a R_{margin} value comparison the bridged tap line performs marginally better. Figure 6.9 however, indicates that for a 2.5 mile line with and without bridged taps, that a higher jitter spectrum is produced for the line with a bridged tap. The better echo canceller performance for the bridged tap case

f_n PLL NATURAL FREQUENCY (KHz)	TUNED R(Rmargin)	MISTUNED R(Rmargin)	ADDITIONAL CONDITION
1.1	84.0 (19.7)	30.0 (-34.3)	EQ. & ABS.
0.77	86.0 (21.7)	32.5 (-31.8)	
0.50	87.5 (23.2)	58.7 (-5.6)	
0.23	87.5 (23.2)	87.5 (23.2)	
0.03	87.5 (23.2)	87.5 (23.2)	
0.50	87.5 (23.2)	59.0 (-5.3)	EQ. & SQR.
0.23	87.5 (23.2)	87.5 (23.2)	
0.03	87.5 (23.2)	87.5 (23.2)	
1.1	74.5 (10.2)	39.5 (-33.8)	NO EQ. & ABS.
0.7	84.9 (20.6)	63.0 (-1.3)	
0.03	87.5 (23.2)	87.5 (23.2)	

Table 6.2: Effect of mistuning prefilter (R , R_{margin}) for mistuned by 2 KHz to 74 KHz center frequency and 20 KHz BW on a 3.5 mile-line with 2 bridged taps

f_n PLL NATURAL FREQUENCY (KHz)	TUNED R (Rmargin) (dB)	MISTUNED R (Rmargin) (dB)	ADDITIONAL CONDITION
1.1	79.4 (25.4)	56.2 (2.3)	BRIDGED TAP
0.77	79.5 (25.5)	62.6 (8.7)	
1.1	50.9 (16.6)	36.1 (1.8)	NO BRIDGED TAP
0.77	52.7 (18.4)	39.7 (5.2)	
0.03	56.3 (22.0)	56.0 (21.7)	

Table 6.3: Effect of mistuning prefilter on EC (R, R_{margin})
for a 2 KHz mistuning, 20 KHz BW on a
2.5 mile line with and without bridged taps

PLL NATURAL FREQUENCY (KHz)	TUNED R (Rmargin) (dB)	MISTUNED R (Rmargin) (dB)
0.50	59.0 (26.7)	59.0 (26.7)
0.23	59.0 (26.7)	59.0 (26.7)
0.03	59.0 (26.7)	59.0 (26.7)

Table 6.4: Effect of mistuning prefilter on EC (R, R_{margin})
for 2 KHz mistuning, 20 KHz BW, on
1.5 mile line with one bridged tap,
no equalizer

may be due in part to the line being better equalized for the worse bridged tap case. Therefore, even though there is more jitter in the bridged tap case, the echo canceller performs better. This will be discussed in more detail in the summary.

6.6 Equalizer Considerations

The choice of the optimal fixed equalizer to be used varies from subscriber loop to subscriber loop. As mentioned, the assumed equalizer constant k up to now is equal to 0.8. Table 6.7 depicts the EC performance for a 1.5 mile line with a bridged tap for varying equalizer constant k values.

Figure 6.10 indicates the superior jitter spectrum for $k=.9$. For this line the amount of jitter clearly depends strongly on the channel/equalizer response (i.e., Intersymbol Interference). For the long line case, Table 6.8 contains the results for a 3.5 mile line with two bridged taps. The best EC performance is for the case of $k=.8$. From the point of view of jitter spectrum, Figure 6.11 indicates best results for $k=.7$.

These results confirm the choice of $k=.8$ for general fixed equalization of subscriber loops and also the sensitivity of the jitter and its influence to the overall end-to-end response including the equalizer. Additionally, the best EC performance values are not always reflected by the best timing jitter spectrum performance. This may in part be due to the shape (i.e.,

PREFILTER BANDWIDTH (KHz)	NO BRIDGED TAP	BRIDGED TAP
5	56.0 (21.7)	59.0 (26.7)
20	54.0 (19.7)	59.0 (26.7)
100	50.0 (15.7)	59.0 (26.7)
144	54.0 (19.7)	59.0 (26.7)

Table 6.5: Effect of presence of bridged taps on line
versus prefilter BW giving (R, R_{margin})
PLL $f_n = .50$, on 2.5 mile line, no equalizer

PREFILTER BANDWIDTH (KHz)	NO BRIDGED TAP	BRIDGED TAP
5	56.0 (21.7)	59.0 (26.7)
20	56.3 (22.0)	59.0 (26.7)
100	56.5 (22.2)	59.0 (26.7)
144	57.3 (23.0)	59.0 (26.7)

Table 6.6: Effect of presence of bridged taps on line
 versus prefilter BW giving (R , R_{margin})
 $PLL f_n = .03$, on 2.5 mile line, no equalizer

EQUALIZER k VALUE	PREFILTER BANDWIDTH (KHz)	10	20	40
0.7		59.0 (36.7)	59.0 (36.7)	59.0 (36.7)
0.8		59.0 (36.7)	59.0 (36.7)	59.0 (36.7)
0.9		59.0 (36.7)	59.0 (36.7)	59.0 (36.7)

Table 6.7: Effect of equalizer constant k values on EC performance (R , R_{margin}) for 1.5 mile line with 1 bridged tap
 $PLR_f = 0.50$, nonlinearity = absolute value for various prefilter bandwidths

<div> <div> <div>PREFILTER BANDWIDTH (KHz)</div> <div>EQUALIZER k VALUE</div> </div> </div>	10	20	40
	56.7	86.1	87.5
0.7	56.7	86.1	87.5
0.8	84.2	87.5	87.5
0.9	63.1	87.5	66.3

Table 6.8: Effect of varying the equalizer constant k (.7, .8, .9) on EC performance (R) for various prefilter BW on a 3.5 mile line with 2 bridged taps, using an equalizer and PLL $f_n = .50$, nonlinear was absolute value

Figure 6.10: Jitter spectrum for various equalizer constants for 1:5 mile line with a bridged tap, 40 KHz wide prefilter, PLL $f_n = 0.50$

Figure 6.10: Jitter spectrum for various equalizer constants for 1.5 mile line with a bridged tap, 40 KHz wide prefilter, PLL $f_n = 0.50$

Figure 6.11: Jitter spectrum for various equalizer constants, on a 3.5 mile line with 2 bridged taps, 25 KHz wide prefilter

the slope) of the echo responses. More discussion on this follows in the summary.

6.7 Choice of Nonlinear Function

The choice of nonlinear function in the timing recovery subsystem (i.e., squarer or absolute value) seems to have a very minor influence on the performance of the echo canceller. Table 6.9 compares the EC performance between the two nonlinear functions for different prefilter bandwidths and PLL natural frequencies. The degradation in EC performance for the PLL $f_n = .5$ case indicates that the squarer function may be superior in the no prefilter configuration. The absolute value function helps reduce the effect of an antisymmetric prefilter since the 8.0 KHz wide prefilter is nonsymmetric (i.e., see Appendix 4). Figure 6.12 illustrates a very small edge in jitter spectrum performance for the absolute value function.

It appears that there is no clear difference in performance for either nonlinear function under consideration, and both perform well under the conditions simulated. Published results [5], [28] favour the absolute value nonlinearity. The nonlinear function of bandpass limiting the signal entering the PLL could be contributing to the reduced advantage that is expected between the two nonlinearities.

PREFILTER BANDWIDTH (KHz)	PLL NATURAL FREQUENCY (KHz) NON- LINEARITY	0.50	0.23	0.03
10	SQR.	79.3(15.0)	87.5(23.2)	87.5(23.2)
	ABS.	84.6(20.3)	87.5(23.2)	87.5(23.2)
20	SQR.	87.5(23.2)	87.5(23.2)	87.5(23.2)
	ABS.	87.5(23.2)	87.5(23.2)	87.5(23.2)
30	SQR.	87.5(23.2)	87.5(23.2)	87.5(23.2)
	ABS.	87.5(23.2)	87.5(23.2)	87.5(23.2)
40	SQR.	87.5(23.2)	87.5(23.2)	87.5(23.2)
	ABS.	87.5(23.2)	87.5(23.2)	87.5(23.2)
80	SQR.	66.7(2.4)	87.5(23.2)	87.5(23.2)
	ABS.	79.2(14.9)	87.5(23.2)	87.5(23.2)
144	SQR.	87.5(23.2)	87.5(23.2)	87.5(23.2)
	ABS.	46.5(-17.8)	87.5(23.2)	87.5(23.2)

Table 6.9: Effect of which nonlinearity (i.e., $(\cdot)^2$ or $||$) on the EC performance (R , R_{margin}) for various prefilter BWs and PLL natural frequencies on a 3.5 mile line with 2 bridged taps

Figure 6.12: Jitter spectrum for the squaring and absolute value nonlinearities on a 2.5 mile line with PLL $f_n = .50$ KHz, and equalization

6.8 The Effect of PLL Natural Frequency

As illustrated already, the natural frequency (i.e., f_n) of the PLL (i.e., related to the loop bandwidth by equations (4.3) and (4.4)) has a marked effect on the EC performance. Table 6.10 summarizes the EC performance for various f_n for the worst case conditions of a mistuned prefilter with and without equalization. It is evident that the greater the f_n beyond a reasonable value (i.e., 0.5 KHz) there is a degradation caused by the increased jitter generated by the PLL. Figures 6.13 and 6.14 demonstrate the increased jitter spectrum arising for larger values of the natural frequency. Of course, the narrower the loop bandwidth of the PLL, the greater the acquisition time and the less robust the system will be.

The choice of PLL parameters such as the natural frequency play a very important part in determining the upper limits of performance that can be expected from the EC. The final choice of the PLL f_n is dependent on a compromise between EC performance and timing recovery robustness (i.e., acquisition and tracking).

6.9 Summary

Considering the simulations that were performed it is evident that a normalized performance factor, R_{margin} (i.e., $R - R_{\text{req}}$) value in the neighbourhood of 20-25 dB can be expected for a reasonably well designed timing recovery subsystem, where due attention has been given to the factors considered above. Thus, for long lines (i.e.,

<div> <div></div> <div>PLL NATURAL FREQUENCY (KHz)</div> </div>	EQUALIZER	NO EQUALIZER
.03	87.5	87.5
.23	87.5	87.5
0.50	58.7	59.0 (squarer)
0.77	32.5	63.0
1.10	30.0	30.5
1.7		28.2

Table 6.10: Effect of a mistuned prefilter (2 KHz) on the EC performance with and without equalization for a 20 KHz BW prefilter on a 3.5 mile line with 2 bridged taps, nonlinearity is absolute value

Figure 6.13: Jitter spectrum for various values of PLL natural frequency f_n on a 3.5 mile line with 2 bridged taps using a 40 KHz BW prefilter, an equalizer and squarer

Figure 6.14: Jitter spectrum for various values of PLL natural frequency f_n on a 3.5 mile line with 2 bridged taps using a 10 KHz BW prefilter, and absolute value

3.5 miles) where the required echo cancellation is high (≈ 60 dB), the EC can achieve in excess of 80 dB of cancellation. For short lines where the required echo cancellation is relatively small (≈ 25 dB), more than 45 dB is realized. The question arises as to why the EC performs so well for the long lines - when it needs to and fairly well - but more than adequately for shorter lines. Figure 6.15 illustrates the relative near, far and achieved residual echo powers¹ for a short (1.5 mile), medium (2.5 mile) and long (3.5 mile) lines each having a bridged tap. Also shown on this figure is the required residual echo power (i.e., such that a 25 dB signal to interference ratio is achieved) and corresponding values of the actually achieved residual echo power. It is clear from this figure that the EC for the short line case has a relatively easy job to do and it does it well. The long line, however, poses a large task for the EC to achieve and it is seen that it is also done well.

What is about the short lines with bridged taps that prevent the echo canceller from performing better? Figure 6.16 illustrates the derivative of the impulse responses of the echo channel for the same lines (i.e., 1.5, 2.5, 3.5 miles with one bridged tap). It is clear that the derivative of the impulse response of shorter channel is higher in magnitude and "spikier" than the

1 The residual echo calculation also subtracted the noise. Thus, in reality the residual error would be slightly above the noise floor.

Figure 6.15: Illustration showing the near-end, far-end and residual echo powers for short (1.5 mile), medium (2.5 mile) and long (3.5 mile) lines with one bridged tap

longer lines. The impact of this can be very significant as seen in equation (2.16), in particular the $B(f)^{(m)}$ term. This term which contributes to the residual mean squared error increases the effect of the derivative of the impulse response as "i" increases or alternatively, the farther along in time of the response. Therefore, for the same jitter spectrum, the impact of jitter on the echo canceller will be accentuated. This turns out to be very fortuitous, in that the worst case loops seem to be less affected by jitter where there is the most to lose. In addition to this, the equalizer that was employed was more suitably matched to long lines, which again, gives assistance where assistance is needed.

Summarizing, the optimum prefilter bandwidth is around 20 KHz. Mistuning of the prefilter has greater adverse effect for long lines and the effect in general may be reduced with a narrow PLL. Bridged tap effects can be minimized by good timing recovery design and equalization. Equalization should be matched to worst case loops. The absolute value nonlinearity was found to have a slight advantage over the squaring function. The PLL was found to perform best with a narrow bandwidth (i.e., f_n) and the narrower the better, although a tradeoff is being made with acquisition and tracking.

Figure 6.16: Plot of the derivative of the impulse response of the echo-path channel for 1.5, 2.5 and 3.5 mile lines with one bridged tap

CHAPTER 7

CONCLUSIONS

7.1 Conclusions

Synchronization system design considerations were investigated with respect to the performance of an echo canceller in the digital subscriber loop environment. The timing recovery system employed a prefilter, a nonlinear element and a second order phase locked loop.

It was found that judicious design of the timing recovery subsystem will lead to good echo canceller performance, for even worst case loops (i.e., long and containing bridged taps). Several factors contribute to the judicious design including, use of a fixed equalizer that is best matched to the worst case lines since the short lines can afford to be over compensated, use of a prefilter that is fairly narrow and centered at one half the baud rate. Also included are the use of an absolute value nonlinear device for a marginal advantage and the use of a phase locked loop with a small natural frequency (i.e., narrow loop bandwidth). It was found that the expected worst case situations posed tighter constraints on the acceptable timing recovery system parameters, but the echo canceller was able to provide satisfactory echo cancellation.

An example system for good performance would have the following parameters:

PLL natural frequency is approximately .2 KHz

Fixed equalizer constant is approximately .8

Prefilter bandwidth is approximately 20 KHz
(Tuned to center at 72 KHz)

Absolute value (i.e., rectifier) nonlinearity

It was found through these simulations that the analytical results of Falconer [19] were in agreement in general with respect to performance of the echo canceller. The results were extended also to include bridged taps. These indicate that the bridged tap impairment may be circumvented by careful timing recovery design.

7.2 Suggestions for Further Work

The simulations carried out for this work were by no means exhaustive. For example different timing recovery techniques should be investigated such as decision directed techniques and some baud rate timing recovery techniques. This investigation limited itself to Bipolar coding and a number of other line codes are worthy of investigation.

Another potential area of study, that is not considered here is the acquisition problem. Due to the close interaction of the echo canceller and the timing recovery the acquisition problem can be quite complex, and may require some special procedures.

An area deserving of more investigation is that of

alternative equalization methods, and the evaluation of the precise requirements of that equalizer.

An area of great importance, is the implementation constraints that are imposed by VLSI design. From a practical viewpoint, the functions that become part of the timing recovery system must be easily implemented in VLSI and should not require too much complexity (i.e., baud rate is already 144 kbps). This brings into play the attraction of baud rate techniques.

A side issue, yet one worthy of more study, is the requirements of the simulation environment for digital communications systems. BLOSIM certainly has features that set it above the standard "Fortran simulation" technique, but it is not optimum yet.

Finally, it would be of interest to investigate the different timing recovery constraints that are imposed by alternative echo canceller designs.

REFERENCES

- [1] P. Gray and D.G. Messerschmitt, "Integrated Circuits for Telephony", Proceedings of IEEE, Vol.68, August 1980, pp.991-1009.
- [2] E.J. Claire, "A Multiple Service Broadband System for the Existing Telephone Local Loop", Proceedings of Zurich Seminar on Digital Communications, 1978.
- [3] A.E. Joel, "The Past 100 Years In Telecommunications Switching", IEEE Communications Magazine, Vol.22, May 1984, pp.64-70.
- [4] S.V. Ahamed, "Simulation and Design Studies of Digital Subscriber Lines", Bell Syst. Tech. J., Vol.61, July-August 1982.
- [5] O. Agazzi, J. Tzeng, D.G. Messerschmitt and D.A. Hodges, "Timing Recovery in Digital Subscriber Loops", submitted for publication.
- [6] S. Ahamed et al., "A Tutorial on Two Wire Digital Transmission in the Loop Plant", IEEE Trans. Commun., Vol.COM-29, November 1981.
- [7] P.J. van Gerwen, N.A.M. Verhoeckx, and T.A.C.M. Claassen, "Design Considerations For a 144 Kbit/s Digital Transmission Unit for the Local Telephone Network", IEEE J. on Sel. Areas in Commun., Vol.SAC-2, March 1984, pp.314-323.
- [8] A. Cantoni, "Clock Recovery in Adaptive Digital Hybrids", Tech. Report EE8402, Dept. of Elect. and Comp. Eng., University of Newcastle, New South Wales, Australia, January 1984.
- [9] D.G. Messerschmitt, "Echo Cancellation in Speech and Data Transmission", IEEE J. on Sel. Areas in Commun., Vol.SAC-2, March 1983, pp.283-297.
- [10] J.A. Franco, "New Line Codes for Subscriber Loops", Master's Thesis, Carleton University, 1984.
- [11] P.J. Gerwen et al., "An Integrated Echo Canceller for Baseband Data Transmission", Philips Tech. Rev., Vol.39, 1980, pp.102-117.

- [12] N.A.M. Verhoeckx et al., "Digital Echo Cancellation for Baseband Data Transmission", IEEE Trans. Acoust., Speech, Signal Processing, Vol.ASSP-27, December, 1979, pp.768-781.
- [13] K.H. Mueller, "A New Digital Echo Canceller For Two-Wire Full Duplex Data Transmission", IEEE Trans. Commun., Vol.COM-24, September 1976, pp.956-962.
- [14] D.D. Falconer and K.H. Mueller, "Adaptive Echo Cancellation/AGC Structures for Two-Wire Full-Duplex Baseband Data Transmission", Bell Syst. Tech. J., Vol.58, September 1979, pp.1593-1616.
- [15] D.D. Falconer, "Adaptive Reference Echo Cancellation", IEEE Trans. Commun., Vol.COM-30, September 1982.
- [16] N. Holte and S. Stueflotten, "A New Digital Echo Canceller for Two-Wire Subscriber Lines", IEEE Trans. Commun., Vol.COM-29, November 1981, pp.1573-1581.
- [17] O. Agazzi, D.G. Messerschmitt and D.A. Hodges, "Nonlinear Echo Cancellation of Data Signals", IEEE Trans. Commun., Vol.COM-30, November 1982.
- [18] M.M. Sondhi and D. Berkley, "Silencing Echos on the Telephone Network", Proceedings of IEEE, Vol.68, August 1980.
- [19] D.D. Falconer, "Timing Jitter Effects on Digital Subscriber Loop Echo Cancellers: Part II - Considerations for Squaring Loop Timing Recovery", submitted for publication.
- [20] D.D. Falconer, "Timing Jitter Effects on Digital Subscriber Loop Echo Canceller: Part I - Analysis of the Effect", submitted for publication.
- [21] K.H. Mueller and M. Muller, "Timing Recovery in Digital Synchronous Data Receivers", IEEE Trans. Commun., Vol.COM-24, May 1976, pp.518-531.
- [22] R.D. Gitlin and J.F. Hayes, "Timing Recovery and Scramblers in Data Transmission", Bell Sys. Tech. J., Vol.54, May 1975, pp.569-593.
- [23] L.E. Franks and J.P. Bubrous, "Statistical Properties of Timing Jitter in a PAM Timing Recovery Scheme", IEEE Trans. Commun., Vol.COM-22, July 1974, pp.913-920.

APPENDIX B

CABLE CHARACTERISTICS DATA

This appendix contains the two data sheets from [35] that provide the characteristics by frequency for:

- (1) 19 AWG PIC cable at 70°F
- (2) 26 AWG PIC cable at 70°F

This information is used along with the theory presented in Appendix A to generate the frequency responses and impulse responses for composite (i.e., more than one cable type) subscriber loops with bridged taps of different gauges and lengths.

APPENDIX A

ABCD PARAMETERS FOR BRIDGED TAPS

Subscriber loops in the telephone loop plant may be composed of sections with different gauges and may contain bridged taps. In order to evaluate the frequency response of such lines, a line model is required.

A transmission line may be modelled using ABCD or hybrid parameters. See Figure A.1 where

$$\begin{pmatrix} V_1 \\ I_1 \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} V_2 \\ I_2 \end{pmatrix}$$

$$A = \cosh \gamma l$$

R = Resistance

$$B = Z_0 \sinh \gamma l$$

L = Inductance

$$C = \frac{\sinh \gamma l}{z}$$

G = Conductance

$$D = \cosh \gamma l$$

C = Capacitance

$$\gamma = \text{propagation constant} (= \sqrt{(R + j2\pi fL)(G + j2\pi fC)})$$

$$Z_0 = \text{characteristic impedance} (= \sqrt{\frac{R + j2\pi fL}{G + j2\pi fC}})$$

$$l = \text{length of the line}$$

[36]

Note, A, B, C, and D are complex quantities accurate tables of γ versus frequency and Z_0 versus frequency for various gauges, temperatures and cable types have been published. See Appendix B [36].

Lines made of a series connection of sections with various parameters may be characterized by matrix multiplication of the individual sections ABCD matrices,

giving a composite ABCD matrix. To generate the characteristics for a line in the opposite direction the A and D terms are interchanged.

A line configured as in Figure A.2 has a Voltage Gain at a particular frequency given by:

$$H = \frac{1}{Z_s(C + \frac{D}{Z_L}) + A + \frac{B}{Z_L}}$$

Consider Figure A.3, depicting a transmission line containing a single bridged tap. The three sections of this line may be characterized by their individual ABCD matrices. Here the loop sees the bridge tap as a parallel connection. An equivalent series connection ABCD matrix may be formed from the original bridged tap ABCD parameters, say A', B', C' and D'.

The conversion is given by:

$$A = 1$$

$$B = 0$$

$$C = Y_0 \tanh \Gamma$$

$$D = 1$$

$$\text{where } Y_0 = \text{admittance} = \sqrt{\frac{C' D'}{A' B'}} = \frac{1}{Z_0}$$

$$\text{and } \Gamma = \text{propagation constant} =$$

$$\text{complex } \log_e \frac{1}{2} (A' + D' + \frac{B'}{Z_0} + Z_0 C')$$

[4]

That is the ABCD matrix for a bridged tap line after conversion is

$$\begin{bmatrix} 1 & 0 \\ Y_0 \tanh & 1 \end{bmatrix}$$

In matrix form:

$$\begin{bmatrix} v_1 \\ I_1 \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} I_2 \\ v_2 \end{bmatrix}$$

Figure A.1: ABCD parameters

Figure A.2: Use of ABCD parameter matrix

Figure A,3: Line with a bridged tap

The equivalent line for Figure A.3 is given by:

$$\text{where } \begin{bmatrix} A_E & B_E \\ C_E & D_E \end{bmatrix} = \begin{bmatrix} A_1 & B_1 \\ C_1 & D_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ CC & 1 \end{bmatrix} \begin{bmatrix} A_2 & B_2 \\ C_2 & D_2 \end{bmatrix}$$

$$\text{where } CC = \sqrt{\frac{C_3 D_3}{A_3 B_3}} \tanh \left[\text{complex } \log_e \left(\frac{1}{2} (A_3 + D_3 + \frac{B_3}{Z_0} + Z_0 C_3) \right) \right]$$

$$\text{where } Z_0 = \sqrt{\frac{1}{\frac{C_3 D_3}{A_3 B_3}}}$$

Matrix reduction in this manner may be carried out for more complex lines, and the net ABCD parameters for the entire line obtained.

The frequency response for a composite cable is determined by evaluating the above matrices at each frequency.

APPENDIX B

CABLE CHARACTERISTICS DATA

This appendix contains the two data sheets from [35] that provide the characteristics by frequency for:

- (1) 19 AWG PIC cable at 70°F
- (2) 26 AWG PIC cable at 70°F

This information is used along with the theory presented in Appendix A to generate the frequency responses and impulse responses for composite (i.e., more than one cable type) subscriber loops with bridged taps of different gauges and lengths.

Cable Characteristics 1Hz to 5MHz

19 GAUGE PIC CABLE AT 70. DEGREES F.

FREQ (HZ)	R (OHMS/MI)	L (MH/MI)	G (MU-10/MI)	C (MU-F/MI)	Z0 (OHMS)	GAMMA (DB/MI)	(DEG/MI)
1.	86.94	0.9861	0.000	0.08300	9132.32	-J*9127.63	0.041
5.	86.94	0.9861	0.001	0.08300	4084.62	-J*4081.48	0.092
10.	86.94	0.9861	0.002	0.08300	2882.76	-J*2885.55	0.131
15.	86.94	0.9861	0.003	0.08300	2359.07	-J*2355.63	0.160
20.	86.94	0.9861	0.004	0.08300	2043.38	-J*2039.68	0.185
30.	86.94	0.9861	0.006	0.08300	1669.00	-J*1664.61	0.226
50.	86.94	0.9861	0.010	0.08300	1293.72	-J*1288.64	0.292
70.	86.94	0.9861	0.013	0.08300	1094.17	-J*1088.33	0.345
100.	86.94	0.9860	0.019	0.08300	916.43	-J*909.60	0.412
150.	86.95	0.9860	0.027	0.08300	749.60	-J*741.37	0.504
200.	86.95	0.9860	0.036	0.08300	650.34	-J*640.91	0.581
300.	86.95	0.9859	0.053	0.08300	532.90	-J*521.46	0.709
500.	86.96	0.9857	0.087	0.08300	415.75	-J*401.08	0.909
700.	86.97	0.9856	0.120	0.08300	353.90	-J*336.59	1.068
1,000.	86.99	0.9853	0.168	0.08300	299.30	-J*278.66	1.263
1,500.	87.03	0.9849	0.247	0.08300	248.78	-J*223.59	1.520
2,000.	87.07	0.9845	0.325	0.08300	219.22	-J*190.30	1.725
3,000.	87.17	0.9837	0.479	0.08300	185.51	-J*150.15	2.061
5,000.	87.42	0.9820	0.760	0.08300	153.94	-J*108.68	2.467
7,000.	87.74	0.9803	1.075	0.08300	138.92	-J*86.50	2.744
10,000.	88.36	0.9777	1.511	0.08300	127.32	-J*66.53	3.015
15,000.	89.74	0.9732	2.225	0.08300	118.60	-J*48.35	3.287
20,000.	91.51	0.9686	2.927	0.08300	114.61	-J*38.26	3.469
30,000.	96.03	0.9531	4.310	0.08300	110.69	-J*27.71	3.770
50,000.	107.61	0.9327	7.017	0.08300	107.73	-J*19.14	4.342
70,000.	120.42	0.9171	9.672	0.08300	106.26	-J*15.51	4.927
100,000.	138.79	0.8972	13.593	0.08300	104.74	-J*12.69	5.761
150,000.	166.50	0.8707	20.012	0.08300	102.94	-J*10.32	7.033
200,000.	189.91	0.8507	26.332	0.08300	101.61	-J*8.94	8.127
300,000.	229.18	0.8256	39.769	0.08300	100.02	-J*7.31	9.964
500,000.	291.46	0.7997	63.114	0.08300	98.32	-J*5.67	12.901
700,000.	342.08	0.7852	87.003	0.08300	97.38	-J*4.80	15.292
1,000,000.	405.89	0.7719	122.247	0.08300	96.52	-J*4.02	18.315
1,500,000.	493.70	0.7591	180.012	0.08300	95.69	-J*3.29	22.482
2,000,000.	567.72	0.7513	236.860	0.08300	95.18	-J*2.05	26.302
3,000,000.	691.90	0.7319	346.725	0.08300	94.56	-J*2.33	31.919
5,000,000.	888.03	0.7318	567.710	0.08300	93.92	-J*1.87	41.333

Figure B.1: Cable characteristics for 19 AWG PIC cable at 70°C

Cable Characteristics 1Hz to 5MHz

26 GAUGE PIC CABLE AT 70. DEGREES F.

FREQ (HZ)	R (OHMS/MI)	L (MH/MI)	C (PF-440/MI)	C (PF-F/MI)	Z0 (OHMS)	GAMMA (DB/MI)	GAMMA (DB/100FT)
1.	440.75	0.9861	0.000	0.08300	20562.46 -J*20551.04	0.093	0.61
5.	440.75	0.9861	0.001	0.08300	9195.60 -J*9190.92	0.208	1.37
10.	440.75	0.9861	0.002	0.08300	6502.18 -J*6498.86	0.294	1.94
15.	440.76	0.9861	0.003	0.08300	5109.30 -J*5106.16	0.361	2.33
20.	440.76	0.9861	0.004	0.08300	4598.12 -J*4595.14	0.416	2.75
30.	440.76	0.9861	0.005	0.08300	3754.56 -J*3751.69	0.510	3.36
50.	440.76	0.9861	0.008	0.08300	2906.86 -J*2905.67	0.658	4.34
70.	440.76	0.9861	0.011	0.08300	2458.60 -J*2455.41	0.779	5.14
100.	440.76	0.9861	0.016	0.08300	2057.43 -J*2053.93	0.931	6.15
150.	440.76	0.9861	0.022	0.08300	1650.47 -J*1676.45	1.139	7.53
200.	440.76	0.9860	0.020	0.08300	1455.83 -J*1451.35	1.315	8.70
300.	440.76	0.9860	0.030	0.08300	1189.51 -J*1184.20	1.610	10.66
500.	440.77	0.9859	0.043	0.08300	922.65 -J*916.00	2.075	13.75
700.	440.78	0.9859	0.044	0.08300	780.91 -J*773.09	2.452	16.33
1,000.	440.79	0.9858	0.115	0.08300	654.74 -J*645.46	2.924	19.56
1,500.	440.81	0.9856	0.164	0.08300	536.48 -J*525.19	3.569	24.04
2,000.	440.83	0.9854	0.210	0.08300	466.25 -J*453.25	4.107	27.56
3,000.	440.88	0.9850	0.299	0.08300	383.35 -J*367.51	4.995	34.36
5,000.	441.01	0.9841	0.466	0.08300	301.19 -J*280.76	6.360	44.55
7,000.	441.15	0.9836	0.625	0.08300	258.16 -J*234.04	7.422	53.99
10,000.	441.39	0.9825	0.853	0.08300	220.58 -J*191.85	8.692	65.90
15,000.	441.87	0.9807	1.213	0.08300	166.46 -J*151.47	10.294	83.56
20,000.	442.45	0.9789	1.558	0.08300	167.08 -J*126.94	11.503	99.83
30,000.	443.88	0.9753	2.217	0.08300	145.71 -J*97.35	13.232	130.60
50,000.	447.81	0.9660	3.458	0.08300	127.25 -J*67.47	15.285	190.10
70,000.	453.09	0.9546	4.634	0.08300	119.21 -J*52.05	16.509	249.33
100,000.	463.34	0.9432	6.220	0.08300	113.56 -J*39.12	17.726	334.24
150,000.	485.60	0.9306	8.993	0.08300	109.61 -J*28.32	19.253	491.26
200,000.	513.04	0.9212	11.550	0.08300	107.80 -J*22.21	20.676	644.17
300,000.	575.17	0.9062	16.436	0.08300	105.92 -J*17.35	23.540	969.55
500,000.	694.61	0.8816	25.633	0.08300	103.97 -J*12.91	29.264	1551.77
700,000.	812.95	0.8614	34.351	0.08300	102.45 -J*10.86	34.477	2142.93
1,000,000.	956.65	0.8381	46.849	0.08300	100.70 -J*9.00	41.198	3010.78
1,500,000.	1124.38	0.8146	66.645	0.08300	97.35 -J*7.42	50.493	4152.71
2,000,000.	1321.07	0.7901	85.614	0.08300	90.39 -J*6.43	58.348	5879.66
3,000,000.	1600.58	0.7423	121.841	0.08300	97.23 -J*5.26	71.551	8115.35
5,000,000.	2044.07	0.7130	190.024	0.08200	96.02 -J*4.05	92.534	14340.97

Figure B.2: Cable characteristics for 26 PWG PIC cable at 70°C

APPENDIX C

ANALOG TO DIGITAL TRANSFORMATIONS

C.1 Transformation of analog loop filter to digital filter.

$$\begin{aligned}
 H(s) &= \frac{k(1 + \tau_2 s)}{1 + \tau_1 s} \\
 &= k \frac{2}{1} + \frac{k \frac{\tau_2}{\tau_1} [\frac{1}{\tau_2} - \frac{1}{\tau_1}]}{\frac{1}{\tau_1} + s} \\
 &= k' + \frac{k'[a-b]}{b+s}
 \end{aligned}$$

$a = 1/\tau_2$
 $b = 1/\tau_1$
 $k' = k \tau_2/\tau_1$

$$h(t) = k' \delta(t) + (a-b) k' e^{-bt} u(t)$$

$$h(n) = h(t) \Big|_{t=nT} \quad T = \text{sampling period}$$

$$h(n) = k' \delta(n) + (a-b) k' e^{-bnT} u(n)$$

$$\begin{aligned}
 H(z) &= k' \sum_{n=0}^{\infty} \delta(n) z^{-n} + (a-b) k' \sum_{n=0}^{\infty} e^{-bnT} z^{-n} \\
 &= k' + \frac{(a-b) k' e^{-S_k T}}{1 - e^{-S_k T} z^{-1}} \quad \text{where } S_k = b = 1/\tau_1
 \end{aligned}$$

Using impulse variance method [37]

$$H(z) = \sum_{k=1}^N \frac{\tau_k T A_k}{1 - e^{-S_k T} z^{-1}}$$

$$\begin{aligned}
 H(z) &= k \frac{\tau_2}{\tau_1} + \frac{(\tau_1 - \tau_2) k T}{\tau_1^2 (1 - z^{-1} e^{-T/\tau_1})} \\
 &= \frac{\theta_0}{\theta_1}
 \end{aligned}$$

$$\therefore \theta_o(n) = \theta_o(n-1) e^{-T/\tau_1} - \theta_1(n-1) k \frac{\tau_2}{\tau_1} e^{-T/\tau_1} + \\ \theta_1(n) \left[k \frac{\tau_2}{\tau_1} + \left(\frac{\tau_1 - \tau_2}{2} \right) kT \right]$$

C.2 Transformation of analog VCO (integrator) to digital form

$$\frac{d\theta_o}{dt} = K_o V_c$$

T = sampling period

$$\theta_o = \int k_o V_c dt$$

an integral may be approximated by a trapezoidal rule [37]

$$\text{i.e., } \theta_o(t) = \int_{t_o}^t \theta'_o(t) dt + \theta_o(t_o)$$

then:

$$\theta_o(nT) = \theta_o((n-1)T) + \frac{T}{2} [k_o V_c (nT) + k_o V_c((n-1)T)]$$

APPENDIX D

IIR Band Pass Filter Frequency Responses

This appendix contains the frequency responses of the various prefilters which vary in bandwidth and center frequency

	Center Frequency (KHz)	Bandwidth (KHz)
Prefilters:	72	10
	72	20
	72	30
	72	40
	72	80
	72	144
	74 and 72	20 (mistuned)

These responses are achieved by cascading two of the aforementioned IIR filters for different "r" values.

Figure D.1: Amplitude response of IIR prefilter centered at 72 KHz with bandwidth ≈ 10 KHz

Figure D.2: Amplitude response of IIR prefilter centered at 72 KHz
with bandwidth = 20 KHz

Figure D.3: Amplitude response of IIR prefilter centered at 72 KHz
centered at 72 KHz with bandwidth $\approx$ 40 KHz

Figure D.4: Amplitude response of IIR prefilter centered at 72 KHz
with bandwidth $\approx$ 80 KHz

Figure D.5: Amplitude response of IIR prefilter centered at 72 KHz
with bandwidth $\approx$ 144 KHz

Figure D.6: Amplitude response of IIR prefilters centered at 72 KHz and 74 KHz with bandwidth ≈ 20 KHz (i.e., mistuned)

APPENDIX E

ECHO AND TRANSMISSION PATH IMPULSE RESPONSE

This appendix contains the impulse responses of the echo and transmission path for some of the lines used in the simulations. All bridged taps are assumed to be 0.5 miles long. The mainline sections are 26 AWG gauge PIC cable at 70°F and the bridged taps are 19 AWG gauge, PIC cable at 70°F.

The following is a list of line configurations for which the impulse response for the echo and transmission channel are plotted.

Configurations

The first 3 plots are for the echo channel for 1.5, 2.5 and 3.5 mile lines with and without bridged taps. The second 3 plots are for the transmission channel as above. (Note: plots show 20 symbol times). (Note: Lines with squares have no bridged taps, lines with octagons have bridge taps (i.e., see plots).

Figure E.1: Impulse response of 1.5 mile line transmission channel with and without a bridged tap. (Plot shows impulse response over 20 symbol times)

Figure E.2: Impulse response of 2.5 mile line transmission channel with and without a bridged tap. (Plot shows impulse response over 20 symbol times)

Figure E.3: Impulse response of 3.5 mile line transmission channel with and without a bridged tap. (Plot shows impulse response over 20 symbol times)

Figure E.4: Impulse response of 1.5 mile line echo channel with and without bridged taps. (Plot is for 20 symbols times)

Figure E.5: Impulse response of 2.5 mile line echo channel with and without bridged taps. (Plot is for 20 symbol times)

Figure E.6: Impulse response of 3.5 mile line echo channel with and without bridged taps. (Plot is for 20 symbol times)

APPENDIX F

COMPUTER SIMULATION PROGRAM LISTINGS

This appendix contains a listing of the user written programs required to carry out the simulations in this work. They are used with a simulation package called BLOSIM [53] under UNIX on a VAX 750. The code is written in C.

The simulation code consists of

- UNIVERSE.T - the topology definition of the complete simulation
- PLL_GAL.T - the topology definition of the phase locked loop GALAXY
- REQUIRED STARS - these are the elements or blocks that are linked together to create a complete simulation

This will be the order in which the different elements of the simulation are presented in this appendix. STAR are in alphabetical order.


```

star noise
param file af_noise
param int 35000
star pr_af_noise

# NEAR TX STUFF
# was 1 0
param float 1 0
star const
#
param float 100
star const
#
param int 0
star delay_near
#
star summer1
#
star fork_sum
#
param float 0 6
star lpf_sum
#
param file resid_echo
param int 35000
star print_diff
#
star fork_echo
#
param file af_diff
param int 35000
star pr_af_diff
#
param file af_echo
param int 35000
star pr_af_echo
#
param int 6
star delay_chan
#
param float 0 6
star lpf_near
#
star fork_near
#
param file near
param int 35000
star pr_near
#
param int 0
star delay_diff
#
param float 0 6
star lpf_far
#
star fork_far
#
param file far

```

```

noise.c
null_pr_to_file.c
const.c
const.c
delay.c
summer.c
fork.c
lpf.c
diff_pr_to_file.c
fork.c
null_pr_to_file.c
null_pr_to_file.c
delay.c
lpf.c
fork.c
pr_to_file.c
delay.c
lpf.c
fork.c
file

```

```

pr_to_file.c
starter_block.c
sign.c
bpf_144.c
bip_codec
forkkk
trans_filter
channel
constj
channel
constt
delay_chan
fork_near
summer
lpf_near
pr_near
delay_near
echo_can
coder2
trans_filter2
tran_filt_from_file2
constz
fork_far
summer1
lpf_far
pr_far
fork_sum
summer
lpf_sum
delay_diff
print_diff
print_diff
now NOISE
connect noise
connect constn
connect pr_af_echo
connect summer
rx and timing recovery
connect lpf
pr_af_diff
echo can

```

```
0 pr_af_echo
1 pr_echo_est
0 fork_echo
0 equalizer
0 const_echo
0 bpf_2t
0 bpf_2t_2
0 abs_value
0 bpf_t
0 sign
0 bpf_t2
0 const_pll
0 pr_af_bpf2
0 pll
1 pr_af_phase
0 pr_af_pll
2 starter_block
0 pr_error
```

```
connect echo_can
connect echo_can
connect pr_af_echo
connect fork_echo
connect equalizer
connect const_echo
connect bpf_2t
connect bpf_2t_2
connect abs_value
connect bpf_t
connect sign
connect bpf_t2
connect const_p11
connect pr_af_bpf2
connect p11
connect p11
connect p11
connect starter_block
```



```

.....
ABS_VALUE
.....
purpose: this star implements an absolute value
function
.....
input 0 x sample
output 0 x sample
model x = | x |
parameters none
.....
#include <math.h>
no_input_fifos 1
no_output_fifos 1
declarations
SAMPLE x
end
main_code
while(length_output_fifo(0) < maxlength_output_fifo(0) &&
length_input_fifo(0) > 0 ) {
    get(0,&x);
    if(x < 0)
        x = 1.0 * x; /* x--> o/p fifo */
    if (put(0,x) < 0) /* lost sample - o/p fifo full */
        return(1);
    return(0);
}
end

```

Figure F.3: Listings of STAR's

```

if(mode == 0) { /* coder desired */
    if(xin == (SAMPLE) 0)
        xout = (SAMPLE) 0;

    else if(xin == (SAMPLE) 1) {
        last_one = - last_one;
        xout = (SAMPLE) last_one;
    }

    else return(2); /* input not 0 or 1 */
}

else { /* decoder desired */
    if(xin == (SAMPLE) 0)
        xout = (SAMPLE) 0;

    else if(xin == (SAMPLE) 1 || xin == (SAMPLE) -1)
        xout = (SAMPLE) 1;

    else return(3); /* input not 0, +1 or -1 */
}

if(put(0,xout) < 0) /* put sample onto output fifo */
    return(2); /* output fifo full */

return(0); /* input fifo empty */
}

end

```

```

.....
BIP_CODER
.....
purpose: this routine is either a bipolar coder or decoder
         depending on an input parameter

input 0 binary data ie 0 or 1 - for coder mode
      bipolar signals - for decoder mode
output 0 bipolar data - for coder mode
      binary data - for decoder mode

model 0 output when input is zero
      1 or -1 alternating when input is 1

parameter 1 mode integer
           = 0 --> coder
           = 1 --> decoder
.....

no_input_fifos 1
no_output_fifos 1

param mode int

state last_one int initialize 1

declarations
SAMPLE xin,xout.

end

initialization_code

/* error if parameter not zero or one */
if(mode != 0 && mode != 1)
    return(1);

end

main_code

while(length_output_fifo(0) < maxlength_output_fifo(0) &&
      length_input_fifo(0) > 0) {
    if(get(0,xin) < 0)
        return(1);

    /* algorithm slightly different for coder and decoder */
}

```

BPF_144

purpose {this star is a IIR band pass filter centered
at "f_not" Hz

input 0 x
output 0 x

param 1 f_not float default 144000
param 2 t_s float default 1/(4*144000)
param 3 r float default .98

#include <math.h>
#define pi 3.1415926

no_input_fifos 1
no_output_fifos 1

param f_not float default 144000
param t_s float default 1.0/(4.0*144000)
param r float default .98

state y_pre float initialize 0.0
state y_pre_pre float initialize 0.0

declarations
SAMPLE x,y

end

main_code

```
/* make sure output fifo not full */
while (length_output_fifo(0) < maxlength_output_fifo(0)) {
    if (length_input_fifo(0) > 0) {
        if (get(0,4x) >= 0) {
            y = y_pre*2.0*r*cos(2.0*pi*f_not*t_s) + x - (r+r*y_pre_pre);
            y_pre_pre = y_pre;
            y_pre = y;
            if (put(0,y) < 0) /* put sample on output fifo */
                return(1); /* output fifo full */
        } /* if */
    } /* while */
    return(0);
}
```

end

DPF_72

purpose: this star is a IIR band pass filter centered
at "f_not" Hz.

```
input 0 x
output 0 x

param 1 f_not float default 72000
param 2 t_s float default 1/(4*144000)
param 3 r float default 90
```

```
#include <math.h>
```

```
#define pi 3.1415926
```

```
no_input_fifos 1
no_output_fifos 1
```

```
param f_not float default 72000
param t_s float default 1.0/(4.0*144000)
param r float default 90
```

```
state y_pre float initialize 0.0
state y_pre_pre float initialize 0.0
```

```
declarations
SAMPLE x,y
```

```
end
```

```
main_code
```

```
/* make sure output fifo not full
checkalloc() : /* check integrity of dyn storage alloc */

while(length_output_fifo(0) < maxlength_output_fifo(0)
    if(length_input_fifo(0) > 0) {
        if(opt(0,6x) > 0) {
            y = y_pre*2.0+pi*cos(2.0*pi*f_not*t_s) * x - (r*y_pre_pre)
            y_pre_pre = y_pre
            y_pre = y
            if(opt(0,y) < 0) /* put sample on output fifo */
                return(1) /* output fifo full */
        } /* if */
    } /* while */
    return(0)
end
```

```

.....
CONST
.....
*
* purpose this routine does nothing but pass samples
* from the input fifo to the output fifo and
* multiply them by the constant parameter
*
* input 0 x
* output 0 x
*
* model x = x * constant
*
* parameter 1 float constant default 1.0
*
.....
no_input_fifos 1
no_output_fifos 1
param constant float default 1.0
declarations
SAMPLE x
end
main_code
/* make sure output fifo not full
while (length_output_fifo(0) < maxlength_output_fifo(0) &&
length_input_fifo(0) > 0 )
/* transfer samples to output fifo from input fifo
if (get(0,&x) >= 0 ) {
x = x * constant
if (put(0,x) > 0 ) /* put sample on output fifo */
return(0) ; /* output fifo full */
}
return(0);
end

```

```

.....
COS_CBN
.....

```

```

.....
purpose: this star outputs the cosine of the input
angle.
.....

```

```

.....
input 0 x samples
output 0 z samples
.....

```

```

.....
model : z = cos( x )
.....

```

```

.....
parameters : none
.....

```

```

.....
description : the first sample output is 1.0
.....

```

```

.....
#include <math.h>
.....

```

```

no_input_fifos 1
no_output_fifos 1
.....

```

```

.....
declarations
SAMPLE x,z
.....

```

```

end
.....

```

```

initialization_code
.....

```

```

z = 1.0 ;
put(0,z) ; /* place initial output in fifo */
/* this is done to facilitate feedback loop in pll_gai */
.....

```

```

end
.....

```

```

main_code
.....

```

```

while( length_input_fifo(0) > 0 &&
length_output_fifo(0) < maxlength_output_fifo(0) ) {
    get(0,&x) ;
    z = cos( x ) ;
    put( 0,z ) ;
}
return(0) ;
end
.....

```

```

.....
DATA_GEN
.....
purpose: this function generates random data
samples +1,0
.....
input : no input fifo
output 0 x samples (+1,-1)
model : x= rnd(+1.0)

parameter 1 ift max_sample default 1000
- this is the maximum number of samples
that will be generated

description : this function uses a pseudo random
number generator to create a sequence
of 1,0's.
.....
#include <math.h>

param max_sample int default 1000
no_input_fifos 0
no_output_fifos 1

state count int initialize 0
state max int float initialize 2147483640.0

declarations
double rnum,temp;
double floor();
SAMPLE x;
end

initialization_code
srand(5000);
end

main_code
while(length_output_fifo(0) < .maxlength_output_fifo(0) &&
count <= max_sample ) {
++count;
temp = (float)count / 100.0;
rnum = (((rand()/(maxint)*2.0)-1.0);
srand((rnum+1.0)*500);
if (rnum <= 0.0)
x = 0.0;
else x = 1.0;
put(0,x);
}
return(0);
end

```



```

.....
LOOP_FILTER
.....
* purpose this filter implements the analog filter [1*tau_2 / (1+tau_1)]
* in sample discrete time the transformation used was the
* impulse invariant
*
* input 0 theta_in
* output 0 theta_out
*
* model
*   theta_in(n) = theta_in(n) * f_filt_gain
*   theta_out(n) = theta_in(n) * (1 + tau_2/tau_1) / tau_1 +
*   tau_2/tau_1 * [theta_out(n-1) - theta_in(n-1)] *
*   exp(-tau_2/tau_1)
*
* parameter 1 float tau_1 defaults to 5e-6
* parameter 2 float tau_2 defaults to 4e-6
* parameter 3 float t_s defaults to 1/(14400*4) sample period
* parameter 4 float f_filt_gain defaults to 5000 hz
*
*.....
#include <math.h>
.....
no_input fifos 1
no_output fifos 1

state theta_in_pre float initialize 0 0
state theta_o_pre float initialize 0 0

param tau_1 float default 5e-06
param tau_2 float default 4e-06
param t_s float default 1.73611e-6
param f_filt_gain float default 5000

declarations
SAMPLE theta_in
SAMPLE theta_out
float temp_var_1
float temp_var_2
end

main_code
temp_var_1 = ((t_s/tau_1) * (1 - (tau_2/tau_1))) * (tau_2/tau_1)
temp_var_2 = (exp(-(t_s/tau_1)) * t_s) / tau_1
while (length input_fifo(0) > 0 &&
length output_fifo(0) < maxlength_output_fifo(0)) {
get(0, theta_in)
theta_outs[theta_o_pre * temp_var_1 + f_filt_gain *
theta_in_pre + theta_in *
temp_var_2] = theta_in_pre +
temp_var_1 * temp_var_2 * f_filt_gain
theta_in_pre = theta_in
if (put(0, theta_out) != 0)
return(0)
}
return(0)
end

```

```

.....
..... ° DIF_PR_TO_FILE
.....
..... purpose: this function prints difference of samples from
..... 2 input files
.....
.....      Input 0 sample data values
.....      1 sample data values
.....
.....      parameter 1 file file_name default stdout
.....      2 int print_after default 0
.....
..... no_input_files 2
..... no_output_files 0
.....
..... state "fp" FILE* Initialize 0
..... state sample_num int default "stdout"
..... param file_name file default "stdout"
..... param print_after int default 0
.....
..... declarations
..... SAMPLE x.y.z
..... FILE *fopen() /* opens file and assigns pointer fp */
..... int i empty
..... end
.....
..... initialization code
..... if ( strcmp( file_name, "stdout" ) == 0 )
.....     fp = stdout
..... else
.....     if ( ! fp = fopen( file_name, "w" ) ) == NULL )
.....         return(1) /* file_name cant be opened to write */
.....
..... main_code
.....
..... // dont start printing each line until there is data in each i/p fifo //
..... // and there is enough room on the output fifo //
..... while ( length_input_fifo(0) > 0 && length_input_fifo(1) > 0 ) {
.....     if ( getc(fifo_0) > 0 && getc(fifo_1) > 0 ) {
.....         ++sample_num
.....         if(sample_num % print_after) {
.....             x = x-y
.....             printf( fp, "%d", (float)x )
.....             // print newline //
.....             fflush( fp );
.....         }
.....     }
..... }
..... return(0) /* at least one input fifo is empty */
.....

```

F14

031410

purpose this routine does nothing but pass samples
from the input fifo to the output fifo and
differentiate them

input 0 x
output 0 x

parameters 1 float eta default 1.0

no input fifo 1

no output fifo 1

state previous float initialized 0.0

param eta float default 1.0

declarations

sample x x out

end

main code

while (length output fifo not full)
length input fifo(0) < maxlength output fifo(0) do
transfer samples to output fifo from input fifo
if (not 0) {
x out = x previous + eta * (input(0) - x out)
if (not 0) {
put sample on output fifo x
output fifo full
return 0
}

end

E-100 FILE

purpose this file is an adaptive differential filter in which the adaptation coefficients are initially read in from a file whose name is given as a parameter

input 0 data sequence (ie binary data) KNOWN bits
input 1 sample sequence of far end received data (ie TO BE EQ)
output 0 sample sequence of far end data with echo removed
output 1 sample sequence of echo estimates

model

$$q(n+1) = q(n) + \gamma(n) \cdot y(n) \cdot b(n)$$
 suggested gamma 0.0047
 q(n) adaptation coefficients
 y(n) sample sequence input (fifo 0)
 x(n) sample sequence of far end data (fifo 1)
 b(n) error signal
 gamma update parameter

parameter 1 num_taps int default 10 number of filter taps
 2 gamma float default 0.0047 step size
 3 startup_gain float default 0.01 gamma for first 5000 samples
 4 coef_file file gain start up coef from here and update every 1000 samples
 5 echo_coef_file run file haven't coef every 1000 samples

does adaptation can be used in echo cancellation

no input files 2
no output files 2

state echo estimate float
 state channel float
 state q float
 state q1 float
 state q2 float
 state q3 float
 state past float
 state sample_num int initialize 0
 state error float initialize 0.0
 state b float initialize 1.0
 state gam float
 state fp FILE
 state samples_read int initialize 0
 state last_update int initialize 0

param num_taps int default 10
 param gamma float default 0.0047
 param startup_gain float default 0.01
 param coef_file

```

param coef_file run file

declarations
SAMPLE x y
float b now
int count 1; taps that's all taps used
double temp floor fb, fabet
FILE *fpopen()
end

Initialization code
/* allocate memory for coefficient vector */
q0 = (float*) calloc(num_taps, sizeof(float))
q1 = (float*) calloc(num_taps, sizeof(float))
q2 = (float*) calloc(num_taps, sizeof(float))
q3 = (float*) calloc(num_taps, sizeof(float))
/* allocate memory for past data samples array */
past = (float*) calloc(num_taps, sizeof(float))
channel = (float*) calloc(4, sizeof(float))
echo_estimate = (float*) calloc(4, sizeof(float))

/* initialize past-sample array to zero */
for (count=0; count < num_taps; ++count) {
    past[count] = 0.0
    q0[count] = 0.0
    q1[count] = 0.0
    q2[count] = 0.0
    q3[count] = 0.0
}

gam = startup_gain
that's all = 0
taps = 4 * num_taps
taps_used = 0

if (fp = fopen(coef_file, "r")) == NULL)
    printf("EC coefficients are starting at ZERO\n")
else {
    printf("GETTING EC coefficients from file\n")
    if (fscanf(fp, "%d %d %d", &samples_used, &taps_used) != EOF) {
        if (samples_used != 0 && taps_used == num_taps) {
            count = 0;
            while (fscanf(fp, "%e %e %e %e", &q0[count], &q1[count], &q2[count], &q3[count]) != EOF)
                ++count;
        }
        printf("coef have been read in, samples_used = %d \n", samples_used)
    }
    /* if */
    fclose(fp);
} /* else */

main code
that's all = 1
while (length_input_fifo(0) > 0 && length_input_fifo(1) > 3)
    while (length_output_fifo(0) < 4 && maxlength_output_fifo(0)) {
        that's all = 0
        if (get(0, &x) > 0) {
            /* to get :1 and -1 by differential decoding */
            b_now = (2.0 * x) + b_last
        }
    }
}

```

```

error = channel[k] - echo_estimate[0]
/* update coefficients */
for (i=0; i<num_taps; ++i)
    q[i] = q[i] + gain*error + past[i]
put(0, error)
break

case 1
for (count=0; count<num_taps; ++count)
    echo_estimate[1] += q[count] * past[count]
put(1, echo_estimate[1])
error = channel[k] - echo_estimate[1]
/* update coefficients */
for (i=0; i<num_taps; ++i)
    q[i] = q[i] + q[i-1] + gain*error + past[i]
put(0, error)
break

case 2
for (count=0; count<num_taps; ++count)
    echo_estimate[2] += q2[count] * past[count]
put(1, echo_estimate[2])
error = channel[k] - echo_estimate[2]
/* update coefficients */
for (i=0; i<num_taps; ++i)
    q2[i] = q2[i] + q2[i-1] + gain*error + past[i]
put(0, error)
break

case 3
for (count=0; count<num_taps; ++count)
    echo_estimate[3] += q3[count] * past[count]
put(1, echo_estimate[3])
error = channel[k] - echo_estimate[3]
/* update coefficients */
for (i=0; i<num_taps; ++i)
    q3[i] = q3[i] + q3[i-1] + gain*error + past[i]
put(0, error)
break

default
printf("ERROR in echo %d\n",
break

} // for i
} // switch

/* print coef every 100 samples */
temp = (float)sample_num / 100
if (temp - floor(temp) == 0)
for (count=0; count<num_taps; ++count)
    printf(" q[%d] = %f\n", count, q[count])
} // while
return(0)
end

```

```

// THIS SECTION SAVES COEF. EVERY 1000 SAMPLES
if (sample_num % 1000 == 0) {
    last_update = 1000
    if (fp = fopen("coef.dat", "w")) {
        printf("updating COEF file sample_num %d\n", sample_num)
        for (count=0; count<num_taps; ++count)
            printf(fp, "%e %e %e %e\n", q[count], q2[count], q3[count],
            temp)
        fclose(fp)
    }
    // try to update coef but couldn't open file
    if (fp = fopen("coef.dat", "w")) {
        printf("updating COEF file sample_num %d\n", sample_num)
        for (count=0; count<num_taps; ++count)
            printf(fp, "%e %e %e %e\n", q[count], q2[count], q3[count],
            temp)
        fclose(fp)
    }
    // try to update coef but couldn't open file
    if (fp = fopen("coef.dat", "w")) {
        printf("updating COEF file sample_num %d\n", sample_num)
        for (count=0; count<num_taps; ++count)
            printf(fp, "%e %e %e %e\n", q[count], q2[count], q3[count],
            temp)
        fclose(fp)
    }
}

// THIS SECTION SAVES COEF. EVERY 1000 SAMPLES
if (sample_num % 1000 == 0) {
    last_update = 1000
    if (fp = fopen("coef.dat", "w")) {
        printf("updating COEF file sample_num %d\n", sample_num)
        for (count=0; count<num_taps; ++count)
            printf(fp, "%e %e %e %e\n", q[count], q2[count], q3[count],
            temp)
        fclose(fp)
    }
    // try to update coef but couldn't open file
    if (fp = fopen("coef.dat", "w")) {
        printf("updating COEF file sample_num %d\n", sample_num)
        for (count=0; count<num_taps; ++count)
            printf(fp, "%e %e %e %e\n", q[count], q2[count], q3[count],
            temp)
        fclose(fp)
    }
    // try to update coef but couldn't open file
    if (fp = fopen("coef.dat", "w")) {
        printf("updating COEF file sample_num %d\n", sample_num)
        for (count=0; count<num_taps; ++count)
            printf(fp, "%e %e %e %e\n", q[count], q2[count], q3[count],
            temp)
        fclose(fp)
    }
}

```


[illegible]

```

.....
FORK
.....
purpose this star takes samples in from a single
      fifo and transfers them to all the output
      fifos

input 0 x samples
output 0 n x samples
model x = x
parameters none
description this function determines the number of
      output fifos by the "no_output_fifos()"
      function therefore the topology defin
      ition determines this
.....
state no fifos int
no_input_fifos 1
declarations
SAMPLE x
int 1, full_output_fifo.
end

Initialization_code
/* assign value to state variable */
no_fifos = no_output_fifos()
if( no_output_fifos() <= 0) return(3).
end

main_code
/* check that no output fifo full */
for(i=0; i<no_fifos; ++i)
  if((length_output_fifo(i) == maxlength_output_fifo(i))
    return(0) /* output fifo i full */
  while(get(O,i) >= 0) {
    /* put sample in each output fifo, and note if any
       output fifo fills up */
    full_output_fifo = 0; /* set to one if fifo full */
    for(l=0; l<no_fifos; ++l)
      if((put(l,x) != 0) /* fifo now full */
        full_output_fifo = 1;
      if(full_output_fifo) return(0);
    return(0); /* input fifo empty */
  }
end

```



```

.....
LOOP_FILTER
.....
purpose this filter implements the analog filter (1-tau_2 / (1+tau_1))
in sample discrete time the transformation used was the
impulse invariant

input 0 theta_in
output 0 theta_out

model
    theta_in(n) = theta_in(n-1) + filt_gain
    theta_out(n) = theta_in(n) * (1-tau_2/tau_1) +
    tau_2/tau_1 * (theta_out(n-1) - theta_in(n-1)) *
    exp(-t_s/tau_1)

    parameter 1 float tau_1 defaults to 5e-6
    parameter 2 float tau_2 defaults to 4e-6
    parameter 3 float t_s defaults to 1/(144000*4) sample period
    parameter 4 float filt_gain defaults to 5000 hz

.....
#include <math.h>
.....
no_input_fifos 1
no_output_fifos 1

state theta_in_pre float initialize 0 0
state theta_o_pre float initialize 0 0

param tau_1 float default 5e-06
param tau_2 float default 4e-06
param t_s float default 1.73611e-6
param filt_gain float default 5000

declarations
SAMPLE theta_in
SAMPLE theta_out
float temp_var_1
float temp_var_2
end

main code
temp_var_1 = ((t_s/tau_1) * (1 - (tau_2/tau_1))) * (tau_2/tau_1)
temp_var_2 = (exp((-1) * t_s/tau_1))
while (length(input_fifo(0)) > 0 &&
length(output_fifo(0)) < maxlength_output_fifo(0)) {
    get(0, theta_in)
    theta_outs(theta_o_pre + (theta_in_pre * filt_gain * tau_2/tau_1) + temp_var_2 *
    (theta_in - temp_var_1 * filt_gain)
    theta_in_pre = theta_in
    theta_o_pre = theta_out
    if (put(0, theta_out) != 0)
        return(0)
}
return(0)
end

```

```

.....
LPP
.....
*
* purpose this routine exercises a low pass filter
* whose 3db point is at 720hz and is set for
* a sampling period = 1/(4*14000) seconds
*
* input 0 x
* output 0 filtered samples
*
* model y(k) = alpha*y(k-1) + (1-alpha)*x(k)
*
* parameter 1 alpha default 986386 (1e for 720hz 3db pt)
*
.....

no_input_fifo 1
no_output_fifo 1

state last_y float initialize 0 0

param alpha float default 986386

declarations
SAMPLE x, new_y
end

main_code
while { length_input_fifo(0) > 0 && length_output_fifo(0) <
maxlength_output_fifo(0) } {
    if (get(0, &x) >= 0) {
        new_y = ((1.0 - alpha) * x) + (alpha * last_y)
        last_y = new_y
        if (put(0, new_y) < 0) /* put sample on output fifo */
            return(1) /* output fifo full */
    }
    return(0)
}
end

```

MULT

purpose this star multiplies the samples from two
input fifos and outputs the result to the
output fifo

input 0: x samples
input 1: y samples
output 0: z samples

model $z = x * y$

parameters none

no input_fifos 2
no output_fifos 1

declarations

SAMPLE x, y, z

end

main code

```
while( length(input_fifo(0)) > 0 &&  
length(input_fifo(1)) > 0 &&  
length(output_fifo(0)) < max_length_output_fifo(0) ){
```

```
    get(0, &x)  
    get(1, &y)  
    z = x * y  
    put(0, z)  
}
```

```
return(0)  
end
```

NOISE

purpose this module generates gaussian noise samples
given the mean and standard deviation required

output 0 noise samples

parameter 1 mean float default 0.0
2 std_dev float default 1.0

#include <math.h>

#define max_int 2147483647

no_input_fifos 0

no_output_fifos 1

param mean float default 0.0

param std_dev float default 0.01

declarations

SAMPLE x

int i

double sum

end

initialization:code

srand(3579)

end

main_code

while (length_output_fifo(0) < maxlength_output_fifo(0)) {

sum = 0.0

for (i=0; i<12; ++i)

sum += (double)rand() / max_int * .5;

x = sum * std_dev + mean;

put(0,x)

}

return(0)

end

```

    }
    return(0) ; /* at least one input fifo is empty */
}
end

```

PR_TO_FILE

purpose: this function prints samples from an input
fifo into a specified file and throws the
samples away.

input 0 sample data values
parameter 1 file file_name default stdout
parameter 2 int print_after default 0

no_input_fifos 1

state fp FILE*
state sample_num int initialize 0
param file_name file default "stdout"
param print_after int default 0

declarations

SAMPLE x;
FILE *fopen() ; /* opens file and assigns pointer fp */
int i, empty

Initialization code

if (strcmp(file_name, "stdout") == 0)

fp = stdout;

else

if (! fp = fopen(file_name, "w")) { fp = NULL; }
return(1); /* file_name cant be opened to write */

end

main_code

/* dont start printing each line until there is data in each 1/p fifo */
/* and there is enough room on the output fifos

while (length_input_fifo(0) > 0) {

if (get(0, &x) >= 0) {

++sample_num;

if (sample_num >= print_after) {

fprint(fp, "%e", (float)x);

/* print newline

fprint(fp, "\n");

};

return(0); /* at least one input fifo is empty */

end

SIGN

purpose: this routine outputs the sign of the incoming sample.

input 0: x
output 0: sign(x)
model: x = sign(x)
parameters: none

no_input_fifos 1
no_output_fifos 1
declarations
SAMPLE x
end

main_code

```
/* make sure output fifo not full
if (length_output_fifo(0) == maxlength_output_fifo(0))
return(0); /* output fifo full */
/* transfer samples to output fifo from input fifo
while (get(0,x) >= 0) {
if (x > 0)
x = 1.0;
else
x = -1.0;
if (put(0,x) != 0) /* put sample on output fifo */
return(0); /* output fifo full */
}
Return(0);
end
```

SQUARER

purpose: this star implements a squaring nonlinear function.

input 0 x sample
output 0 x sample
mode 1
parameters: none

#include <math.h>

no_input_fifos 1
no_output_fifos 1

declarations
SAMPLE x;

end

main_code

while (length_output_fifo(0) < maxlength_output_fifo(0) &&
length_input_fifo(0) > 0) {

get(0, &x)

x = x * x

put(0, x)

return(1)

return(0)

end

/* x -> o/p fifo */
/* lost sample - o/p fifo full */

STARTER_BLOCK

purpose: this stat outputs a given number of zero samples the first time it is called, then does nothing.

input 0: x samples
output 0: x samples

model: x[n] = 0 n < samples_delay

parameter 1: samples_delay
-this is number of zero samples to dump

param samples_delay int

no_input_fifos 1

no_output_fifos 1

state zero_count int initialize 0

declarations

int return_code

SAMPLE x

end

main_code

/* Put out zero samples until zero_count = delay */

if (zero_count < samples_delay)

while (zero_count < samples_delay) {

/* try to put out a zero sample */

return_code = put(0,0,0);

if (return_code >= 0)

/* zero was put out, so increment zero_count */

++zero_count;

if (return_code != 0)

return(0); /* output fifo now full */

/* enough zeros have been put out, so do nothing */

if (zero_count >= samples_delay) {

while (length_input_fifo(0) > 0 &&

length_output_fifo(0) < maxlength_output_fifo(0)) {

get(0,0,x);

put(0,x);

}

return(0);

end

SUMMER

purpose: this star sums the samples from two
input fifos and outputs the result to the
output fifo.

input 0 x samples
input 1 y samples
output 0 z samples
model : z = x + y
parameters : none

no_input_fifos 2
no_output_fifos 1
declarations
SAMPLE x,y,z;
end

main_code
while length_input_fifo(0) > 0 &&
length_input_fifo(1) > 0 &&
length_output_fifo(0) < maxlength_output_fifo(0) {
get(0,x);
get(1,y);
z = x + y;
put(0,z);
}
return(0);
end

TRAN_FILE_FROM_FILE

purpose: this star takes in sampled data and filters it with the filter coefficients coming from a file

input 0 sampled data stream
output 0 filtered samples

mode:

$y[k] = \sum_{n=0}^N h[n] * x[k-n]$

parameter 1 int num_taps default 40 (must be even)
parameter 2 file file_name

description: this block is being implemented for a 40 tap filter.

#include <math.h>
#include "plotter.c"

define pi 3.14159

no_input_fifos 1
no_output_fifos 1

state h float
state old float
state sample_num int
state fp FILE*

param num_taps int default 40
param file_name file

declarations

SAMPLE x,y
int count;
FILE *fpopen();

end

initialization_code

/* allocate memory for coefficient array */

h = (float*) calloc(num_taps, sizeof(float));
/* storage for previous input samples */

old = (float*) calloc(num_taps, sizeof(float));
/* initialization to zero
for (count = 0; count < num_taps; count++)
old[count] = 0.0;
h[count] = 0.0;
/* set up coefficients */

if (fp fopen(file_name, "r")) == NULL,
return(1); /* can't open file name */

count = 0
/* assume want to throw away every second sample in data file
file is a phase and we want only a phase */

while(fscanf(fp "%f %f", &x, &y)) != EOF && count < num_taps {
h[count] = x;
count++;
}

if (count < num_taps) {
return(2); /* not enough coefficients have been read */
}

sample_num

end
main code

while(length input_fifo(0) > 0 && length output_fifo(0) < maxlength output_fifo(0))

if (get(0, &x) > 0) {
for (count = num_taps; count > 1; count--)
old[count+1] = old[count];
old[0] = x;
}

else
return(10);

sample_num

for (count = 0; count < num_taps; count++) {
x = h[count] * old[count];
printf("%f\n", old[count] + x);
}

if (put(0, x) < 0)

return(11);

return(0);

end

TRANSMIT_FILTER

purpose: this routine takes a sample from the input fifo and outputs it a number of times depending on the duty cycle and the number of phases required.

input 0 samples
output 0 N samples

model : o(t) = I(t) n

parameter 1 int n default 4
2 int duty default 2

description- n is the number of phases that are needed for each input bit
duty is the number of the n phases that should be equal to the input bit and the rest (ie. n-duty bits) will be set to zero.

no_input_fifos 1
no_output_fifos 1

param n int default 4
param duty int default 2

declarations
SAMPLE x
int i
end

main_code

```
while( (length_output_fifo(0) + n) <= maxlength_output_fifo(0) &&
length_input_fifo(0) > 0 ) {
    if( get(0,x) < 0 )
        return(1);
    for( i=0; i<duty; ++i)
        if( put(0,x) < 0 )
            return(2); /* shouldn't happen */
    for( i=0; i < (n-duty); ++i)
        if( put(0,0) < 0 )
            return(3);
    return(0);
}
end
```

```

*****
VCO
*****
purpose: this star implements a voltage controlled
oscillator.
input 0 theta_in
output 0 phi_out
time_error = time_error_pre + time_error_pre*omega_o*t_s + theta_in*gain
model: phi_out = phi_out_pre + omega_o*t_s + theta_in*gain
parameter 1 float t_s default 1/(4*144000)
           - sampling period
           2 float gain default 1.0
           - vco gain
           3 float omega_o default 2*pi*144000.0
           - vco center frequency
           4 float phase_error_limit default 1.0 is no limit
*****

#define pi 3.14159
no_input_fifos 1
no_output_fifos 2
state phi_out_pre float initialize 0.0
state theta_in_pre float initialize 0.0
state time_error_pre float initialize 0.0

param t_s float default 1.73611e-6
param gain float default 1.0
param omega_o float default 9.04779e+5
param phase_error_limit float default 1.0

declarations
SAMPLE theta_in, phase_error
SAMPLE phi_out, time_error, small_delta
end

main_code
while (length(input_fifo(0)) > 0 &&
length(output_fifo(0)) < maxlength(output_fifo(0)) &&
length(output_fifo(1)) < maxlength(output_fifo(1))) {
get(0, theta_in);
phase_error = ((theta_in - theta_in_pre) * t_s * gain) / 2.0;
if (phase_error > phase_error_limit)
phase_error = phase_error_limit;
else if (phase_error < (-1.0) * phase_error_limit)
phase_error = -1.0 * phase_error_limit;
phi_out = phase_error + phi_out_pre + omega_o*t_s;
}

```

```

if (phi_out >= (2*pi))
phi_out = phi_out - (2.0 * pi)

phi_out_pre = phi_out;
theta_in_pre = theta_in;
time_error = phase_error / omega_o
small_delta = time_error / time_error_pre
if (put(0, phi_out) < 0)
return(1);
if (put(1, small_delta) < 0)
return(2);
return(0);
end

```

APPENDIX G

JITTER SPECTRUM ESTIMATION PROGRAM

```

.....
S_DELTA_F

This program generates the spectral response of a set
of data that is contained in a specified file. currently
the number of samples used to calculate the expectation
is the maximum that it can use.
This program makes use of periodogram theory.

The resolution is currently set for 72 kHz in 1.0 KHz increments.
The results are then plotted in db and the absolute values
are printed after the graph.

!!!!!! NOTE Input values are multiplied by 1e6 to keep magnitude
of numbers large enough for single precision math.
To offset this 12 dB are subtracted from the resultant
spectrum values.
.....
char file_name[20];
double to_plot[10000], to_plot_2[10000], data[40000], dec_data[10000];
int start, last;
int count = 0;

typedef struct {
    double re, im;
} COMPLEX;

#include <stdio.h>
#include "plotter.c"

read_data()
{
    printf("file_name = ");
    scanf("%s", file_name);
    printf("start point = ");
    scanf("%d", &start);
    printf("last point = ");
    scanf("%d", &last);

    main(argc, argv)
    int argc;
    char *argv[];
    {
        FILE *fp, *fopen();
        char *strcpy();
        int num_pts;
        int NUM_FREQ = 72;
        int NUM_EXPECT = 500;
        float PI = 3.1415926;
        float T;
        COMPLEX X[10000];
        int N, i, j, m, n, M, k;
        float dec_size;
        int dec_data;
        float expect[10000];
        double x, xr, xi, sin(), cos(), floor(), sqrt(), log10();
    }
}

```

```

/* read in command line arguments if they are given */
if(argc == 1)
    read_data();
else if(argc == 2) {
    strcpy(file_name, argv[1]);
    start = 0;
    last = 5000;
}
else if(argc == 4) {
    strcpy(file_name, argv[1]);
    sscanf(argv[2], "%d", &start);
    sscanf(argv[3], "%d", &last);
}
else
    printf("ERROR in read_data\n");

if((fp=fopen(file_name, "r")) == NULL) {
    printf("ERROR with file\n");
}
else {
    while(fscanf(fp, "%E", &x) != EOF && count < last) {
        ++count;
        if(count >= start)
            data[count-start] = x * 1000000.0; /* to keep workable magnitude */
    }

    num_pts = count - start;
    dec_size = (int) floor((double) num_pts / 4.0);
    T = 1.0 / 144000.0;

    /* DECIMATE INPUT DELTA'S TO THE BAUD RATE (ie 1/4) */
    for(i=0; i<dec_size; ++i) {
        dec_data[i] = data[i*4];
    }

    /* FIND EXPECTATION OF DELTA(N)'s.
    for(n=1; n<NUM_EXPECT; ++n) {
        k=0;
        M=0;
        expect[n] = 0.0;
        while(k*n < dec_size) {
            ++M;
            expect[n] += dec_data[k] * dec_data[k*n];
            ++k;
        }
        expect[n] = expect[n] / M;
    }

    /* CALCULATE REAL AND IMAG. PARTS OF SPECTRAL DENSITY
    for(j=0; j<NUM_FREQ; ++j) {
        F = 1000.0 * j; /* freq increment in Hz
        X[j].re = 0.0;
        X[j].im = 0.0;
        for(i=0; i<NUM_EXPECT; ++i) {
            X[j].re += expect[i] * cos(-2.0*PI*F*i*T);
            X[j].im += expect[i] * sin(-2.0*PI*F*i*T);
        }
    }
}

```

Figure G.1: Spectral Estimation Program Listing

```

to_plot[j] = 0.0 ;
}
for( j=0 ; j<NUM_FREQ ; ++j){
    xr = X[j].re ; X[j].re ;
    xi = X[j].im ; X[j].im ;
    x = xr + xi ;
    to_plot[j] = sqrt(x) ;
    to_plot[j] = to_plot[j] * 144000.0 * 1e-12 ;
}
/* NOW TO LOG SCALE -----*/
for( i=0 ; i<NUM_FREQ ; ++i) {
    if(to_plot[i] == 0.0 )
        to_plot[i] = 1.0e-32 ;
    to_plot_2[i] = log10(to_plot[i]) - 12.0 ;
}

```


END

2 2 0 4 8 6

FIN