9950-1570

MOTOROLA INC.

31 January 1990

In reply refer to:
DLN-017/5767

2 / 77

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, California 91109

Attention: J. H. McConkey
Contract Negotiator Specialist
M/S 190-207

Subject: Contract 958596
Final Report Submittal

Reference: (a) DLN-011/5767, Software Breadboard Study, 30 November 1989

Enclosure: (1) Software Breadboard Study Final Report, 31 January 1990 (2 copies)

Gentlemen:

Enclosure (1) is forwarded in accordance with Article 2 (b) (2) of the subject contract. This submittal includes the verbal comments received from JPL on the reference (a) submittal.

Please contact the undersigned, 602-732-3619, for any clarifications you may desire.

Sincerely,

MOTOROLA INC., GEG

J. C. McDearmon for

D. L. Nelson
Contract Administrator
TT&C Program Office

(NASA-CR-190473) SOFTWARE BREADBOARD STUDY Final Report (Motorola) 177 p

N92-30757

Unclas

G3/33 0104858

31 January 1990

SOFTWARE BREADBOARD STUDY
FINAL REPORT

Strategic Electronics Division
Aerospace Electronics Office
Aerospace RF Section

This document was prepared for JPL
under contract No. 958596

Prepared By:

C. Nuckolls 1/29/90

C. Nuckolls
Project Leader

M. Frank
Project Engineer

Approved By:

P. C. Bartholomew

P. Bartholomew, Manager
Aerospace RF Section

Robert K Hansen

R. Hansen
Chief Engineer

MOTOROLA INC.
Strategic Electronics Division
2501 S Price Rd.
Chandler, AZ 85248-2899

PRECEDING PAGE BLANK NOT FILMED

INTRODUCTION

This report summarizes the results of a study performed for the Jet Propulsion Laboratory (JPL) under contract 958596. The overall goal of this study was to develop new concepts and technology for the Comet Rendezvous Asteroid Flyby (CRAF), Cassini, and other future deep space missions which maximally conform to the Functional Specification for the NASA X-Band Transponder (NXT), FM513778 (preliminary, revised July 26, 1988). The study is composed of two tasks.

The first task was to investigate a new digital signal processing technique identified in a previous program, "A Transponder Study" (contract no. 958377). This technique involves the processing of 1-bit samples and has the potential for significant size, mass, power and electrical performance improvements over conventional analog approaches. The entire X-band receiver tracking loop was simulated on a digital computer using a high-level programming language as shown conceptually in Figure A. The approach shown in this figure had the best performance in terms of size, mass, power and cost of all those analyzed in the "A Transponder Study" final report. Simulations on this "software breadboard" showed the technique to be well-behaved and a good approximation to its analog predecessor from threshold to strong signal levels in terms of tracking-loop performance, command signal-to-noise ratio and ranging signal-to-noise ratio. The successful completion of this task paves the way for building a hardware breadboard, the recommended next step in confirming this approach is ready for incorporation into flight hardware.

The second task in this study was to investigate another technique identified during the "A Transponder Study" program. It was shown in the "A Transponder Study" final report that a substantial size, weight and power savings could be obtained by using sampling techniques to implement down-conversion mixers. This sampling mixer approach is also shown conceptually in Figure A. The approach provides considerable simplification in the synthesis of the receiver first LO over conventional phase-locked multiplier schemes and in this block diagram approach, provides down-conversion for an S-band emergency receive mode without the need of an additional LO. The objective of this study was to develop methodology and models to predict the conversion loss, input RF bandwidth and output RF bandwidth of a series GaAs FET sampling mixer and to breadboard and test a circuit design suitable for the X and S-band down-conversion applications shown in Figure A. Considerably more effort than planned was required to develop models and methodology to predict sampling mixer performance. This was primarily due to the non-linear nature of the sampling mixer which required lengthy time-domain SPICE analysis and the immaturity of present day GaAs FET switching models. Models and methodology sufficient to match test results obtained on a existing L-band sampling mixer circuit were

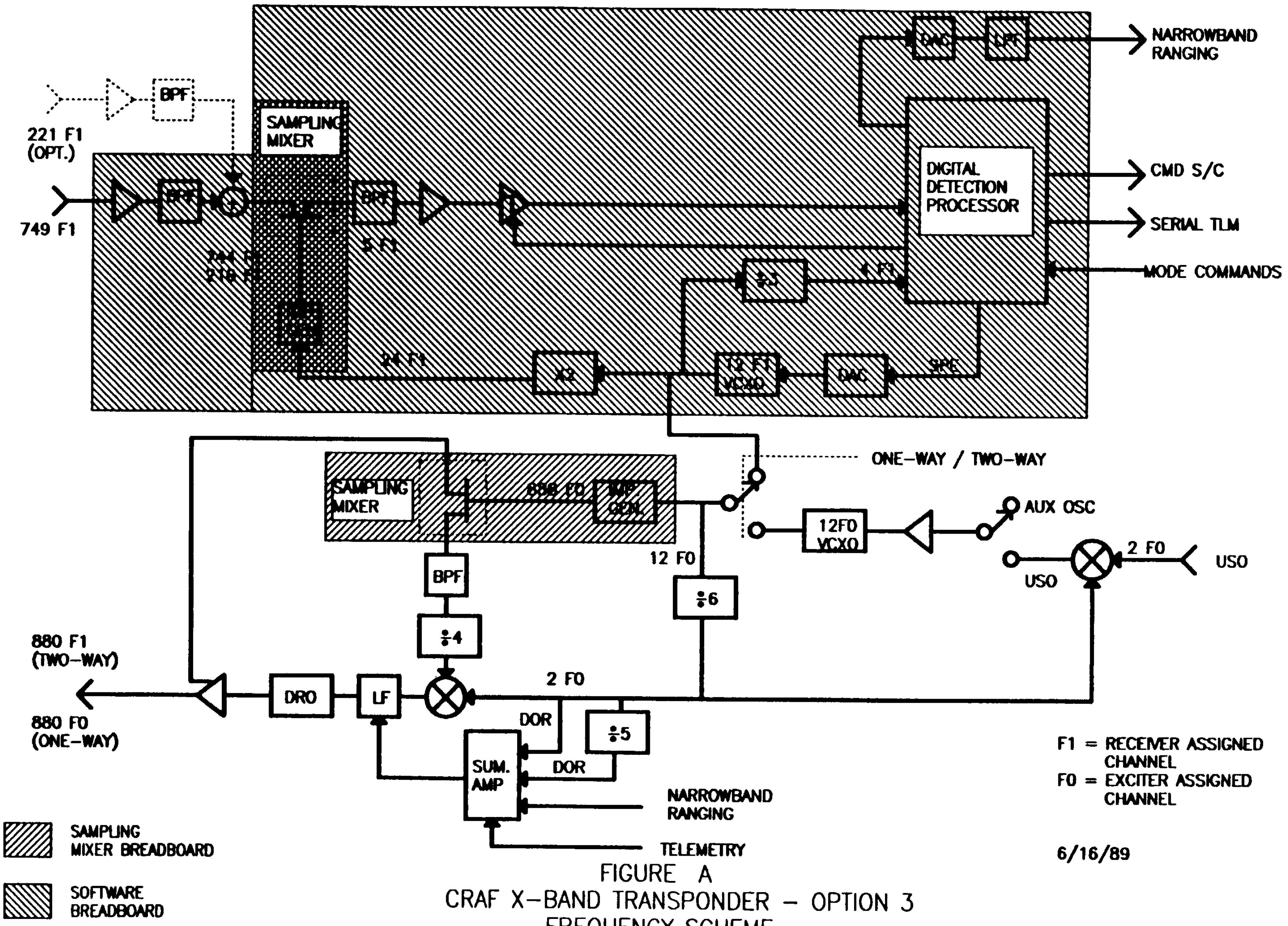


FIGURE A
CRAF X-BAND TRANSPONDER – OPTION 3
FREQUENCY SCHEME

developed and additional circuit modifications were found which give a flat IF passband. Analysis of X-band sampling mixer circuits based on the L-band model suggests that an implementation with the conversion loss of a traditional diode mixer can be achieved. The study was terminated by JPL at this point when it became apparent that a breadboard could not be built with the remaining funds. Further work done during the preparation of this report shows that X-band sampling mixer implementations will likely require Microwave Monolithic Integrated Circuit (MMIC) or hybrid Microwave Integrated Circuit (MIC) technologies to achieve the required low values of parasitic inductance and capacitance. These realizations would have been beyond the scope of the original funding. Further work to build an X-band MIC hybrid sampling mixer is recommended as the most cost effective way to develop this important technology into flight readiness.

This report is divided into two sections. Part A describes the simulation model and analysis results for the software breadboard task. Part B describes the methodology, models and analysis results for the sampling mixer task. Conclusions and recommendations for further work can be found at the end of each part, as applicable.

TABLE OF CONTENTS

Page

PART A SOFTWARE BREADBOARD STUDY

TABLE OF CONTENTS

(Continued)

LIST OF FIGURES

Page

Page

Page

LIST OF TABLES

PART B SAMPLING MIXER STUDY

LIST OF FIGURES

LIST OF TABLES

PART A

SOFTWARE BREADBOARD STUDY

I. Introduction

In this report, a summary of the testing of a software breadboard of the digital signal processing technique developed in the previous program, "A Transponder Study," (contract no. 958377) will be presented. The digital signal processing involves using a 1 bit analog-to-digital (A/D) converter to sample the input signal, and then processing these samples using digital techniques. A block diagram of the proposed circuit is shown in Fig. 1.

The software breadboard is a Monte Carlo simulation of the block diagram of Fig. 1. The analog signals in the block diagram were represented using double precision floating point variables, and the digital signals in the block diagram were represented using integer variables whose word size corresponded to the hardware implementation.

The goals of this simulation study include verifying the results of "A Transponder Study" (ATS), as well as investigating some areas of concern which were not thoroughly analyzed in the study. The results of ATS which will be verified are:

a) The signal-to-noise (SNR) degradation from using 1 bit sampling, and

b) The linear response of the 1 bit A/D for negative SNR.

The areas of concern which were identified in ATS include:

a) At positive SNR levels, what are the effects of the 1 bit nonlinearity on command and ranging intermodulation?

b) At positive SNR levels, what are the effects of the 1 bit nonlinearity on the phase-locked loop transient behavior?

c) What are the effects, and what trade-offs can be made on the VCO phase jitter due to the digital-to-analog converter?

The outline of this report is as follows: in Section II, a description of the program used to perform the Monte Carlo simulation of the block diagram of Fig. 1 will be given. This section will also discuss some of the models against which the outputs of the simulation program were compared. In Section III, the results of the simulation study will be given, and in Section IV, conclusions and recommendations will be discussed.

II. Simulation Program

A. Program Description

As stated in the introduction, the simulation program is a software implementation of the block diagram of Fig. 1. For this reason, the functions of the block diagram will be discussed, and their software implementations will be noted.

The input signal is assumed to be a sine wave phase modulated by both command and ranging tones. The input signal takes the form:

$$s(t) - A\cos(\omega_0 t + B_c \cos\omega_c t + B_r \cos\omega_r t) \qquad (1)$$

where, unless otherwise noted,

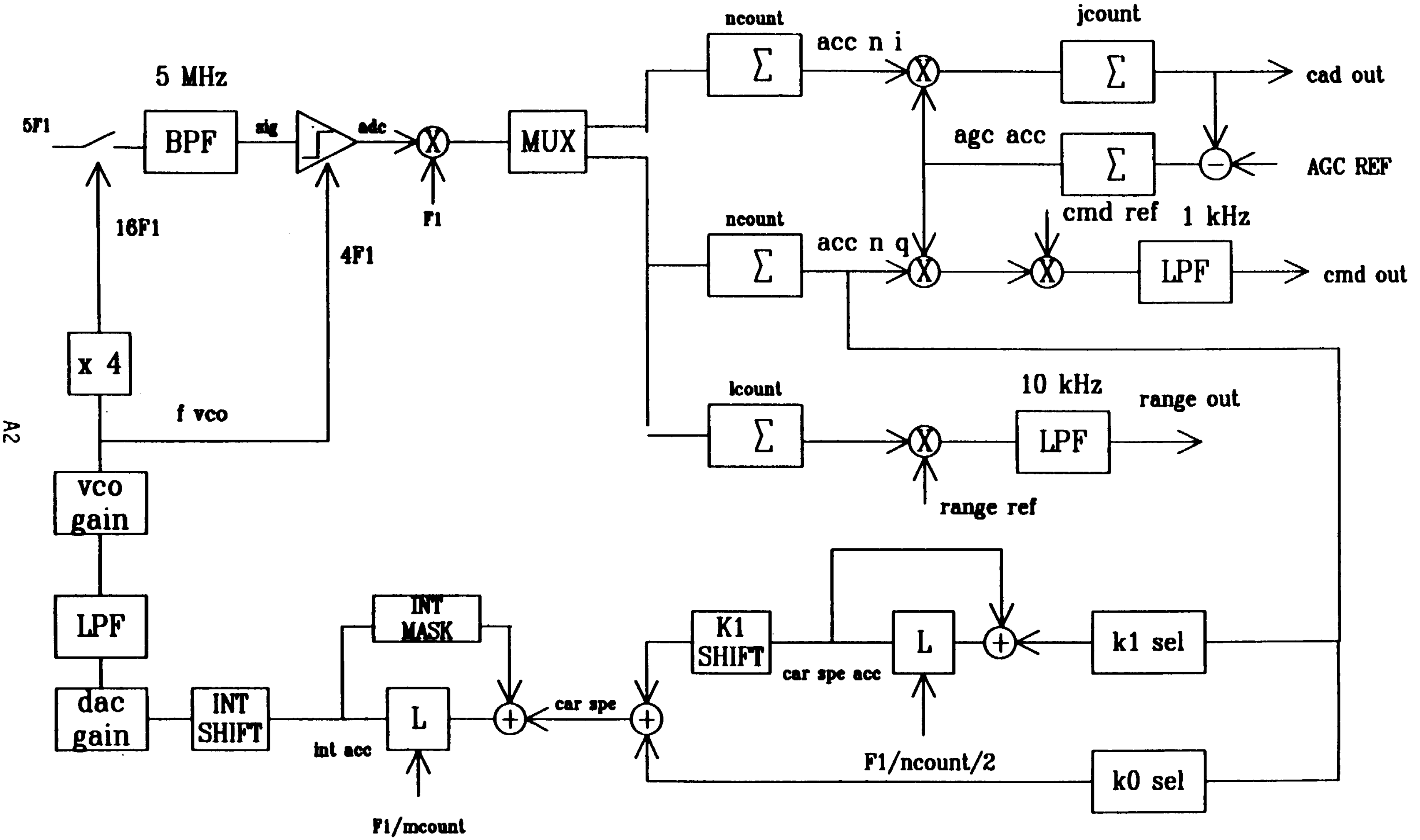


Fig. 1. Software breadboard block diagram.

f_0 = carrier frequency = 5*F1
F1 = 9.56558642 MHz
B_c = command modulation index = 0.9 rads
f_c = command frequency = 16 kHz
B_r = ranging modulation index = 0.785 rads
f_r = ranging frequency = 1 MHz

Zero mean, Gaussian noise is added to the input signal. The noise generator has a variance of 1, so that

$$\sigma^2 = N_0 B = 1 \qquad (2)$$

where B is the Nyquist rate which is equal to one half the simulation sampling frequency (i.e., 16*F1). Note that the input signal-to-noise density, which is an input parameter to the program is given by

$$\text{SND} = \frac{A^2}{2\ N_0} = \frac{A^2\ B}{2}\ \text{dB-Hz}. \qquad (3)$$

The input signal and noise are filtered by a 2 pole digital Butterworth filter. The filter's magnitude and phase responses are shown in Figs. 2 and 3, respectively. The filtered signal is then sampled by a 1 bit A/D whose output is +1 if the signal is greater than zero, and -1 if the signal is less than zero.

The digitized samples are then converted to baseband by inverting the sign of every other pair of samples. The odd and even samples are then separated into I and Q channels, respectively. The I and Q channels are filtered by integrate and dump (I/D) filters of length 'ncount'. Unless otherwise noted, 'ncount' is set to 256, so that the two-sided noise bandwidth of the I/D filters is

$$B_n = \frac{4*F1}{2*ncount} \approx 75\ \text{kHz}. \qquad (4)$$

Note that this value of 'ncount' requires a 9 bit word at the output of the I/D filter (-256 -> 255). A separate I/D filter is used in the Q channel as the ranging filter. The length of this filter is equal to 'lcount' which unless otherwise noted is set to 8. This value results in a noise bandwidth of

$$B_n = \frac{4*F1}{2*lcount} \approx 2.4\ \text{MHz}. \qquad (5)$$

This value of 'lcount' will give approximately 3 dB of attenuation to a 1 MHz ranging tone.

The output of the in-phase I/D filter is sent to a coherent automatic gain control (AGC) and then to a final I/D filter whose output gives the coherent amplitude detection (CAD). The length of the final I/D filter is equal to 'jcount' (nominally equal to

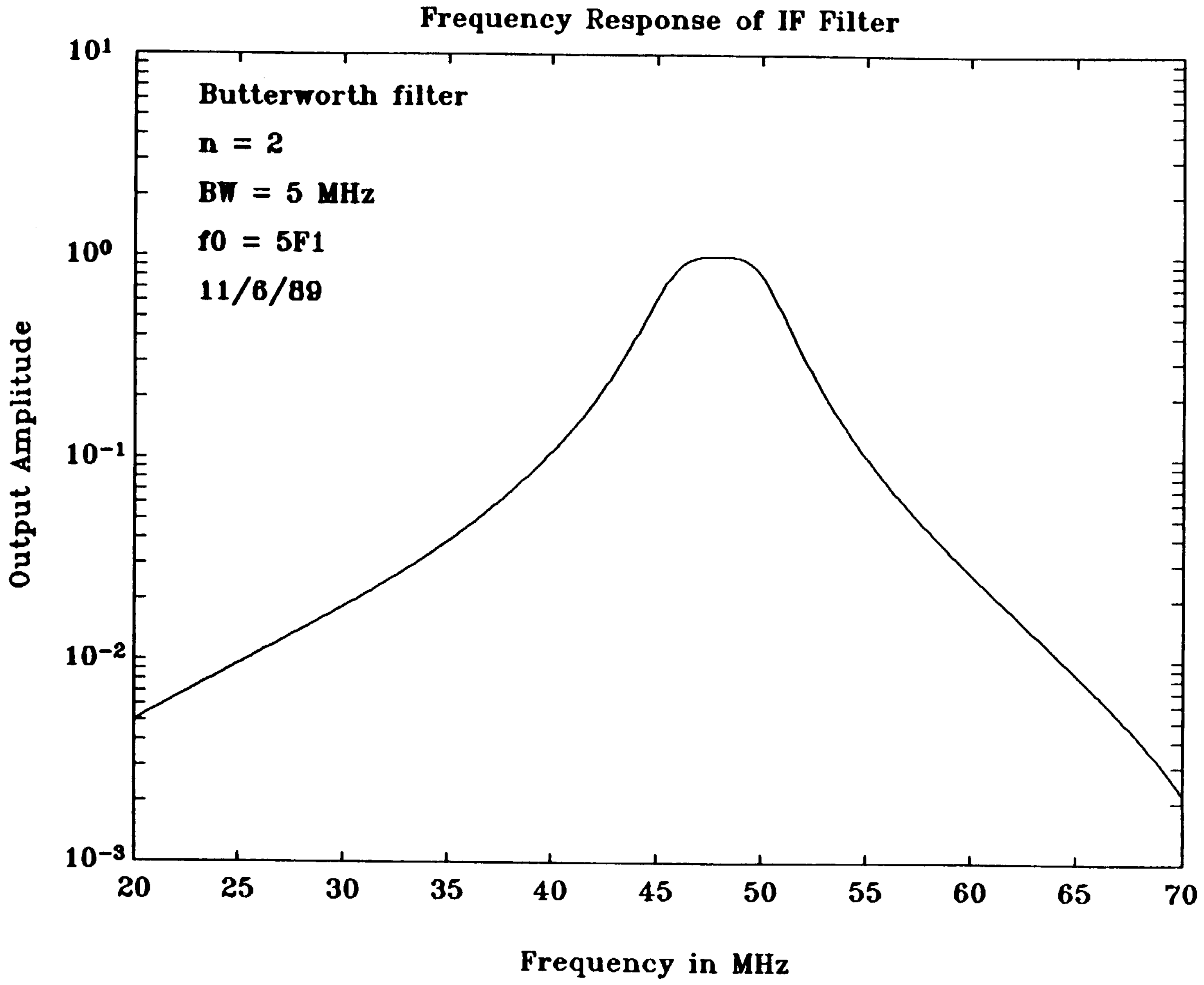


Fig. 2. Magnitude response of IF bandpass filter.

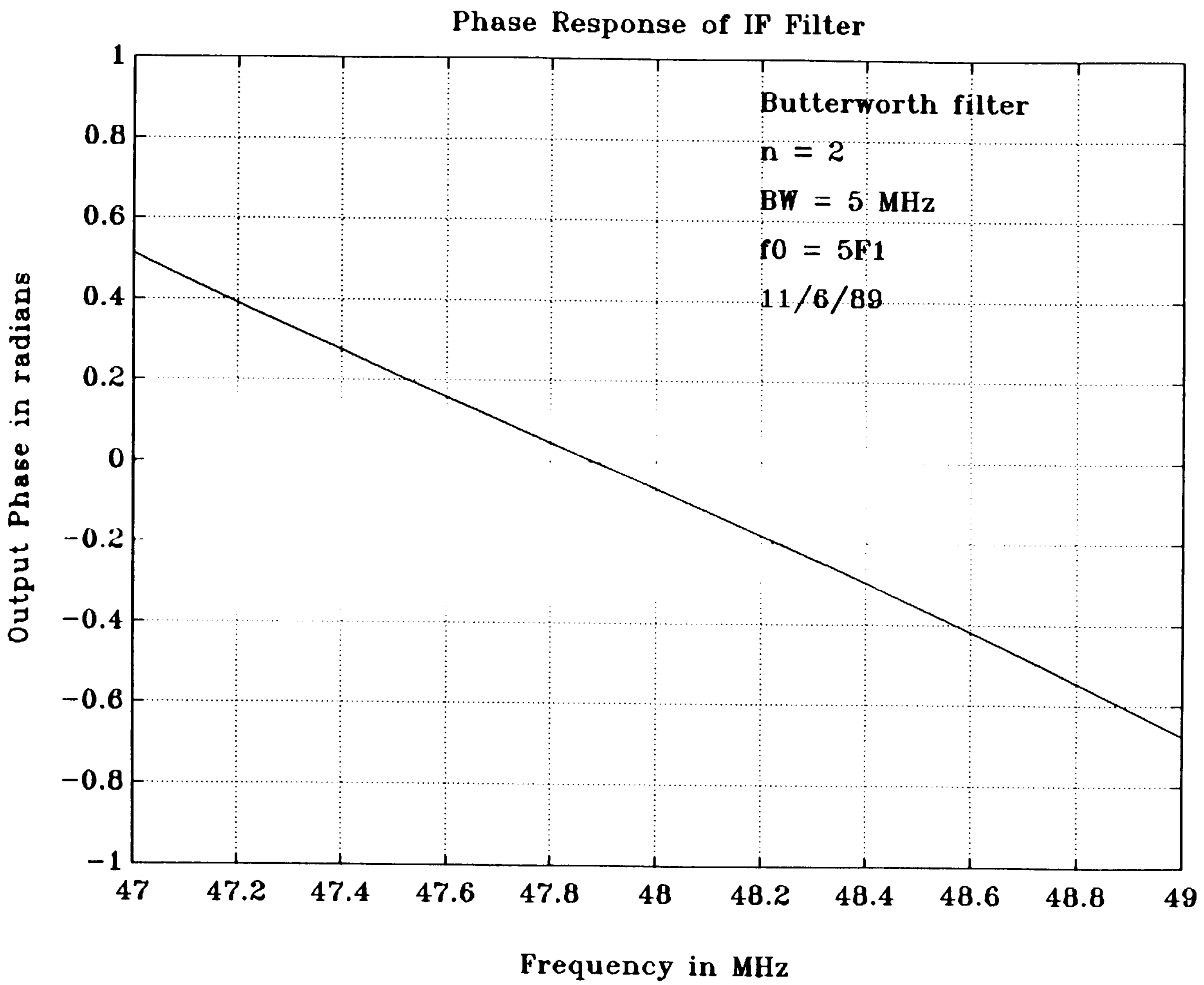


Fig. 3. Phase response of IF bandpass filter.

10), so that the noise bandwidth of the CAD output is

$$B_n = \frac{4*F1}{2*ncount*lcount} \approx 7.5 \text{ kHz.} \qquad (6)$$

The CAD signal is at baseband, so that the SNR ratio of this output can be calculated by finding the sample mean and sample standard deviation (described in the next section) directly.

The output of the quadrature I/D filter is also AGC'd. The AGC output would normally be bandpass filtered and sent to the command detection unit (CDU). However, in the software breadboard we are interested in obtaining statistics of the command signal. Therefore, we instead mix the command signal to DC and low pass filter the mixer output. The magnitude response of the digital low pass filter is shown in Fig. 4. Note that the two-sided noise bandwidth of this filter is 2 kHz. The estimated command signal-to-noise ratio is obtained from the samples output from the low pass filter.

A similar down-convert and low pass filter operation is used on the ranging filter output. The magnitude response of the ranging low pass filter is shown in Fig. 5, note that its two-sided noise bandwidth is 20 kHz. The estimated SNR of the ranging signal is obtained from the samples of the low pass filter output.

The quadrature I/D filter output is used as the phase error in the phase-locked loop (PLL). A block diagram representation of the PLL is shown in Fig. 6. This linear model will be used to obtain baseline (phase and frequency) step responses with which to compare the outputs of the software simulation. A z-transform representation of the PLL loop filters is shown in Fig. 7. Note that 'k1_sel' and 'k0_sel' are selectable integer gains used to set the response of the loop filter. Also note that this digital filter approximates an analog integral plus proportional loop filter. In addition to 'k1_sel' there is also an additional effective gain of 'K1_SHIFT'. Therefore, the total gain of this branch is given by

$$k1\_gain = k1\_sel * 2^{-K1\_SHIFT} \qquad (7)$$

The shift in (7) is necessary to insure that 'car_spe' (see Fig. 1) is a 20 bit word.

The 20 bit word, 'car_spe' is input to the interpolator, whose function is to reduce the number of bits down to that needed by the digital-to-analog converter (DAC). The update rate of the interpolator and, therefore, the DAC is F1/mcount. For the nominal value of 'mcount' = 2, this update rate is approximately 4.8 MHz. The output of the DAC is low pass filtered and subsequently input to the VCO to close the loop. Note that although the VCO puts out a frequency which sets the A/D sampling rate, it is possible to determine the effective phase of the VCO. This effective VCO phase is just the carrier phase at the (VCO) sample times.

The preceding discussion has been a brief summary of the operation of the software breadboard. In the following section we

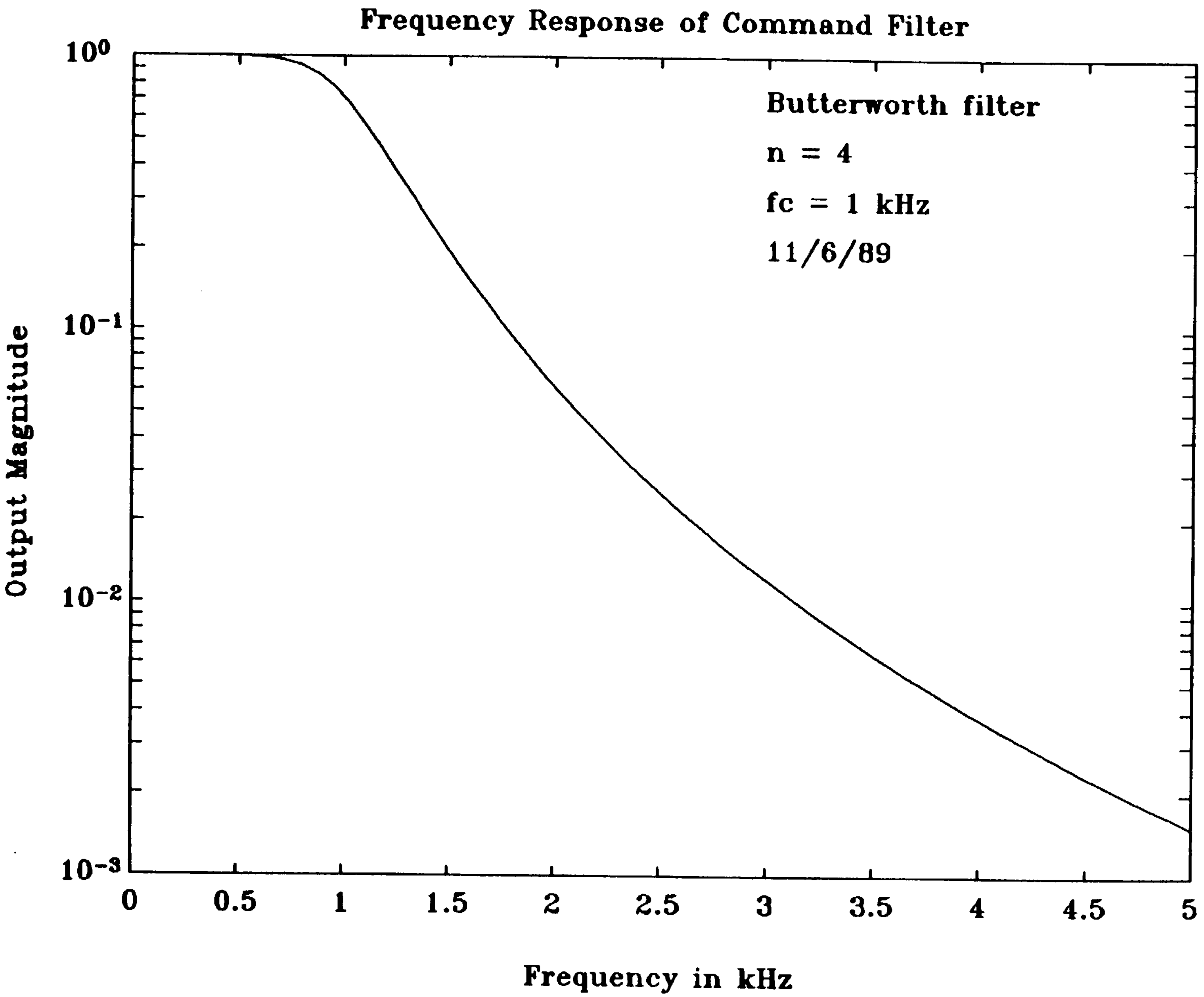


Fig. 4. Magnitude response of command low pass filter.

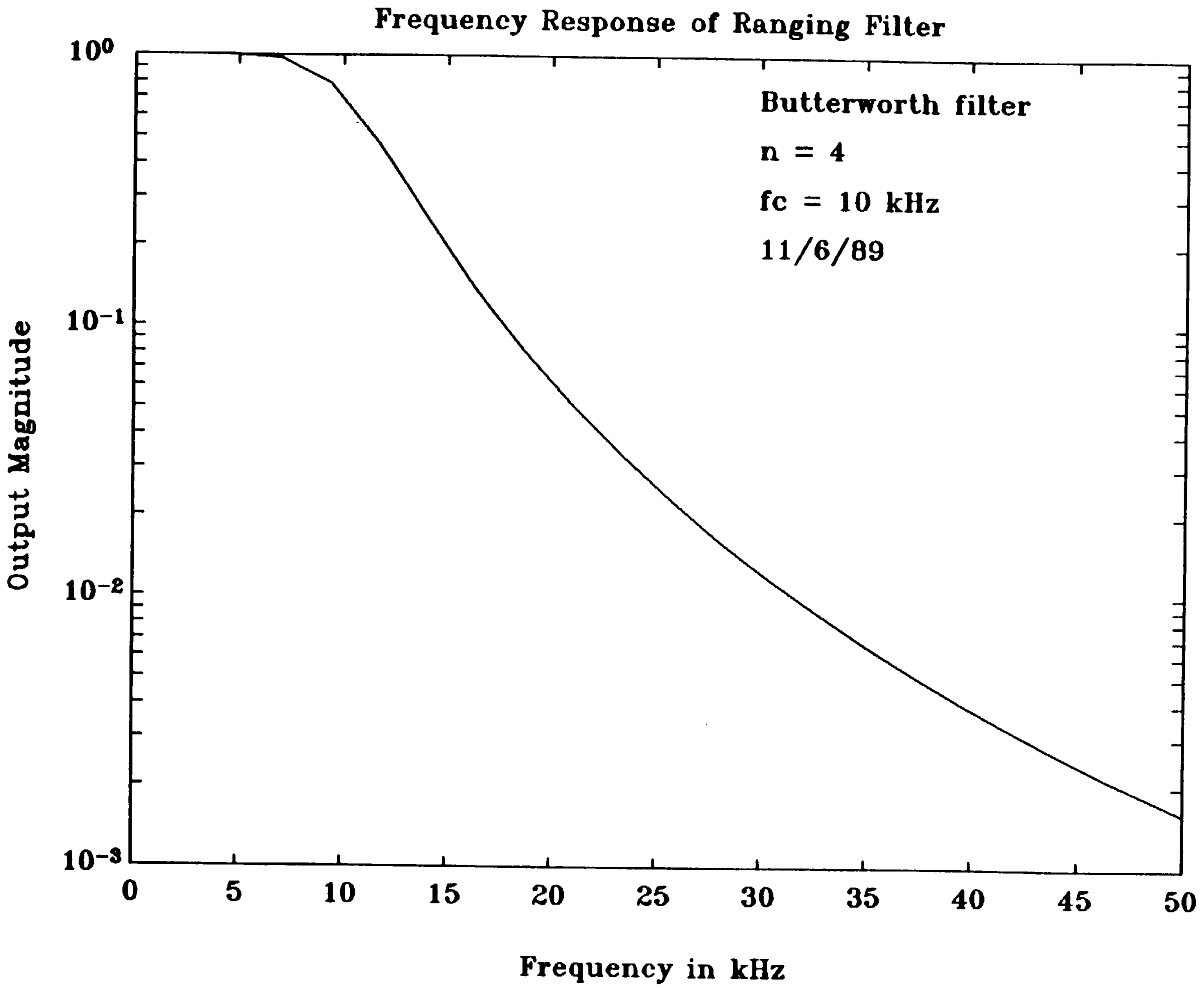
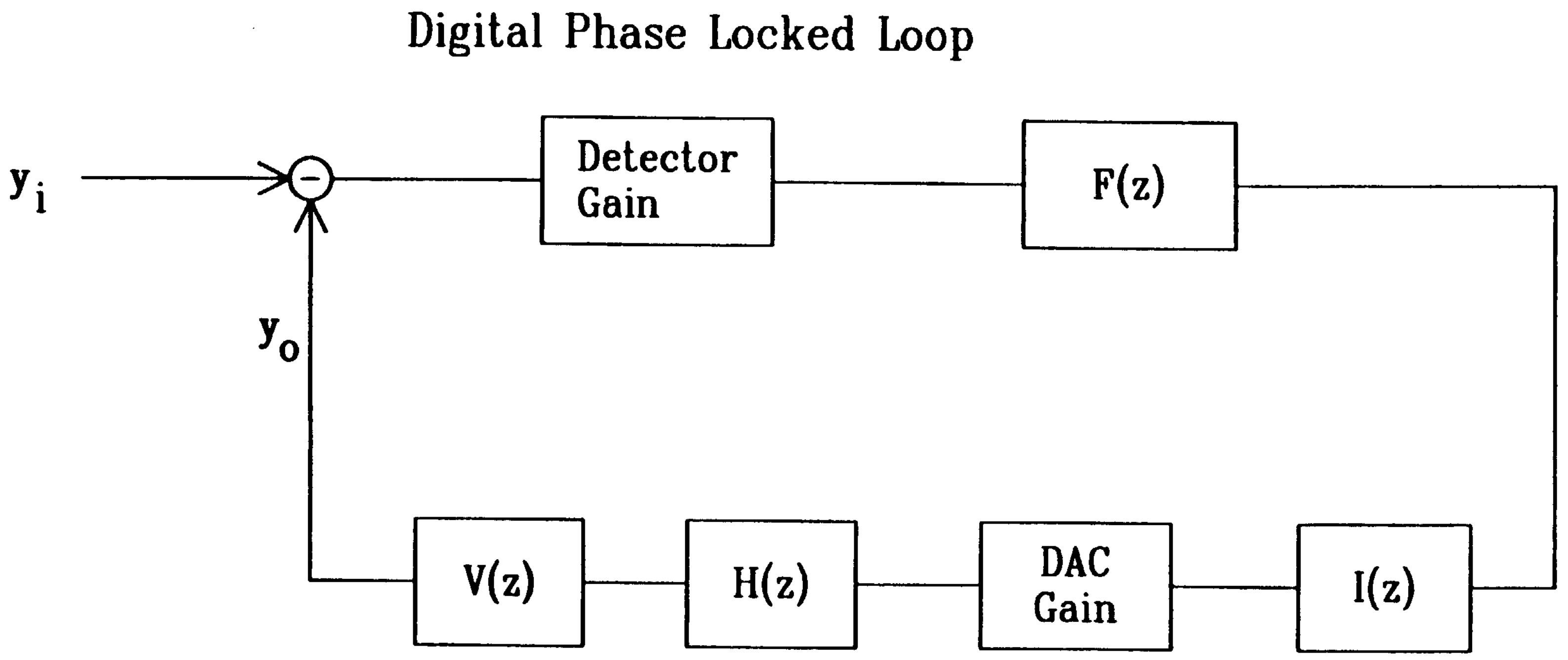


Fig. 5. Magnitude response of ranging low pass filter.



Detector Gain = (output signal level)*ncount
F(z) = Digital loop filter
I(z) = Interpolator filter
DAC Gain = D/A Converter Gain
H(z) = DAC low pass filter
V(z) = VCO transfer function

Fig. 6. Z-transform linearized model of carrier phase-locked loop.

Digital loop filter

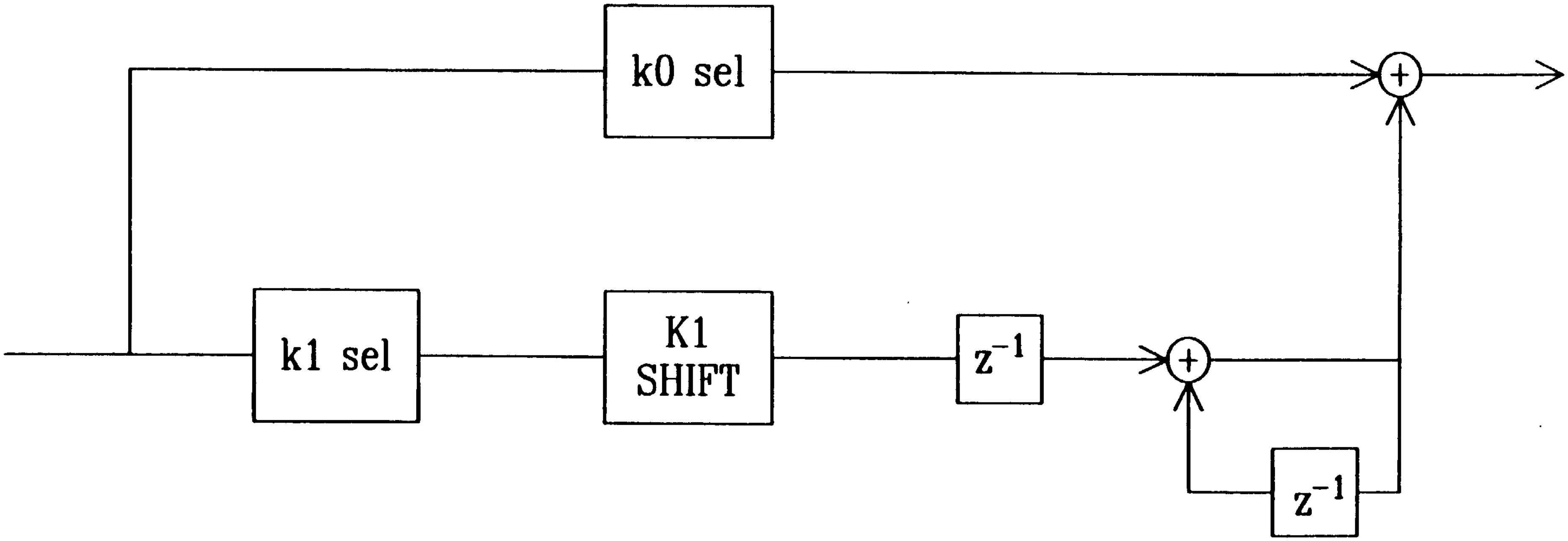


Fig. 7. Loop filter block diagram.

will outline the methodology of testing the breadboard to insure that it accurately reflects the hardware.

B. Software Breadboard Analysis

1) Introduction

In order to validate the integrity of the software breadboard, we need to develop some (possibly linearized) models of the breadboard and then compare the software simulation outputs to the outputs of these models. Three models which will be considered here to predict (and compare) the performance of the software breadboard are:

a) A linearized digital phase-locked loop [BOM87] model which will be used to predict the step response and stability of the PLL,

b) An equivalent analog PLL model which will be used to predict the frequency acquistion of the PLL, as well as the phase jitter of the loop, and

c) A statistical model of the output signal-to-noise ratio based on coherent 1 bit samples of the received signal plus noise.

The following subsections will discuss these models in more detail.

2) Digital Phase-Locked Loop Linear Model

In this section the z-transform model of the phase-locked loop used in the software breadboard will be given. For more background on this model or for a survey of digital PLLs, in general, see [BOM87] and [LIN81].

The block diagram of the digital PLL was given in Fig. 6. The loop filter was given in Fig. 7. For our analysis, we will model the detector gain as the signal level output from the 1 bit A/D times the I/D filter DC gain (i.e., 'ncount'):

$$K_{DET} = (\text{output signal level}) * \text{ncount} \qquad (8)$$

The output signal level can be predicted from the input signal level and the input noise variance as [ATS89]:

$$\text{output signal} = 1 - 2*Q\left\{\frac{A}{\sigma}\right\} \qquad (9)$$

where A is the amplitude of the input signal, and σ is the standard deviation of the input noise.

The interpolator will be modeled as a straight gain of the form

$$K_{INT} = 2^{-(\text{INT\_SHIFT})} \qquad (10)$$

similarly, the DAC will contribute a gain of the form

$$K_{DAC} = 2^{-(dac\_bits - 1)} \quad \text{volts/bit} \tag{11}$$

We will assume that the cut-off frequency of the DAC filter is comfortably above the open loop bandwidth of the loop and can therefore be ignored. We can group all of these gains together to get

$$K = K_{DET} * K_{INT} * K_{DAC} \tag{12}$$

The VCO gain is given in units of rad/s/volt. The transfer function of the VCO is [BOM87]

$$V(z) = \frac{K_{VCO} * T}{z - 1} \tag{13}$$

where T (= ncount/2/F1) is the sampling time of the loop. Using the above definitions, and referring to Figs. 6 and 7, a set of open loop discrete time state equations can be written for the loop:

$$\begin{aligned} x(k+1) &= A*x(k) + B*u(k), \\ y(k) &= C*x(k) + D*u(k) \end{aligned} \tag{14}$$

where the state coefficient matrices are given by

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ T*K_{VCO} & T*K_{VCO} & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} K*k1\_sel \\ 0 \\ T*K_{VCO}*k0\_sel*K \\ 0 \end{bmatrix}$$

$$C = [0 \;\; 0 \;\; 1 \;\; 1], \quad D = [0] \tag{15}$$

the system equations were given in state variable form instead of transfer function form because the control system analysis package (i.e., MATLAB) we will be using prefers the state equation form. For example, to obtain the open loop bode plot for the digital PLL, the following MATLAB commands would be issued [MAT87]:

```
w            = logspace(-1,pi);
[mag,phase]  = dbode(A,B,C,D,1,w);
loglog(w,mag);                                    (16)
```

once the state coefficient matrices have been defined, MATLAB can also be used to find the impulse, step, and ramp responses as well as finding gain and phase margins of the loop.

3) Analog Phase-Locked Loop Model

The equivalent analog model of the PLL is shown in Fig. 8 [BOM87]. The following approximate relationships exist between the

Equivalent Analog Phase-Locked Loop

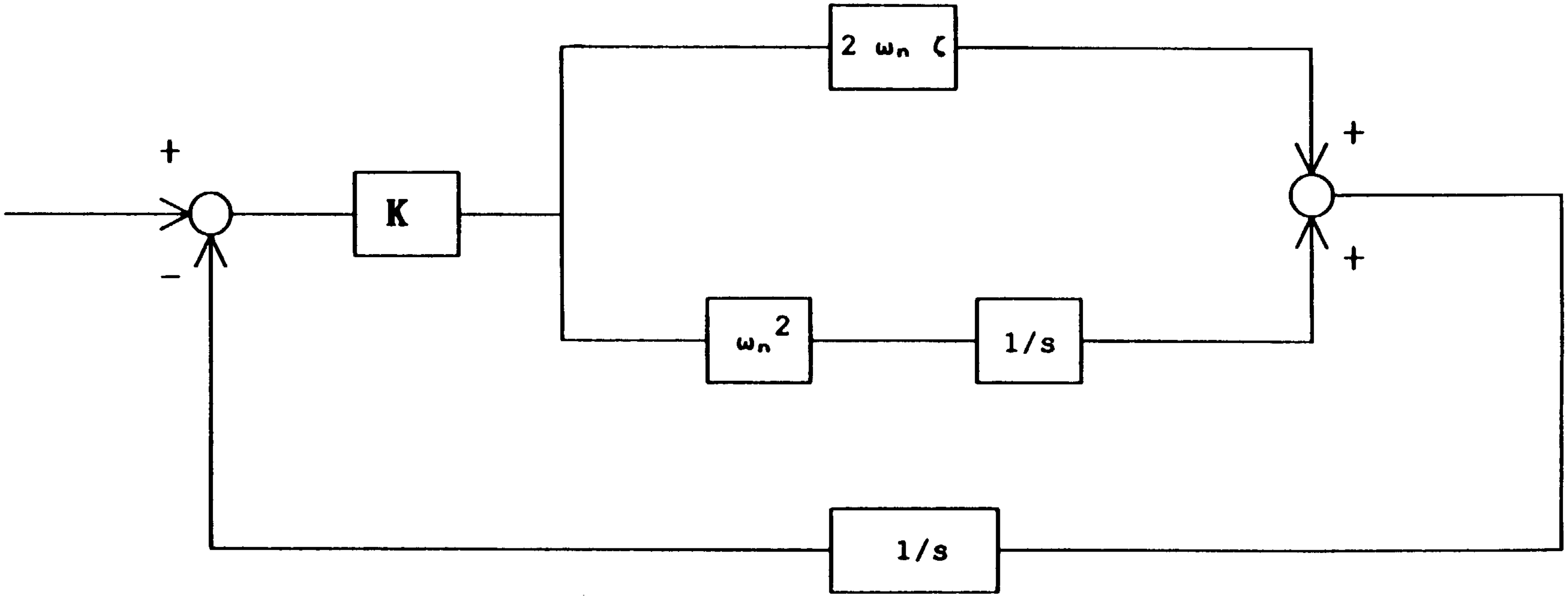


Fig. 8. Analog linearized model of carrier phase-locked loop.

components of the analog and digital PLLs:

$$\omega_n = \left[\frac{K_{VCO}*K*kl\_gain}{T} \right]^{1/2},$$

$$\zeta = \frac{k0\_sel}{2} \left[\frac{K_{VCO}*K*T}{kl\_gain} \right]^{1/2} \tag{17}$$

It is also well-known that the one-sided noise bandwidth of this second-order loop is

$$B_L = \frac{\omega_n}{2} \left\{ \zeta \cdot \frac{1}{4\zeta} \right\} \tag{18}$$

The VCO phase variance of the analog PLL is [ZIE85]:

$$\sigma_\theta^2 = \frac{2 * N_0 * B_L}{A^2} \tag{19}$$

This variance will serve as a baseline against which the digital PLL will be compared. We expect, in general, the phase variance of the digital PLL to be higher than (19) predicts because of the additional quantization noise.

We will also use the analog PLL to predict the frequency acquisition of the loop. During frequency acquisition, the PLL is known to operate outside the linear range of the phase detector. The loop's response can be written in terms of the following nonlinear ordinary differential equation (ODE) [BLA76]

$$\tau_1 \frac{d^2\phi}{dt^2} + K*\tau_2 \cos\phi \frac{d\phi}{dt} + K \sin\phi = 0 \tag{20}$$

where,

$$\tau_1 = \frac{K}{\omega_n^2}, \quad \tau_2 = \frac{2\,\zeta}{\omega_n} \tag{21}$$

and ϕ is the phase error. If we express (20) as a set of two coupled ODEs, the system can be solved numerically (as a function of time) using the Runge-Kutta routine on MATLAB. One such set of equations is

$$\dot{x}_1 = \frac{-K*\tau_2}{\tau_1} x_1 \cos(x_2) - \frac{K}{\tau_1} \sin(x_2)$$

$$\dot{x}_2 = x_1 \qquad (22)$$

where $x_2 = \phi$, the phase error. MATLAB returns a set of state vectors which can be plotted against time in order to observe frequency acquisition of the analog PLL.

4) Statistical Noise Model

a) SNR calculations

The model to be used for the prediction of the output signal-to-noise ratio is shown in Fig. 9. Only the in-phase channel (CAD) is being considered. Implicit in this model is the fact that the PLL is in lock, so that the input signal is being coherently sampled at the peaks of the sine wave. This model was analyzed in [ATS89], so we give only a brief summary of the results of that study here. Note that the SNR results were derived under the assumption of small input SNR, and will therefore only be valid as long as this assumption holds true.

The mean value of the signal output from the I/D filter is given by

$$\bar{y} = \sum_{i=1}^{M} \bar{D}$$

$$= M*\bar{D} \qquad (23)$$

where $\bar{D}$ is the mean value of the signal component out of the 1 bit sampler. This mean value can be expressed in terms of the Q() function as

$$D = 1 - 2*Q\left\{\frac{A}{\sigma}\right\} \qquad (24)$$

where σ is the standard deviation of the input noise. The noise variance out of the I/D filter is given by

$$\sigma_y^2 = \frac{2}{\pi} \sum_{i=1}^{M} \sum_{j=1}^{M} 2 \sin^{-1}[r_n(i-j)] - \sin^{-1}[r_n(i-j-1)] - \sin^{-1}[r_n(i-j+1)] \qquad (25)$$

where $r_n(i)$ is the normalized autocorrelation of the input noise:

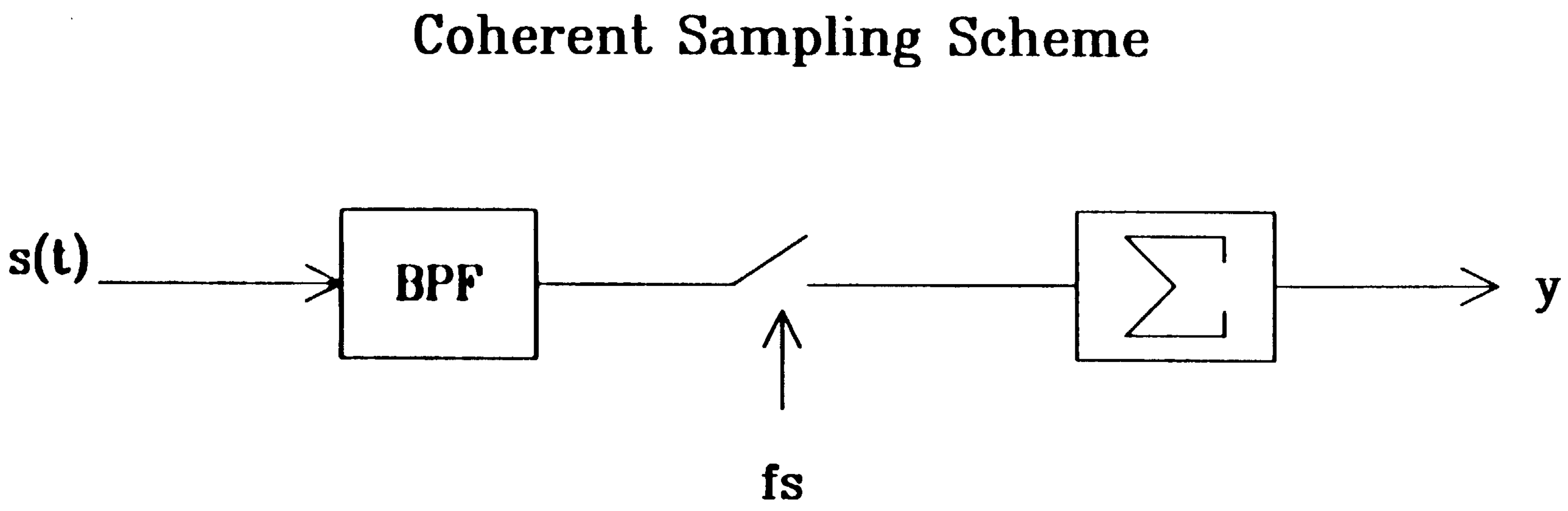


Fig. 9. Block diagram model of coherent sampling.

$$r_n(i) := \frac{R_n(i)}{\sigma^2} \qquad (26)$$

whose form is dependent on the 1 bit sampler pre-filter [ATS89].

b) Modulation of input signal

In order to calculate output SNRs, we need to know how much signal power is in each of the modulating signals. The input modulated signal is of the form

$$s(t) = A\cos(\omega_0 t + B_c \cos\omega_c t + B_r \cos\omega_r t). \qquad (27)$$

For coherent sampling, the signal in the I channel takes the form

$$\begin{aligned} s(t) &= A\cos(i2\pi + B_c \cos\omega_c i t_s + B_r \cos\omega_r i t_s) \\ &= A\cos(B_c \cos\omega_c i t_s + B_r \cos\omega_r i t_s) \\ &= A\cos(B_c \cos\omega_c i t_s) * \cos(B_r \cos\omega_r i t_s) \\ &\quad - A\sin(B_c \cos\omega_c i t_s) * \sin(B_r \cos\omega_r i t_s) \qquad (28) \end{aligned}$$

The cosine and sine functions in (28) can be expanded in terms of a Bessel series, see, for example, [ABR72]. From this expansion we can calculate the signal level of the various harmonic components. Table 1. summarizes the signal components for a command modulation index of 0.9, and a ranging modulation index of 0.785.

Table 1. I Channel Frequency Components

Frequency	Coefficient	Power (CMD mod=0.9, rangemod=0.785)
0	$J_0(B_c)J_0(B_r)$	-3.25 dB
$\omega_r \pm \omega_c$	$2J_1(B_c)J_1(B_r)$	-10.61
$\omega_r \pm 3\omega_c$	$2J_1(B_r)J_3(B_c)$	-39.61
$2\omega_c$	$-2J_0(B_r)J_2(B_c)$	-15.85
$4\omega_c$	$2J_0(B_r)J_4(B_c)$	-51.29

A similar development for the Q channel results in Table 2.

Table 2. Q Channel Frequency Components

Frequency	Coefficient	Power (CMD mod=0.9, rangemod=0.785)
ω_c	$2J_0(B_r)J_1(B_c)$	-3.20 dB
$3\omega_c$	$-2J_0(B_r)J_3(B_c)$	-32.21
ω_r	$2J_0(B_c)J_1(B_r)$	-4.64
$3\omega_r$	$-2J_0(B_c)J_3(B_r)$	-36.10
$\omega_r \pm 2\omega_c$	$-2J_1(B_r)J_2(B_c)$	-23.26

These tables will be referenced later when SNR data is presented.

c) SNR estimation from simulation data

The output from the Monte Carlo simulation will be used to estimate the output SNRs. In order to make these estimations, we shall use the sample mean and sample standard deviation [NET85]:

$$\bar{y} = \frac{\sum_{i=1}^{n} y_i}{n}, \qquad (29)$$

$$\sigma_y = \left[\frac{\sum_{i=1}^{n} (y_i - \bar{y})^2}{n-1} \right]^{1/2} \qquad (30)$$

so that the estimated signal-to-noise ratio is

$$SNR_{est} = 20*\log\left\{ \frac{\bar{y}}{\sigma_y} \right\} \qquad (31)$$

These estimators are good for Gaussian data, which we will assume to have under a central limit theorem [BEA88]. The reference, [NET85], gives formulas for obtaining confidence intervals for the above estimators. In the simulation program, samples are taken of the outputs of the digital filters, and then (29) through (31) are used to estimate the SNRs (for CAD, CMD, and ranging) from these samples.

d) Ranging phase estimator

In order to determine the ranging phase delay of the digital signal processing, a maximum likelihood (ML) estimator is used to determine the phase of the output ranging signal. The ML estimator for a signal in white Gaussian noise is the solution of [WHA71]

$$\int_0^T [r(t) - s(t,\hat{\theta})] \frac{\partial s(t,\hat{\theta})}{\partial \theta} dt = 0 \qquad (32)$$

where r(t) is the signal plus noise, and $\hat{\theta}$ is the estimate of the phase. Using

$$s(t,\theta) = \cos(\omega_r t + \hat{\theta}) \qquad (33)$$

and substituting into (32), the phase estimate is found to be

$$\hat{\theta} = \tan^{-1} \left\{ \frac{-\int_0^T r(t) \sin\omega_r t \, dt}{\int_0^T r(t) \cos\omega_r t \, dt} \right\} \qquad (34)$$

The Cramer-Rao lower bound on the variance of this estimator is

$$\sigma_\theta^2 \geq \left\{ \frac{A^2 T}{N_0} \right\}^{-1} \qquad (35)$$

Note that an ML estimator asymptotically approaches the Cramer-Rao bound on the variance [WHA71]. In the simulation program, the output of the ranging filter is multiplied by the sine and cosine of the accumulated ranging phase. These products are then summed to approximate the integrals in (33), and the phase estimate is obtained from the arctangent as in (33).

5) Conclusions

In this section some models were presented to predict the performance of the software breadboard. These models are not only important as a check on the accuracy of the software implementation, but they also give baselines against which the breadboard can be compared. In the next section, the results of testing the software breadboard will be given, and comparisons will be made against the models we have presented here.

III. Software Breadboard Evaluation

A. Simulation of the Carrier Loop

1) Introduction

It was felt that the testing of the software breadboard carrier loop should be done first for the following reasons:

a) Testing of the carrier loop should give the most thorough check on how accurately the software matches the hardware, and should therefore be done prior to other testing.

b) Correct operation of the carrier loop is essential for proper functioning of the receiver.

The loop will be tested by verifying that the phase step response matches that of the linearized model for negative SNR (at the A/D input). Once the loop has been tested under this condition, positive SNR cases will be considered, and conclusions will be drawn. In addition to the phase step response testing, the frequency acquisition performance of the loop will be considered for both the weak and strong signal cases.

Unless otherwise noted, the loop filter gains will be constant for all testing. These loop filter gains were selected to meet the specifications for the NASA X-band transponder (spec. no. FM 513778) at the threshold signal level (signal to noise density = 15.4 dB-Hz). The following loop gains were obtained to meet the specification requirements:

k0_sel = 10
k1_sel = 3
K1_SHIFT = 10 bits
$K_{INT} = 2^{-10}$
$K_{DAC} = 2^{-9}$
$K_{VCO} = 1570796$

The detector gain for this input signal level can be found from (8) and (9) to be:

$K_{DET} = 0.7607$

so that

$K = 1.45*10^{-6}$

Substituting these values into (17) and (18) the following parameters were obtained

$\omega_n = 22.3$ rad/s
$\zeta = 0.51$
$B_L = 11.2$ Hz

2) Phase Step Response of Carrier Loop

a) Negative SNR phase step response

The simulation program was operated with a signal-to-noise density of 60 dB-Hz at the input to the A/D prefilter. The SNR at the output of the 5 MHz bandpass filter is approximately -7 dB. The loop was initialized with the carrier phase in lock. After 1 ms of operation, the input carrier phase was stepped by $\pi/4$ radians. For this input SNR, the loop parameters are found from (17) and (18) to be:

$$\omega_n = 282 \text{ rad/s}$$
$$\zeta = 5.8$$
$$B_L = 822 \text{ Hz}$$

Fig. 10 shows the phase step response of the simulation program under the above conditions. Fig. 11 gives the comparable phase step response for the linearized digital model of Fig. 6. Note that except for the absence of noise in the linearized model response, the two curves agree quite well. Fig. 12 shows the phase step response of the simulation program with the carrier modulated by both command and ranging tones. The phase step response for this case also looks normal, although the response is slightly slower due to the 3.25 dB decrease in detector gain (cf. Table 1). Note also the presence of the command subcarrier in the output response.

b) Positive SNR phase step response

The phase step response for negative SNR agreed quite well with the linearized model. In this subsection step responses will be generated for positive SNR conditions where it may not be justified to assume a linearized model. The first case considered will have an input signal-to-noise density of 80 dB-Hz. This value translates into an SNR of 13 dB at the A/D input.

The simulation output for the 80 dB-Hz case is shown in Fig. 13. At this high SNR level, the output of the 1 bit A/D is dominated by the signal, so that the detector gain is theoretically equal to 'ncount' = 256. Fig. 14 shows the linearized model step response for this value of detector gain. The linearized model shows a much slower step response than the simulation output. It was hypothesized in [ATS89] that at strong signal levels the S-curve of the phase detector would no longer be sinusoidal and consequently the detector gain would increase beyond its predicted level. Fig. 15 is a result of trying to quantify this increased detector gain. Detector gains were chosen using trial and error in the linearized model in order to achieve a response similar to the simulation response. The detector gain used in Fig. 15 is equal to 2.5 times the nominal value. Fig. 16 is the simulation response with command and ranging tones modulating the carrier. Again the decreased carrier power results in a slower response than the unmodulated case.

Fig. 17 is the simulation step response for an input signal-to-noise density of 100 dB-Hz. This level of signal is the highest

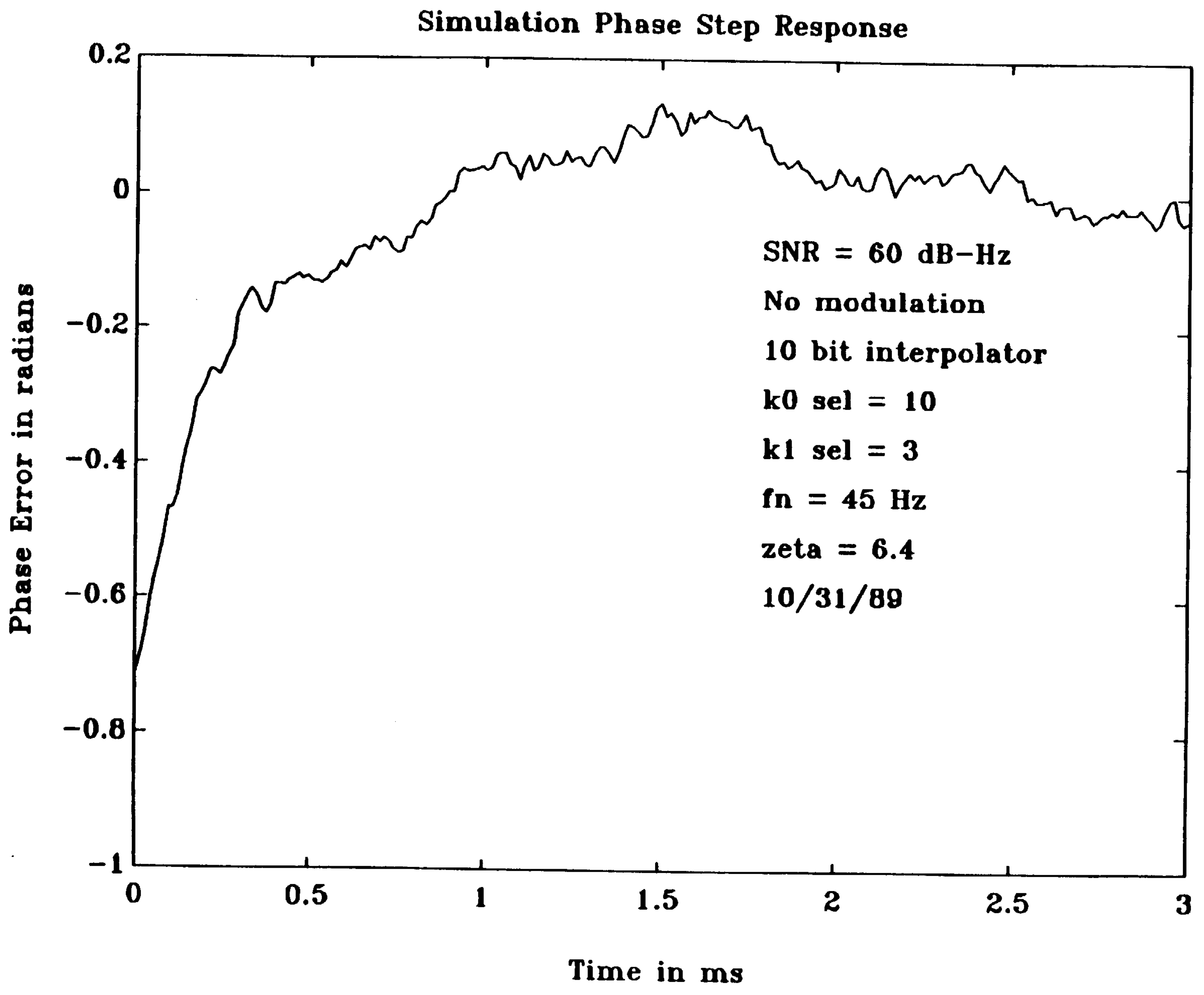


Fig. 10. Phase step response of software breadboard, negative SNR.

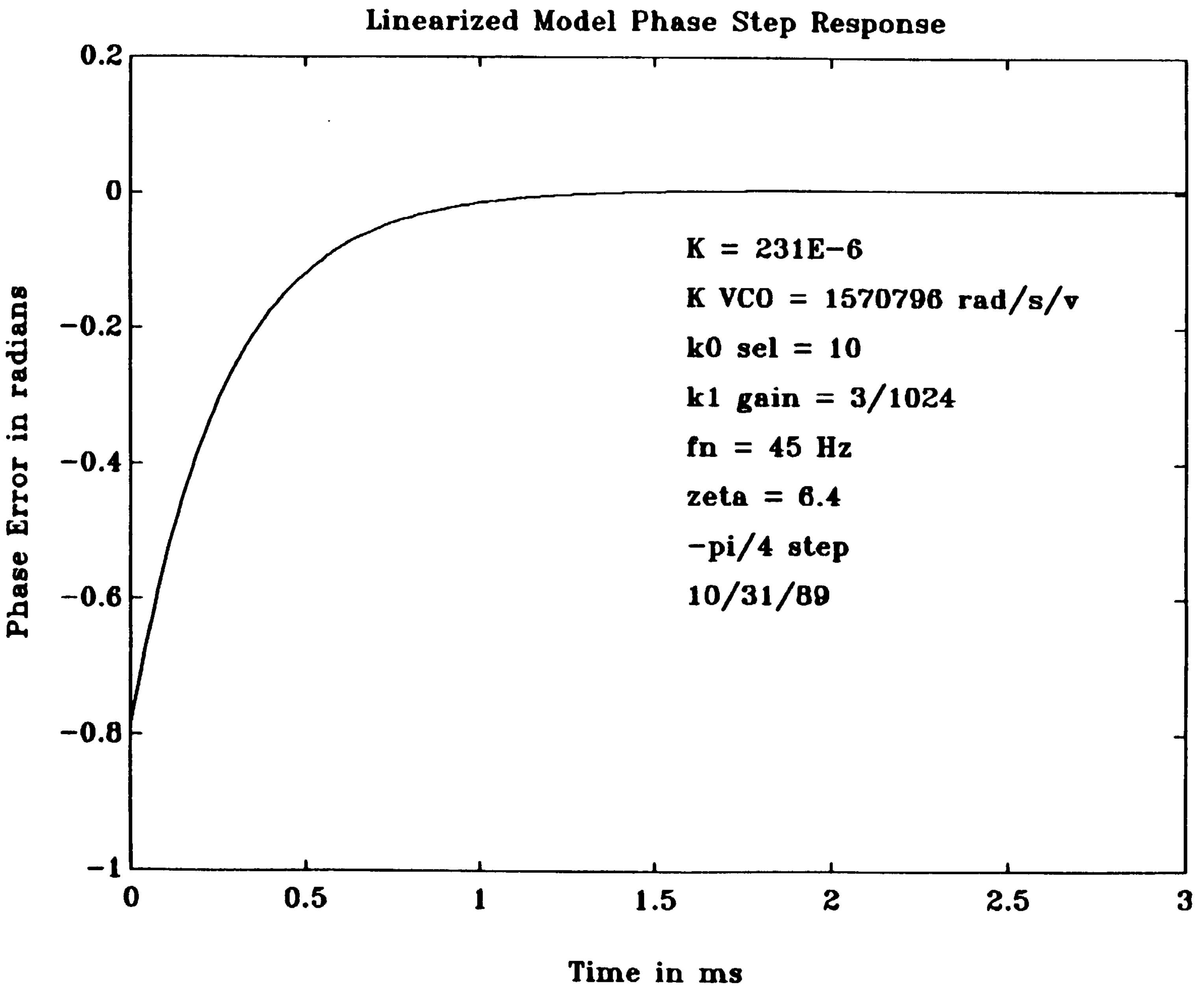


Fig. 11. Phase step response of z-transform model, negative SNR.

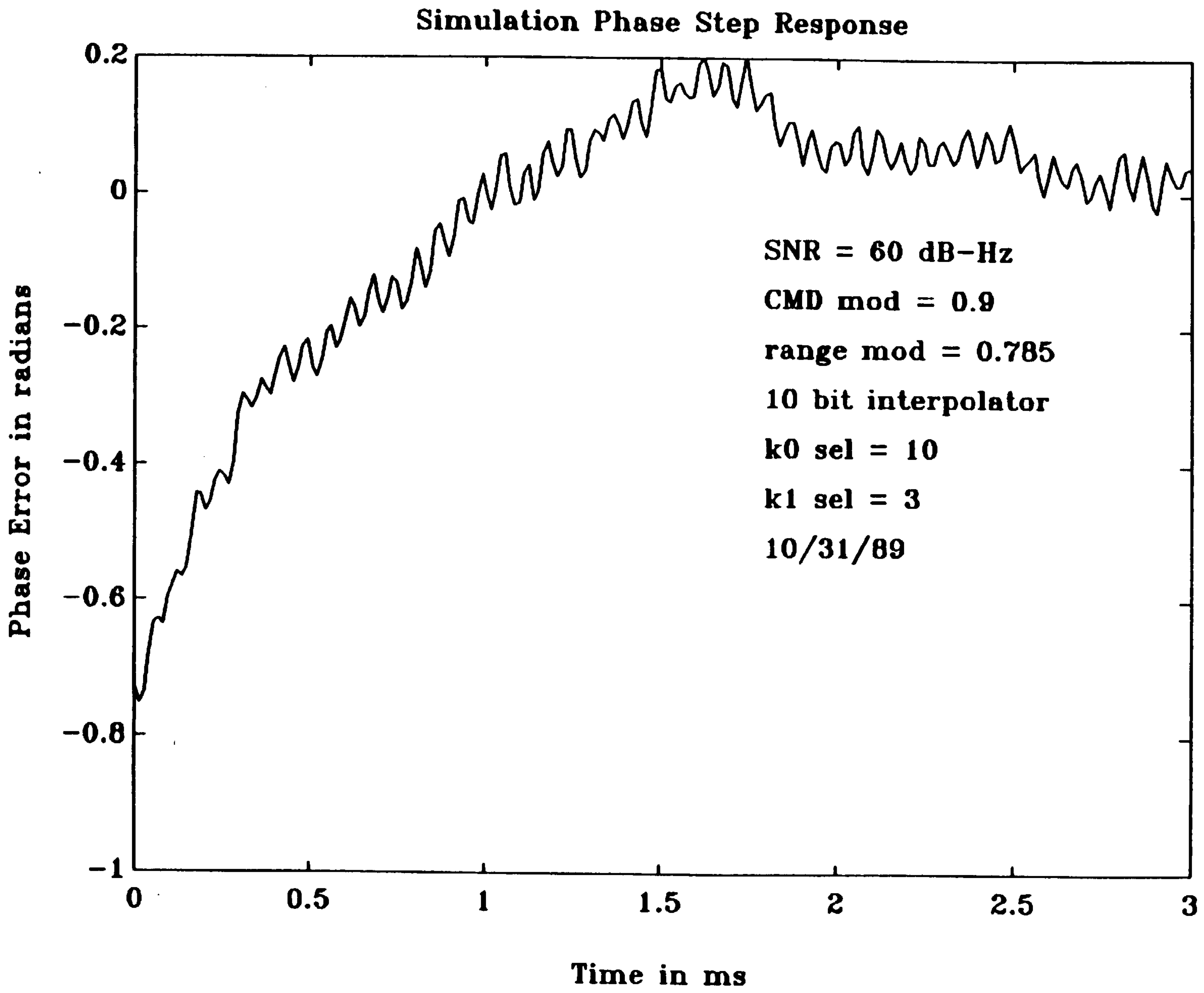


Fig. 12. Phase step response of softwar e breadboard, negative SNR, command and ranging modulation.

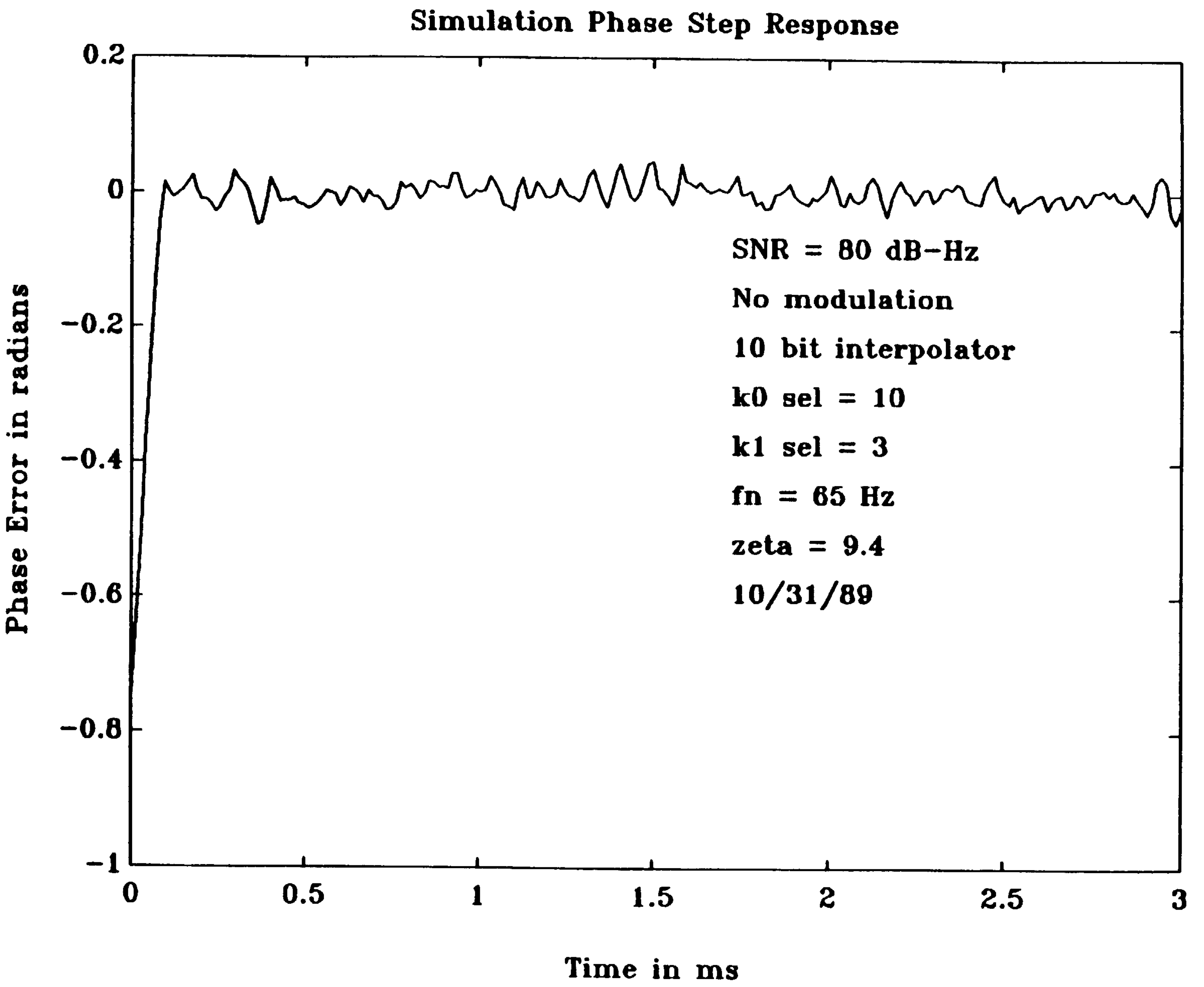


Fig. 13. Phase step response of software breadboard, positive SNR.

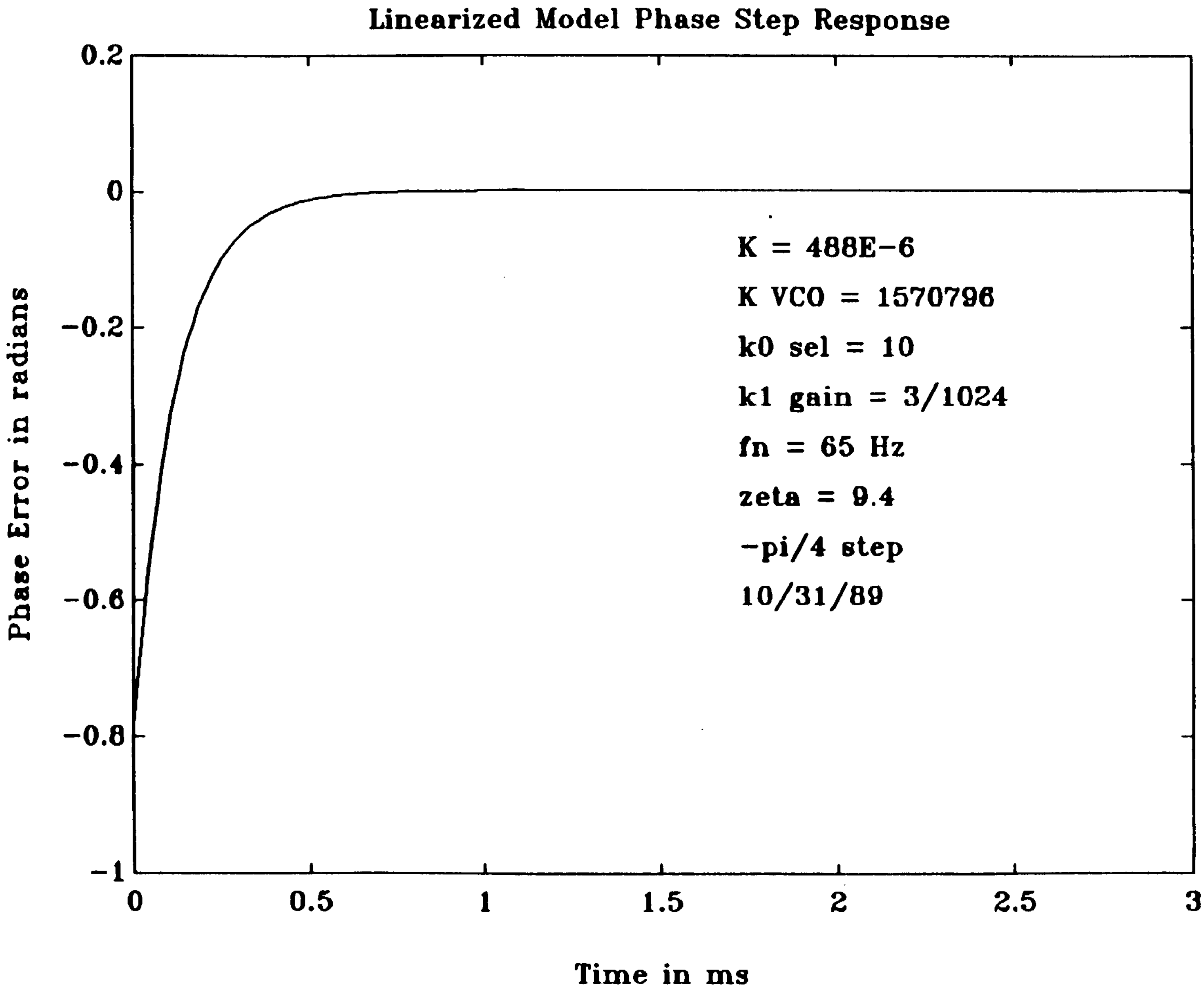


Fig. 14. Phase step response of z-transform model, positive SNR.

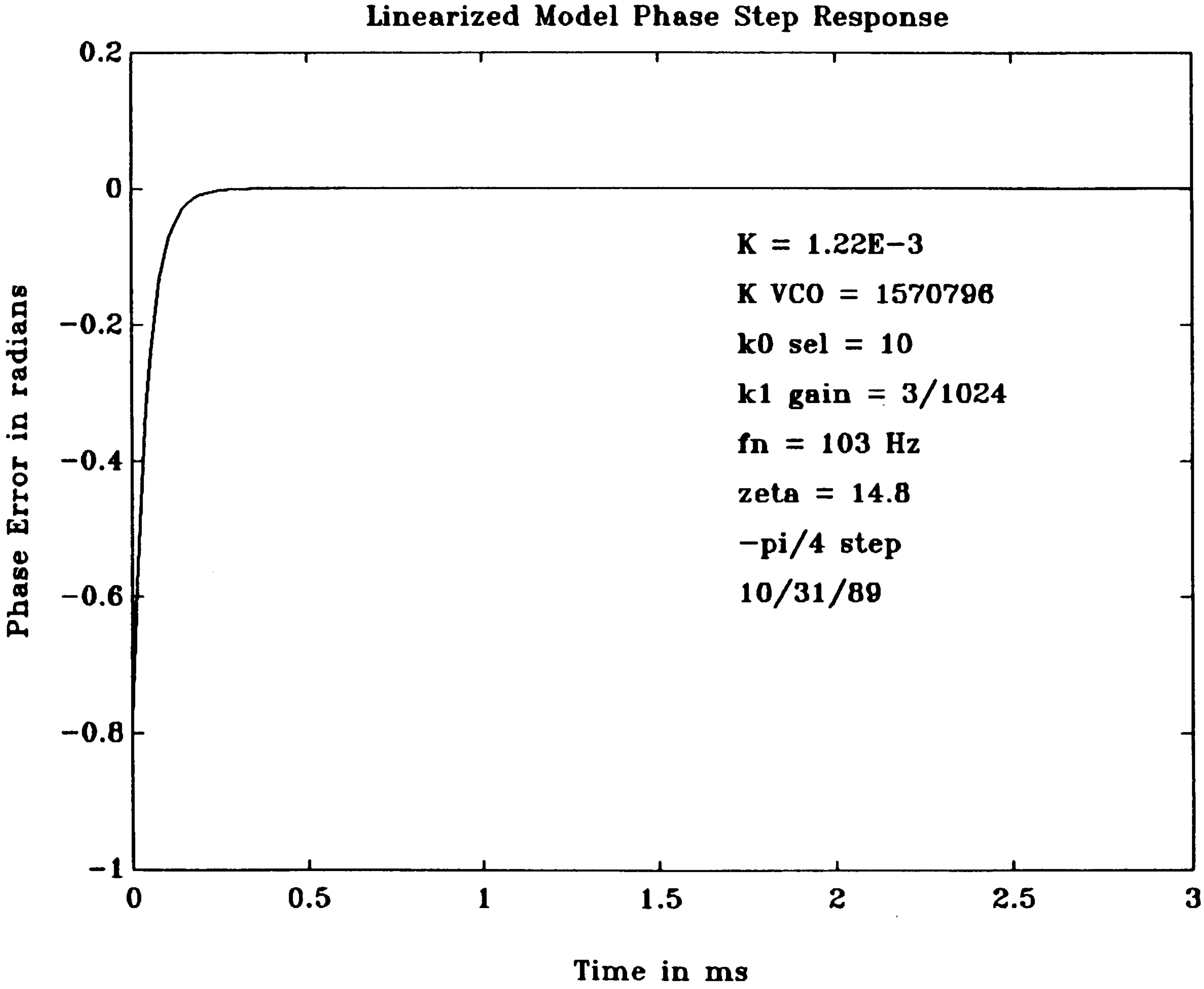


Fig. 15. Phase step response of z-transform model, positive SNR, increased detector gain.

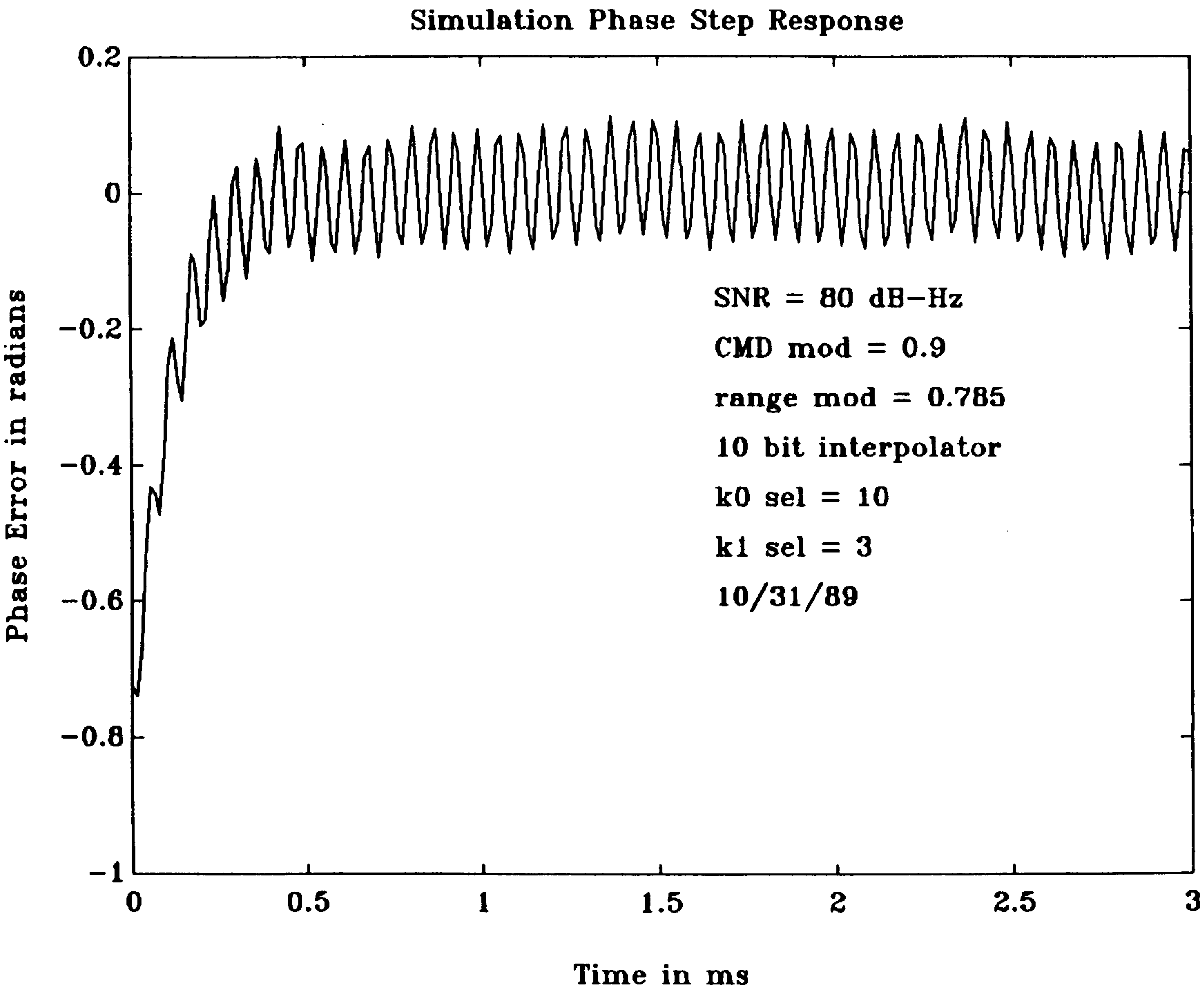


Fig. 16. Phase step response of software breadboard, positive SNR, command ranging modulation.

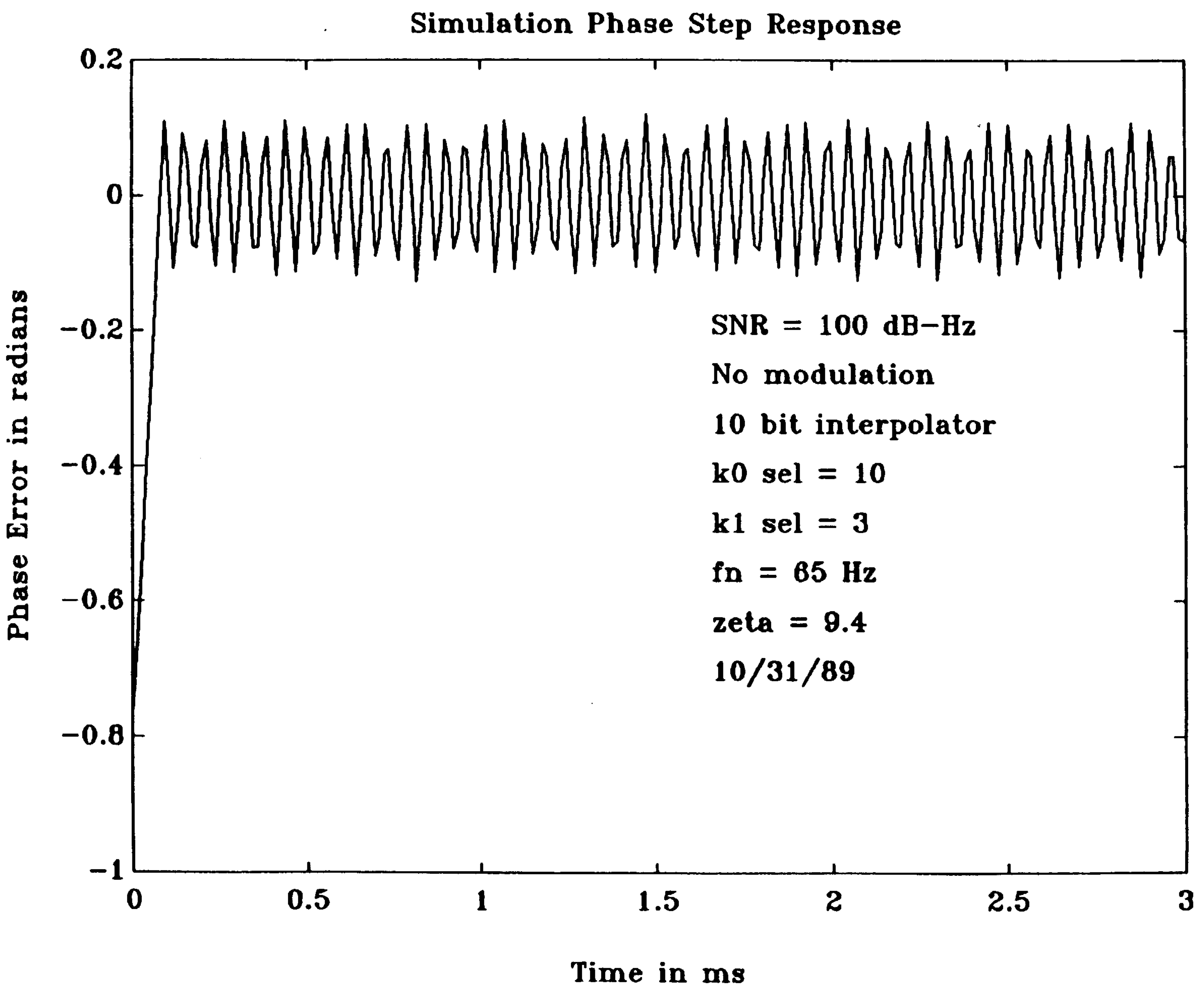


Fig. 17. Phase step response of software breadboard, maximum SNR.

value expected to be seen by the receiver. The rise time for this case is comparable to the 80 dB-Hz case. Therefore, it appears that the detector gain levels off after 80 dB-Hz. Note, however, the limit cycle oscillation (at approximately 19 kHz, 1/4 the loop sampling rate) at this signal level. This limit cycle is a result of the 'bang-bang' operation of the loop when little noise is present. With command and ranging modulation, the limit cycle is no longer present. The oscillation in Fig. 18 is actually the command tone at 16 kHz.

3) Stability of the Phase-Locked Loop

The stability of the PLL can be analyzed using the linearized model of Fig. 6. The major stability concern is that at strong signal levels the detector gain will increase thereby increasing the open loop crossover frequency. If this crossover frequency approaches the sampling frequency of the loop, then the zero order hold will start to contribute appreciable phase lag which will decrease the phase margin of the loop.

As a baseline for comparison, the open loop magnitude and phase responses of the loop at threshold signal level are shown in Figs. 19 and 20. Note that the crossover frequency of this loop is at approximately 4.3 Hz and that the open loop zero is at approximately 3 Hz. The phase margin at crossover is about 53 degrees.

The corresponding plots for strong signal level are shown in Figs. 21 and 22. The crossover frequency is at approximately 3 kHz. The open loop zero is so far below crossover that the slope of the magnitude curve is -1 at crossover. The phase curve shows that the zero order hold starts to contribute a phase lag at higher frequencies, but there is still a more than adequate phase margin of 83 degrees at crossover. Therefore, the stability of the PLL is not a real concern at strong signal.

4) Frequency Acquisition of the Phase-Locked Loop

a) Negative SNR frequency acquisition

The simulation program was operated with a signal-to-noise density of 60 dB-Hz at the input to the A/D prefilter. The loop was initialized with the carrier phase in lock. After 1 ms of operation, the input carrier frequency was stepped by 1 kHz. Note that in the simulation plots, the carrier phase appears to reach a steady state value of 0.3 radians. This is not really the steady state phase error, but is a result of the process by which samples of the output phase error were taken in the simulation program. Of course, for a second order loop, the phase error would approach zero.

In order to determine the process by which the loop acquires, we plot the open loop frequncy response of this loop in Fig. 23. Note that the crossover frequency is at approximately 600 Hz. Therefore, we can predict that the loop will not acquire a 1 kHz offset frequency without slipping cycles. Fig. 24 shows the acquisition behavior of the simulation under the above conditions. For comparison, Fig. 25 shows the theoretical acquisition of the equivalent analog PLL. This figure was obtained by numerically

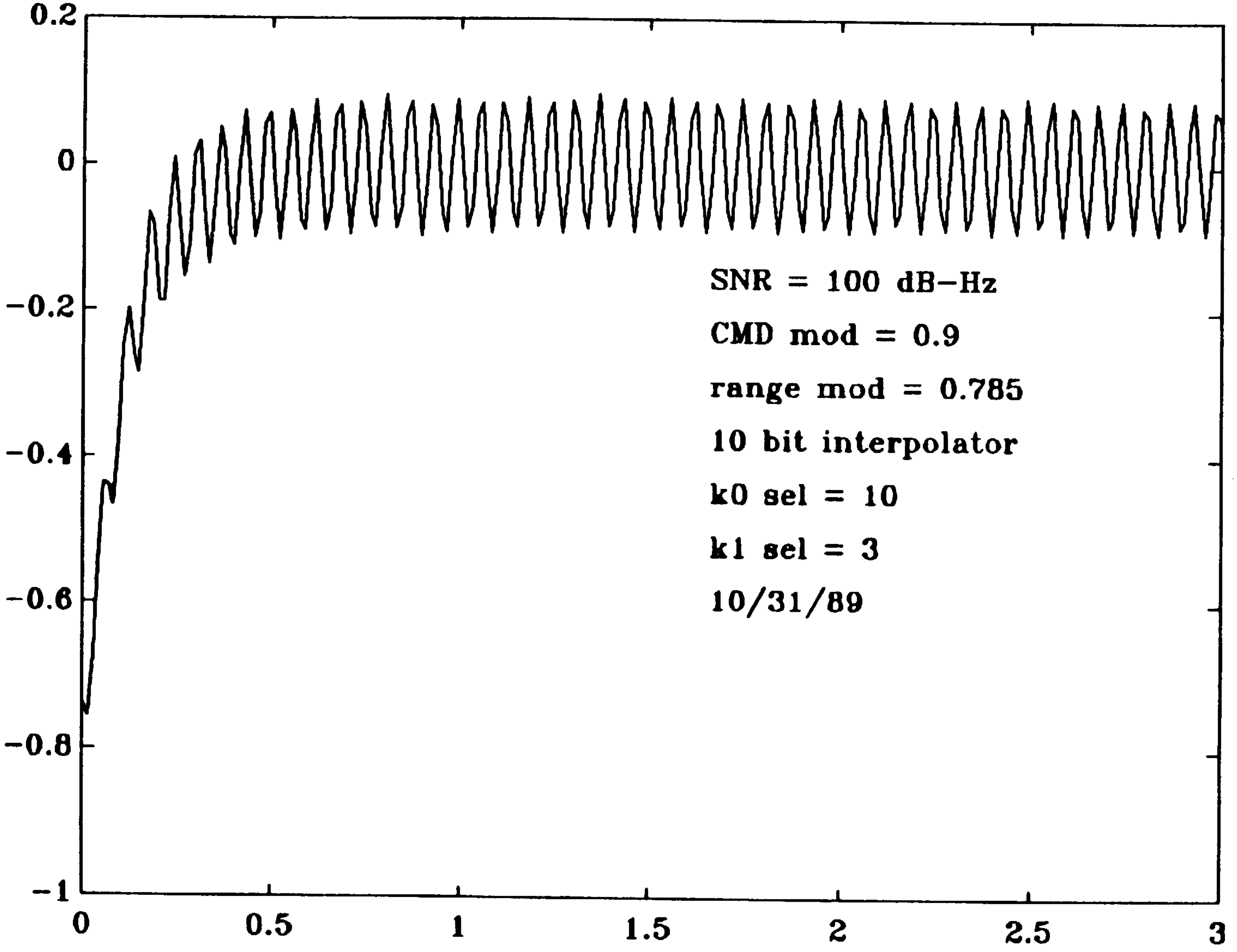


Fig. 18. Phase step response of software breadboard, maximum SNR, command and ranging modulation.

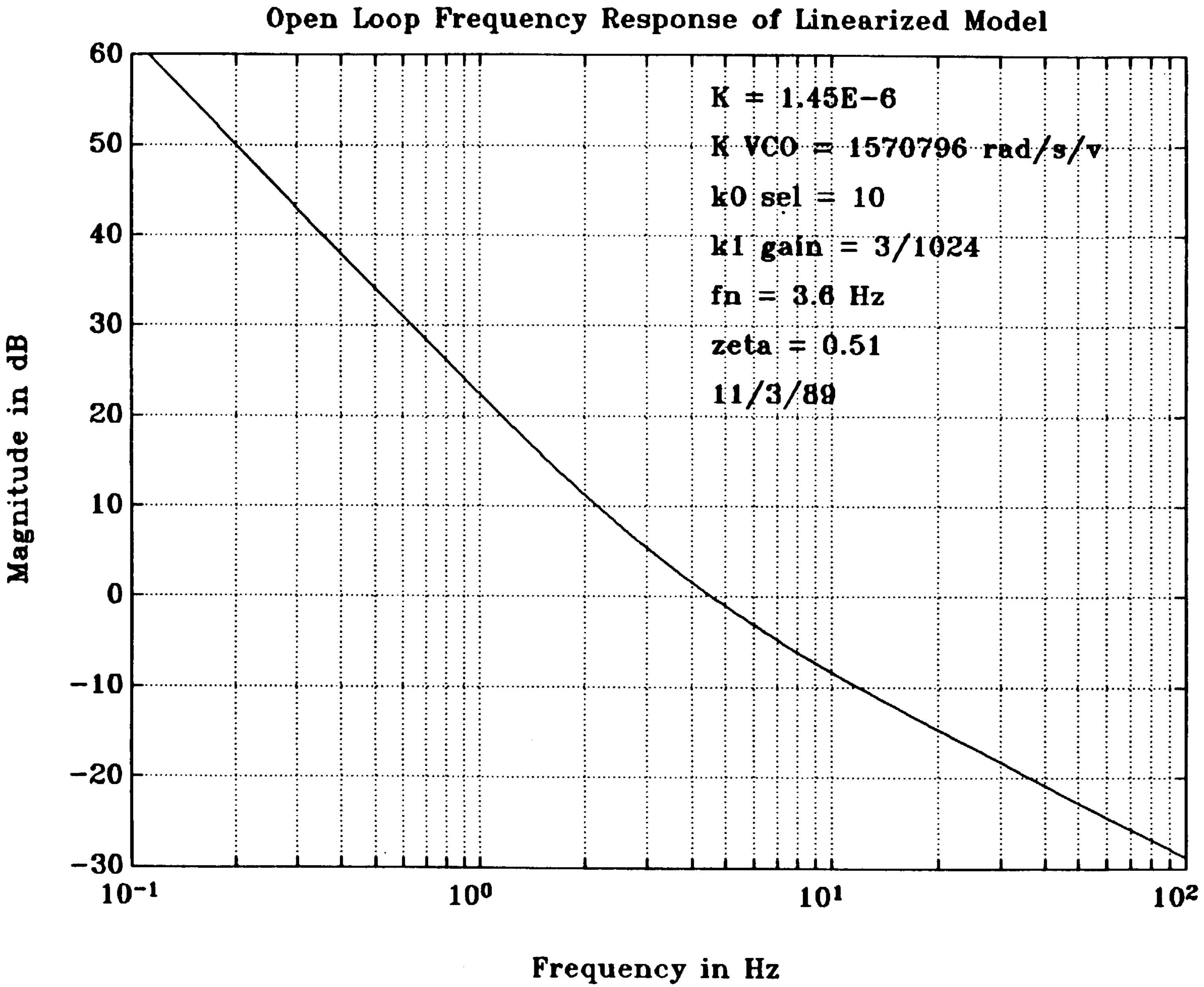


Fig. 19. Open loop magnitude response of z-transform model, threshold signal level.

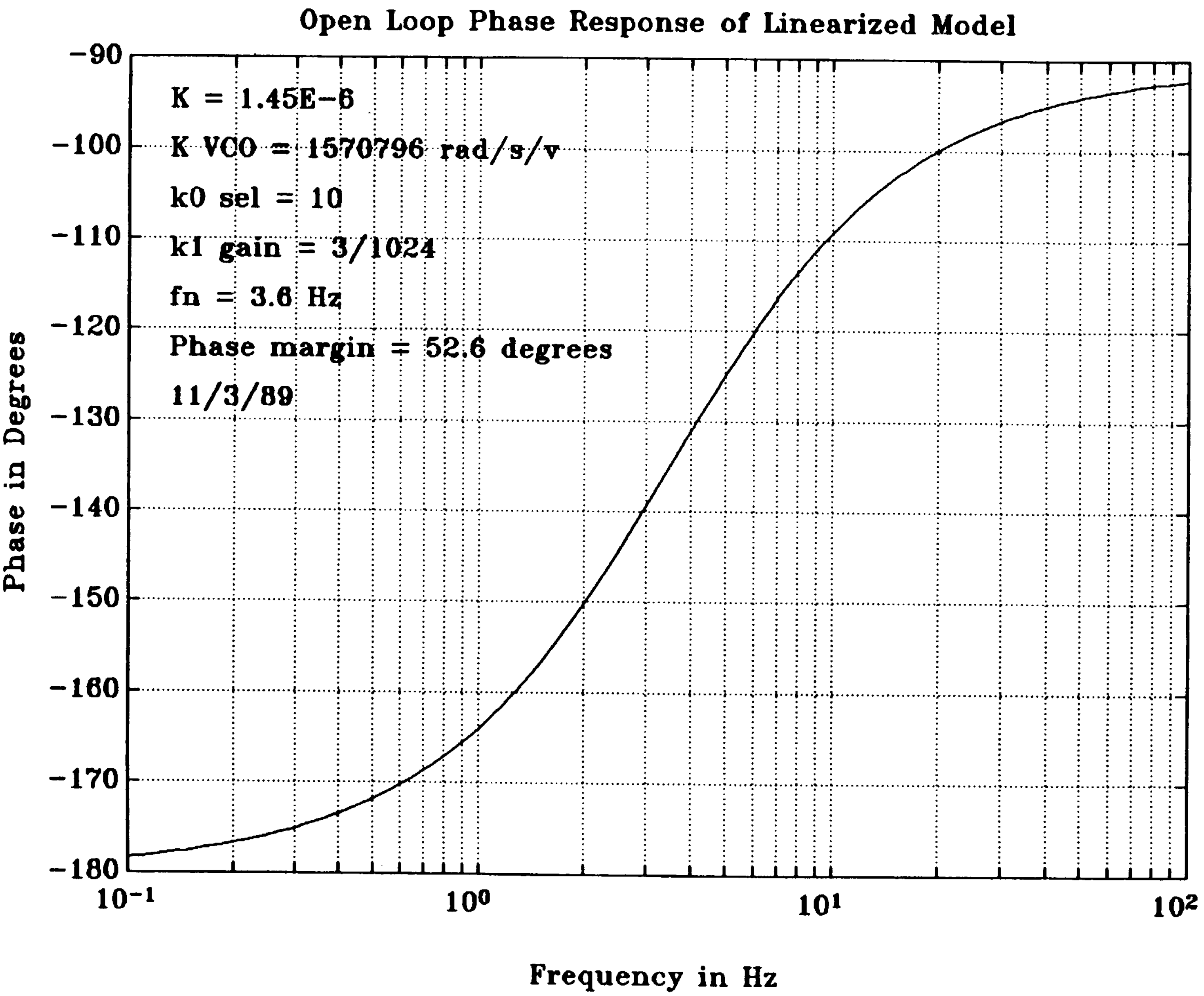


Fig. 20. Open loop phase response of z-transform model, threshold signal level.

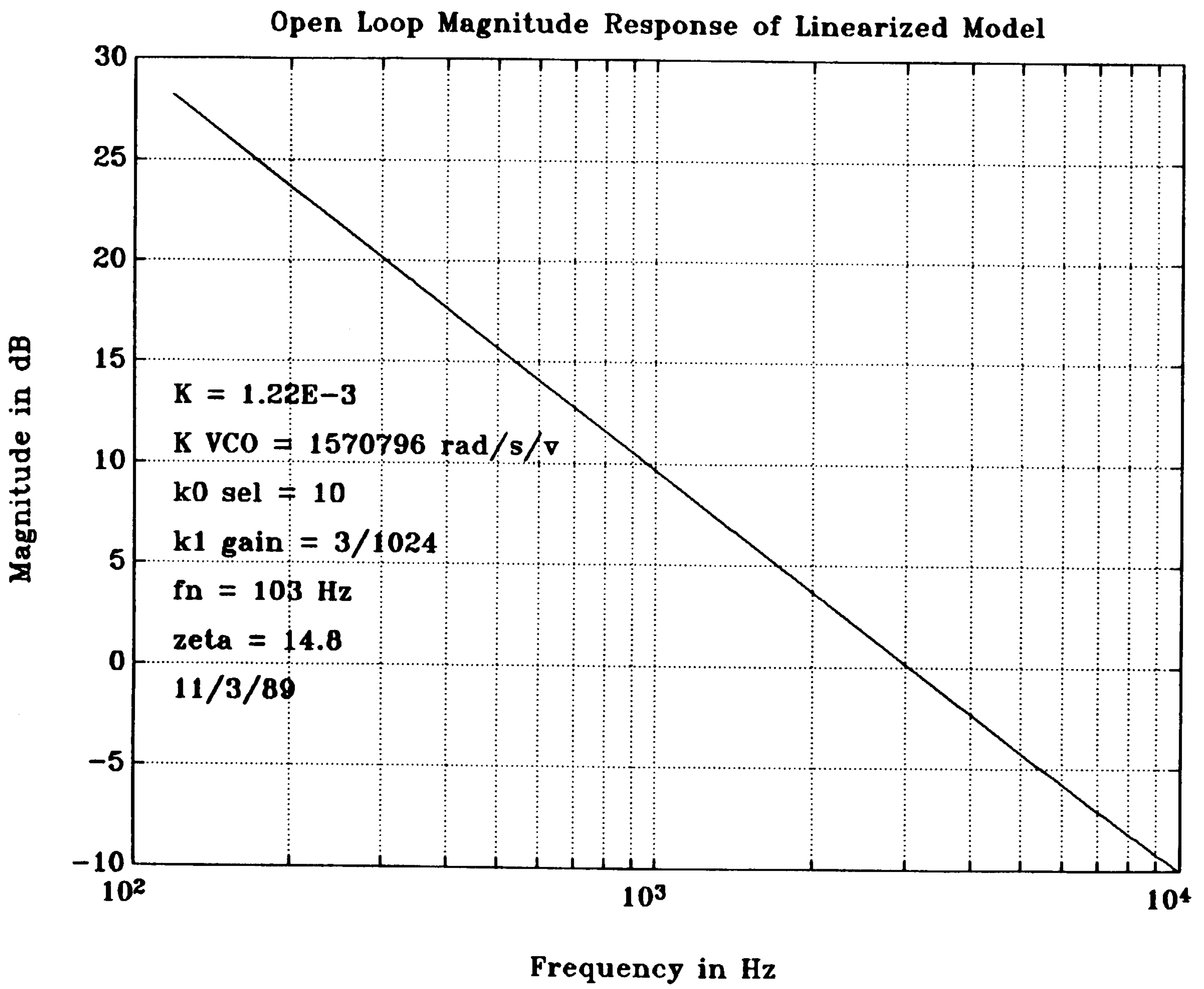


Fig. 21. Open loop magnitude response of z-transform model, maximum signal level.

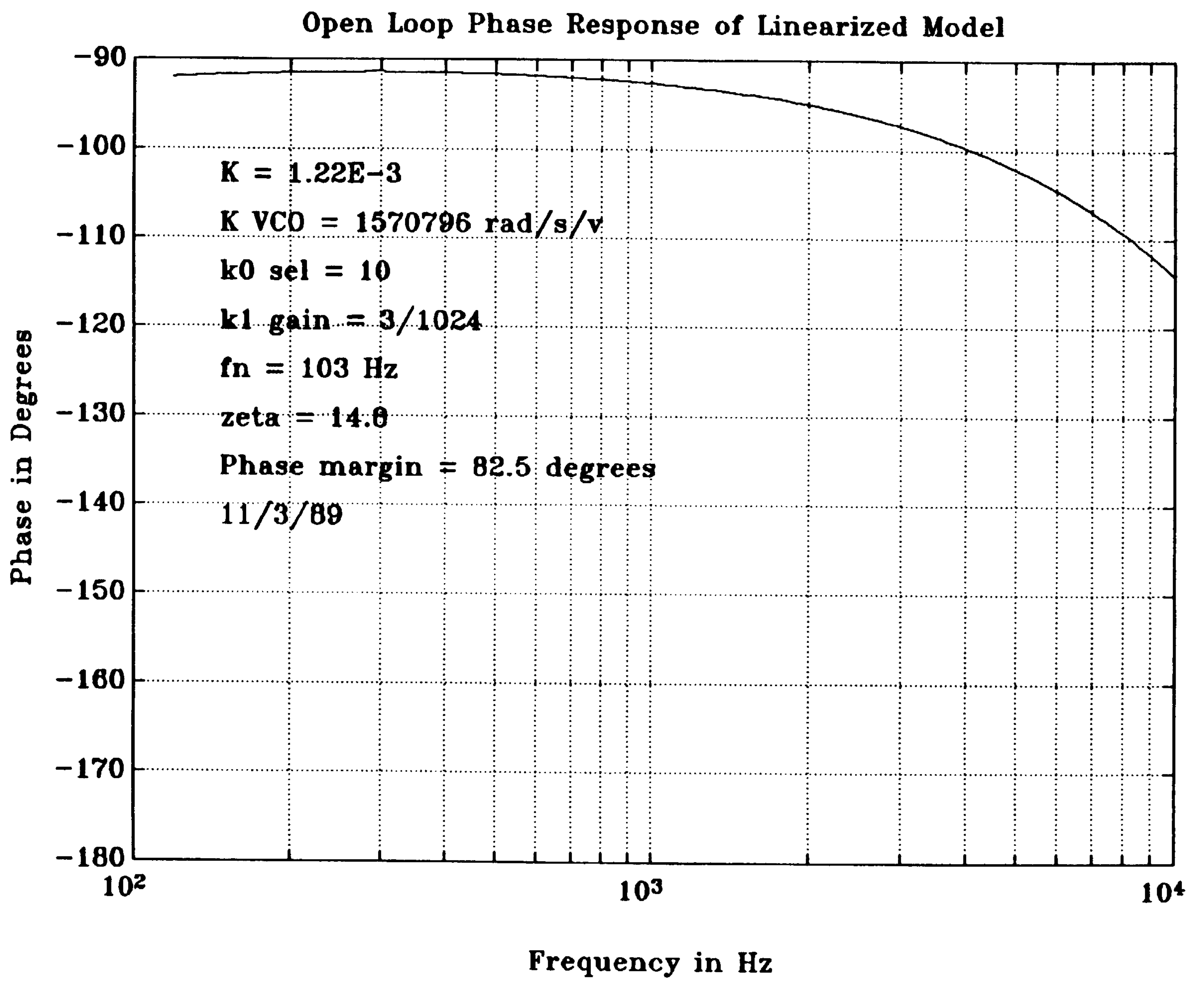


Fig. 22. Open loop phase response of z-transform model, maximum signal level.

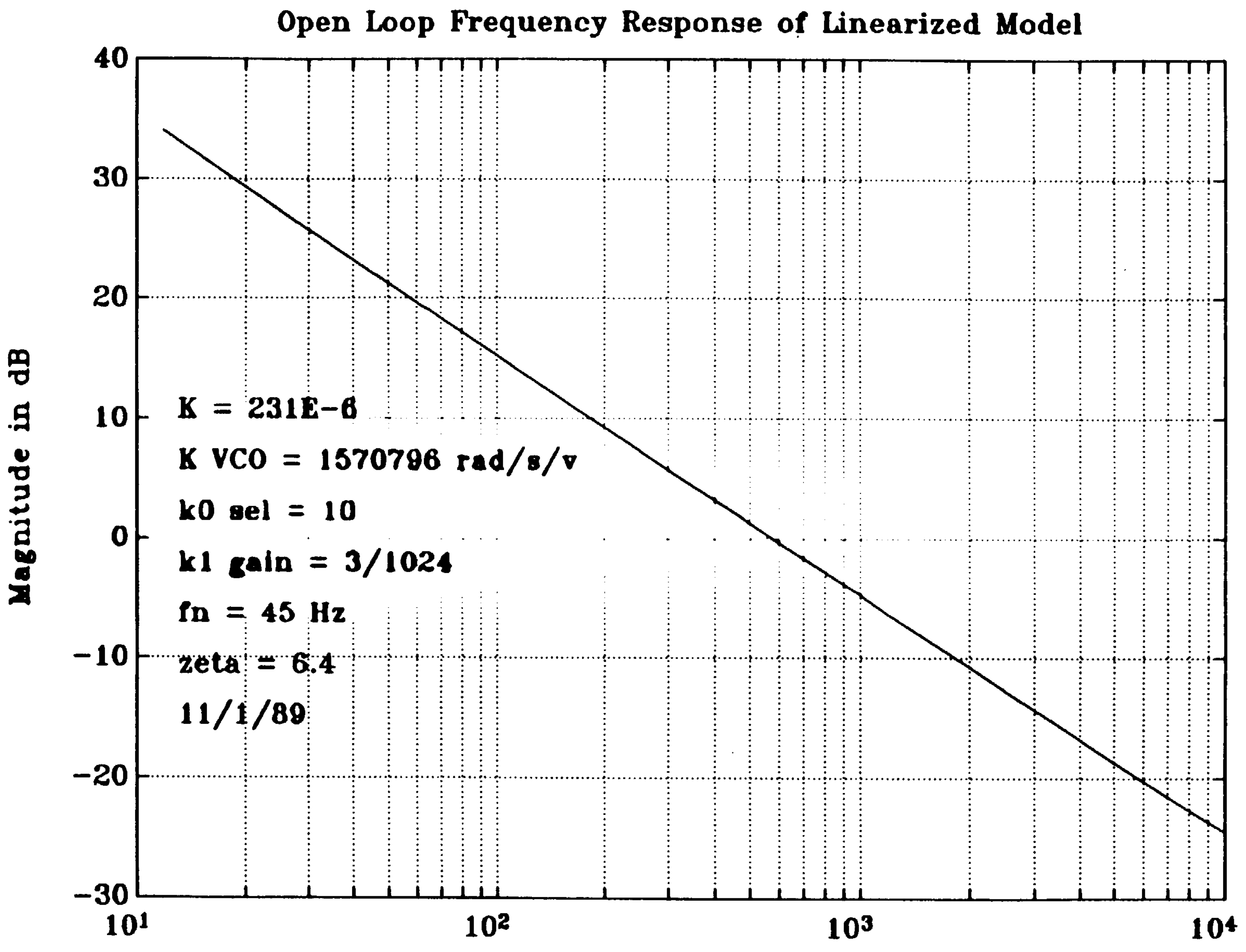


Fig. 23. Open loop magnitude response of z-transform model, negative SNR.

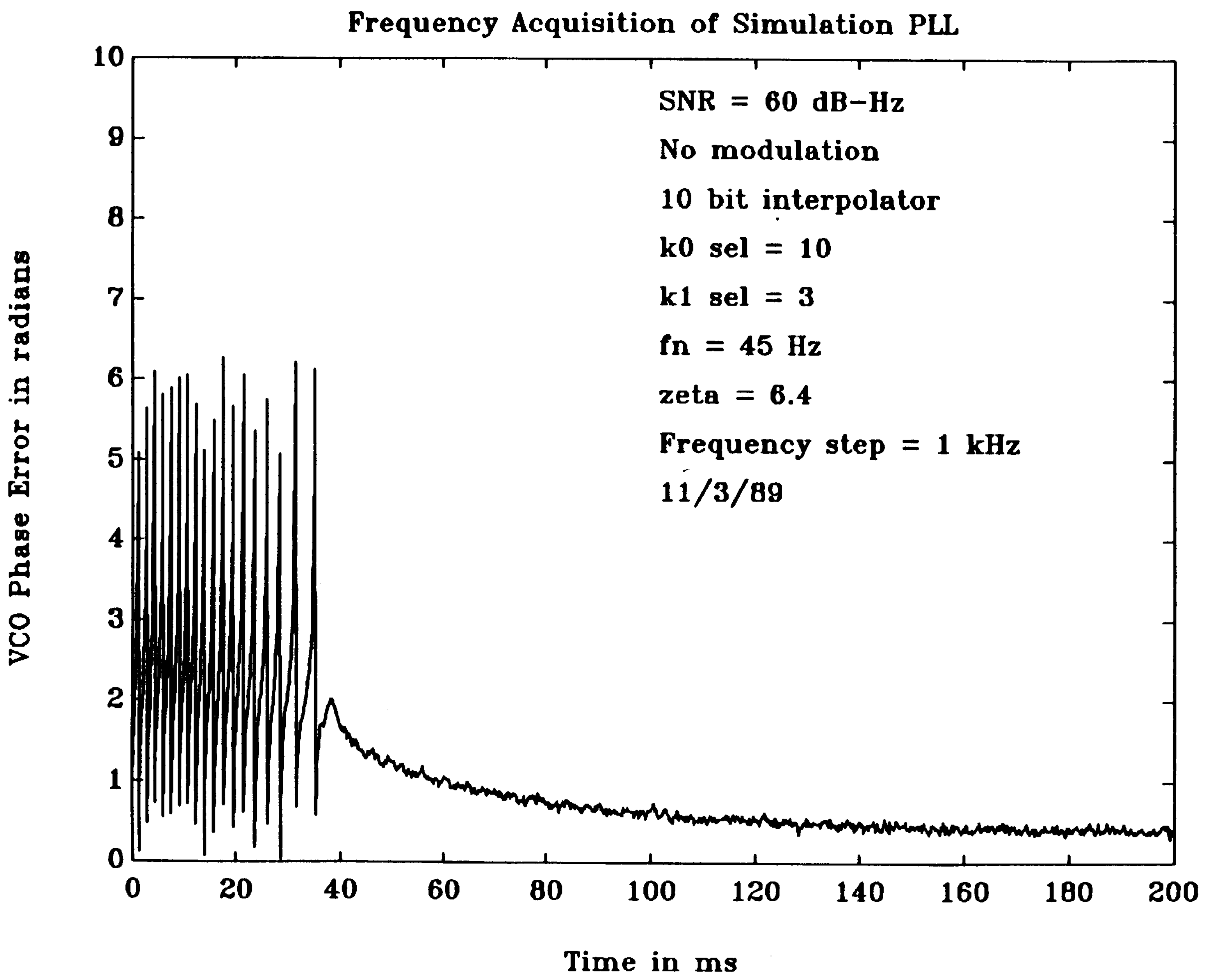


Fig. 24. Frequency acquisition of software breadboard, negative SNR, 1 kHz step.

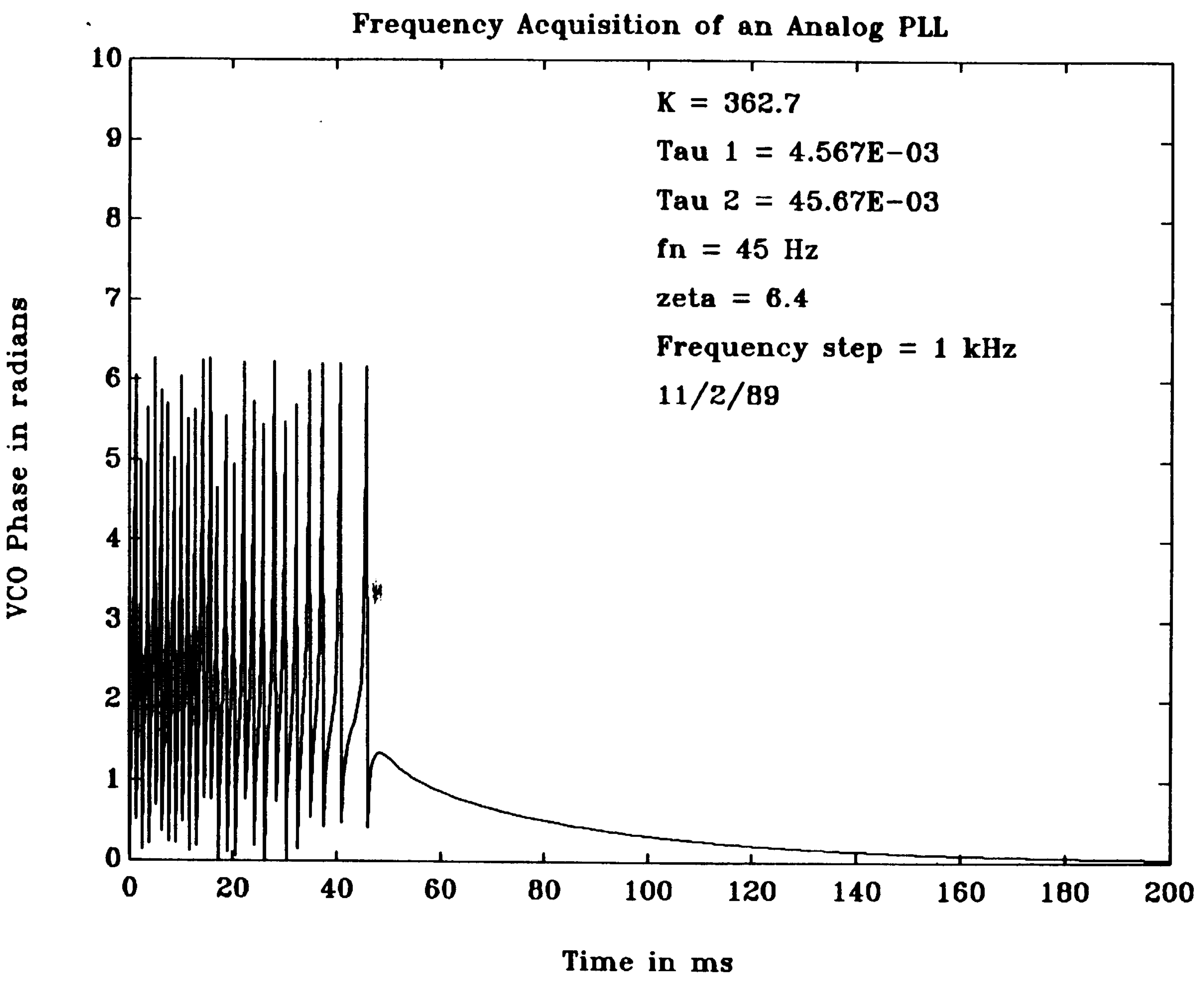


Fig. 25. Frequency acquisition of analog model, negative SNR, 1 kHz step.

solving (22). The equivalent analog PLL acquires sligthtly faster than the software breadboard, but recall that the analog PLL assumed a noise free model.

The approximate formula for acquisition time is given by [BLA76]

$$T_{acq} = \frac{\Delta\omega^2}{2*\zeta*\omega_n^2} \qquad (36)$$

which, for this loop gives

$$T_{acq} = 136 \text{ ms.}$$

This is approximately the time when the two loops stop slipping cycles. The additional time is needed for the completion of the frequency acquisition for the loop operating in the linear mode.

b) Positive SNR frequency acquisition

The simulation program was operated with a signal-to-noise density of 80 dB-Hz at the input to the A/D prefilter. A frequency step was input as in the 60 dB-Hz case. The open loop magnitude response for this 80 dB-Hz case is shown in Fig. 26. The crossover frequency is at 3 kHz. In generating this figure, the phase detector gain was set 2.5 times higher than its nominal gain per the results of the strong signal phase step response (cf. Fig. 15). Fig. 27 shows the simulation response for a 2 kHz frequency step input with Fig. 28 showing the equivalent analog loop response. With the step frequency inside the crossover frequency of the loop, one would expect that the loop would acquire without slipping cycles, as in Fig. 28. However, the simulation does slip cycles because of either the noise present in the simulation, or the phase detector gain is lower than was thought. Figs. 29 and 30 show cases where the step frequency (at 4 kHz) is beyond the crossover frequency causing both loops to slip cycles before acquiring. The discrepancy for this strong signal pull-in cannot be adequately explained from the models developed.

5) Bandwidth Expansion of the Carrier Loop

One behavioral aspect of a phase-locked loop which needs to be characterized is the bandwidth expansion as a function of increasing SNR. In an analog loop, the expansion curve is directly affected by the presence of an automatic gain control (AGC) or an ideal limiter in front of the loop (see, for example, [BLA76] Figs. 9.2 and 9.3). Note that the expansion curves are nearly identical for the AGC and the ideal limiter [BLA76]. Therefore, since the block diagram of Fig. 1 uses a hard limiter as an analog-to-digital converter, it is to be expected that the bandwidth expansion of the software breadboard should be similar to the analog PLL with AGC.

Unfortunately, it was found to be difficult to accurately obtain the noise bandwidth of the software breadboard. One method which was attempted involved frequency modulating the input

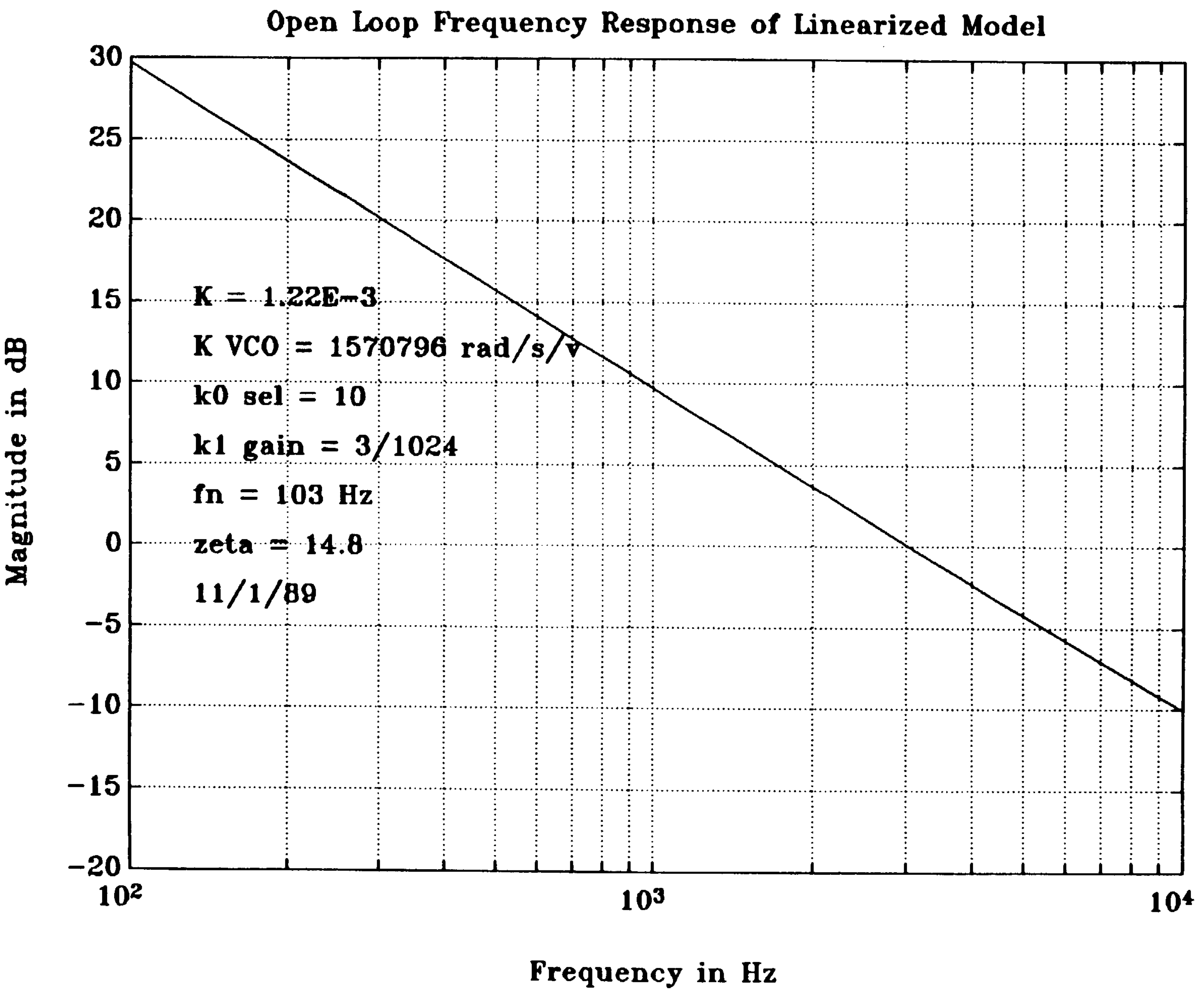


Fig. 26. Open loop magnitude response of z-transform model, positive SNR.

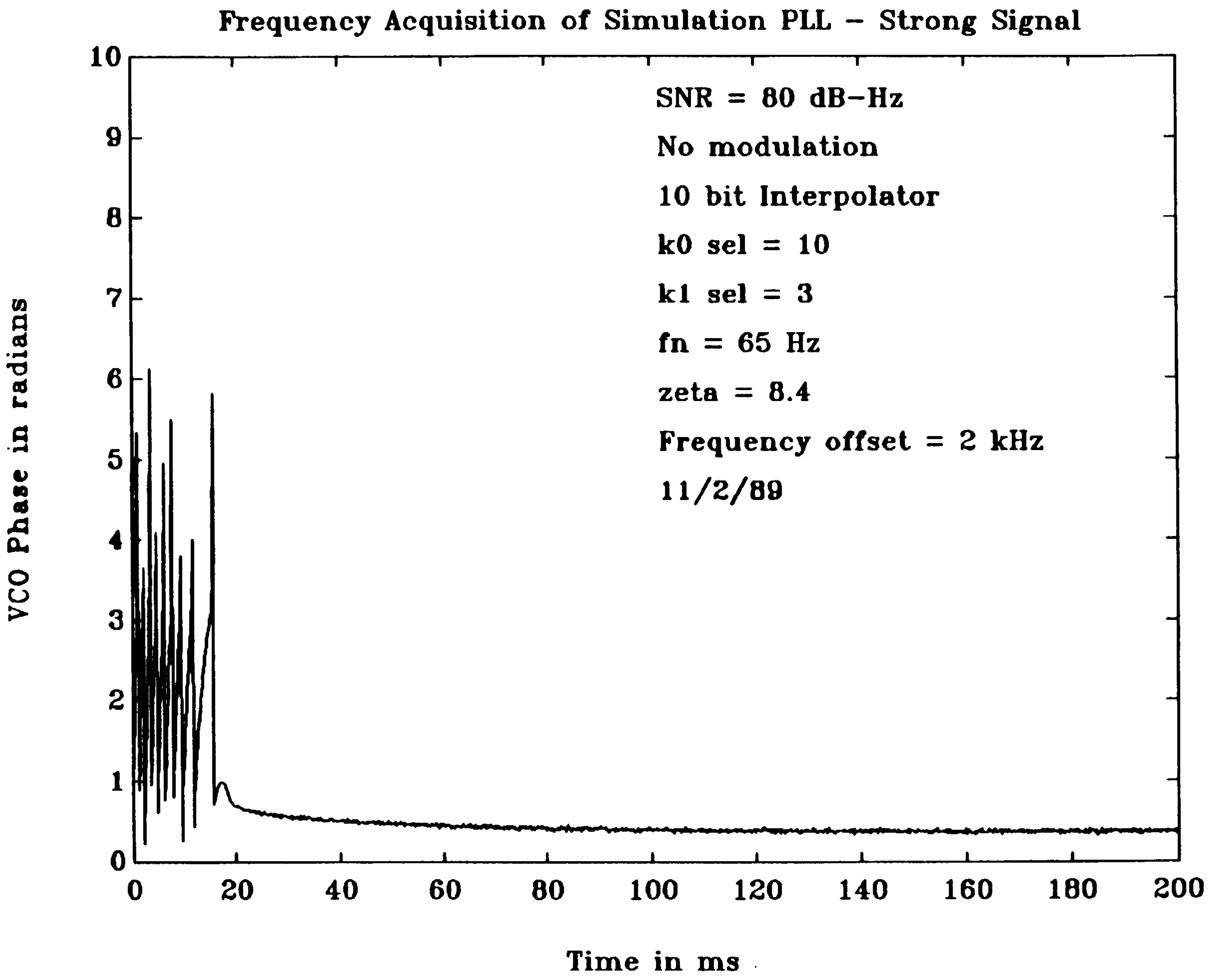


Fig. 27. Frequency acquisition of software breadboard, positive SNR, 2 kHz step.

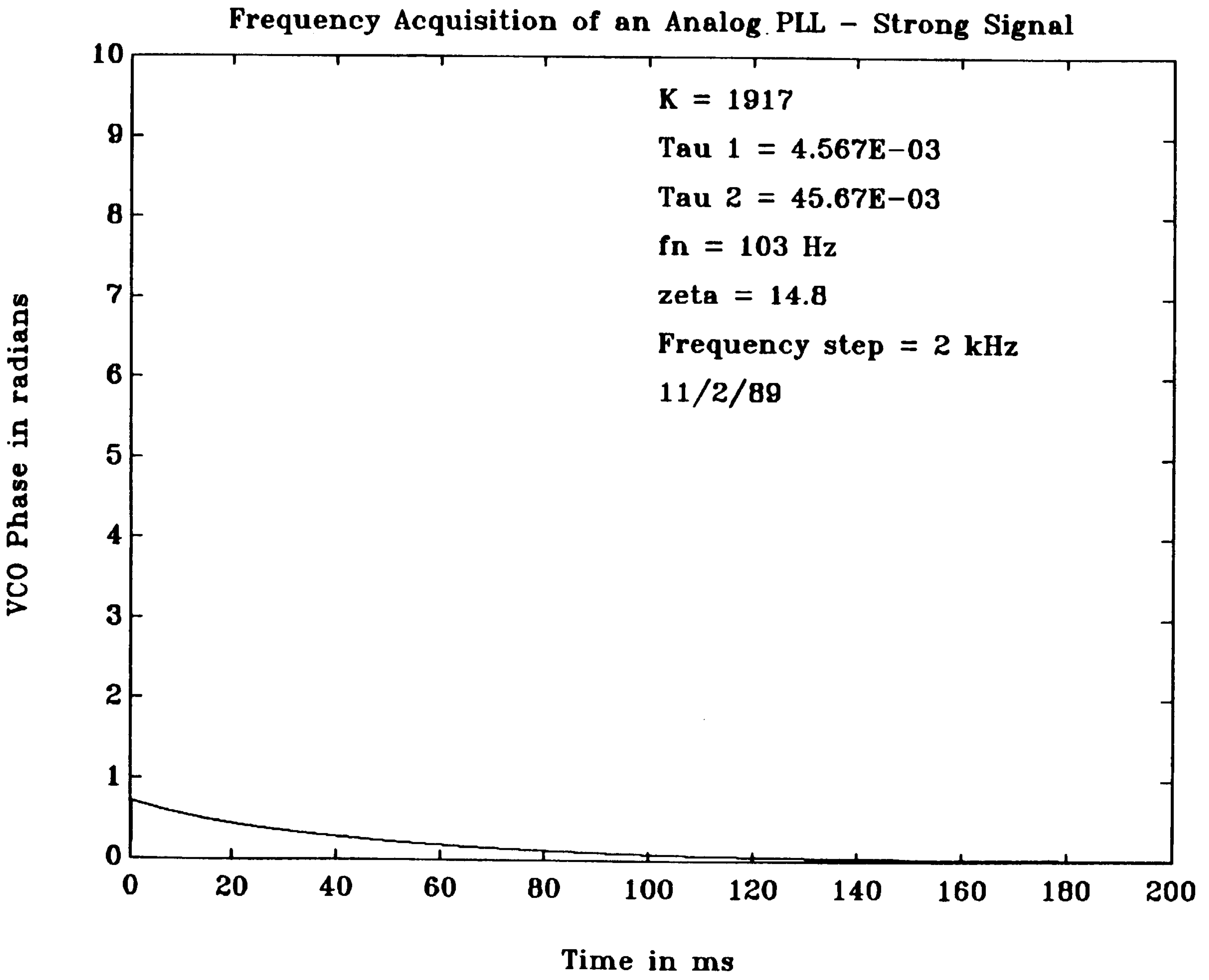


Fig. 28. Frequency acquisition of analog model, positive SNR, 2 kHz step.

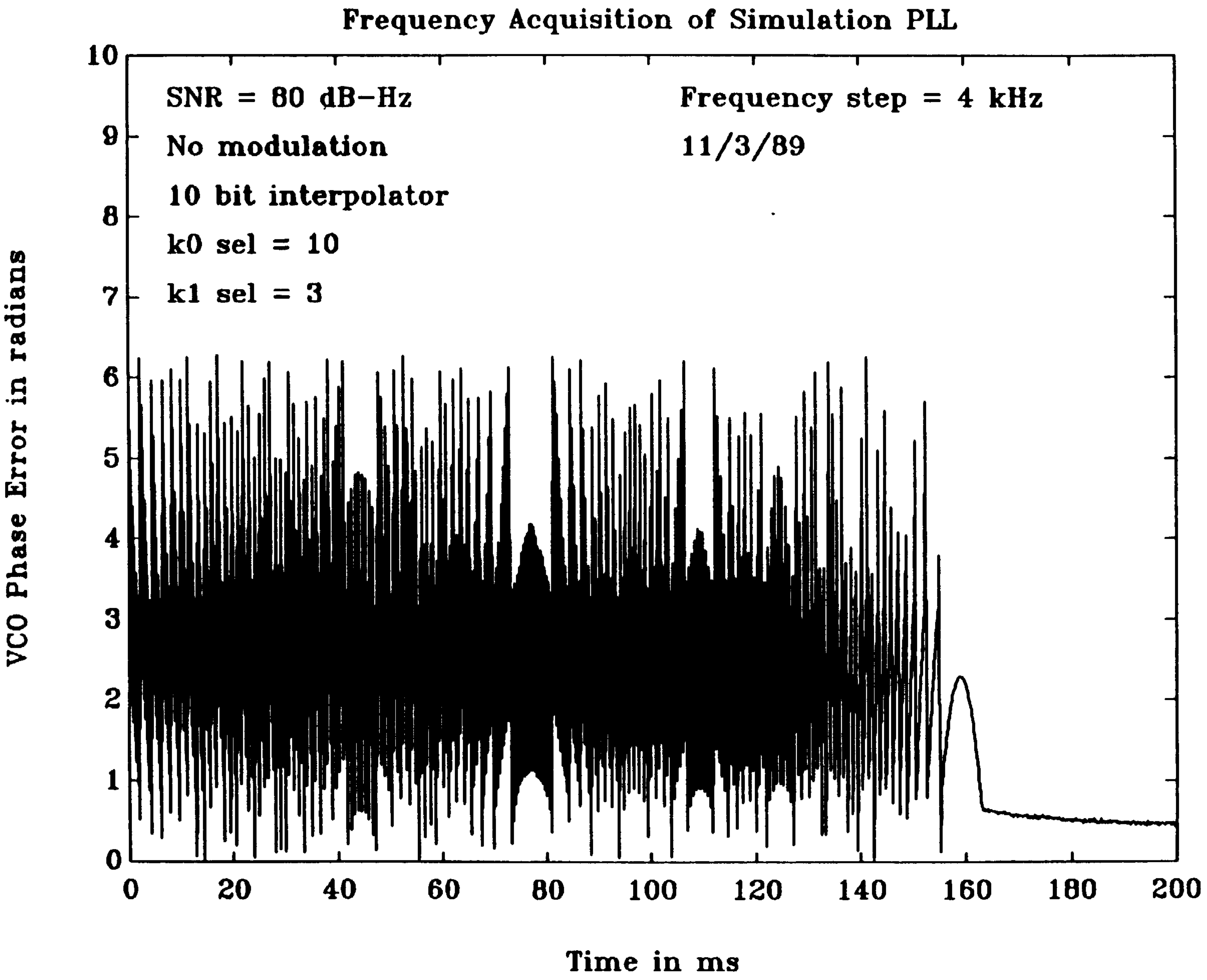


Fig. 29. Frequency acquisition of software breadboard, positive SNR, 4 kHz step.

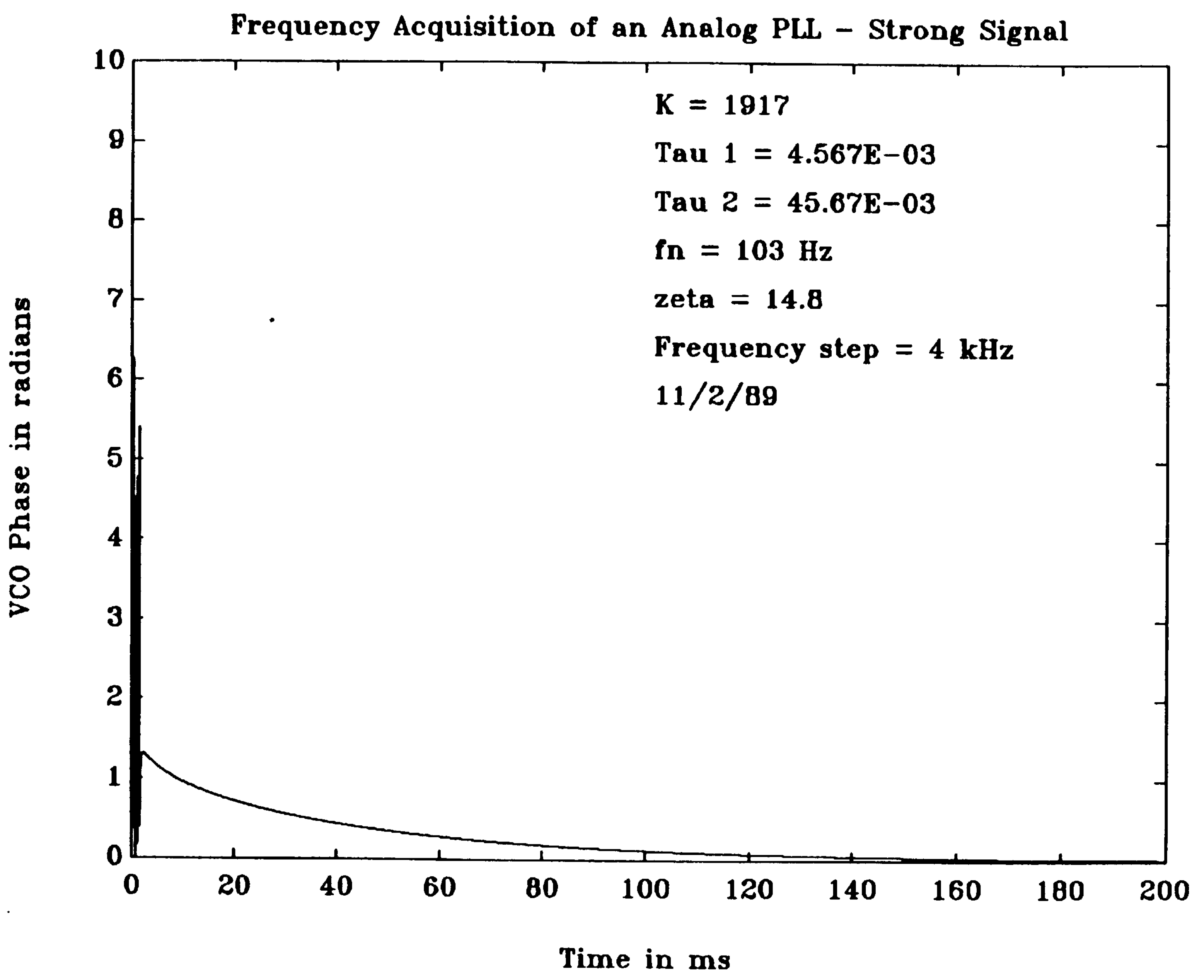


Fig. 30. Frequency acquisition of analog model, positive SNR, 4 kHz step.

carrier, and then plotting the loop response as a function of modulation frequency (see, for example, [BLA76], Fig. 6.16). The frequency response should peak at the loop's natural frequency, which could then be used to calculate the noise bandwidth. This approach was found to be infeasible because the response tends to flatten out at large values of ζ (i.e., large SNR) and therefore did not give a clear indication of the natural frequency.

Another method for determining the bandwidth of the loop is to plot the step response of the loop, and infer the loop bandwidth from the shape of the step response. This method was felt to be somewhat subjective, but should give an approximate value for the loop bandwidth. Figs. 31 - 34 show phase step responses of the software breadboard for different input SNRs. Also included in the figures are step responses of the digital linearized model of Fig. 6. For the linearized model, the phase detector gain was selected to give a close match to the simulation response. The fitted linearized models can then be used to determine the loop bandwidths. For signal-to-noise densities below 60 dB-Hz (-7 dB SNR), the 1 bit A/D can be assumed to operate as a linear device with the phase detector gain given by (8) and (9). From Figs. 31 - 34, and using (8) and (9), the loop bandwidth can be determined as a function of input SNR. Fig. 35 shows a plot of this relationship. The bandwidth at maximum SNR is approximately twice what would be obtained with an analog loop with AGC.

The preceding results were obtained with no modulation of the carrier. Figs. 36 - 39 show step responses with command modulation of the carrier. Note that these plots indicate that these plots indicate that the loop expands to a lesser degree with modulation of the carrier. Fig. 40 is a plot of the loop expansion as a function of input SNR obtained as above. Also shown in this figure is the expansion which would be obtained with an analog loop with AGC. Note that the two curves do not significantly differ.

6) Conclusions

The digital carrier loop as implemented in the software breadboard performed very much like its analog equivalent. The only real diffference occurred at strong signal levels (about 65 dB above the threshold level). At strong signal levels the apparent phase detector gain was determined to be about 2.5 times the predicted gain. This increased gain was estimated by comparing the step response and frequency acquisition of the breadboard to those obtained using equivalent models. The detector gain does appear to level off at the value

$$K_{DMax} = 2.5 * \mathrm{ncount} \qquad (37)$$

It achieves this gain at an input signal-to-noise density (at the prefilter input) of 80 dB-Hz. The exact mechanism for this increased gain is not precisely known. However, an increase in phase detector gain does not appear to occur when modulation of the carrier is present (see Fig. 40).

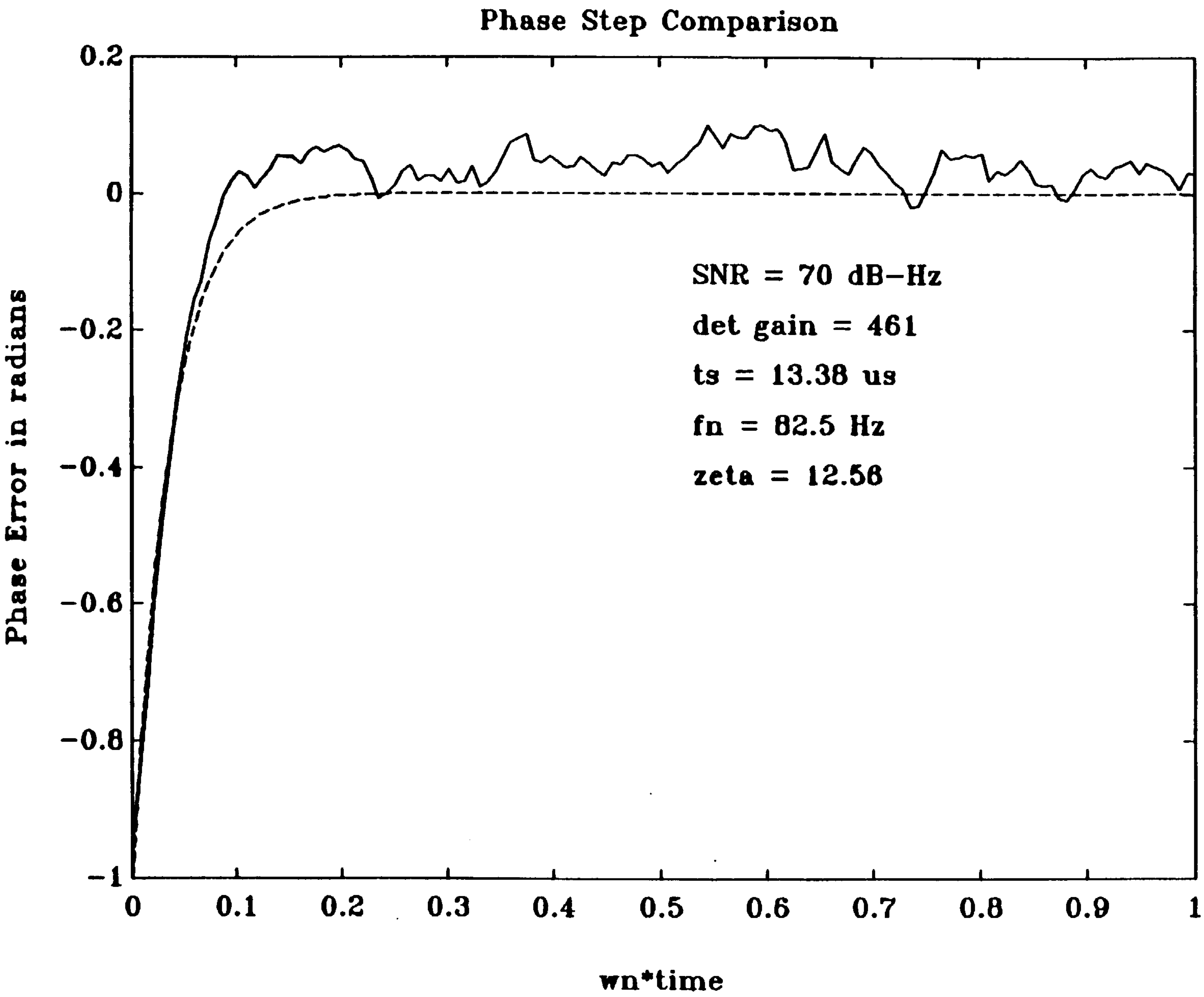


Fig. 31. Phase step comparison, no modulation, signal-to-noise density = 70 dB-Hz.

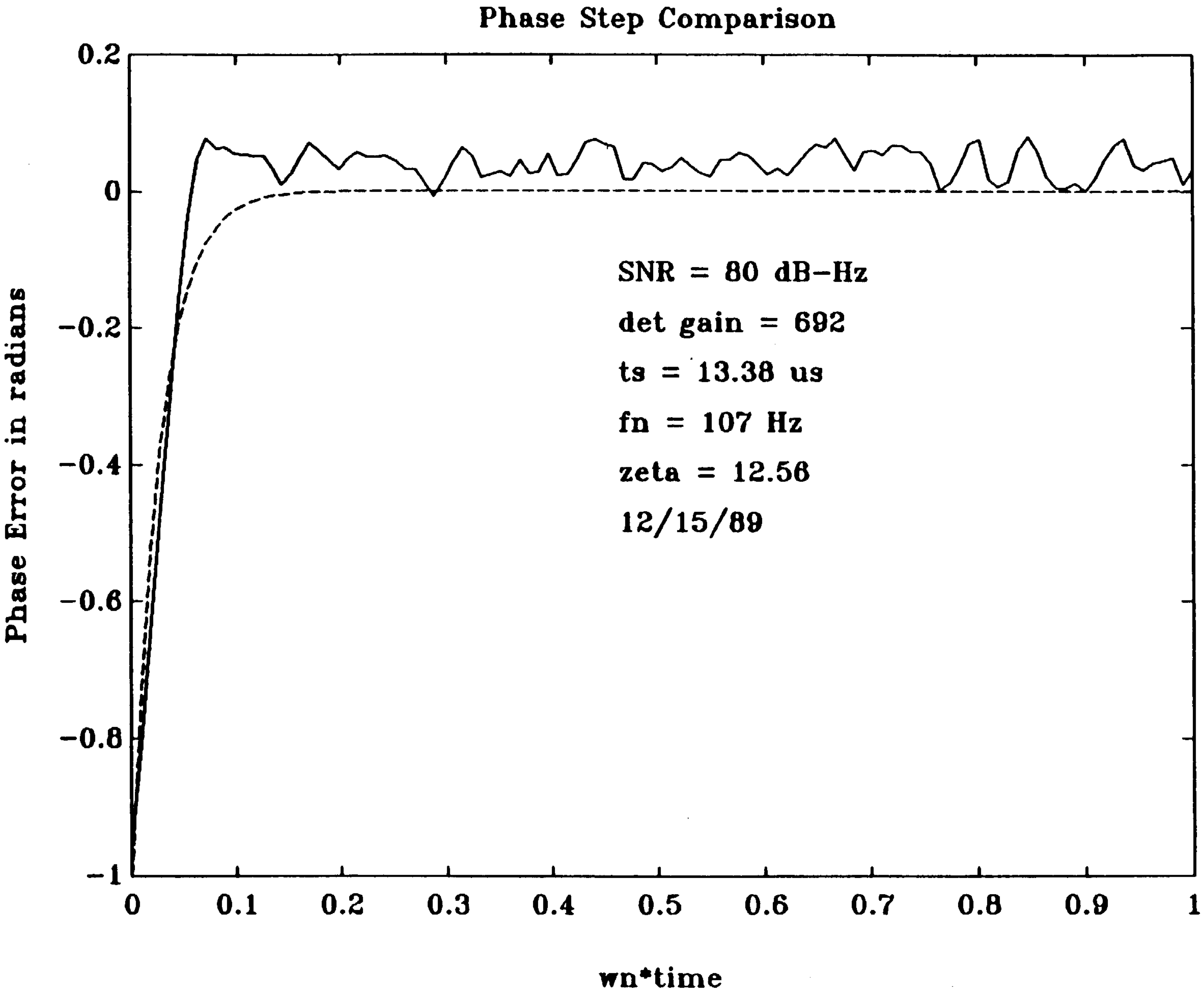


Fig. 32. Phase step comparison, no modulation, signal-to-noise density = 80 dB-Hz.

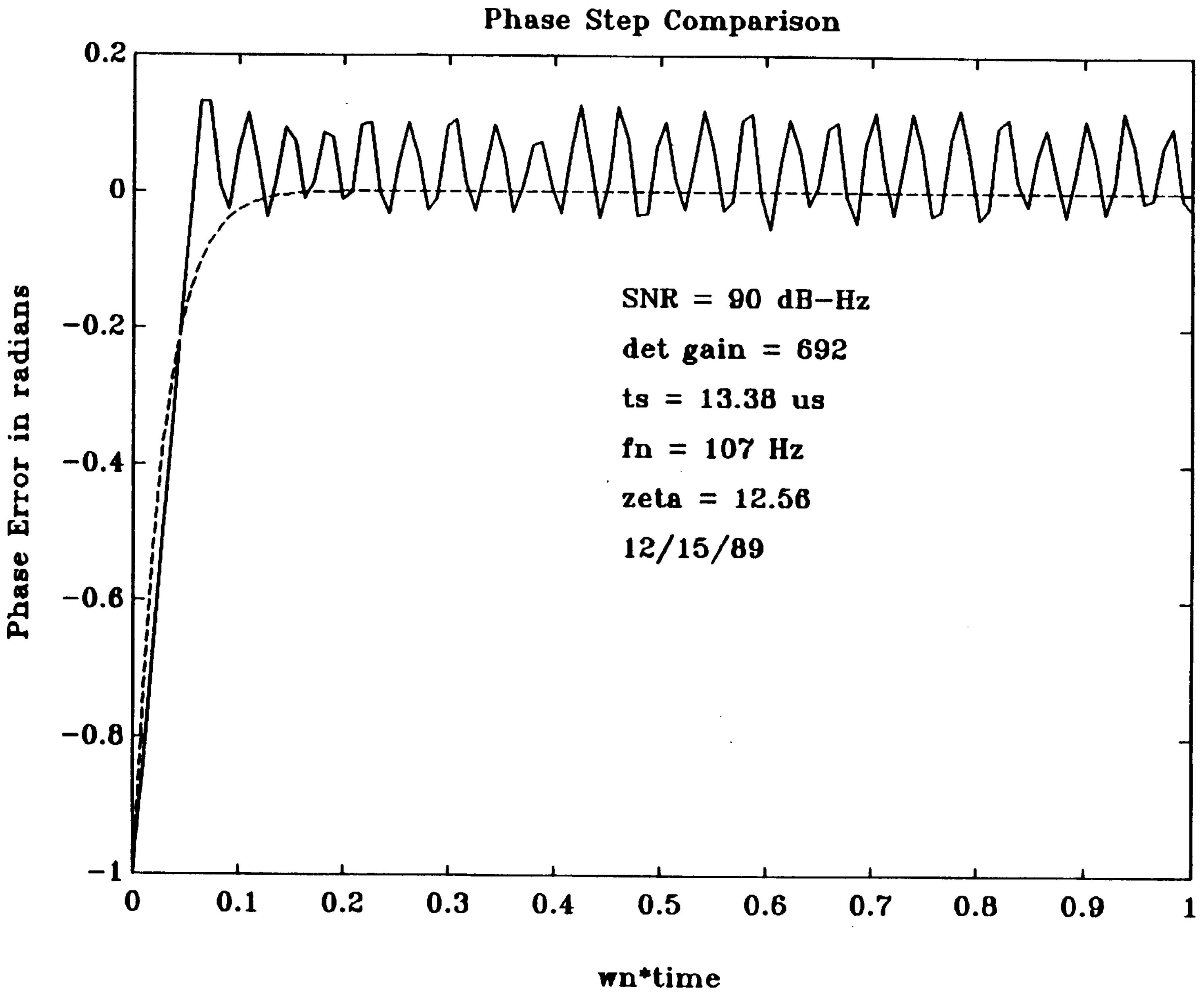


Fig. 33. Phase step comparison, no modulation, signal-to-noise density = 90 dB-Hz.

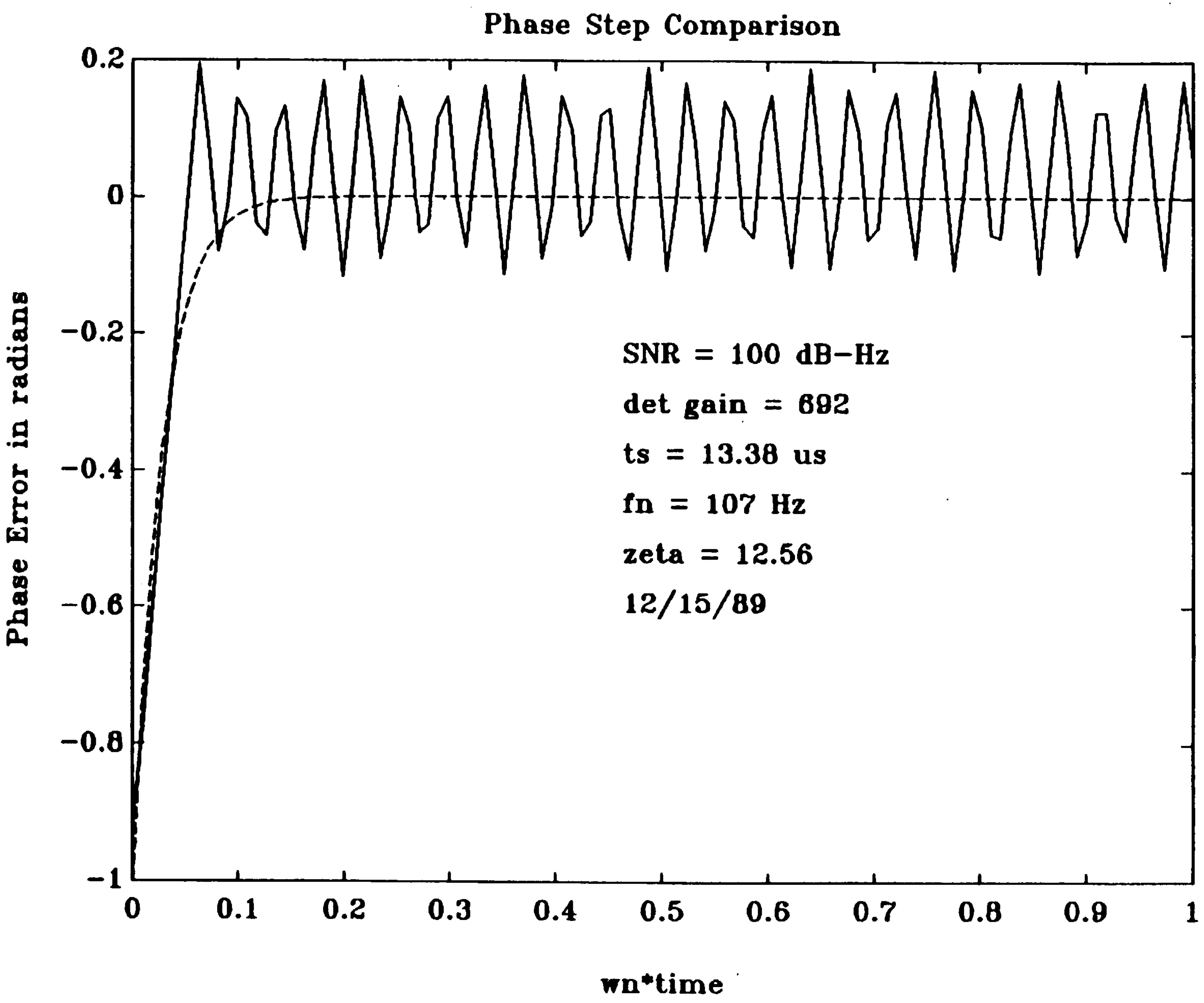


Fig. 34. Phase step comparison, no modulation, signal-to-noise density = 100 dB-Hz.

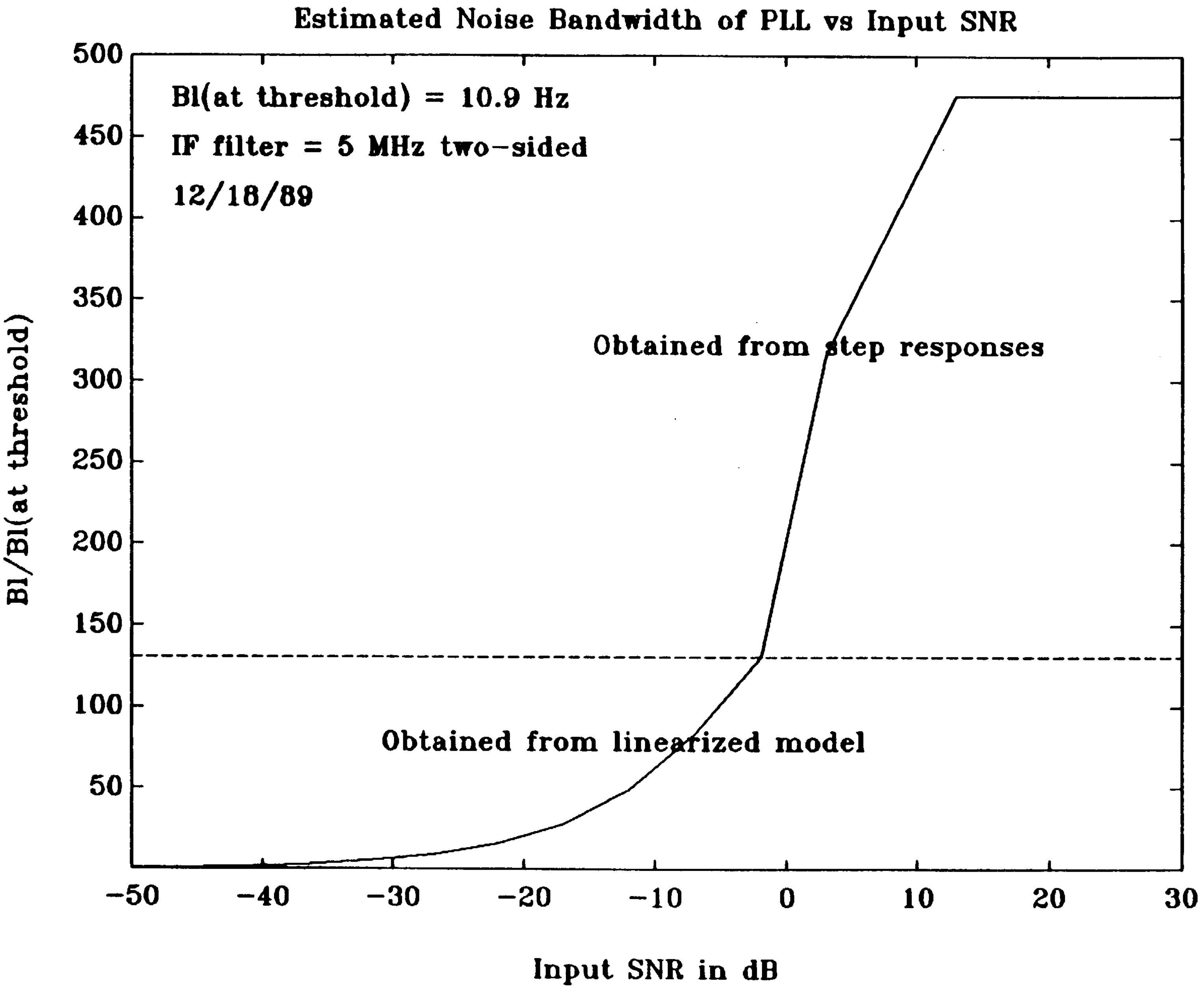


Fig. 35. Bandwidth expansion of software breadboard as a function of SNR, no modulation.

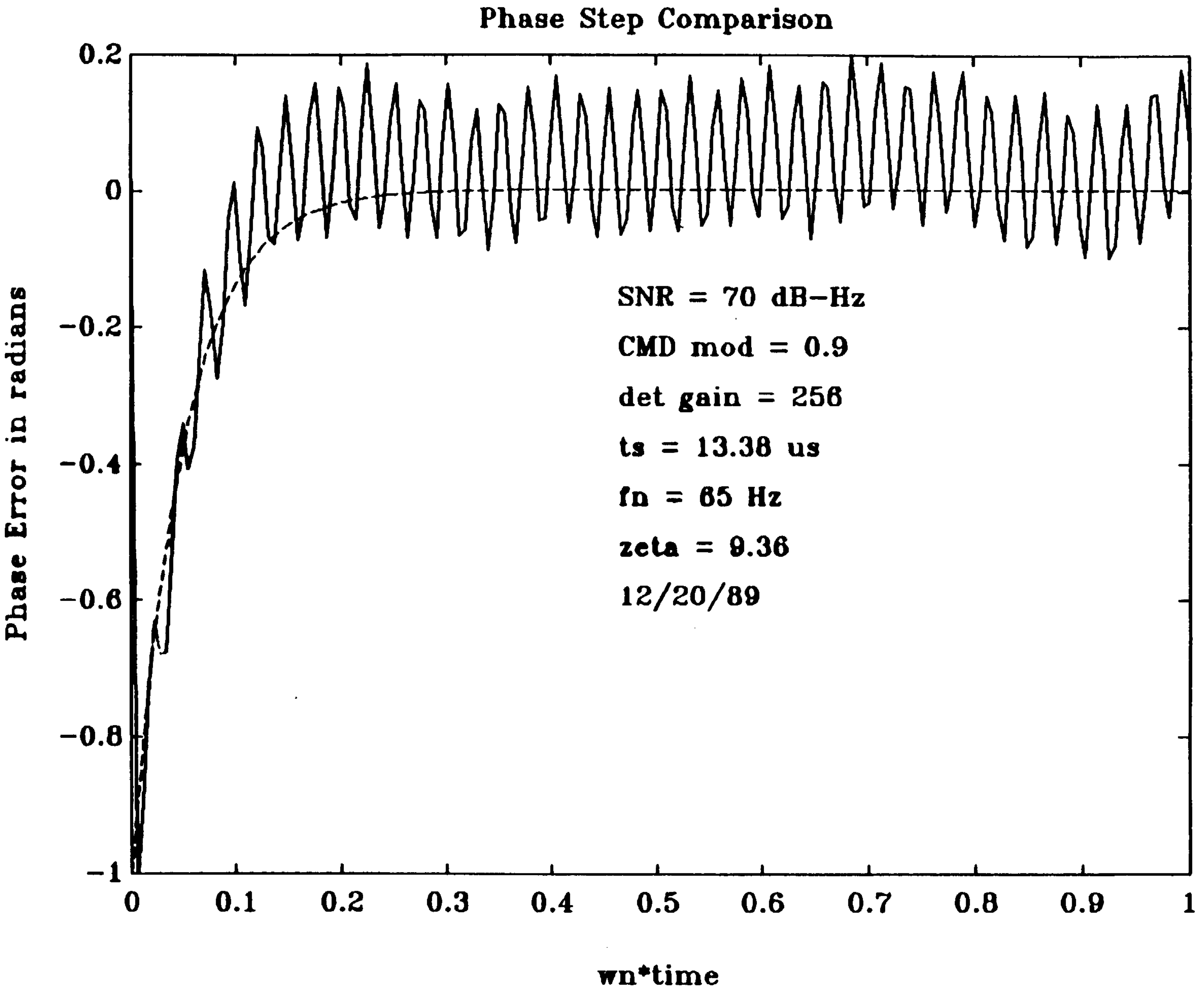


Fig. 36. Phase step comparison, CMD modulation, signal-to-noise density = 70 dB-Hz.

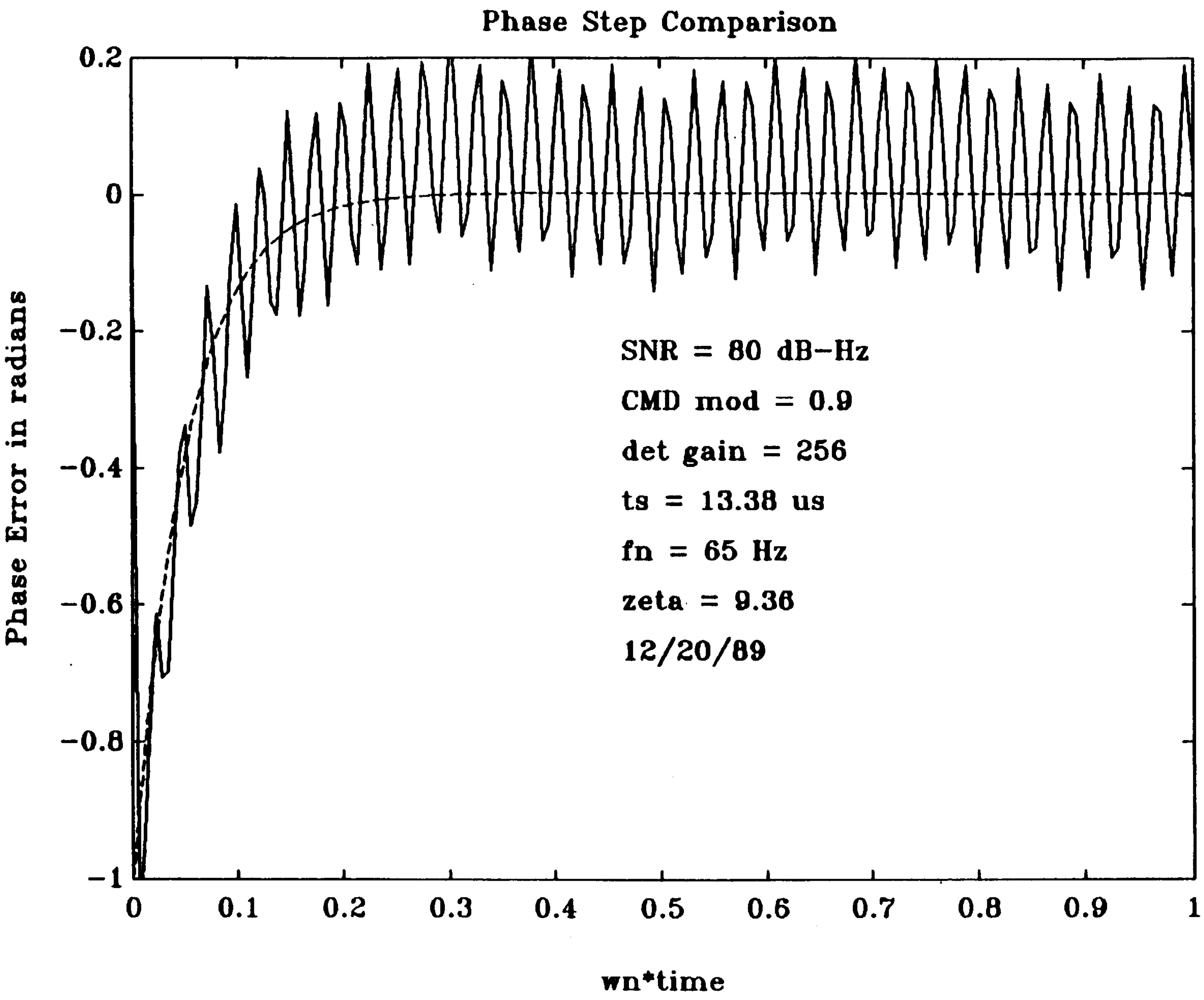


Fig. 37. Phase step comparison, CMD modulation, signal-to-noise density = 80 dB-Hz.

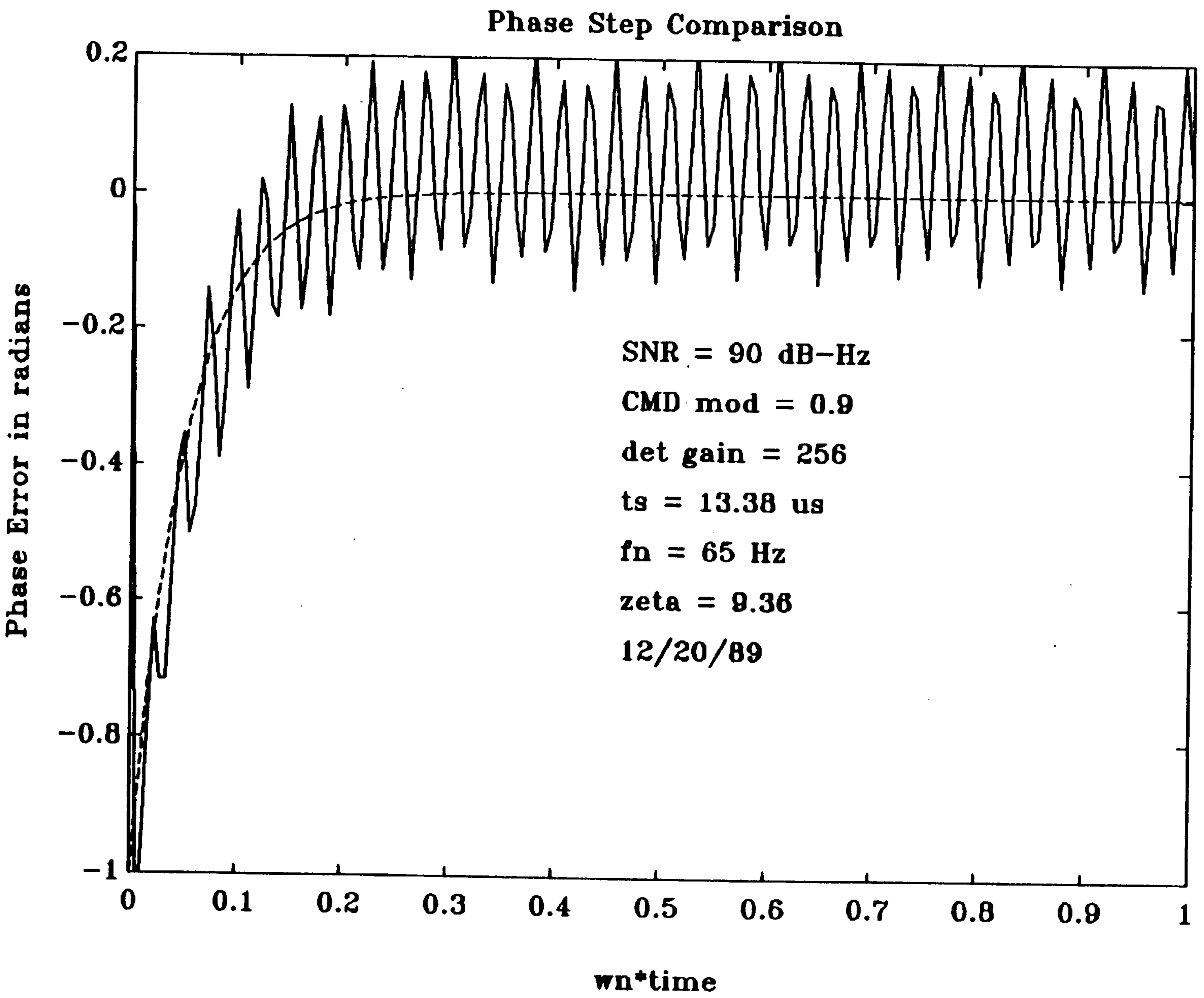


Fig. 38. Phase step comparison, CMD modulation, signal-to-noise density = 90 dB-Hz.

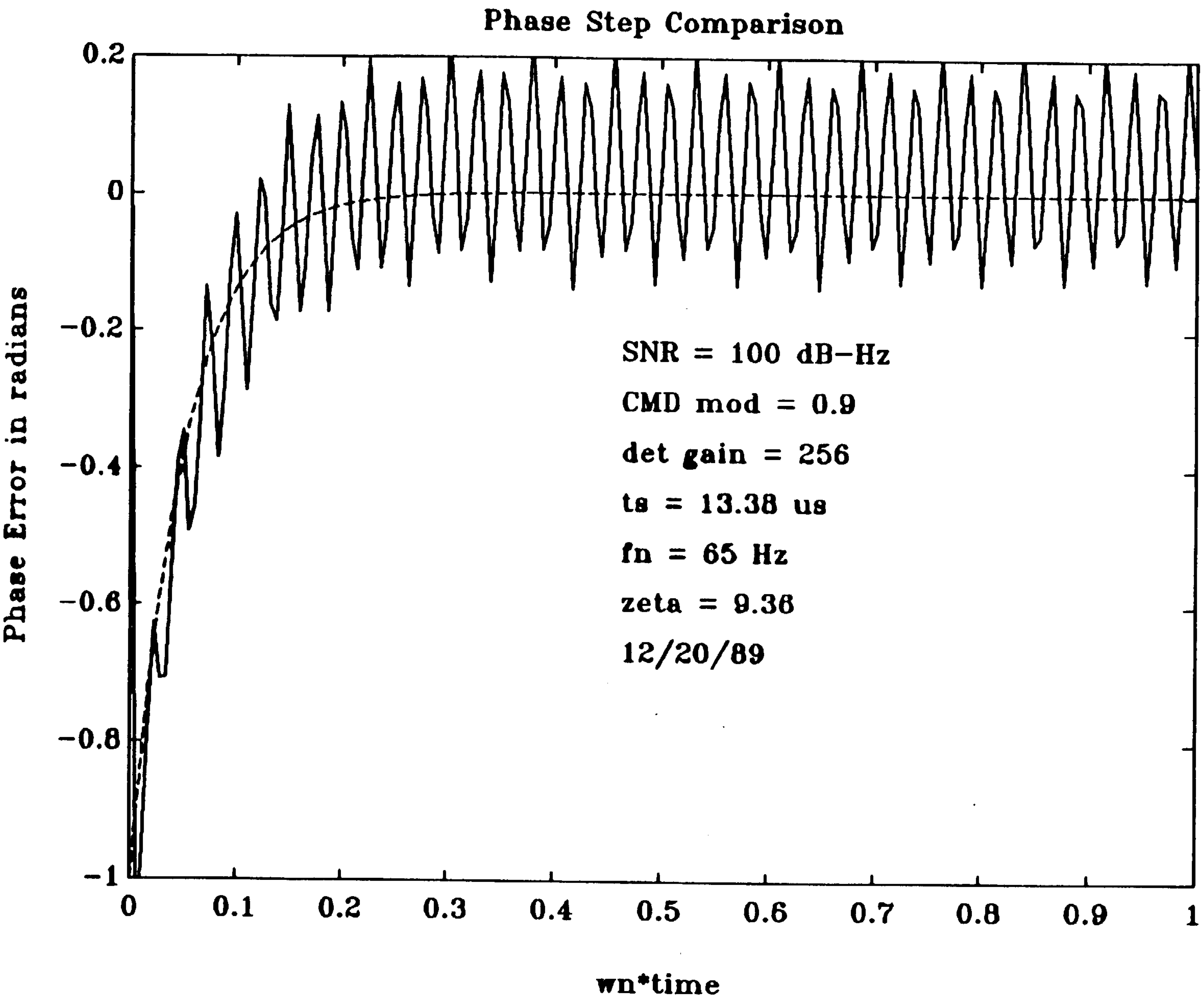


Fig. 39. Phase step comparison, CMD modulation, signal-to-noise density = 100 dB-Hz.

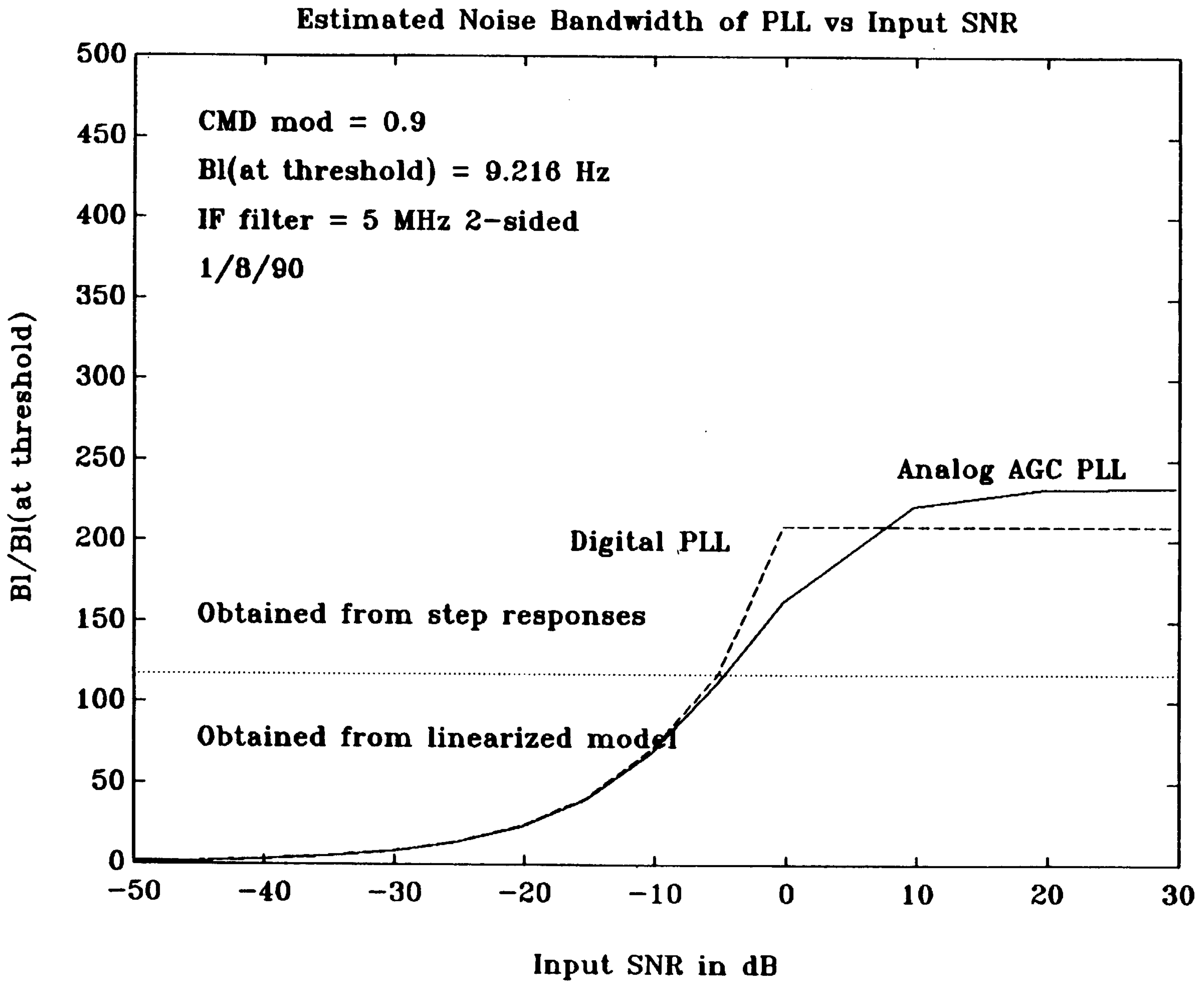


Fig. 40. Bandwidth expansion of software breadboard as a function of SNR, CMD modulation.

B. Output Signal-to-Noise Ratios

The output SNRs for the coherent amplitude detection (CAD), command, and ranging signals were estimated using the sample mean and sample standard deviation as discussed in Section II. Statistics were taken on the software breadboard for input signal-to-noise densities ranging from 30 to 100 dB-Hz. The simulation results are summarized in Table 3.

Table 3. Output SNRs

SNR in	CMD Mod	Range Mod	Output BW	1 Bit Loss	Mod. Loss	Filt. Loss	Calc. SNR	Simul SNR
30 dB-Hz	0.9	0.785	7.5 (CAD)	1.2- 1.5	3.25	0	-10.49- -10.19	-10.2
30	0.9	0.785	2.0 (CMD)	1.2- 1.5	3.20	0	-7.40- -7.10	-8.7
30	0.9	0.785	20.0 (RNG)	1.2- 1.5	4.64	3	-22.15- -21.85	-20.6
50	0	0	7.5 kHz (CAD)	1.2- 1.5	0	0	12.76- 13.06	14.0
50	0.9	0.785	7.5 (CAD)	1.2- 1.5	3.25	0	9.51- 9.81	10.7
50	0.9	0.785	2.0 (CMD)	1.2- 1.5	3.20	0	12.60- 12.30	12.1
50	0.9	0.785	20.0 (RNG)	1.2- 1.5	4.64	3	-2.15- -1.85	-1.0
60	0	0	7.5 kHz (CAD)	1.2- 1.5	0	0	22.76- 23.06	24.4
60	0.9	0.785	7.5 (CAD)	1.2- 1.5	3.25	0	19.51- 19.81	20.7
60	0.9	0.785	2.0 (CMD)	1.2- 1.5	3.20	0	22.60- 22.90	21.4
60	0.9	0.785	20.0 (RNG)	1.2- 1.5	4.64	3	8.16- 8.46	8.9
80	0	0	7.5 (CAD)	1.2- 1.5	0	0	43.36- 43.66	83.1

80	0.9	0.785	7.5 (CAD)	1.2- 1.5	3.25	0	40.11- 40.41	38.5
80	0.9	0.785	2.0 (CMD)	1.2- 1.5	3.20	0	43.20- 43.50	40.9
80	0.9	0.785	20.0 (RNG)	1.2- 1.5	4.64	3	28.76- 29.06	25.2
100	0	0	7.5 (CAD)	1.2- 1.5	0	0	63.36- 63.66	N/A
100	0.9	0.785	7.5 (CAD)	1.2- 1.5	3.25	0	60.11- 60.41	42.2
100	0.9	0.785	2.0 (CMD)	1.2- 1.5	3.20	0	63.20- 63.50	55.4
100	0.9	0.785	20.0 (RNG)	1.2- 1.5	4.64	3	48.76- 49.06	34.5

The calculated values for the output SNRs were obtained as follows

$$\text{SNR}_{\text{calc}} = \text{SNR}_{\text{in}} - 10*\log(\text{output BW}) - \Sigma(\text{losses}) \qquad (38)$$

and are given in dB. The 1 bit loss represents that predicted in the report [ATS89]. This report predicted the losses for both an ideal and single pole filter. Since in the software breadboard a double pole prefilter is used, it was felt that the loss would lie between these two values.

Of most significance in Table 3. are the results for the negative input SNR cases, since at positive SNR levels (i.e., >65 dB-Hz), the output SNR is well above the design threshold value. For this reason, Table 4 gives 99% confidence intervals [NET85] on the results for the 30 and 50 dB-Hz cases.

Table 4. SNR 99% Confidence Intervals

SNR	Signal	Sample Mean	Sample Std.	# Data Points	99% SNR
30	CAD	25.77	83.26	5000	[-11.2 -9.28]
30	CMD	231.63	629.57	5000	[-9.68 -7.79]
30	Rng	0.0264	0.2821	5000	[-22.0 -19.4]
50	CAD	266.43	77.35	500	[10.6 10.9]
50	CMD	238.52	59.57	500	[12.0 12.1]
50	Rng	0.2544	0.2866	500	[-1.48 -0.608]

As can be seen from the table, we can have good confidence in our data for these cases.

C. Voltage Controlled Oscillator Phase Jitter

It is of interest to measure the jitter on the phase of the VCO in order to determine the effects of the quantization noise on receiver and transmitter specifications. The jitter will be estimated using the sample standard deviation, (30). The interpolator at the input to the DAC was shown in Fig. 1. For the cases being considered, the interpolator will feed back the 10 lower bits and send the higher bits to the DAC. The jitter will be measured with 8 and 10 bits of DAC quantization. The number of DAC bits will affect the DAC gain per (11), so that the loop filter will have to be adjusted accordingly to obtain similar loop parameters. Table 5. summarizes the phase jitter results obtained from the software breadboard.

Table 5. VCO Phase Jitter

SNR in	CMD Mod	Range Mod	DAC Configuration	Analog Phase Jitter*	Simulation Phase Jitter
50 dB-Hz	0.	0.	8 bit Interp.	2.8778 deg rms	3.0062 deg rms
50	0.	0.	10 bit Interp.	3.1872	3.4354
60	0.	0.	8 bit Interp.	1.5529	1.8960
60	0.	0.	10 bit Interp.	1.7306	2.1848
80	0.	0.	8 bit Interp.	0.2251	0.8087
80	0.	0.	10 bit Interp.	0.2512	0.9736
100	0.	0.	8 bit Interp.	0.0225	3.6382
100	0.	0.	10 bit Interp.	0.0251	4.2002

*Note: This number changes from 8->10 bits because BL changes due to the redistribution of loop gains which are integer numbers.

Note that at negative SNRs, the jitter is only slightly greater than that contributed by the input noise, but at positive SNRs, the quantization noise tends to dominate. At the 100 dB-Hz level, the limit cycle which was observed in Fig. 17 causes the high value of phase jitter.

D. Command and Ranging Intermodulation Effects

The predicted levels of the various frequency components which would be output from a linear phase detector were given in Tables 1. and 2. The frequency components of the signals output from the software breadboard of the 1 bit detector will be presented in this section. Of particular interest will be the case of positive SNR where the 1 bit A/D no longer acts like a linear device.

In order to observe the frequency spectra, a 1024 point FFT was used to process the data at the ranging filter output. In addition, another I/D filter of the same length as the ranging filter was added to the I channel, and the output of this filter was also processed using the FFT. The FFTs were performed over 20, 1024 point blocks of data, and the magnitude spectrums of all the blocks were averaged in order to reduce the noise variance in the FFTs.

Figs. 41 and 43 are the results of this FFT processing on the I and Q channels, respectively. Figs. 42 and 44 are corresponding plots assuming an ideal sampling phase detector. These plots were obtained by normalizing the I channel DC level to 5.45 dB (see Fig. 41), and then using Tables 1. and 2. to determine the other frequency components relative to the DC value. In addition to the numbers given in the tables, an additional 3 dB was subtracted from those tones near the ranging frequency to account for the I/D filter roll off. Note that the spectrums obtained using the FFTs give more qualitative results than quantitative results since the amplitudes of the tones will depend on the number of samples taken and the resolution of the FFT. However, as can be seen, the theoretical and experimental spectrums show close agreement for this 60 dB-Hz case.

Figs. 45 through 52 are comparable results for the 80 and 100 dB-Hz cases. These spectra show that the sideband tones are definitely higher than for the ideal phase detector. At these high signal levels, the nonlinear nature of the 1 bit A/D causes intermodulation between the command and ranging tones.

Finally, Fig. 53 shows the spectrum of the Q channel output focused on the 16 kHz command tone. The spectrum of this figure was obtained at the maximum signal level of 100 dB-Hz. This figure shows in detail that there is no evidence of the 19 kHz limit cycle (cf. Section III.A.2) with the command modulation on.

E. Ranging Filter Output Spectra

Since under normal operating conditions, the carrier will not be modulated by both command and ranging tones, it is of interest to plot the ranging filter output spectra in either the command or ranging mode. To this end, spectra were obtained for various ranging frequencies and modulation indices. Table 6 summarizes the spectrum plots which were obtained using the software breadboard.

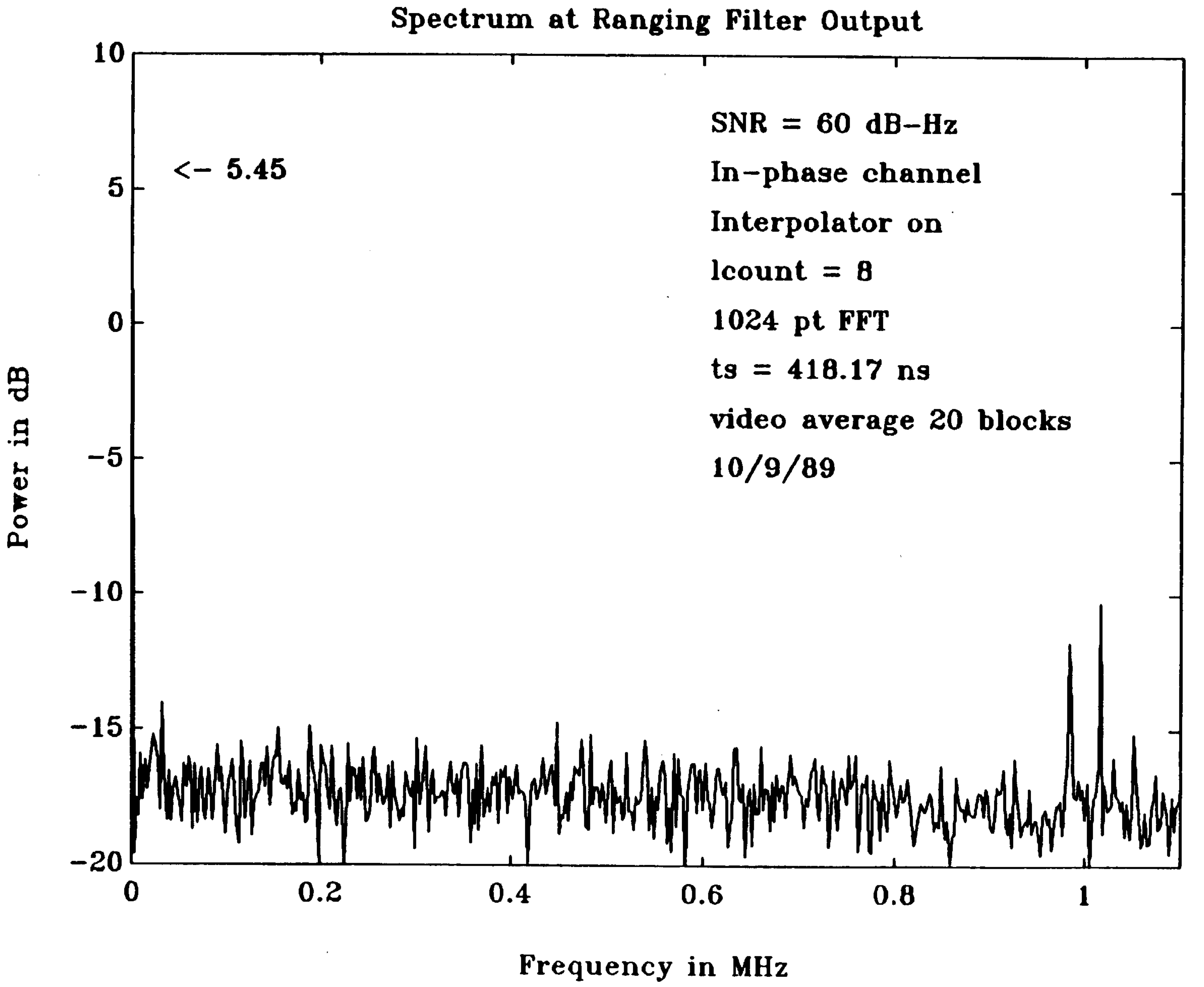


Fig. 41. Software breadboard I channel magnitude spectrum , negative SNR.

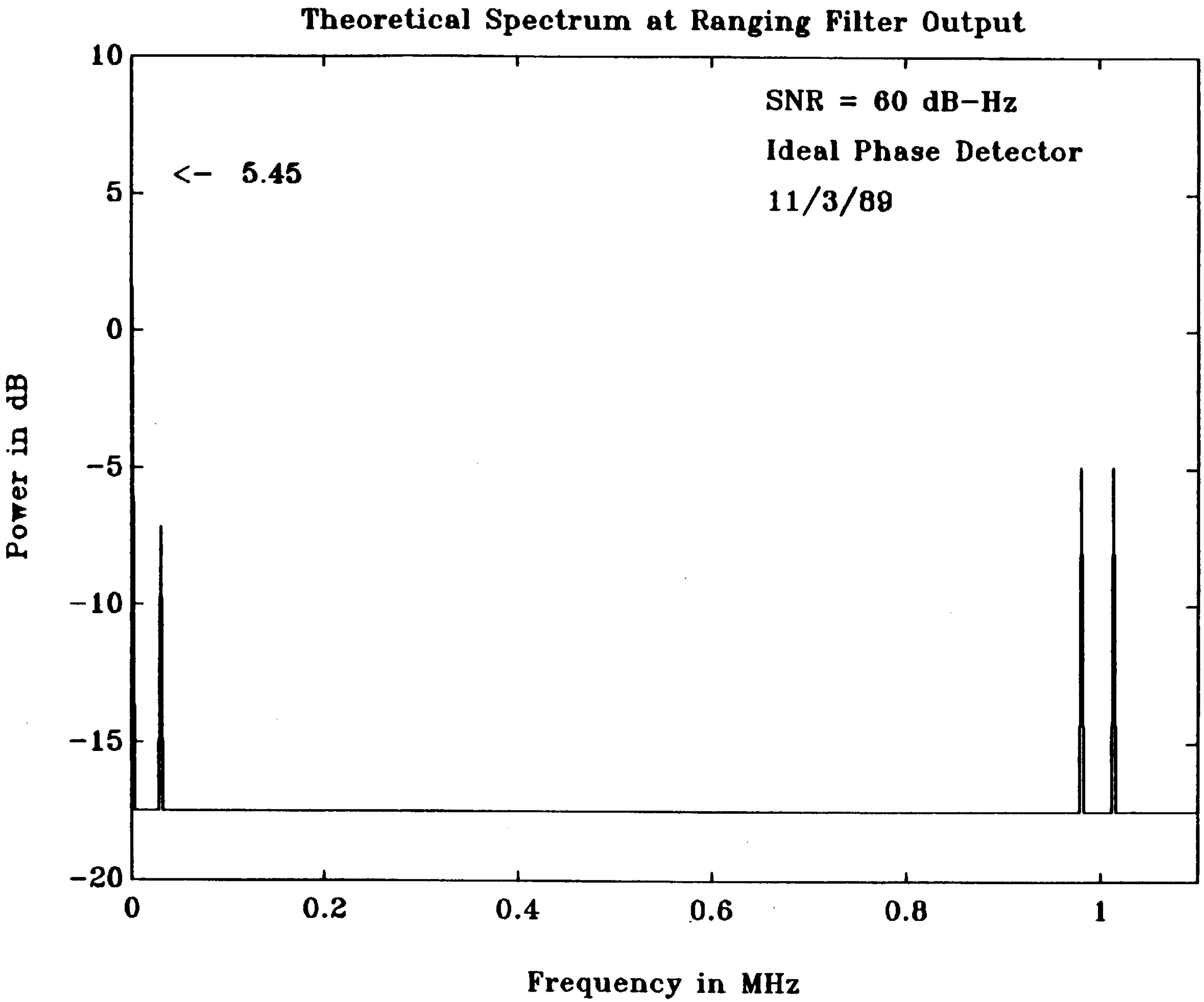


Fig. 42. Ideal I channel magnitude spectrum, negative SNR.

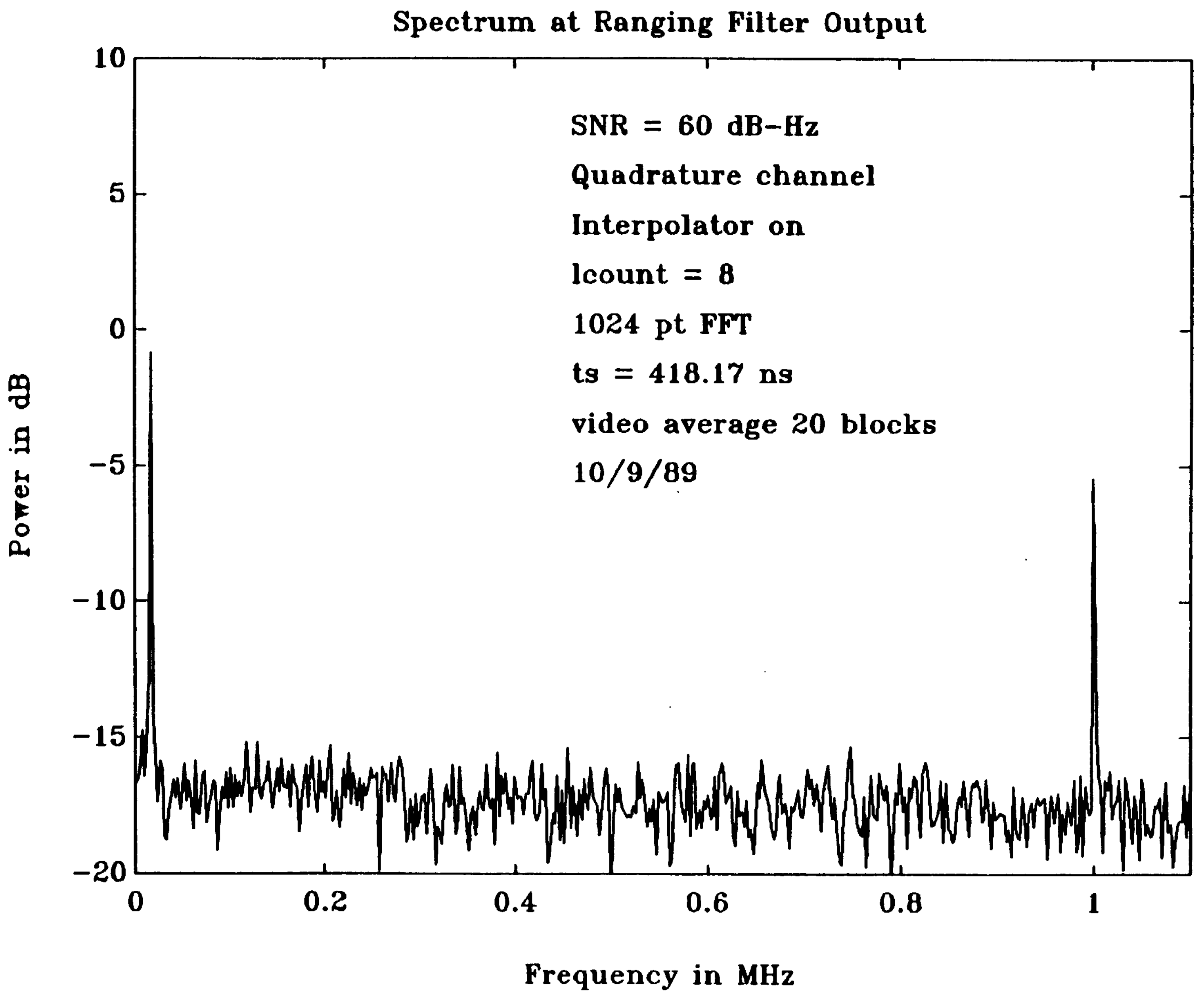


Fig. 43. Software breadboard Q channel magnitude spectrum, positive SNR.

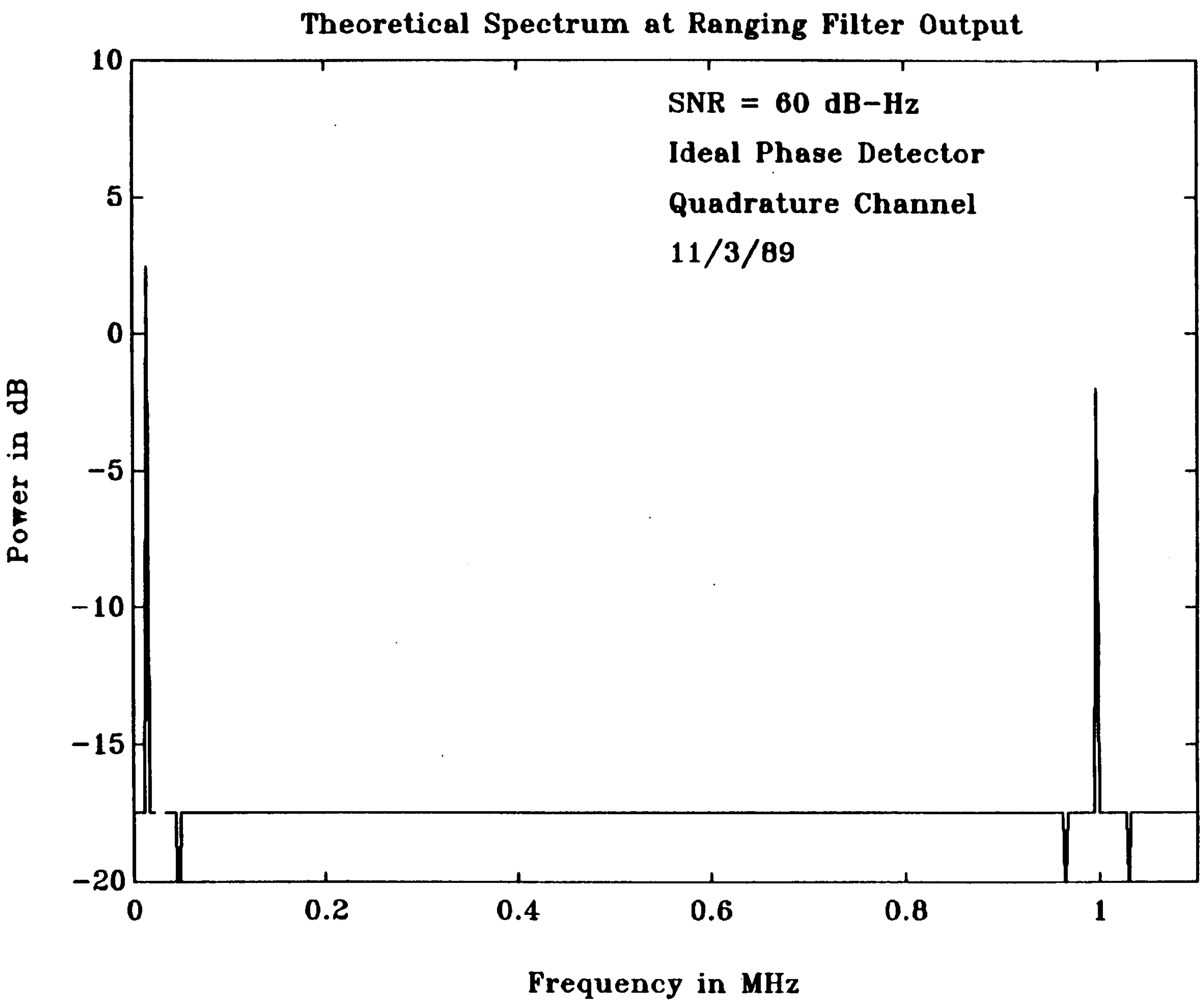


Fig. 44. Ideal Q channel magnitude spectrum, positive SNR.

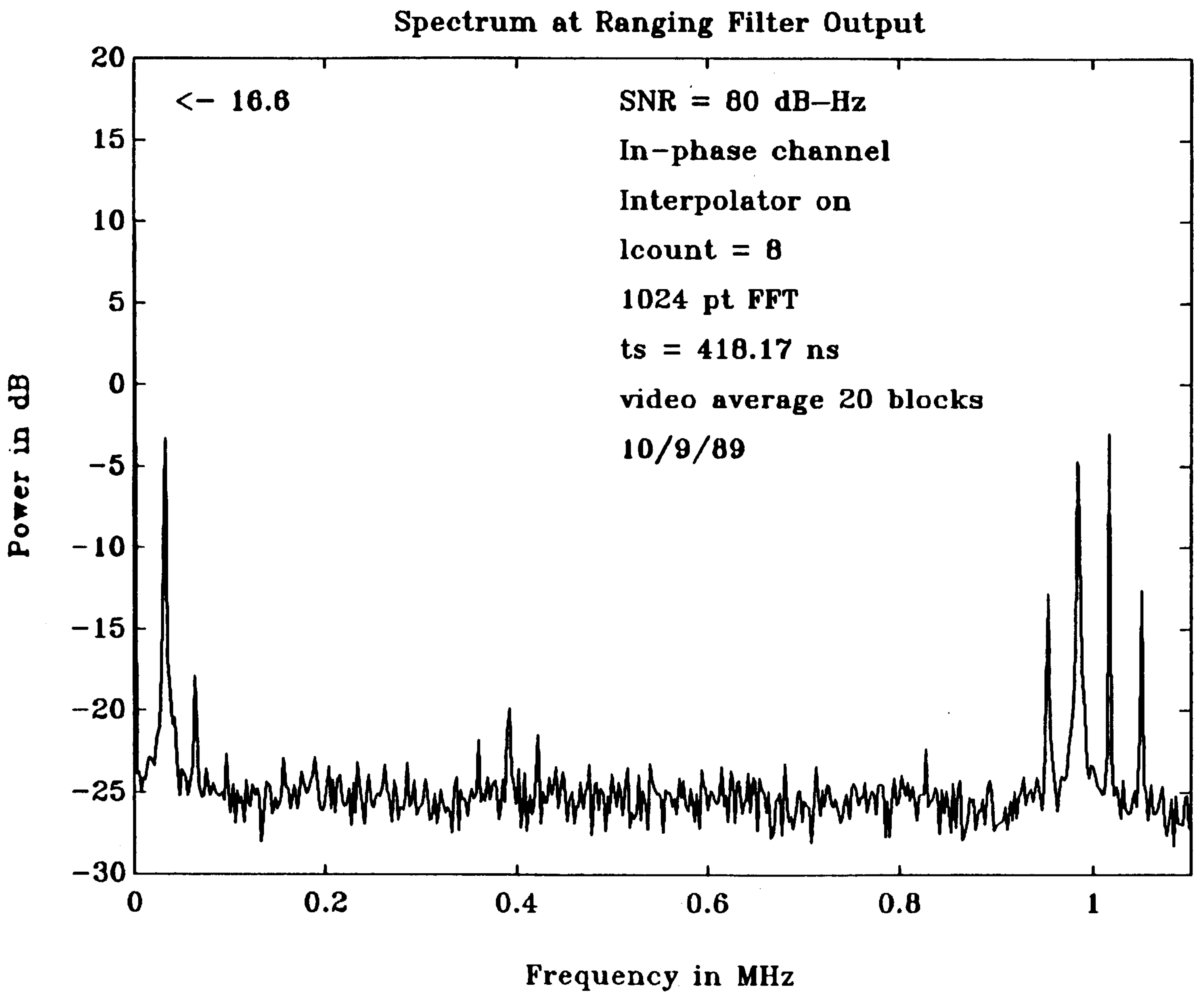


Fig. 45. Software breadboard I channel magnitude spectrum, positive SNR.

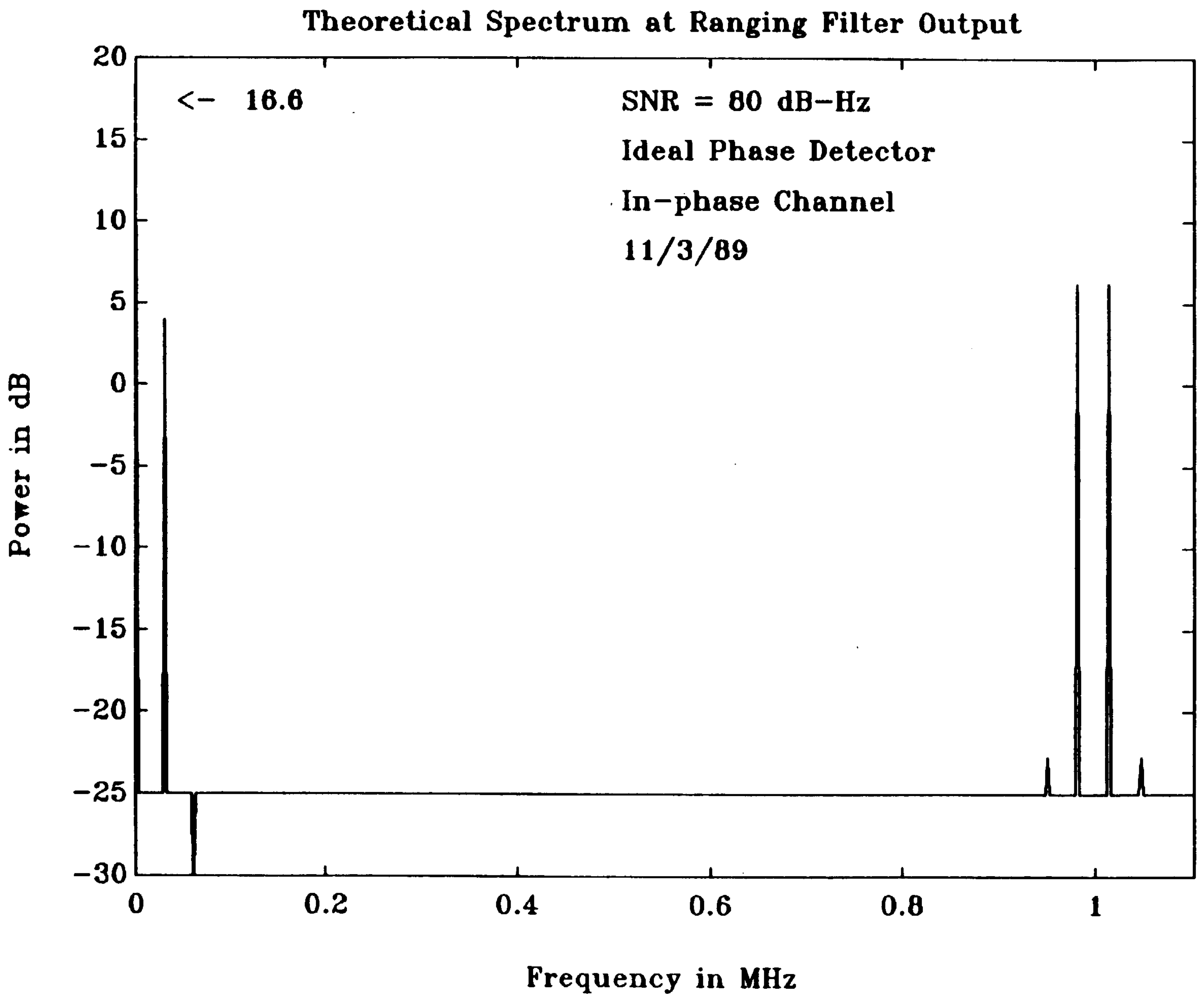


Fig. 46. Ideal I channel magnitude spectrum, positive SNR.

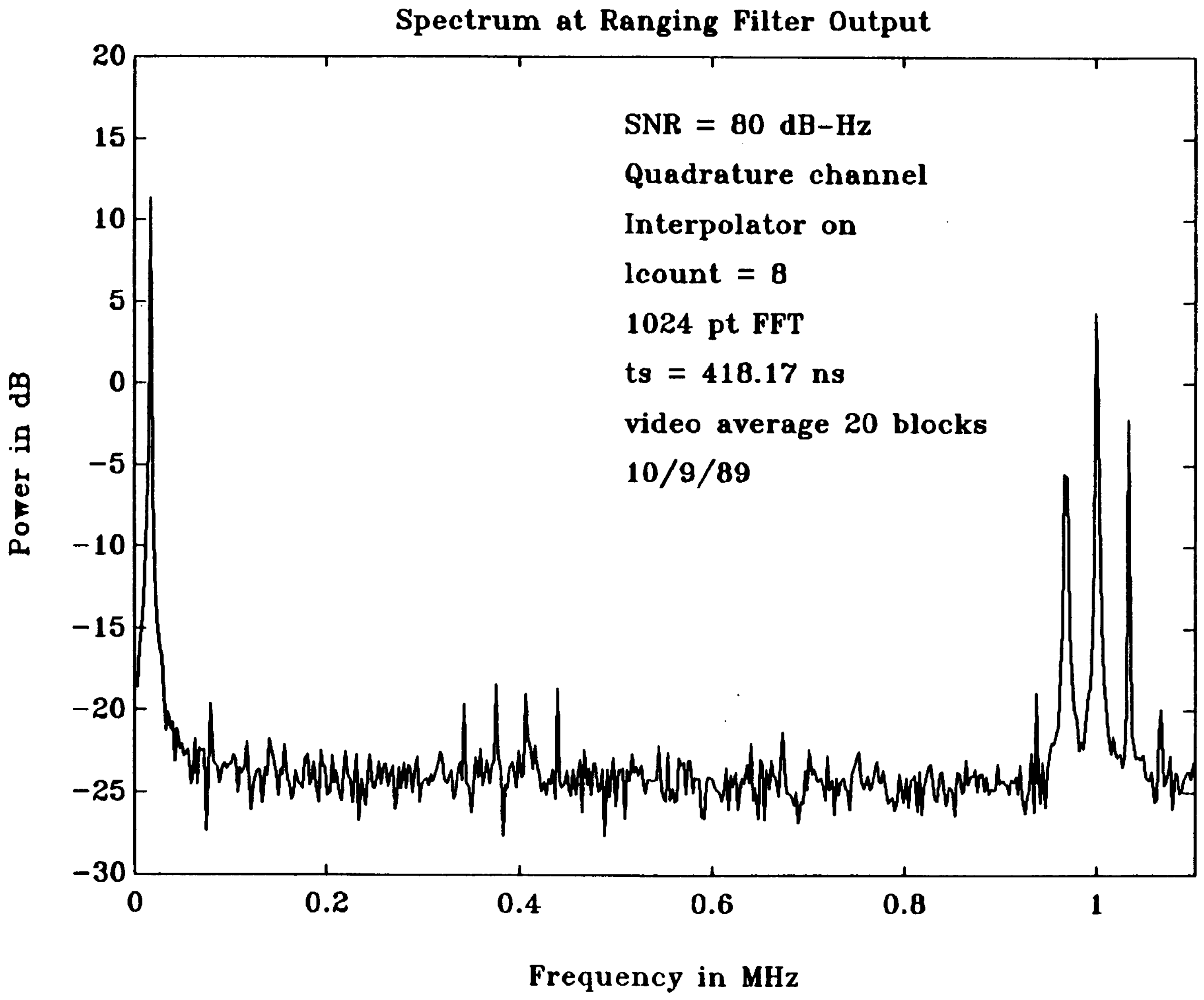


Fig. 47. Software breadboard Q channel magnitude spectrum, positive SNR.

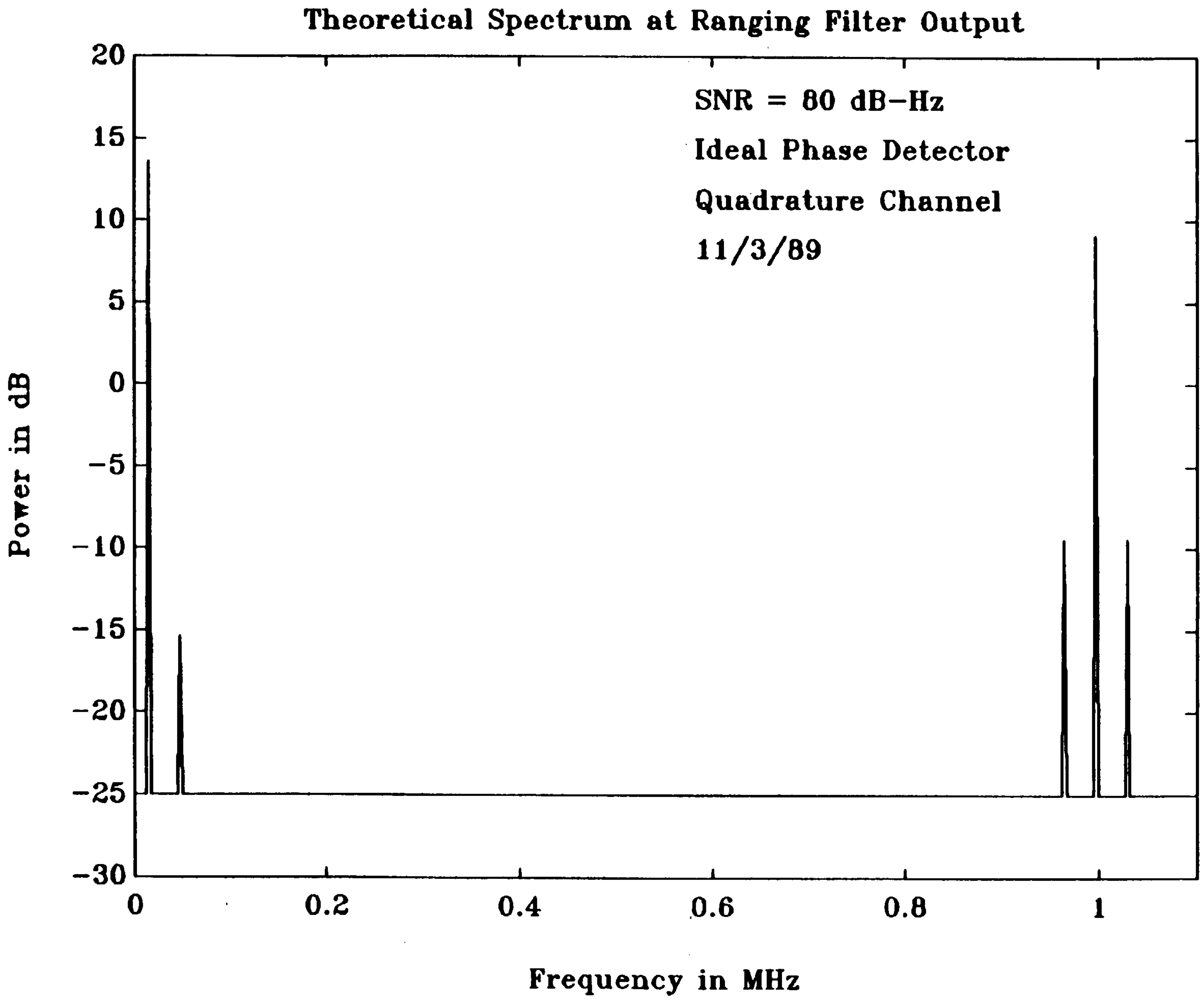


Fig. 48. Ideal Q channel magnitude spectrum, positive SNR.

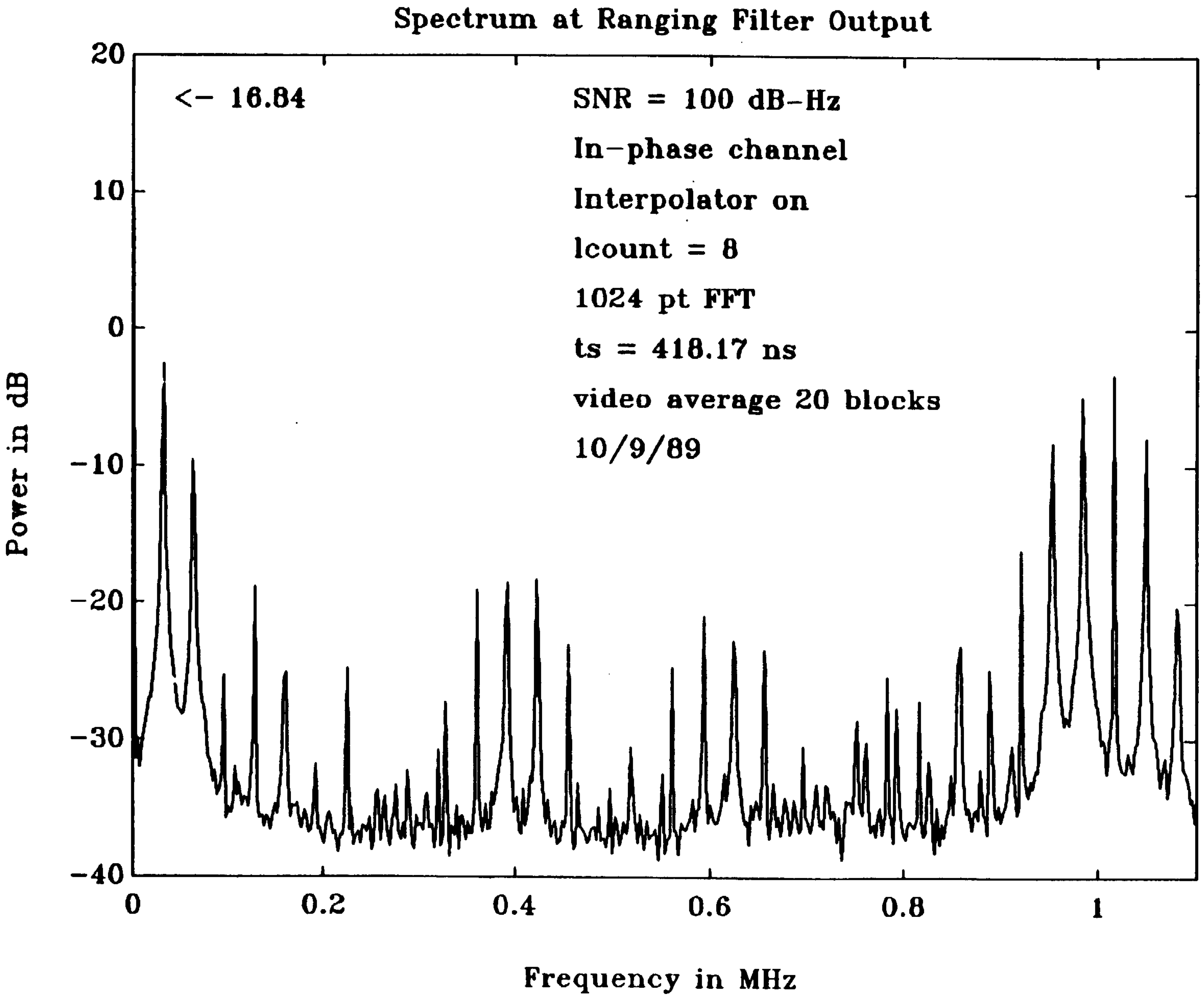


Fig. 49. Software breadboard I channel magnitude spectrum, maximum SNR.

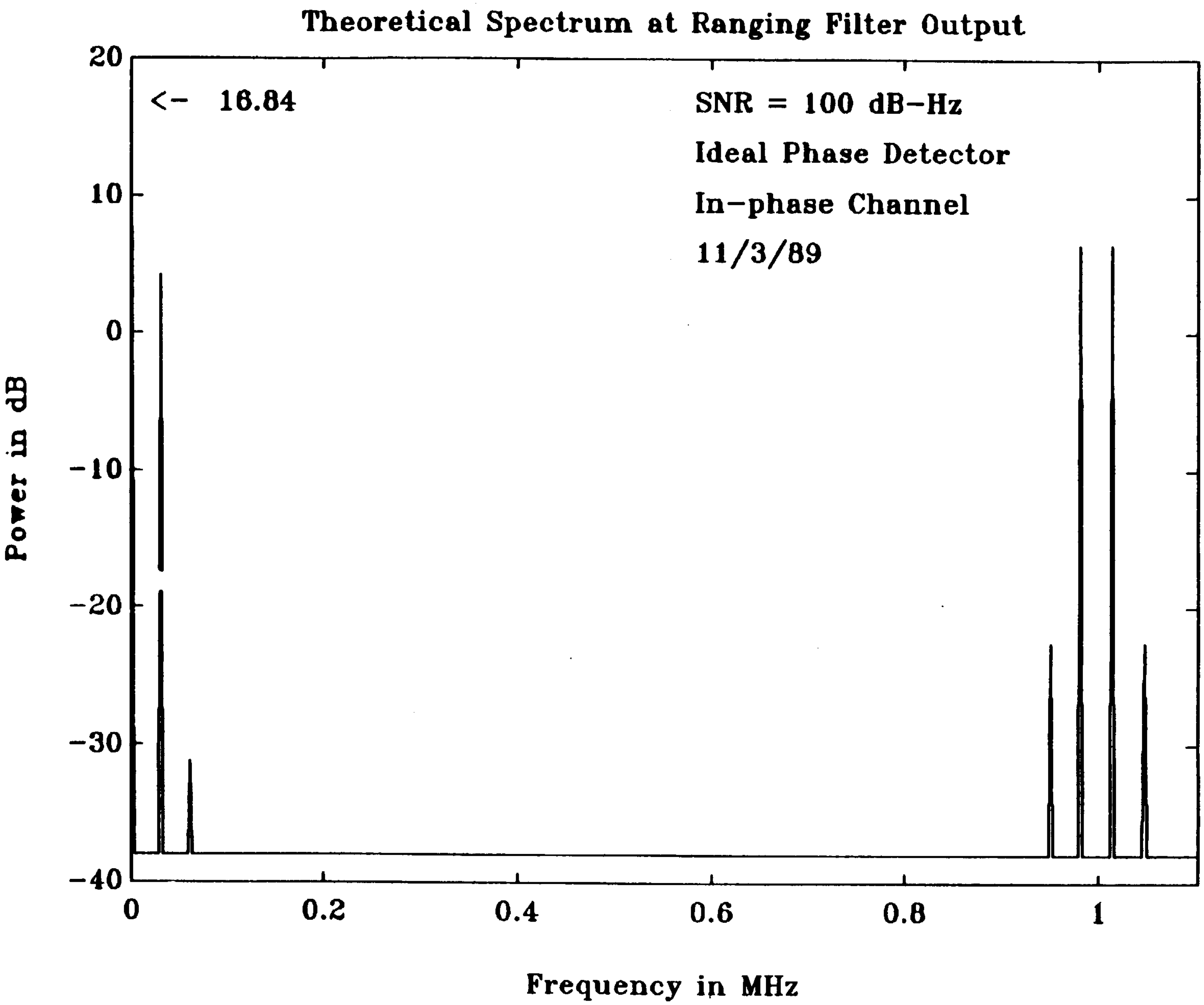


Fig. 50. Ideal I channel magnitude spectrum, maximum SNR.

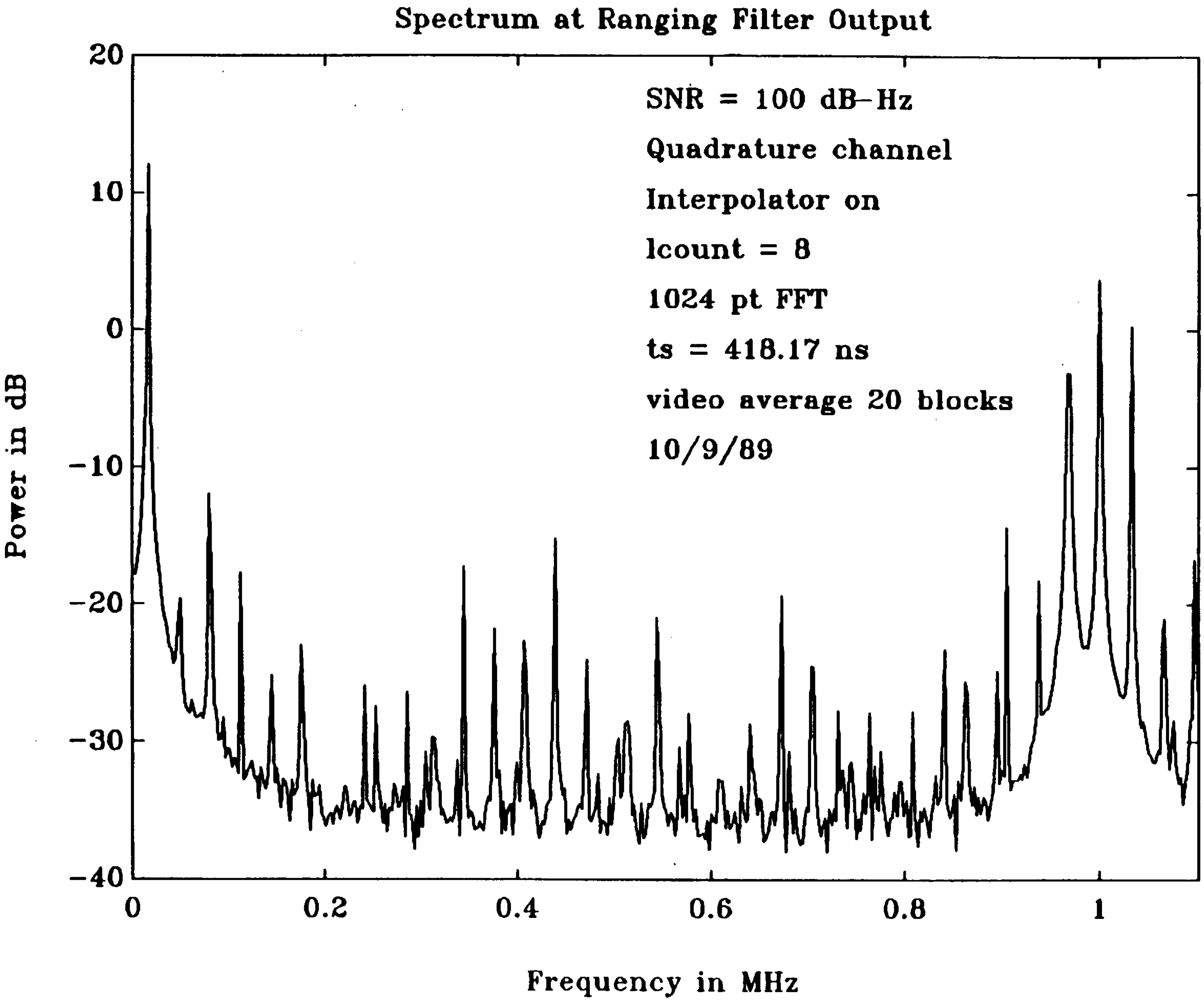


Fig. 51. Software breadboard Q channel magnitude spectrum, maximum SNR.

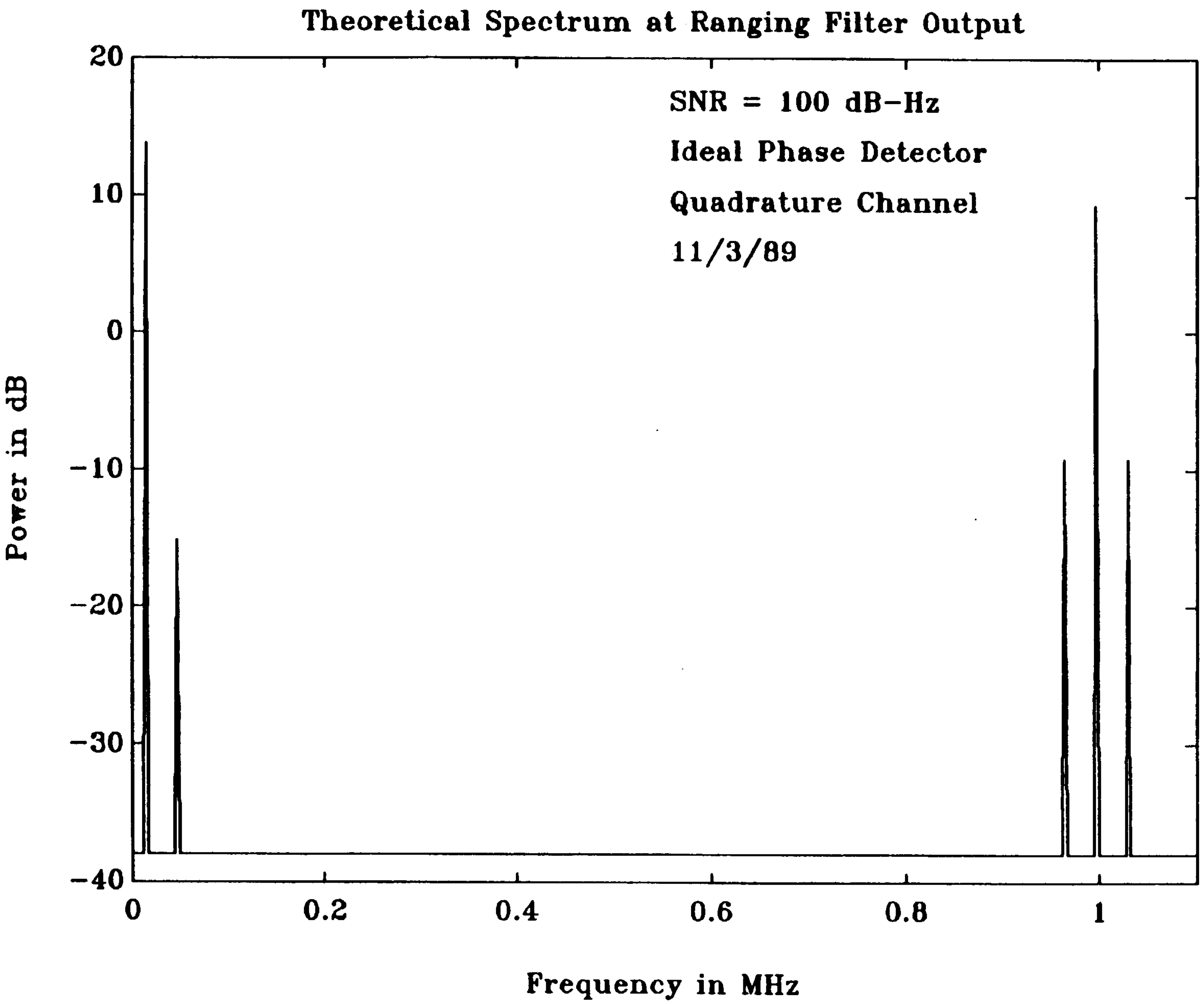


Fig. 52. Ideal Q channel magnitude spectrum, maximum SNR.

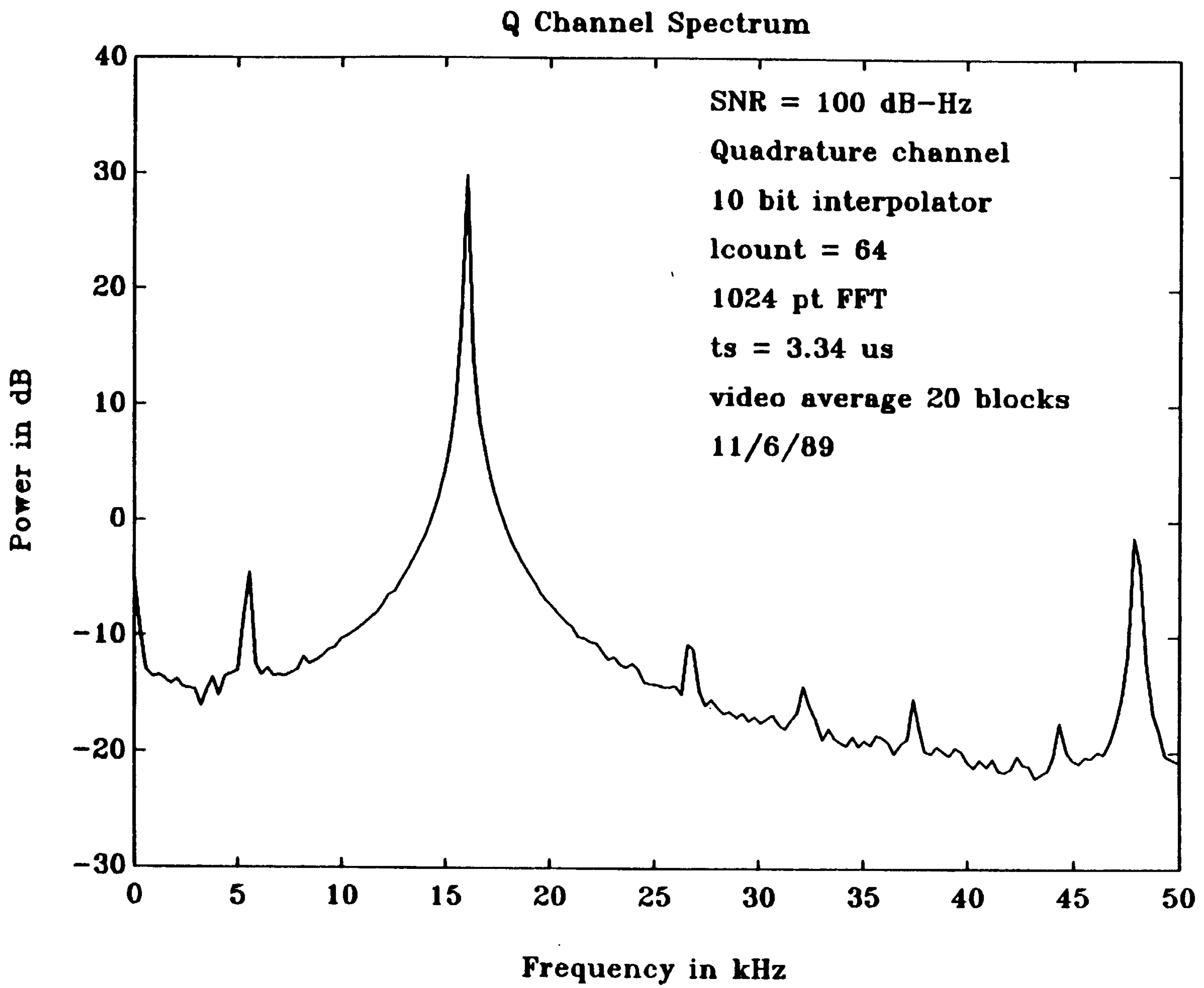


Fig. 53. Software breadboard Q channel magnitude spectrum, maximum SNR showing command tone.

Table 6. Spectrum Plots

SNR	CMD mod	CMD freq	Range mod	Range freq	Fig. #
60	-	-	0.785	1 MHz	54
100	-	-	0.785	1 MHz	55
60	-	-	1.3	1 MHz	56
100	-	-	1.3	1 MHz	57
60	-	-	0.785	100 kHz	58
100	-	-	0.785	100 kHz	59
60	-	-	1.3	100 kHz	60
100	-	-	1.3	100 kHz	61
60	-	-	0.785	10 kHz	62
100	-	-	0.785	10 kHz	63
60	-	-	1.3	10 kHz	64
100	-	-	1.3	10 kHz	65
60	-	-	0.785	1 kHz	66, 67
100	-	-	0.785	1 kHz	68, 69
60	-	-	1.3	1 kHz	70, 71
100	-	-	1.3	1 kHz	72, 73

In this table, the 60 dB-Hz cases represent linear operation of the 1 bit A/D, while the 100 dB-Hz cases represent maximum input signal level. The ranging frequencies considered vary from 1 kHz to 1 MHz, with modulation indices of 0.785 and 1.3 radians. Note that for a 1 kHz ranging frequency, additional expanded plots are given in order to resolve the 1 kHz tone at the output. In all cases where the input SNR was equal to 60 dB-Hz, no harmonics of the fundamental frequency are seen in the output spectra. However, at the 100 dB-Hz level, odd harmonics of the fundamental frequency are clearly present in the output. Note that at 1 MHz and 100 kHz ranging frequencies, aliasing causes the higher harmonics to be folded back in at lower frequencies.

The frequency response of the linearized model of Fig. 6 can be used to predict the response of the breadboard to different ranging tone frequencies. Fig. 74 shows the phase detector output versus ranging frequency for the linearized model. This figure was obtained under the assumptions of strong signal and operation of the loop as in the bandwidth expansion curve of Fig. 40 (modulation on). Note that Fig. 74 indicates that a 1 kHz ranging tone will be attenuated by 4 dB at strong signal level. Little attenuation will be experienced by tones at frequencies above 10 kHz.

F. Ranging Phase Delay

One of the principal advantages of using a digital receiver over its analog equivalent is the reduction in ranging delay variations over time and temperature. The delay experienced by the ranging signal is caused by two separate filters (i.e., one analog, the other digital). The first delay contributor is the analog bandpass prefilter, whose phase response was shown in Fig. 3. For a 1 MHz tone, the phase delay is approximately 0.58 radians. The second delay contributor is the ranging I/D filter whose phase delay is

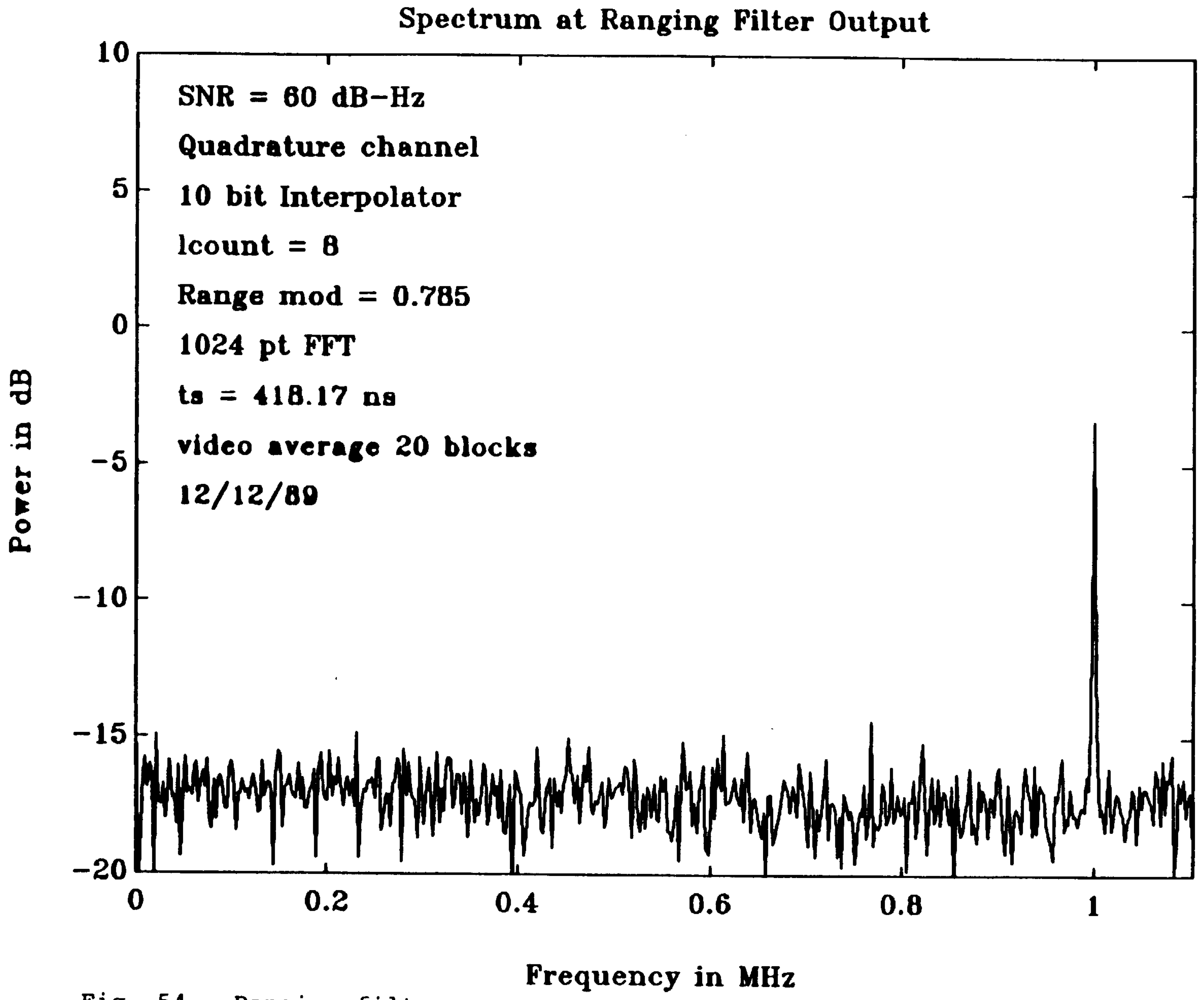


Fig. 54. Ranging filter spectrum, f = 1 MHz, mod = 0.785, signal-to-noise density = 60 dB-Hz.

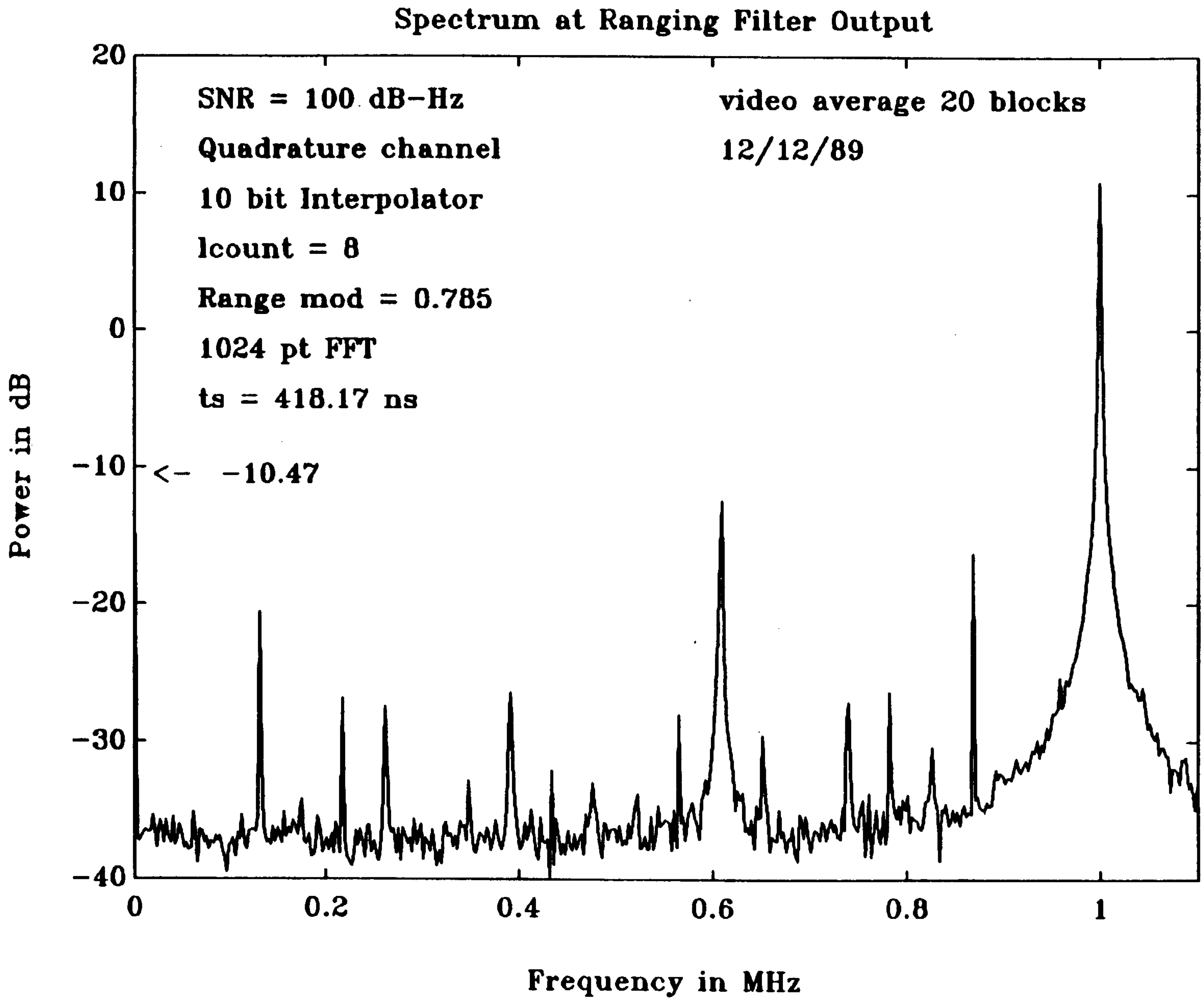


Fig. 55. Ranging filter spectrum, f = 1 MHz, mod = 0.785, signal-to-noise density = 100 dB-Hz.

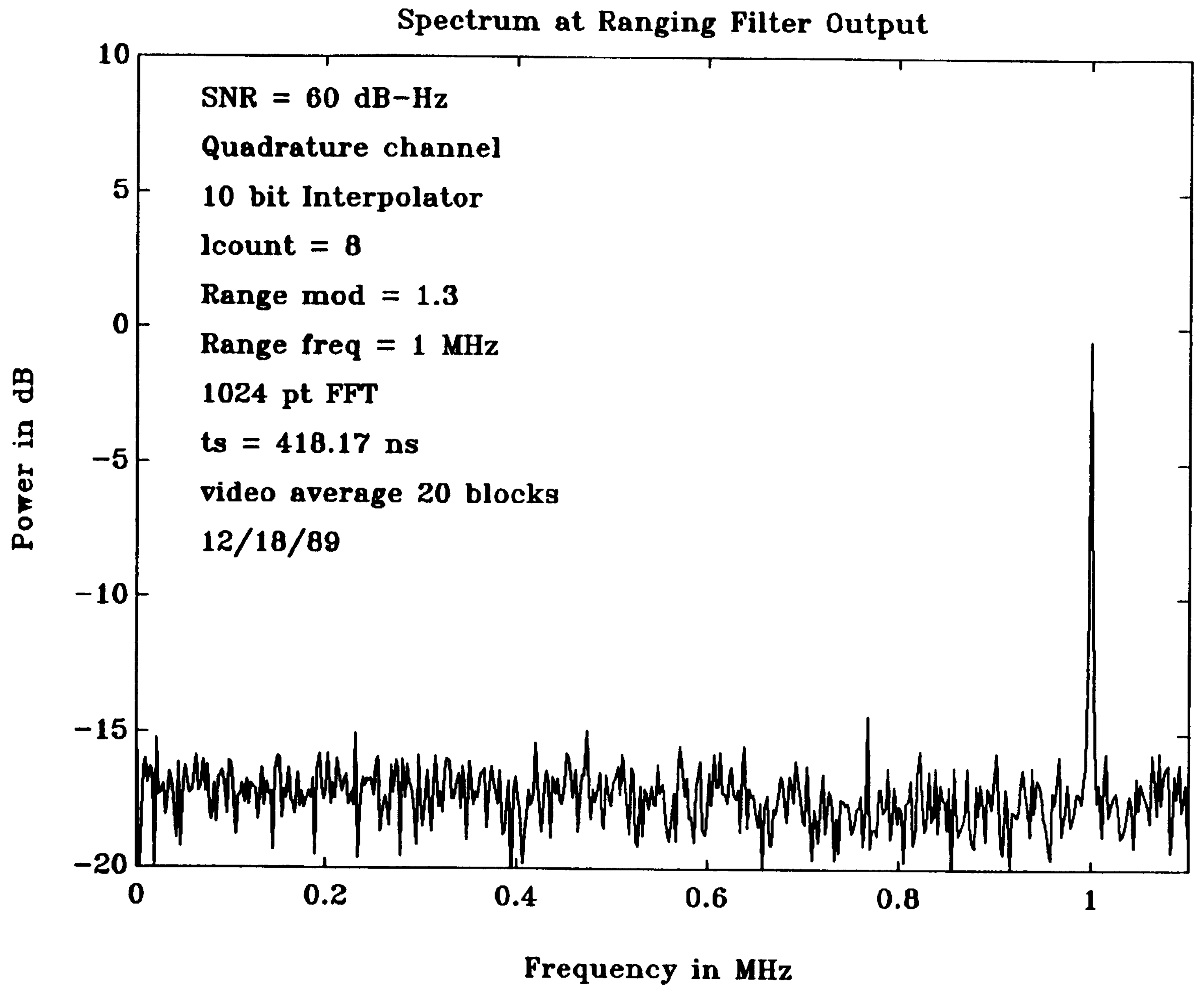


Fig. 56. Ranging filter spectrum, f = 1 MHz, mod = 1.3, signal-to-noise density = 60 dB-Hz.

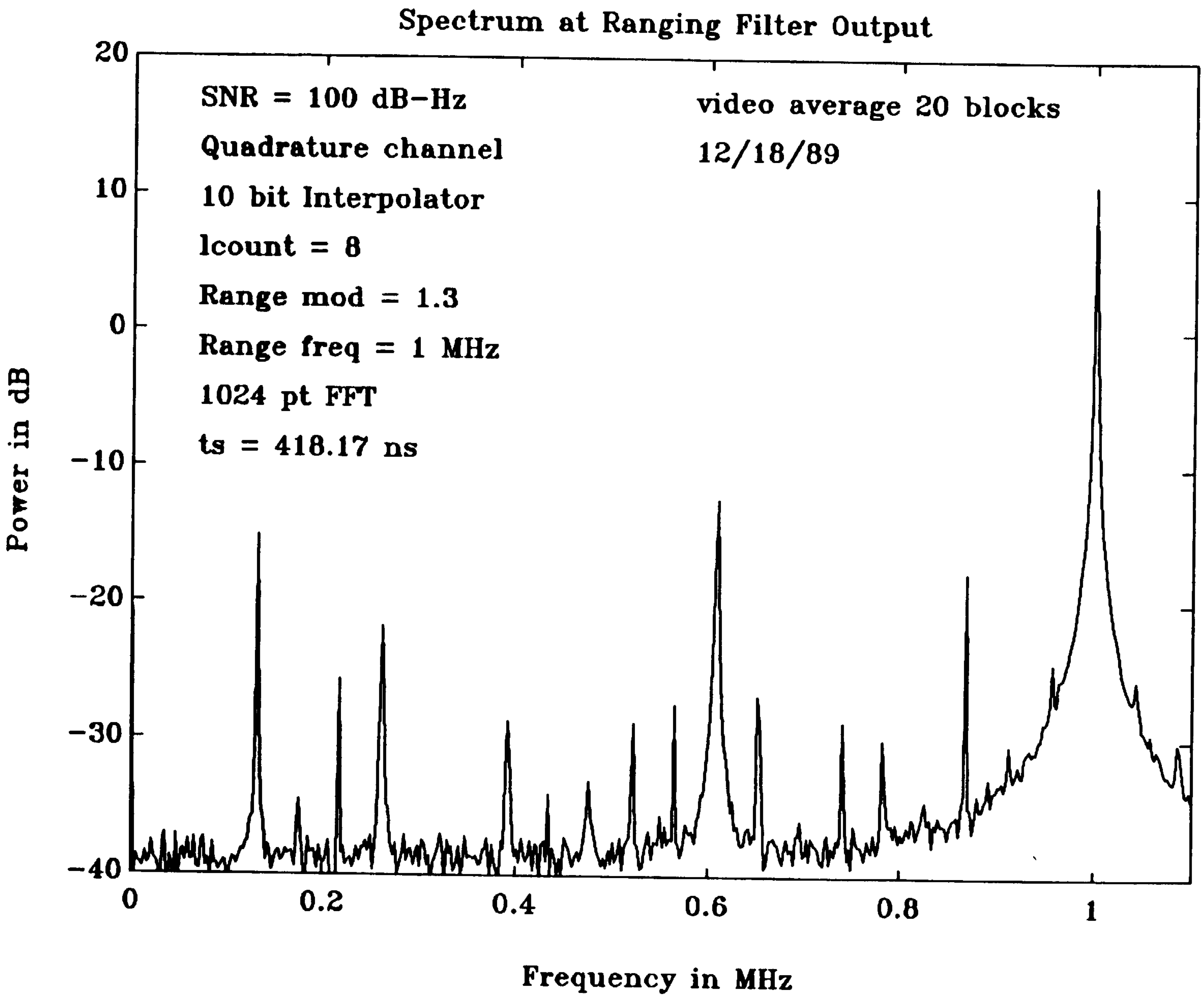


Fig. 57. Ranging filter spectrum, f - 1 MHz, mod = 1.3, signal-to-noise density = 100 dB-Hz.

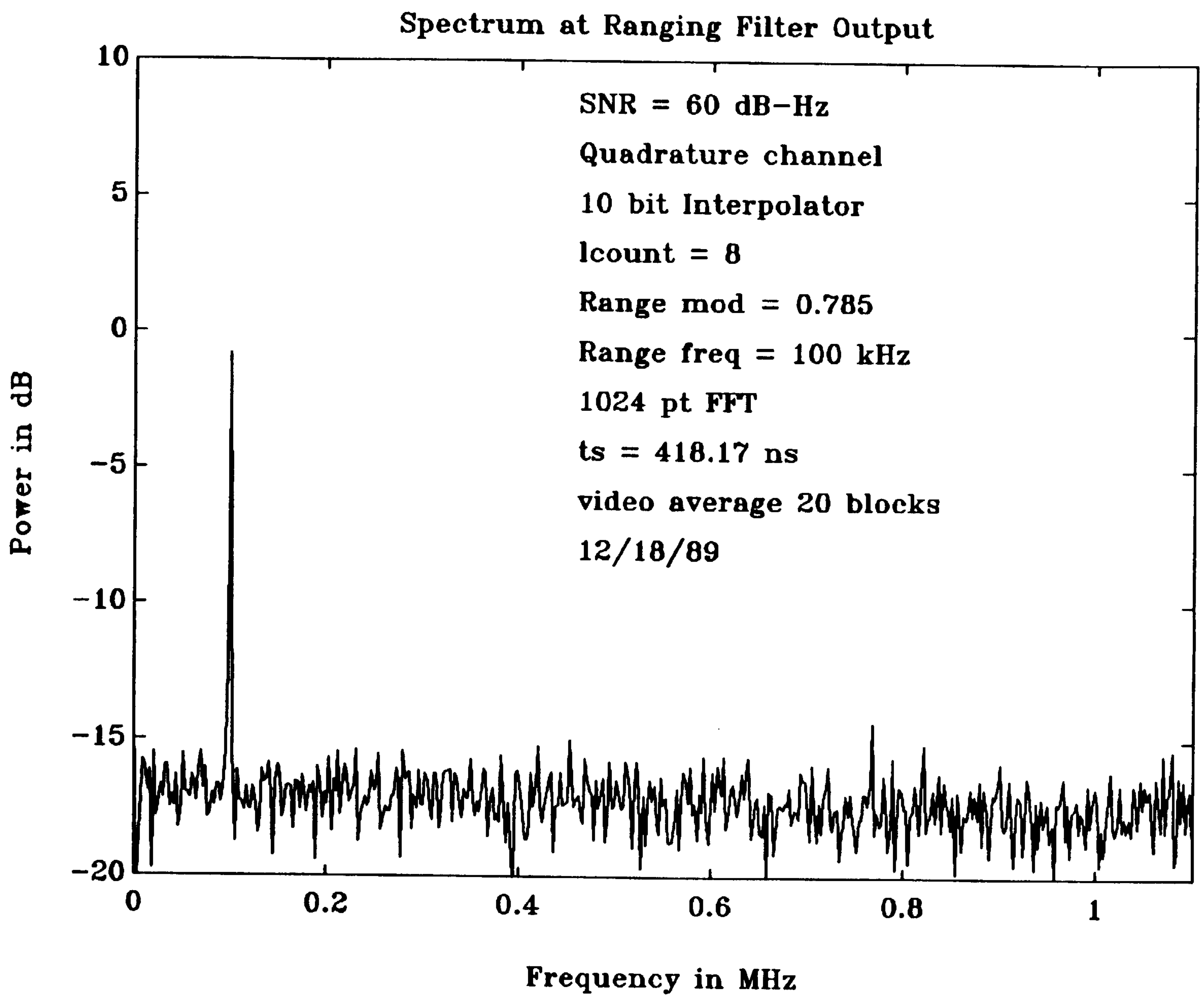


Fig. 58. Ranging filter spectrum, f = 100 kHz, mod = 0.785, signal-to-noise density = 60 dB-Hz.

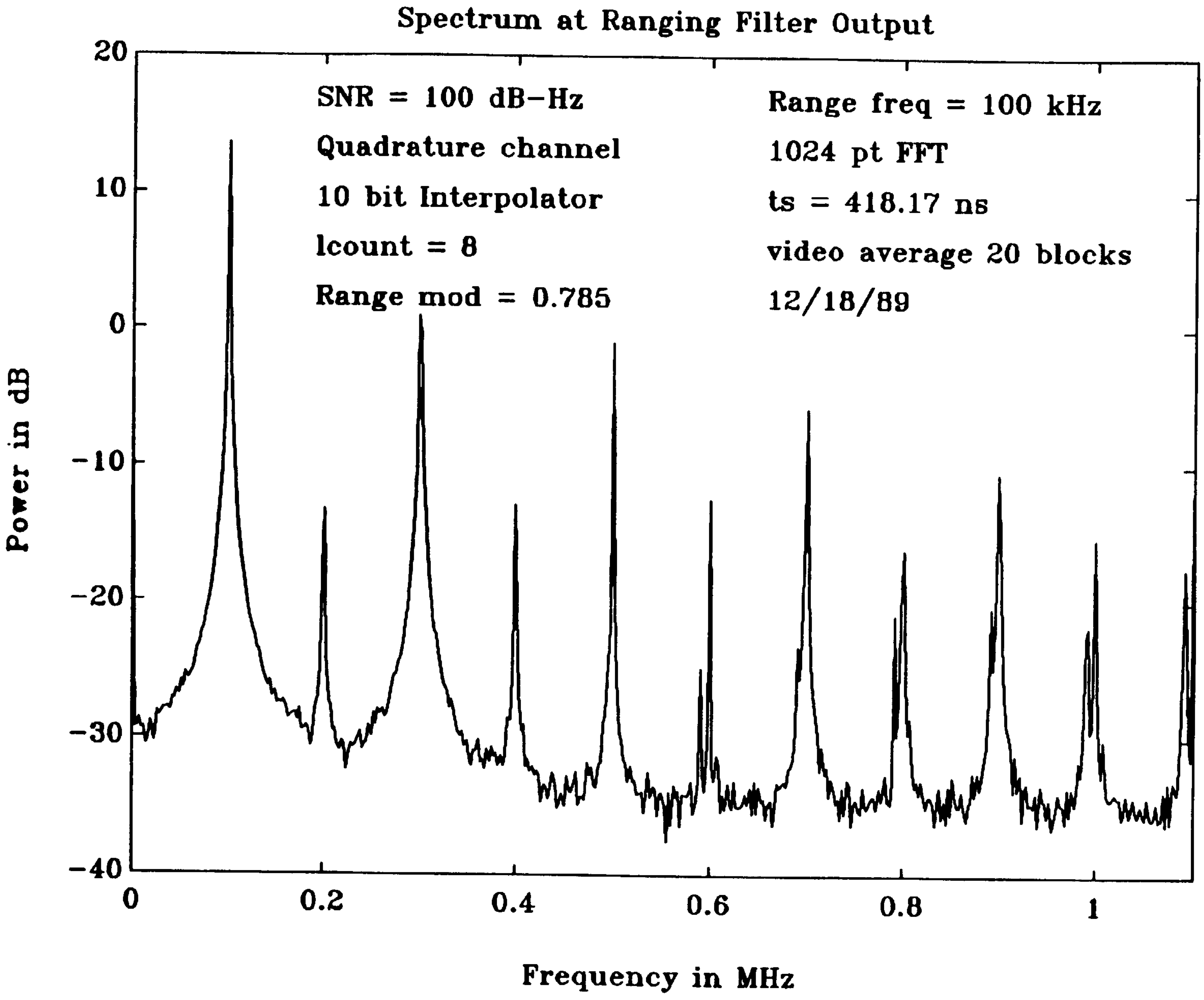


Fig. 59. Ranging filter spectrum, f = 100 kHz, mod = 0.785, signal-to-noise density = 100 dB-Hz.

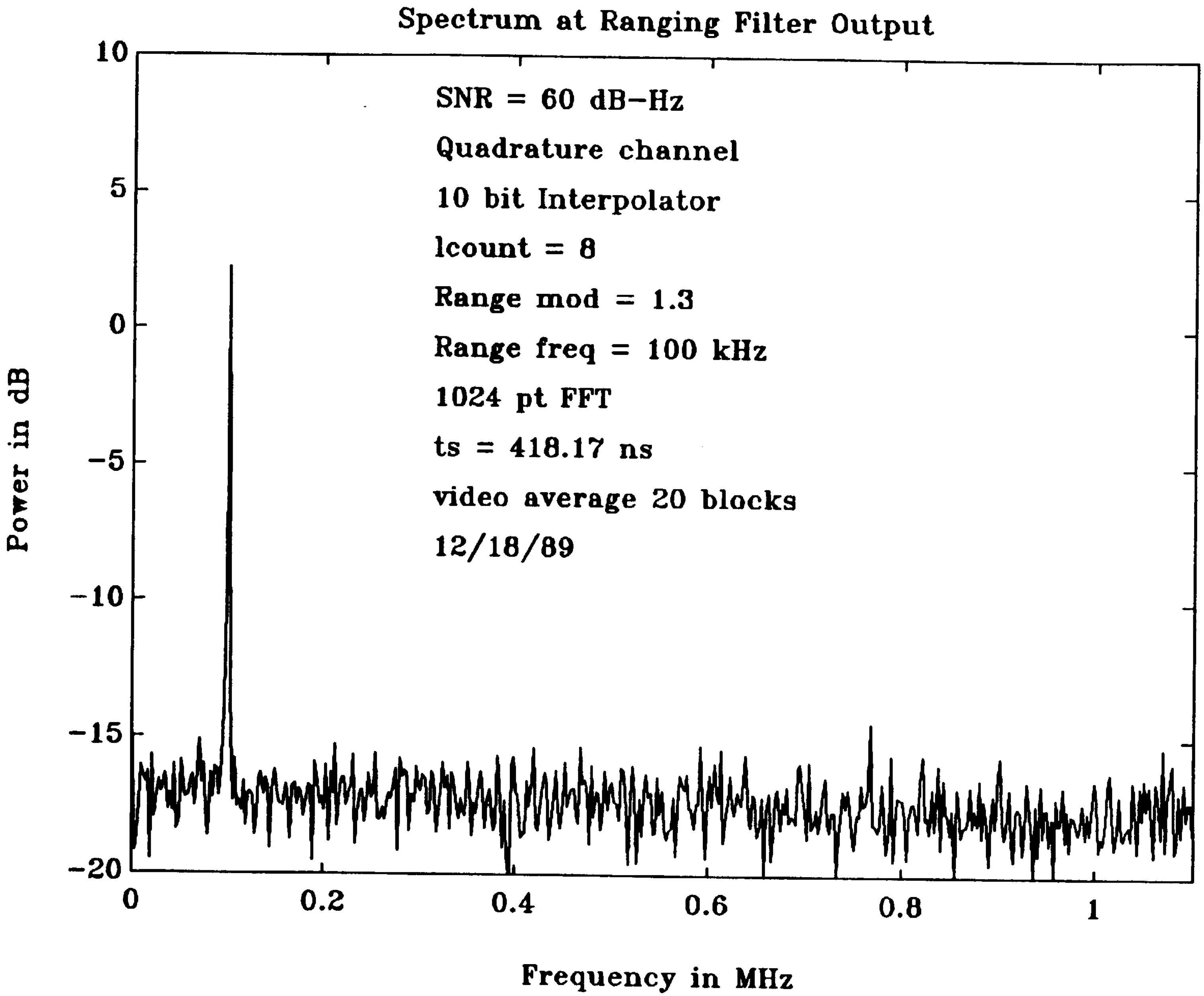


Fig. 60. Ranging filter spectrum, f = 100 kHz, mod = 1.3, signal-to-noise density = 60 dB-Hz.

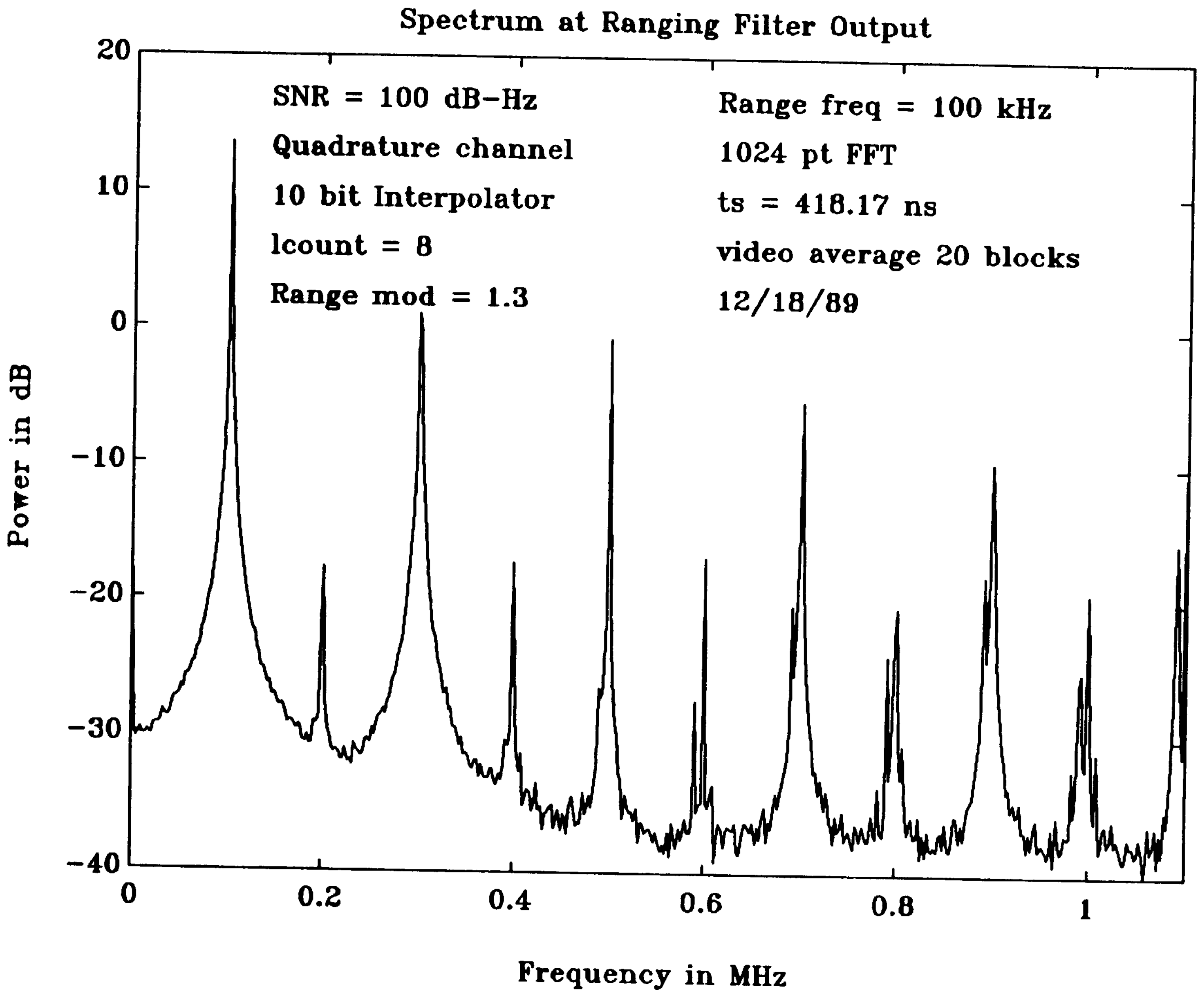


Fig. 61. Ranging filter spectrum, f = 100 kHz, mod = 1.3, signal-to-noise density = 100 dB-Hz.

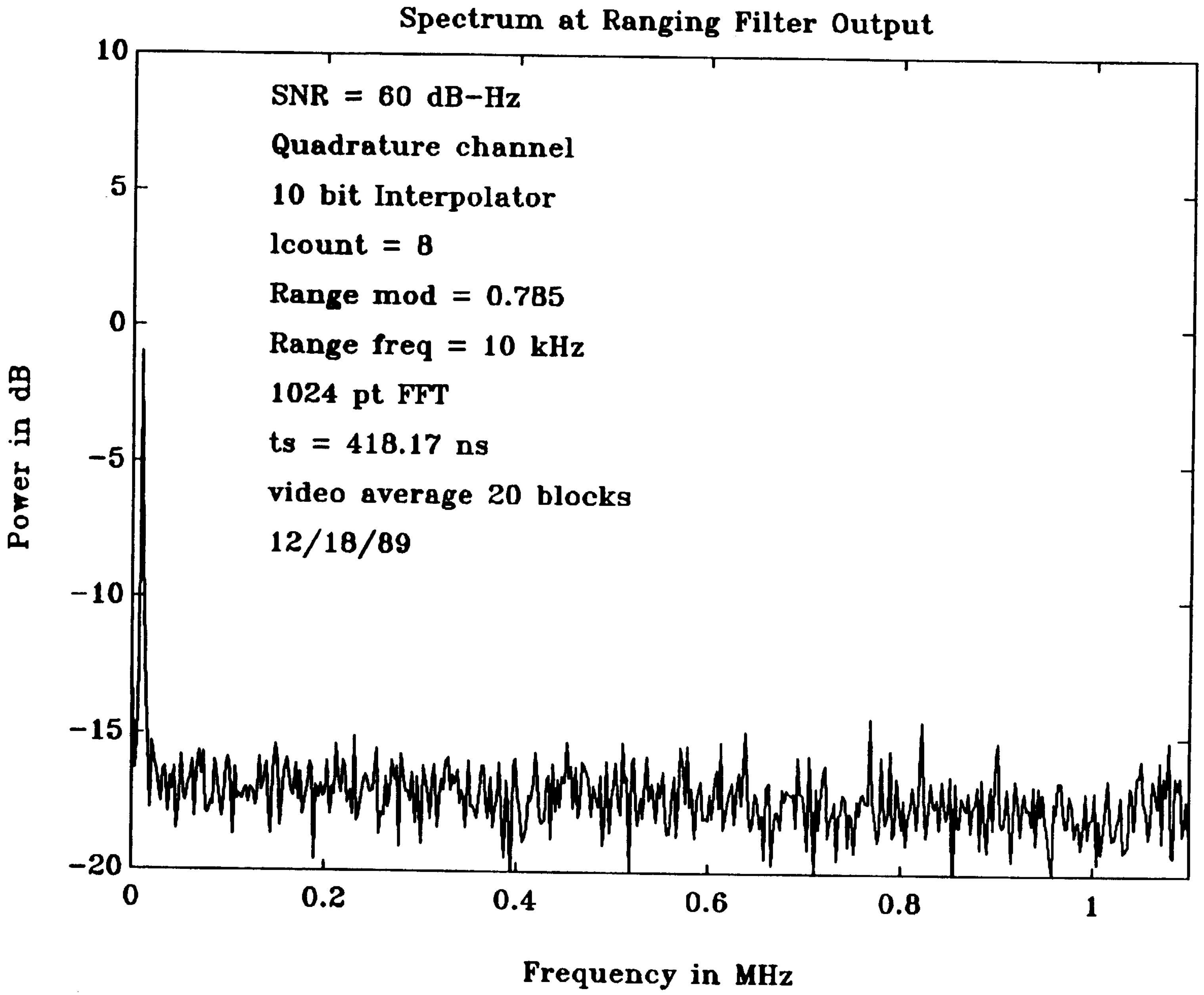


Fig. 62. Ranging filter spectrum, f = 10 kHz, mod = 0.785, signal-to-noise density = 60 dB-Hz.

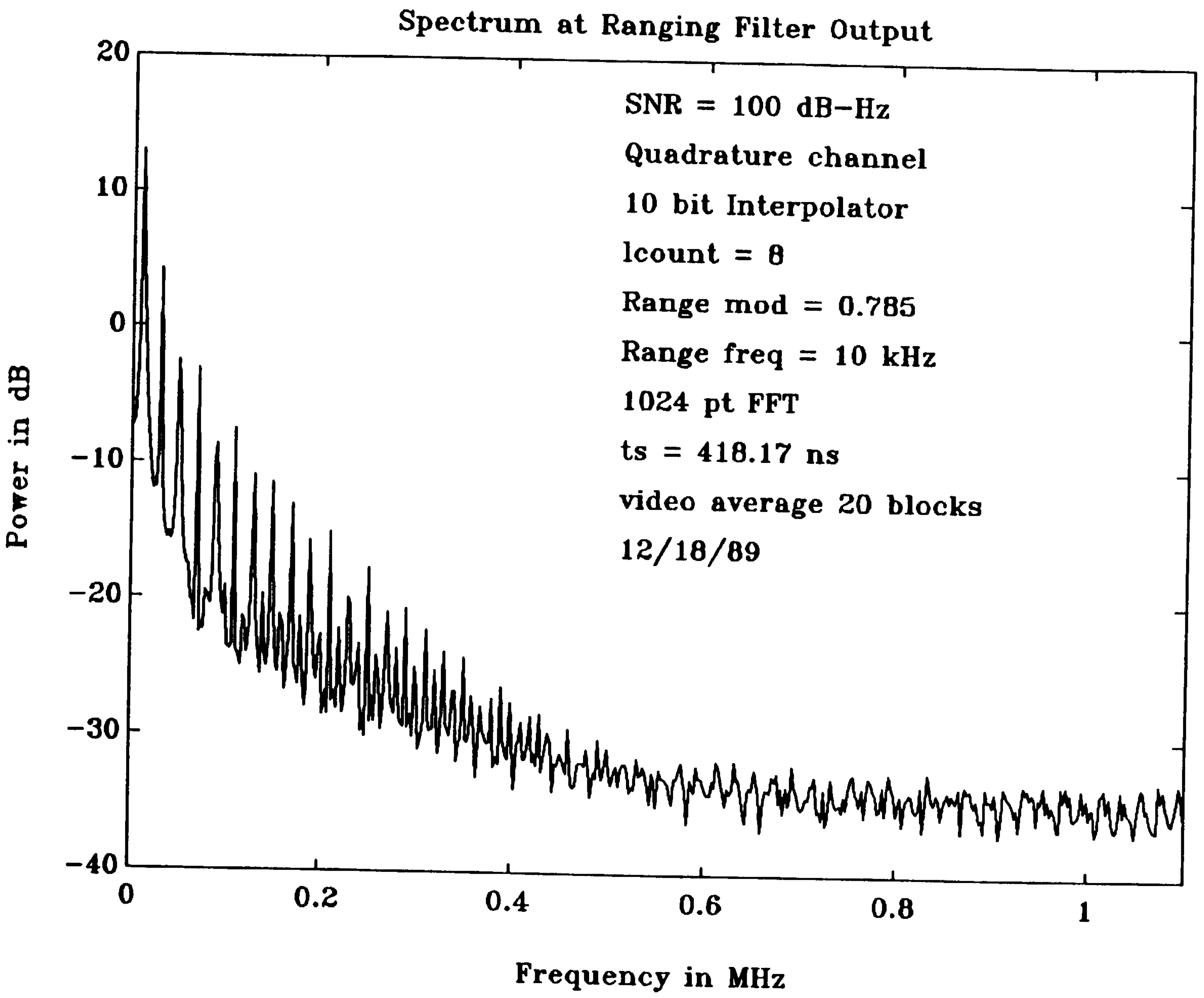


Fig. 63. Ranging filter spectrum, f = 10 kHz, mod = 0.785, signal-to-noise density = 100 dB-Hz.

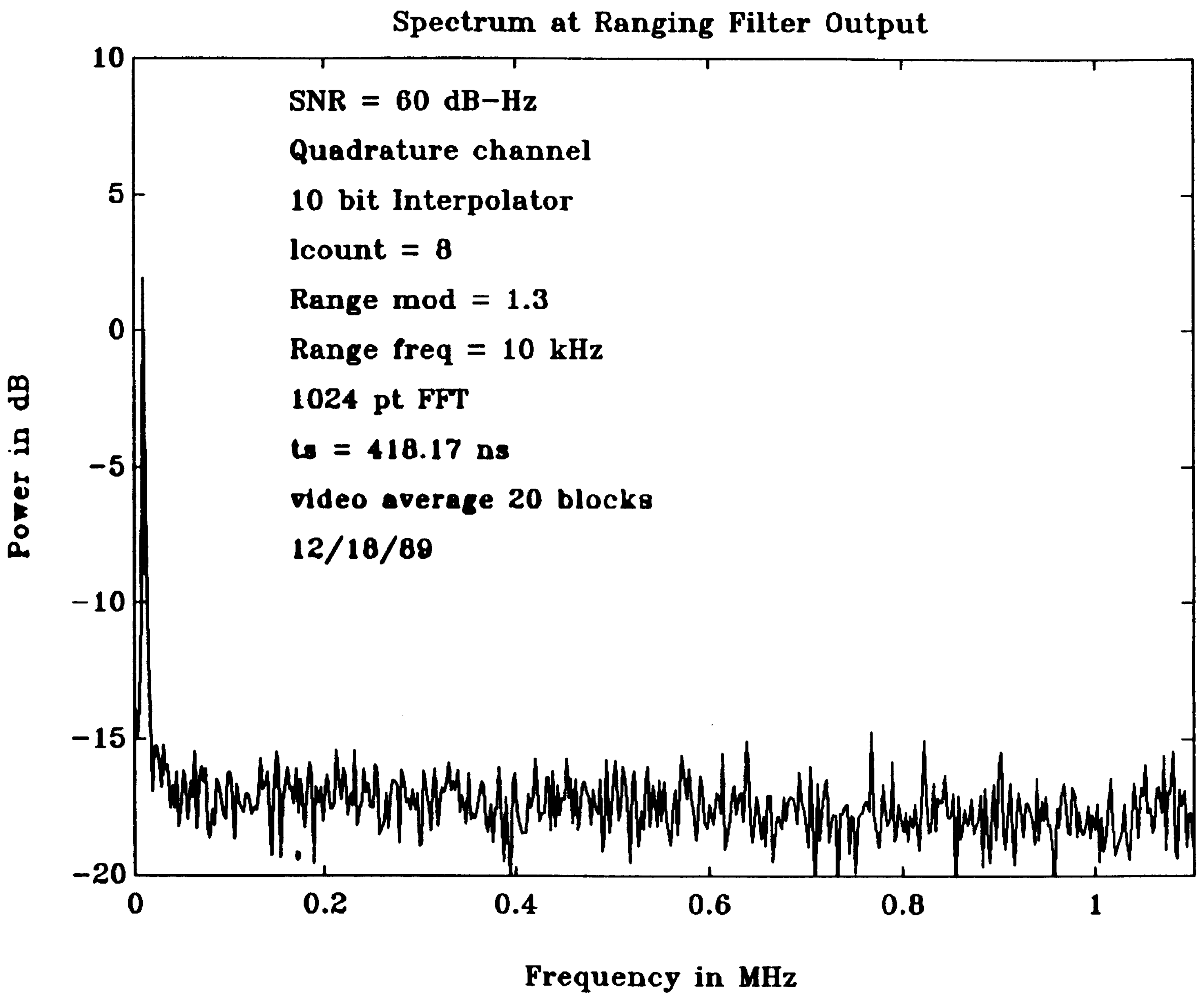


Fig. 64. Ranging filter spectrum, f = 10 kHz, mod = 1.3, signal-to-noise density = 60 dB-Hz.

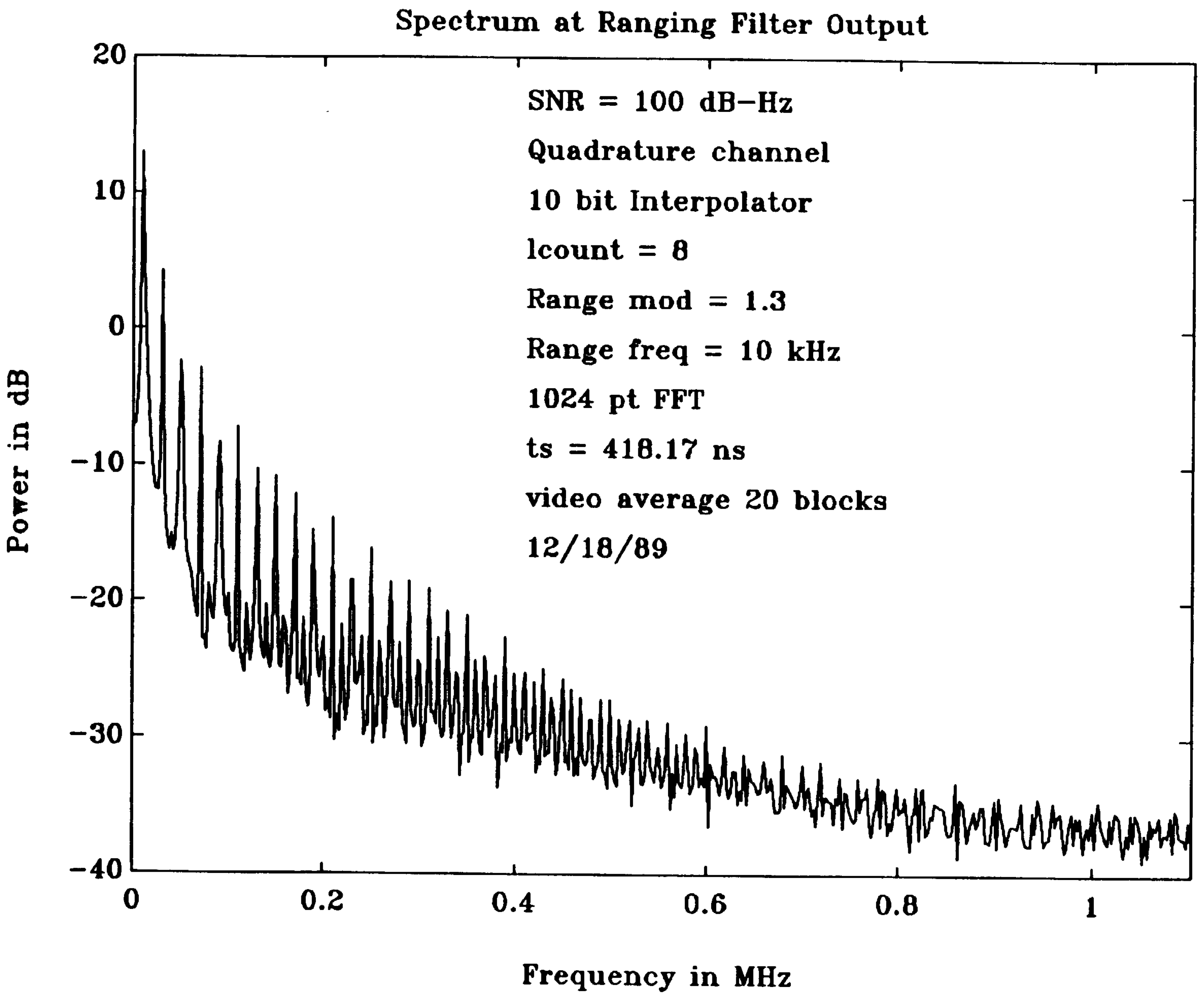


Fig. 65. Ranging filter spectrum, f = 10 kHz, mod = 1.3, signal-to noise density = 100 dB-Hz.

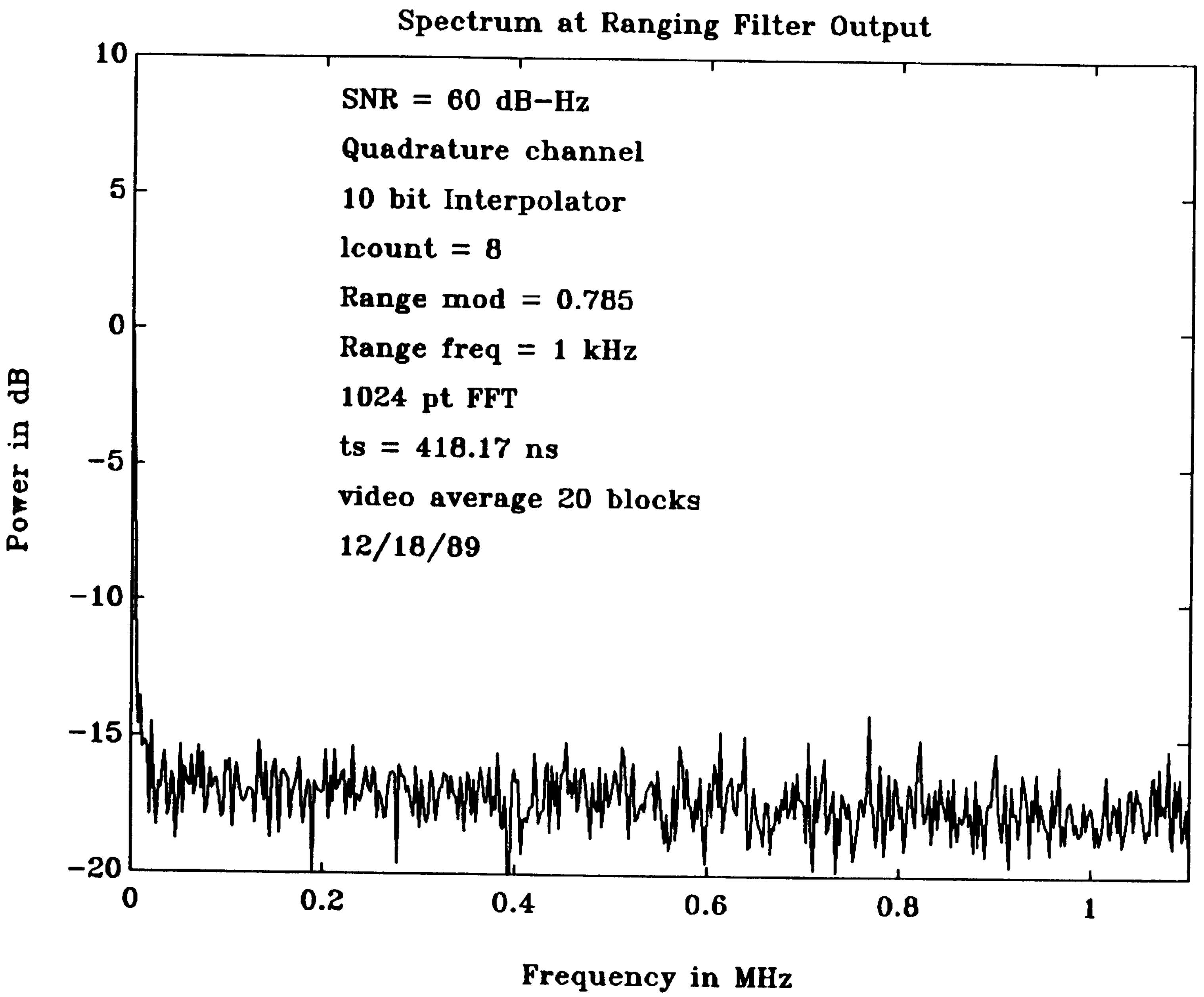


Fig. 66. Ranging filter spectrum, f = 1 kHz, mod = 0.785, signal-to-noise density = 60 dB-Hz.

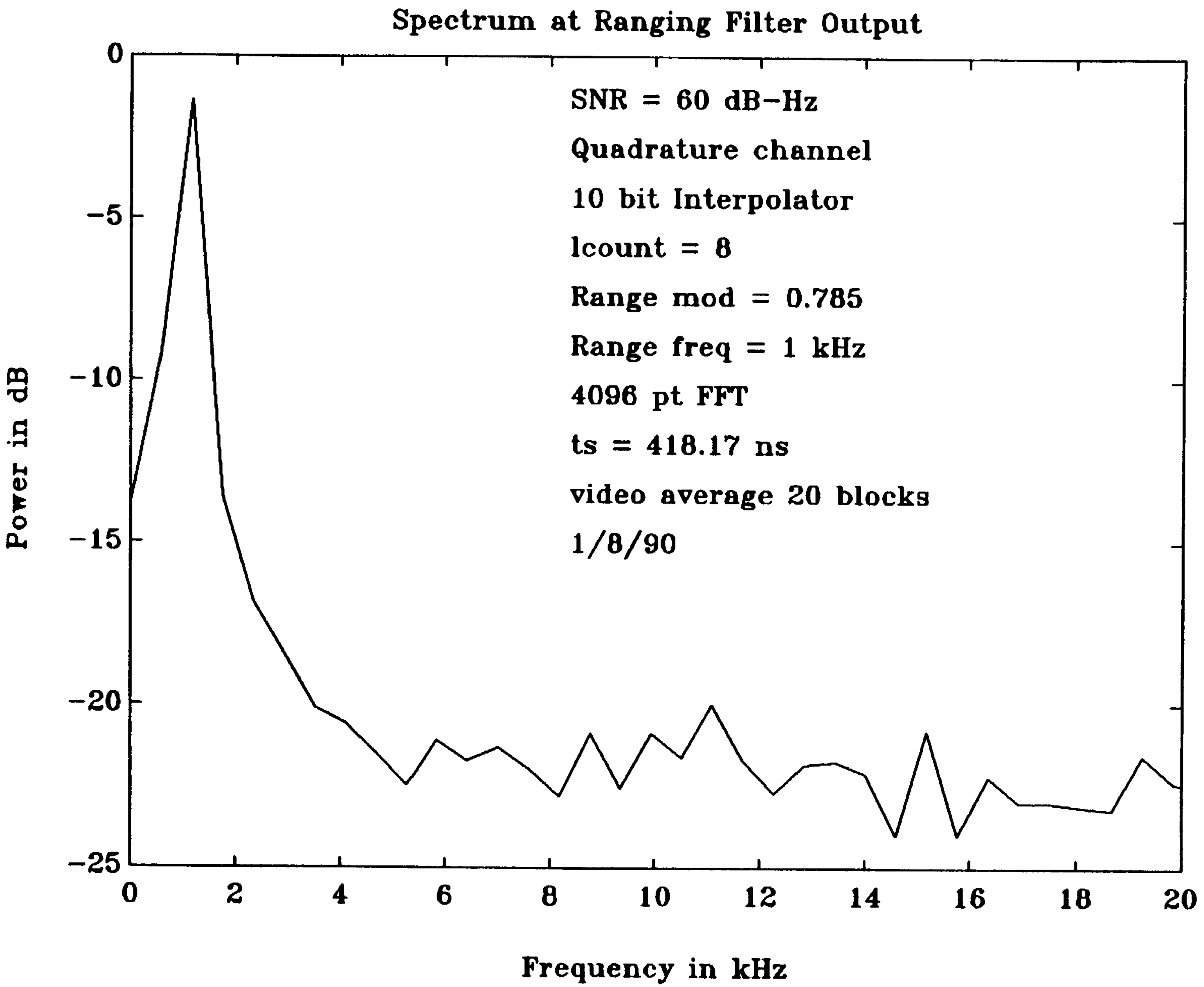


Fig. 67. Ranging filter spectrum, f = 1 kHz, mod = 0.785, signal-to-noise density = 60 dB-Hz, expanded scale.

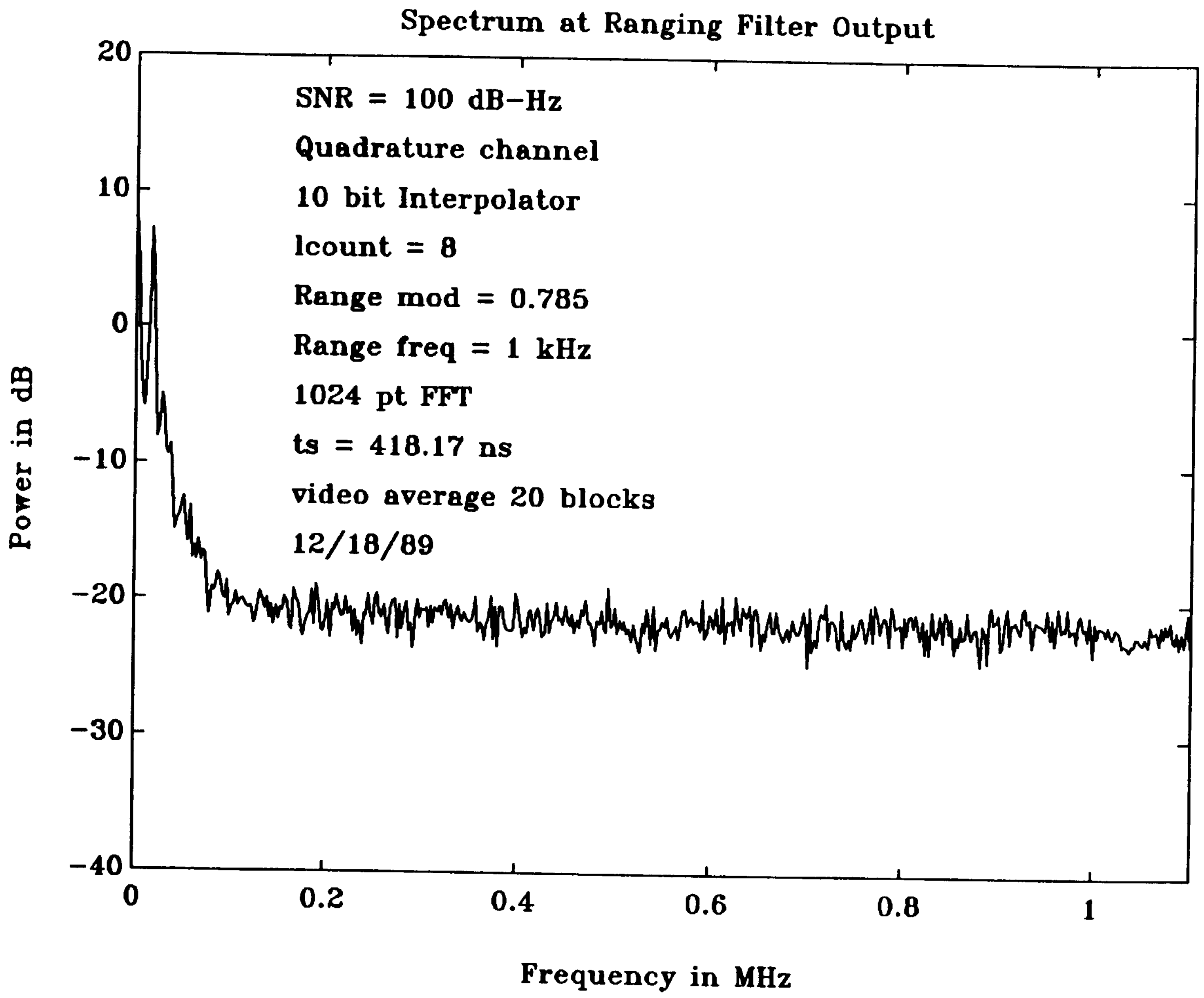


Fig. 68. Ranging filter spectrum, f = 1 kHz, mod = 0.785, signal-to-noise density = 100 dB-Hz.

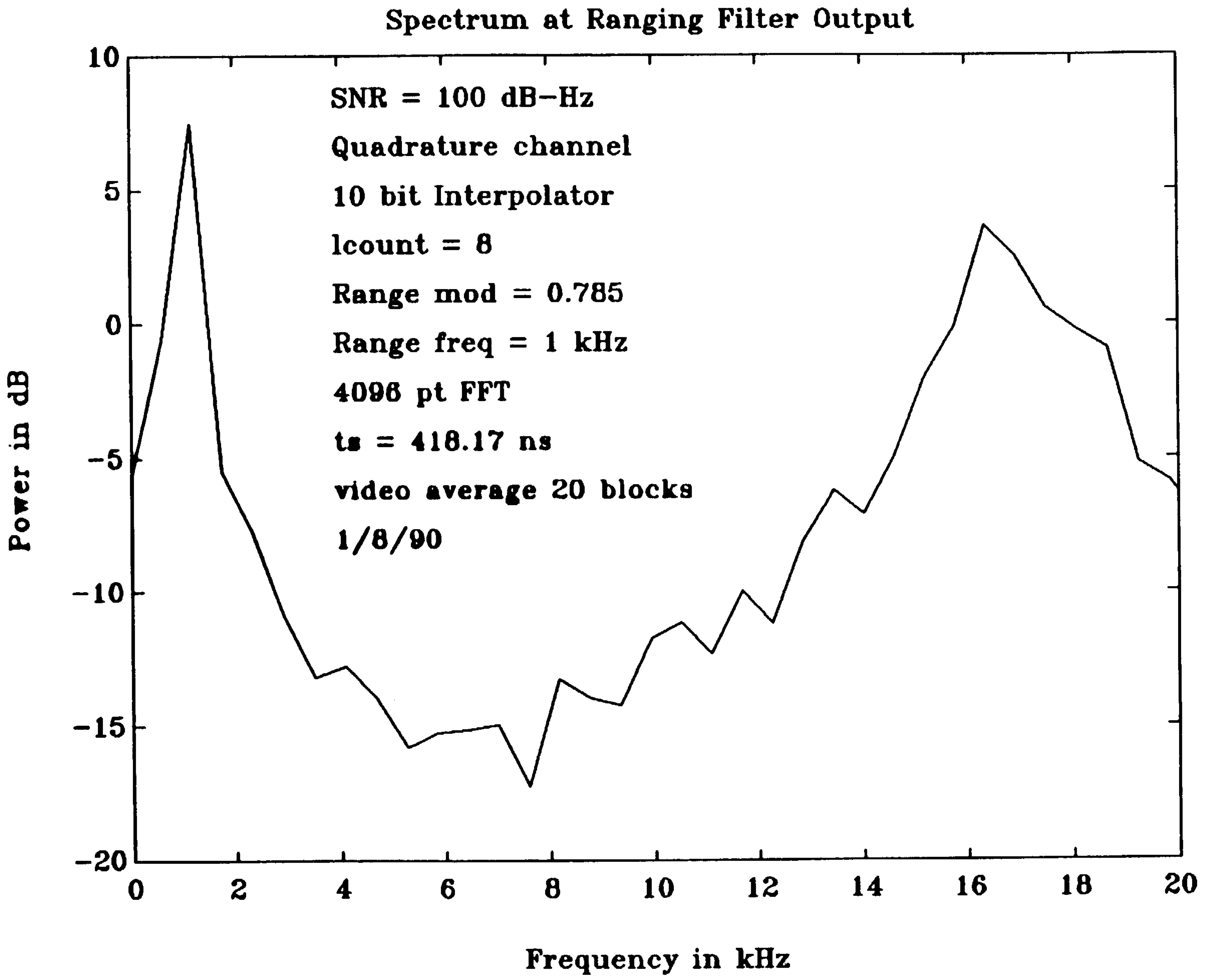


Fig. 69. Ranging filter spectrum, f = 1 kHz, mod = 0.785, signal-to-signal density = 100 dB-Hz, expanded scale.

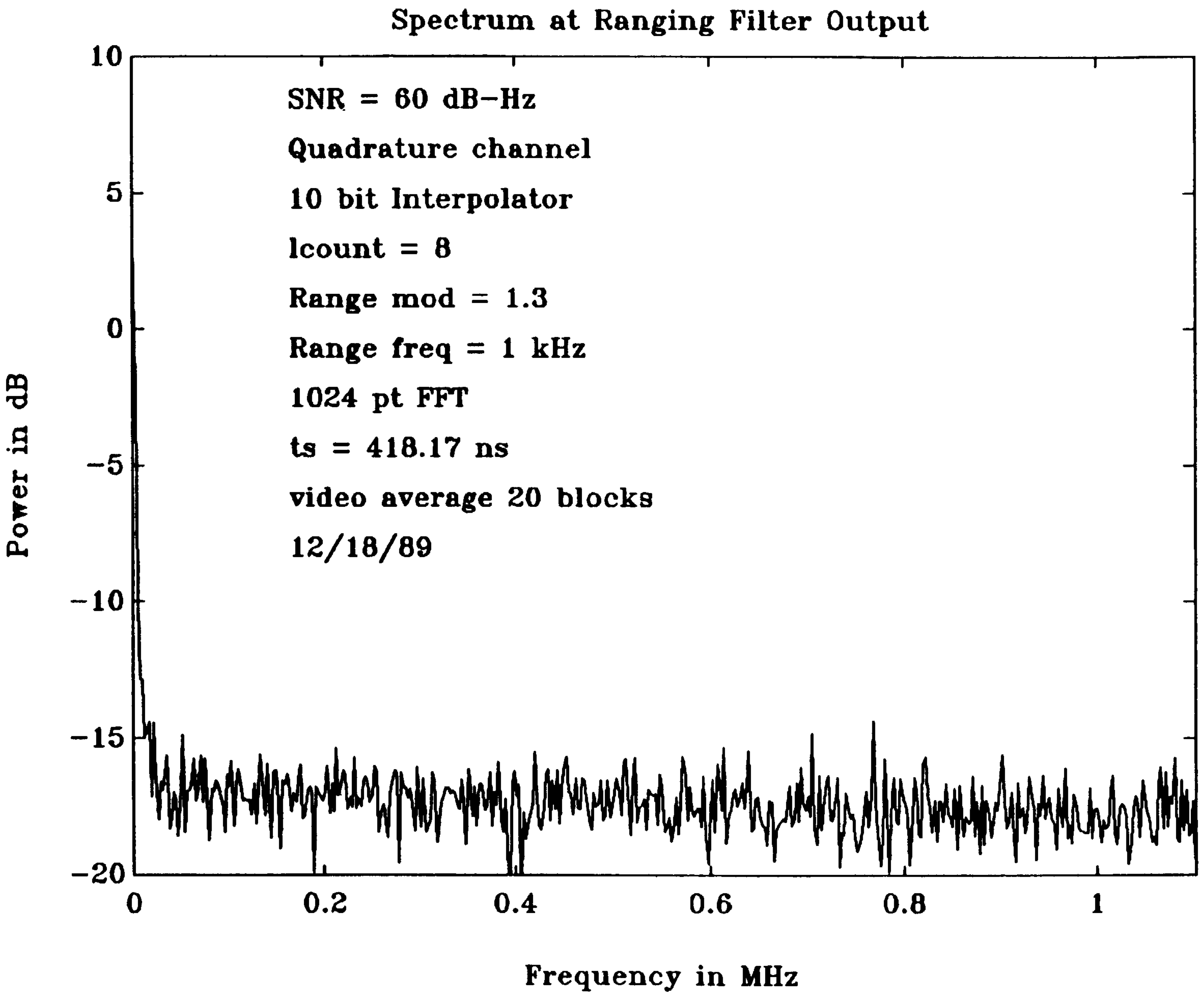


Fig. 70. Ranging filter spectrum, f = 1 kHz, mod = 1.3, signal-to-noise density = 60 dB-Hz.

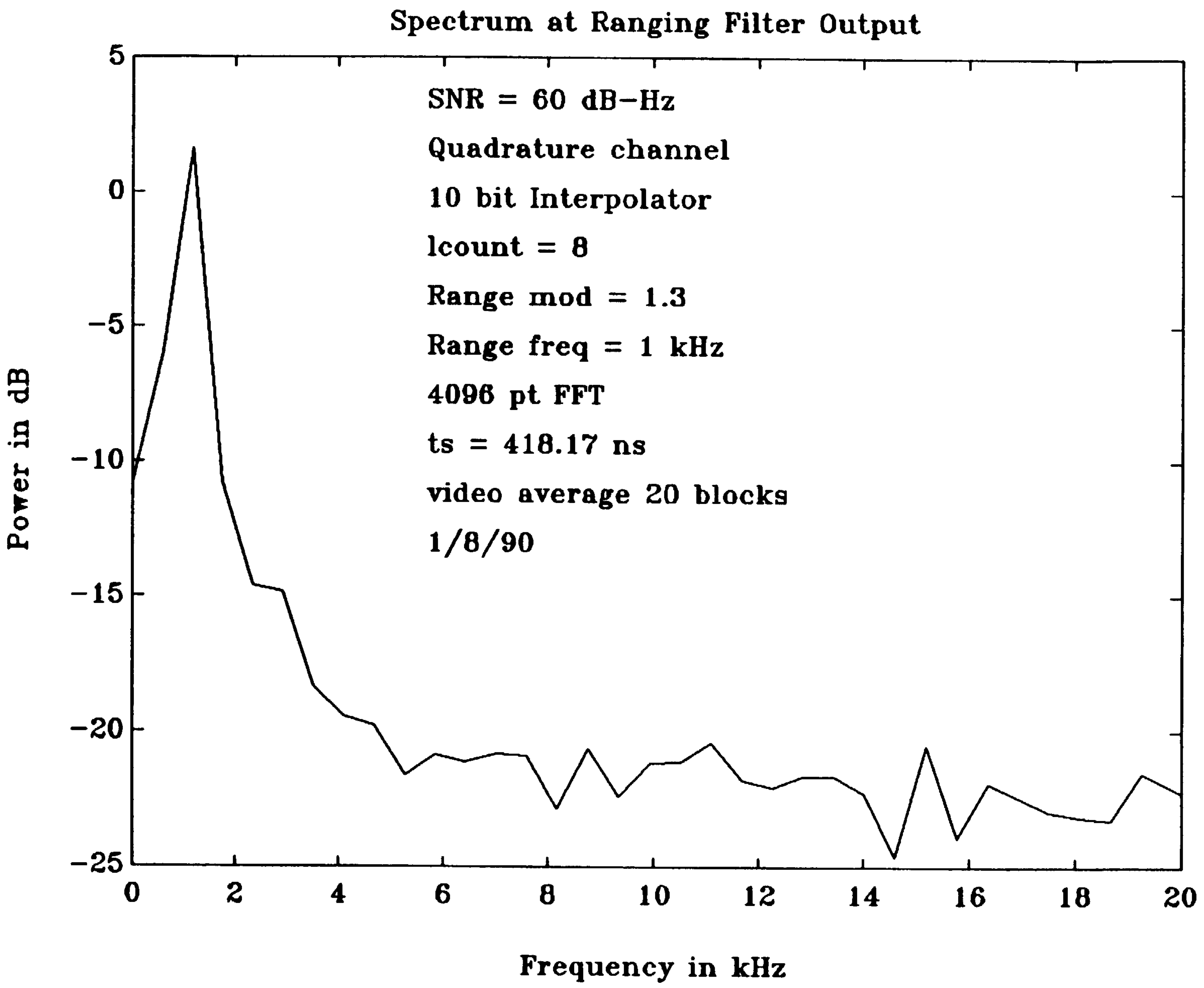


Fig. 71. Ranging filter spectrum, f = 1 kHz, mod = 1.3, signal-to-noise density = 60 dB-Hz, expanded scale.

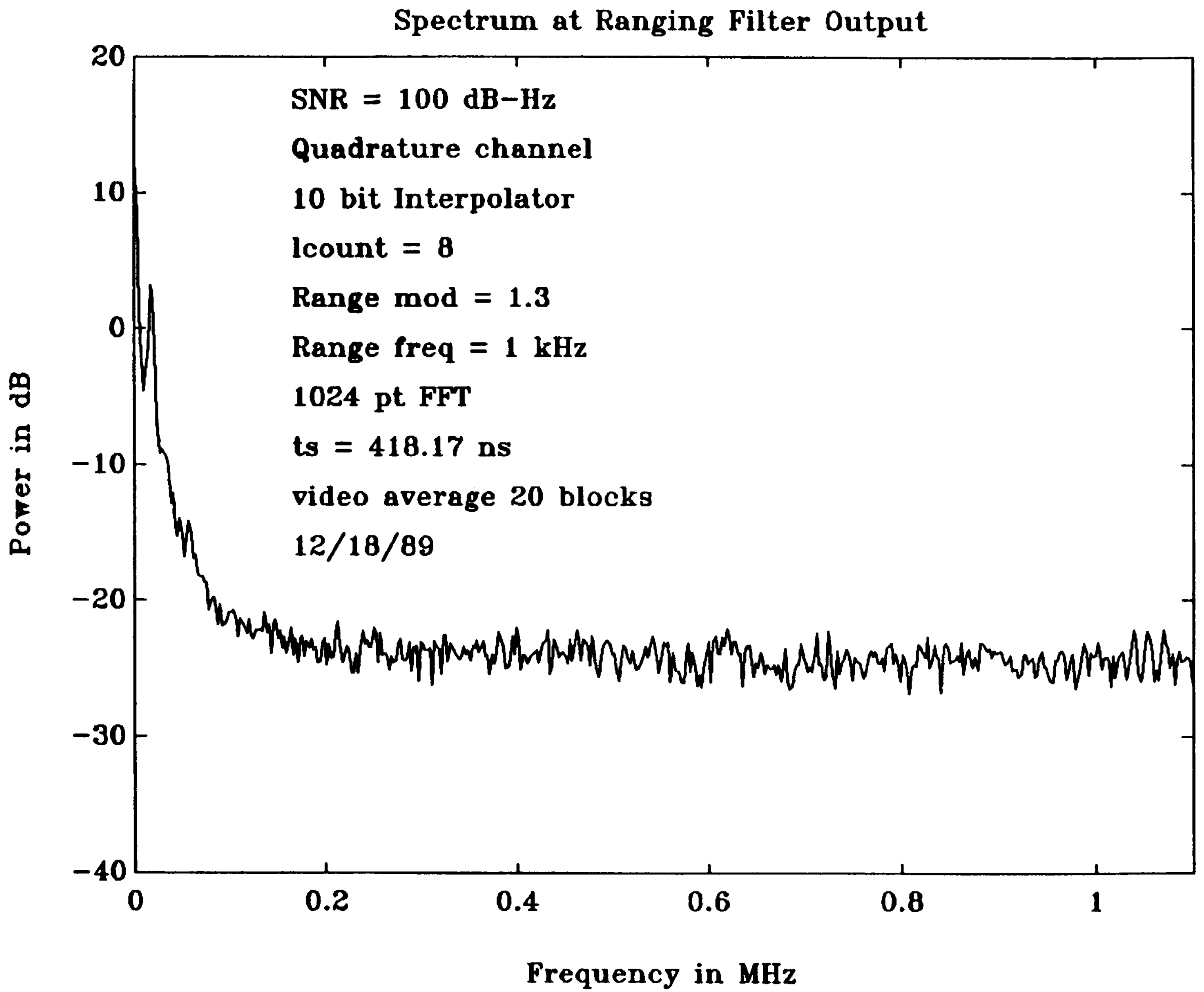


Fig. 72. Ranging filter spectrum, f = 1 kHz, mod = 1.3, signal-to-noise density = 100 dB-Hz.

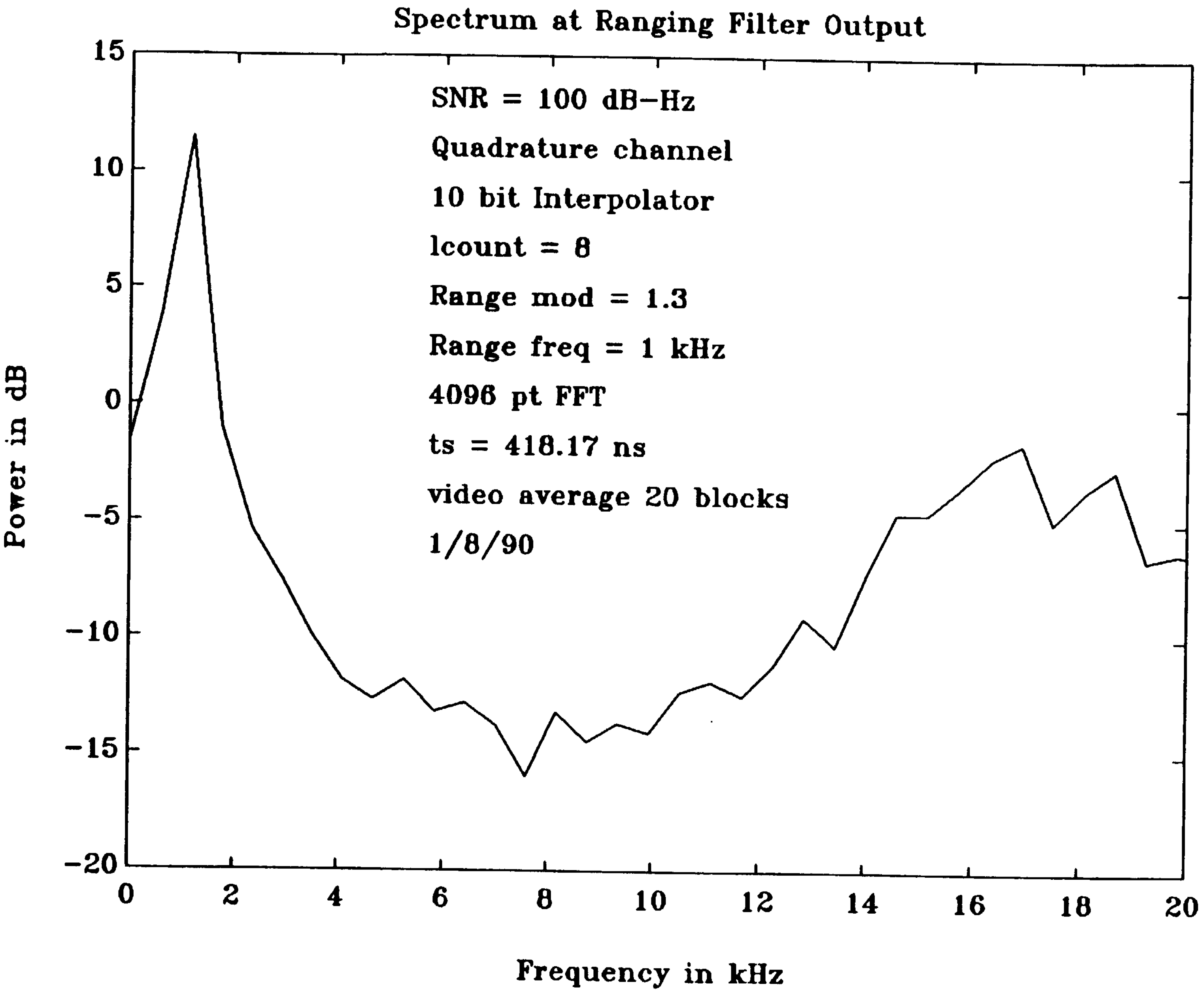


Fig. 73. Ranging filter spectrum, f = 1 kHz, mod = 1.3, signal-to-noise density = 100 dB-Hz, expanded scale.

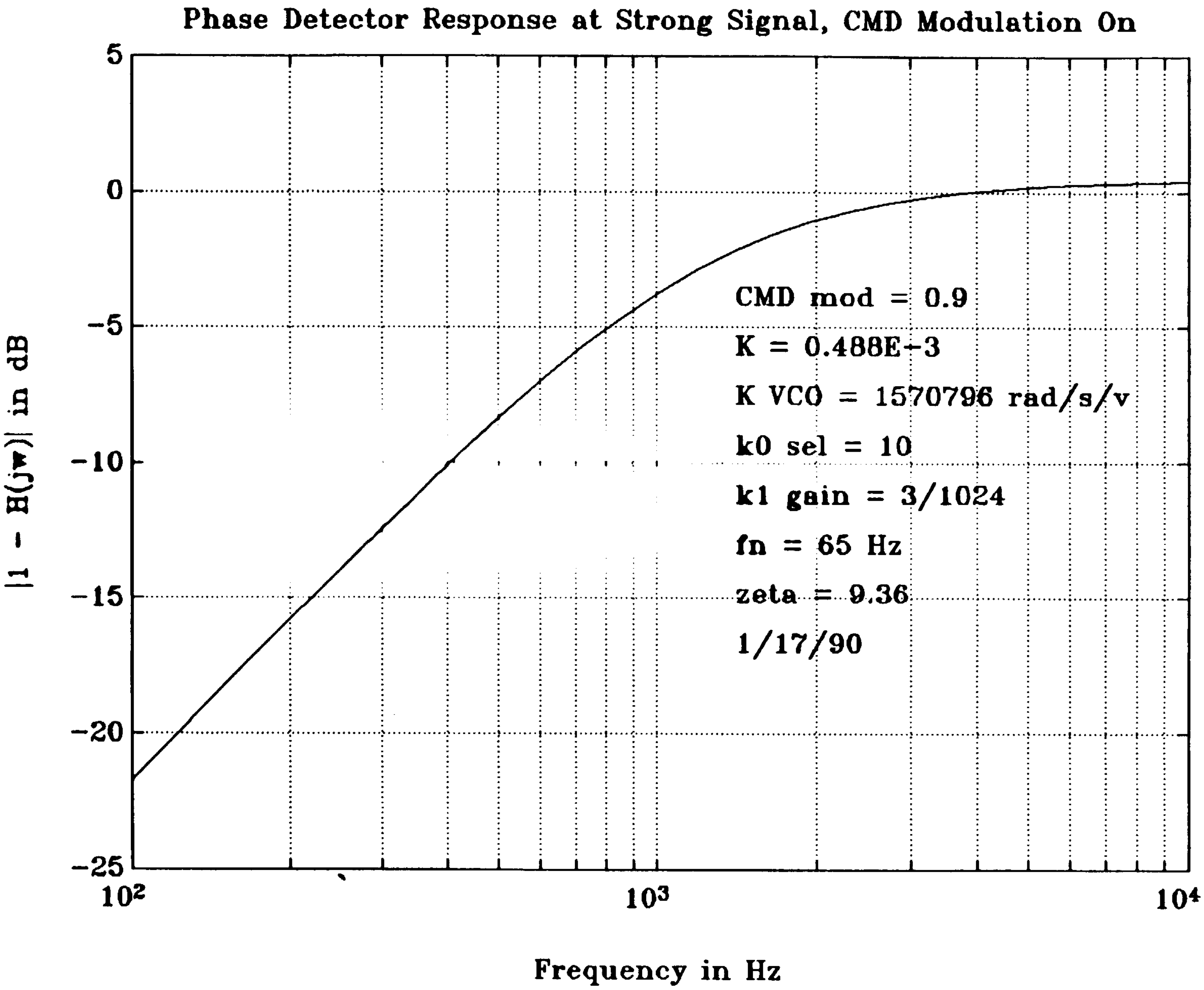


Fig. 74. Phase detector response at strong signal, obtained from linearized model.

$$\theta = \frac{\mathrm{lcount} - 1}{4\mathrm{Fl}} \omega_r \qquad (39)$$

which, for lcount = 8, is equal to 1.15 radians. The magnitude and phase responses of the ranging I/D filter are shown in Figs. 75 and 76. The I/D delay will not be affected by temperature or aging of parts. It is, however, tied to the carrier frequency and will therefore be subject to doppler shifts. These shifts will be small with respect to 4Fl (the sampling time), and will also be predictable. For a 1 MHz ranging tone, the phase delay through the entire circuit is approximately 1.73 radians (or 275 ns).

One open issue in using the 1 bit processing scheme is whether the ranging delay changes as a function of signal level when the SNR is positive. In order to estimate the output ranging phase, the maximum likelihood estimator of (34) was used on the software breadboard. Fig. 77 shows the result of this estimation for signal-to-noise densities of 60, 70, 80, 90, and 100 dB-Hz. Also shown in this figure are 1 sigma bounds on the estimates obtained from (35). As can be seen from the figure, the ranging delay does not appear to vary as a function of signal level.

IV. Conclusions and Recommendations

In this report the testing of the software breadboard implementation of a 1 bit digital receiver was presented. The results of the Monte Carlo simulation were compared against equivalent models in order to verify the accuracy of the simulation. The simulation outputs compared quite well with the outputs of the models. In addition, the simulation results showed good agreement with the previous study, [ATS89].

The major goal of this study was to use the software simulation to predict the performance of a hardware receiver at strong signal levels. It is known that at positive SNR, the 1 bit A/D no longer acts like a linear device. The question that needed to be answered was whether or not this nonlinearity degraded the performance of the receiver beyond acceptable levels. To this end, the performance of the phase-locked loop was tested at strong signal levels. The loop was seen to track phase steps, as well as acquire frequency offsets. The loop responses compared favorably to the responses of linear models. It was also shown that the increased phase detector gain at strong signal level did not cause a stability problem.

With no modulation and positive carrier-to-noise ratio, the phase-locked loop appeared to possess a limit cycle. A limit cycle is a phenomenon which occurs in a control loop containing a nonlinearity (i.e., the phase detector). Whether this limit cycle is a cause for concern would have to be determined by JPL. Some methods for reducing the effects of the limit cycle include: 1) changing the loop filter gain constants at high SNR to reduce the loop gain, 2) changing the loop sampling rate in order to change the frequency of the limit cycle, and 3) using a coherent AGC to reduce the loop gain at strong signal.

The 1 bit nonlinearity was seen to cause intermodulation of command

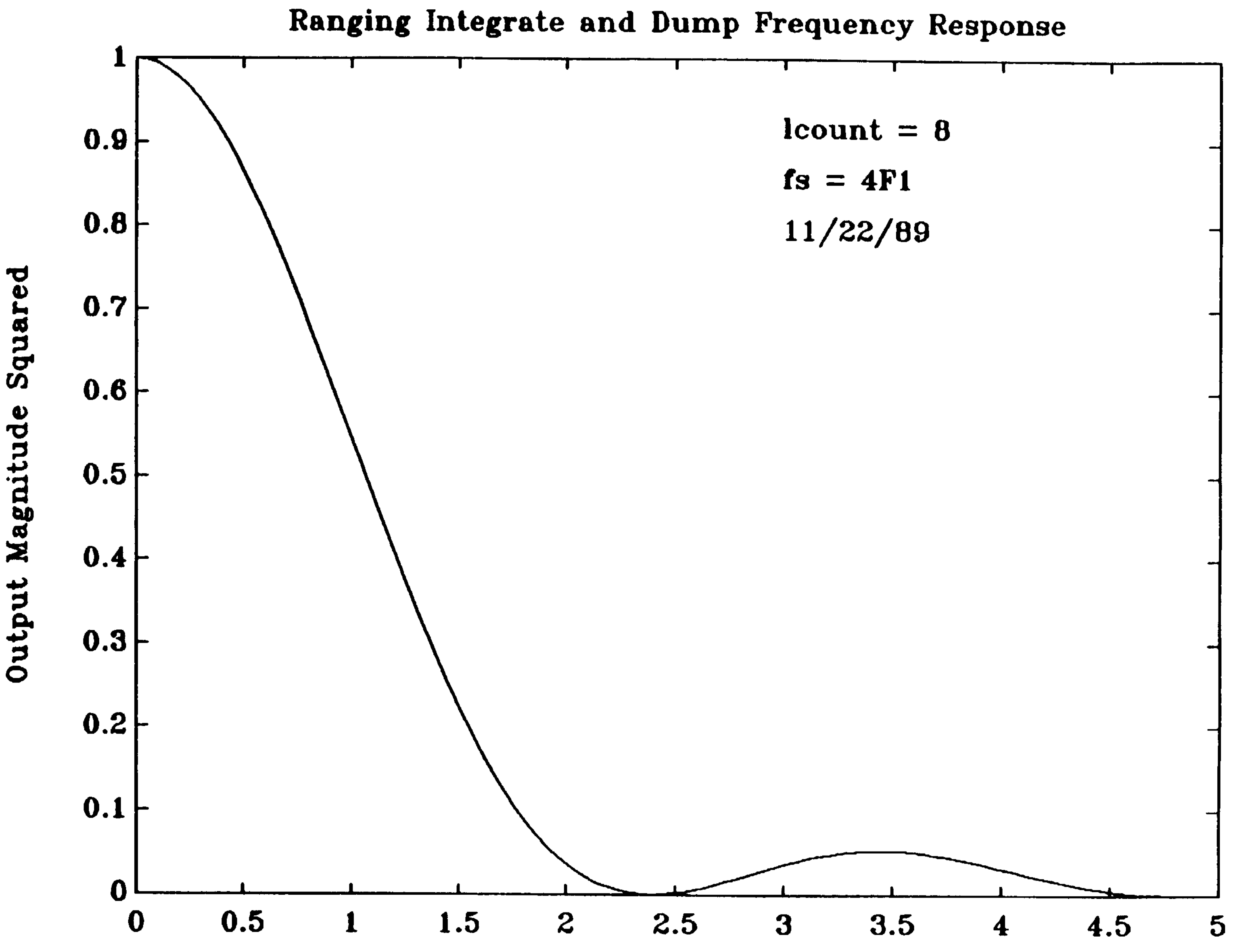


Fig. 75. Magnitude response of ranging integrate and dump filter.

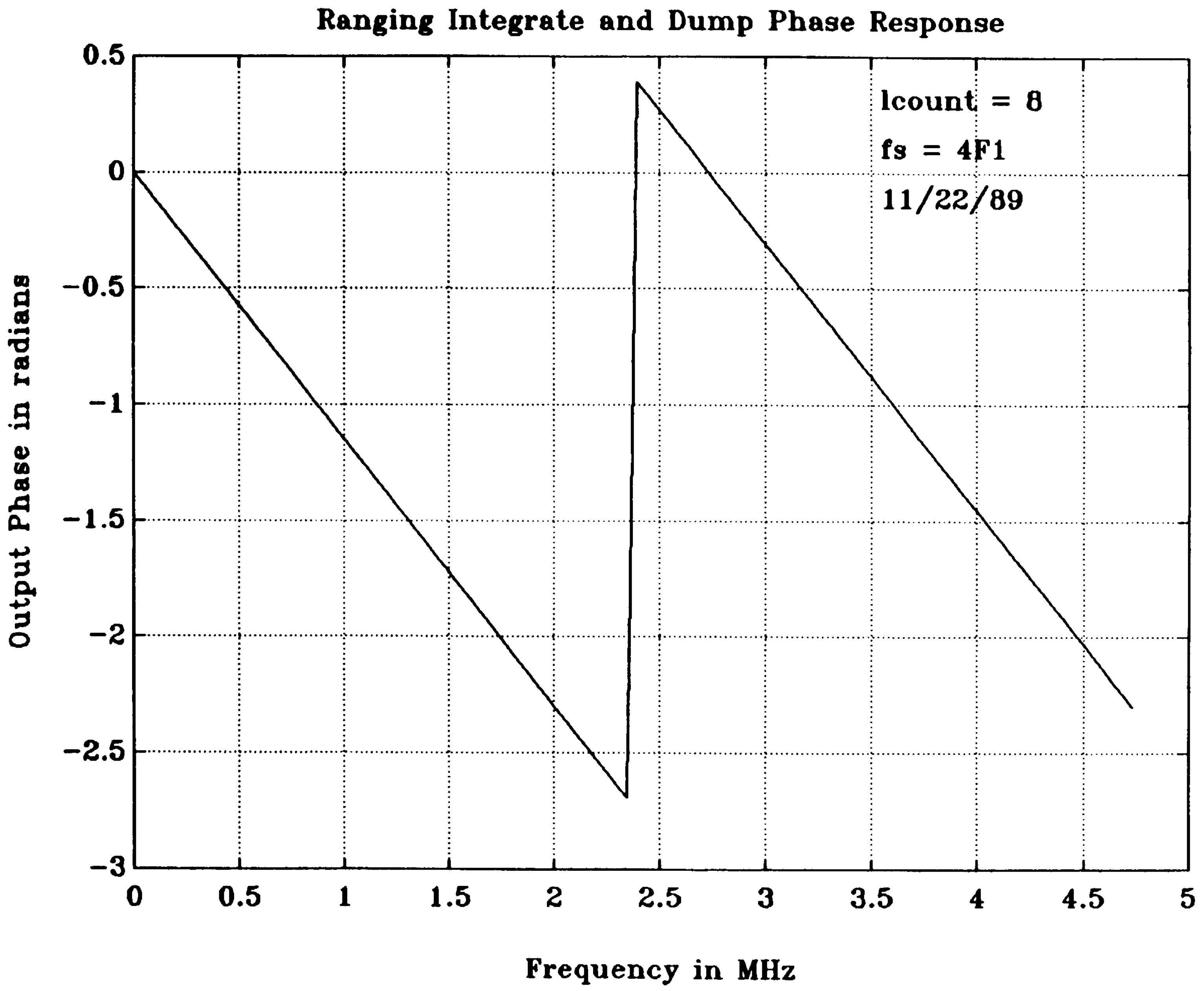


Fig. 76. Phase response of ranging integrate and dump filter.

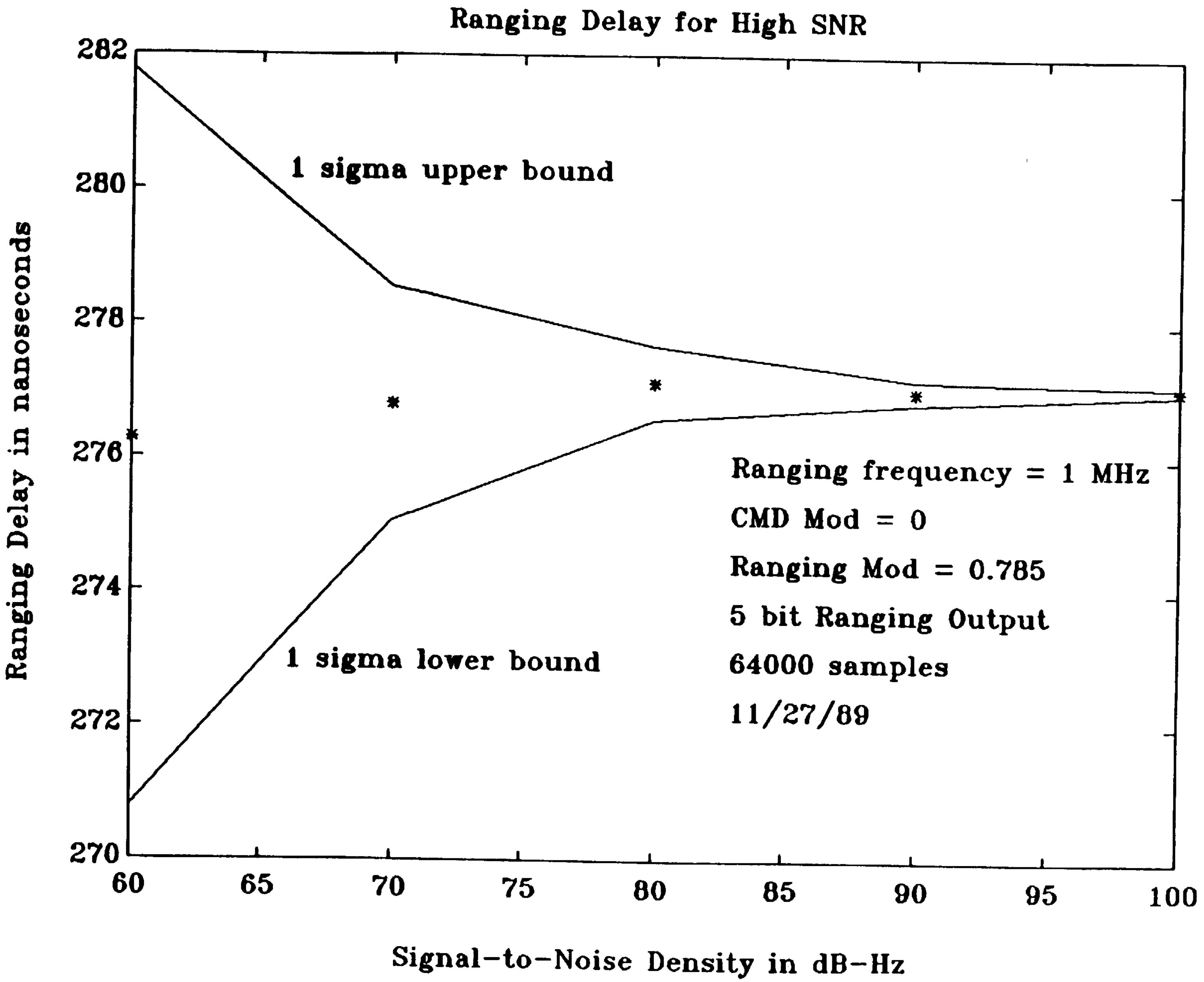


Fig. 77. Software breadboard ranging delay and 1 sigma bounds

and ranging signals at strong input signal levels. Loss of desired signal was apparent. Whether this strong signal ranging performance is affected by additional internal products is heavily dependent on the deep space network (DSN) ranging demodulator design. This effect would therefore require further evaluation by JPL.

Another issue of concern was the effect of the DAC quantization on the phase jitter of the PLL. It was shown in this study that with only 8 bits of quantization, the jitter compared favorably with the equivalent analog model, and that the quantization noise was significantly below the thermal noise in the loop. Finally, it was shown that the reduced dependence of the ranging phase delay on temperature and aging effects continues to be a strong selling point of the digital receiver, and that the ranging delay is not affected by signal level.

Not tested in this study was the performance of the noncoherent AGC. This AGC is shown in the block diagram of Fig. 1. It was felt that the AGC would have no impact on the testing that was performed. In addition, since it is not clear whether a coherent AGC should be used in lieu of the noncoherent AGC; the AGC performance should be evaluated in a future study after a decision on which AGC should be used has been made.

Some areas requiring further study include: 1) selection of the type of AGC, 2) inclusion of CDU functions into the breadboard, and 3) eventual hardware breadboarding of the block diagram.

BIBLIOGRAPHY

[BOM87] D. Boman, Performance of a Low Sampling Rate Digital Phase-Locked Loop, Master's Thesis, Arizona State University, May 1987.

[LIN81] W.C. Lindsey and C.M. Chie, "A survey of digital phase-locked loops," Proc. IEEE, vol. 69, pp. 410-431, Apr. 1981.

[ATS89] "A transponder study final report," Motorola Strategic Electronics Division Final Report Contract No. 958377, June 1989.

[MAT86] "MATLAB user's guide," The Math Works, South Natick, MA, 1986.

[ZIE85] R.E. Ziemer and R.L. Peterson, Digital Communications and Spread Spectrum Systems, New York: Macmillan Publishing Co., 1985.

[BLA76] A. Blanchard, Phase-Locked Loops: Application to Coherent Receiver Design, New York: John Wiley and Sons, 1976.

[NET85] J. Neter, W. Wasserman, and M.H. Kutner, Applied Linear Statistical Models: Regression, Analysis of Variance, and Experimental Designs, Homewood, IL: Irwin, 1985.

PART B

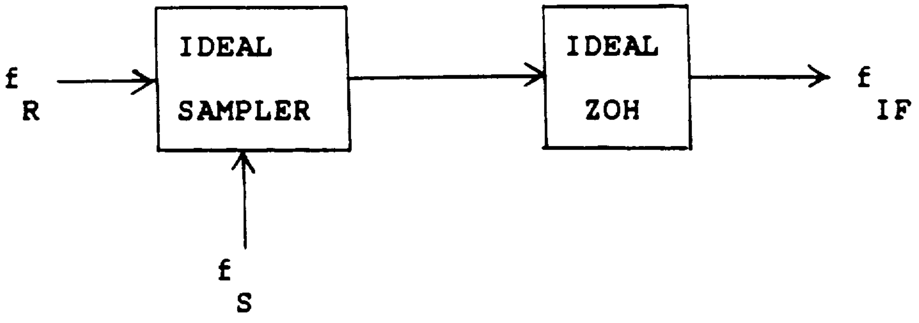
SAMPLING MIXER STUDY

I. IDEAL SAMPLING MIXER MODEL / PERFORMANCE

In this section an ideal model for the sampling mixer is developed, and performance based on this model is predicted. Sections I, II and III are written in MathCAD 2.09.

A. SAMPLING MIXER MODEL

Consider an ideal sampler in cascade with an ideal zero-order hold (ZOH).



f_R = RF Input Frequency

f_S = LO (Sampling) Frequency

Assuming independance between the sampler and hold, it can be shown that [1] the output spectrum of the sampler is

$$F_1(j2\pi f) = \tau f_S \left[F(j2\pi f) + \sum_n \left[\frac{\sin\left[n\pi\tau f_S\right]}{n\pi\tau f_S} \right] \right]$$

$$x F\left[j2\pi\left[f - nf_S\right]\right] + F\left[j2\pi\left[f + nf_S\right]\right]\Big]$$

for:

$n = 1$ to ∞

τ = width of sampling interval

$F(j2\pi f)$ = input RF spectrum

For a down-conversion mixer output frequency near dc, the loss of the sampler is

$$L_1 = 20LOG\left[\tau f_S \left[\frac{SIN\left[n\pi\tau f_S\right]}{n\pi\tau f_S}\right]\right]$$

The ZOH transfer function is found following the method outlined by Schwarz [2]. Assume the input to the ZOH is a train of rectangular pulses of width τ and amplitude h. If $\tau << (1/f_S)$ the rectangular pulses can be approximated by impulses of weight $h\tau$. The output of the ZOH is a pulse of width $(1/f_S)$ and height h. Thus

$$H_2(\omega) = \frac{Y(\omega)}{X(\omega)} = \frac{\int_0^{\frac{1}{f_S}} he^{-j\omega t}\,dt}{\int_0^{\frac{1}{f_S}} h\tau\delta(t)\,dt}$$

where $\delta(t)$ is the unit impulse function

$$H_2(j2\pi f) = \left[\frac{-1}{j2\pi f}\right]\left[e^{\frac{-j2\pi f}{f_s}} - 1\right]$$

$$= \frac{\left[\frac{e^{\frac{-j\pi f}{f_S}}}{\pi\tau f}\right]\left[e^{\frac{j\pi f}{f_S}} - e^{\frac{-j\pi f}{f_S}}\right]}{2j}$$

$$= \left[\frac{1}{\tau f_s}\right]\left[\frac{\sin\left[\frac{\pi f}{f_s}\right]}{\frac{\pi f}{f_s}}\right]\left[e^{\frac{-j\pi f}{f_S}}\right]$$

The first two terms are the magnitude of $H_2(j2\pi f)$ and the last term is is the angle. The dc gain of the ZOH is

$$H_2(0) = \lim_{f \to 0} = \frac{1}{\tau f_s} = \frac{\text{hold time}}{\text{sample time}}$$

The loss of the ZOH can be expressed as

$$L_2 = 20\log\left[\left[\frac{1}{\tau f_s}\right]\left[\frac{\sin\left[\frac{\pi f}{f_s}\right]}{\frac{\pi f}{f_s}}\right]\right]$$

Thus the cascaded response of the sampler and ZOH is

$$H_{SM}(j2\pi f) = \left[\frac{\sin\left[n\pi\tau f_s\right]}{n\pi\tau f_s}\right]\left[\tau f_s\right]\left[\frac{1}{\tau f_s}\right]\left[\frac{\sin\left[\frac{\pi f}{f_s}\right]}{\frac{\pi f}{f_s}}\right]$$

$$= \frac{\sin\left[n\pi\tau f_s\right]\sin\left[\frac{\pi f}{f_s}\right]}{n\tau f\pi^2}$$

This equation represents the best performance attainable with an ideal sampling mixer consisting of an ideal impulse sampler cascaded with an ideal ZOH.

B. THEORETICAL PERFORMANCE

In this section an ideal or upper bound on performance is calculated for the sampling mixer using the expressions derived in section A to compare against the predicted circuit performance in section II.

The loss for an ideal sampling mixer was shown to consist of two terms. The first term

$$L1= \tau f_s \frac{\sin(n\pi f_s \tau)}{n\pi f_s \tau}$$

represents the loss of the sampler at an RF input frequency of nf_s.

This loss is virtually independent of IF frequency (ie constant over all IF frequencies) for IF bandwidths << nf_s". The variables are:

n= harmonic of LO frequency which gives desired IF when added or subtracted from the RF input frequency

f_s = LO (sampling) frequency

τ= width of sampling pulse

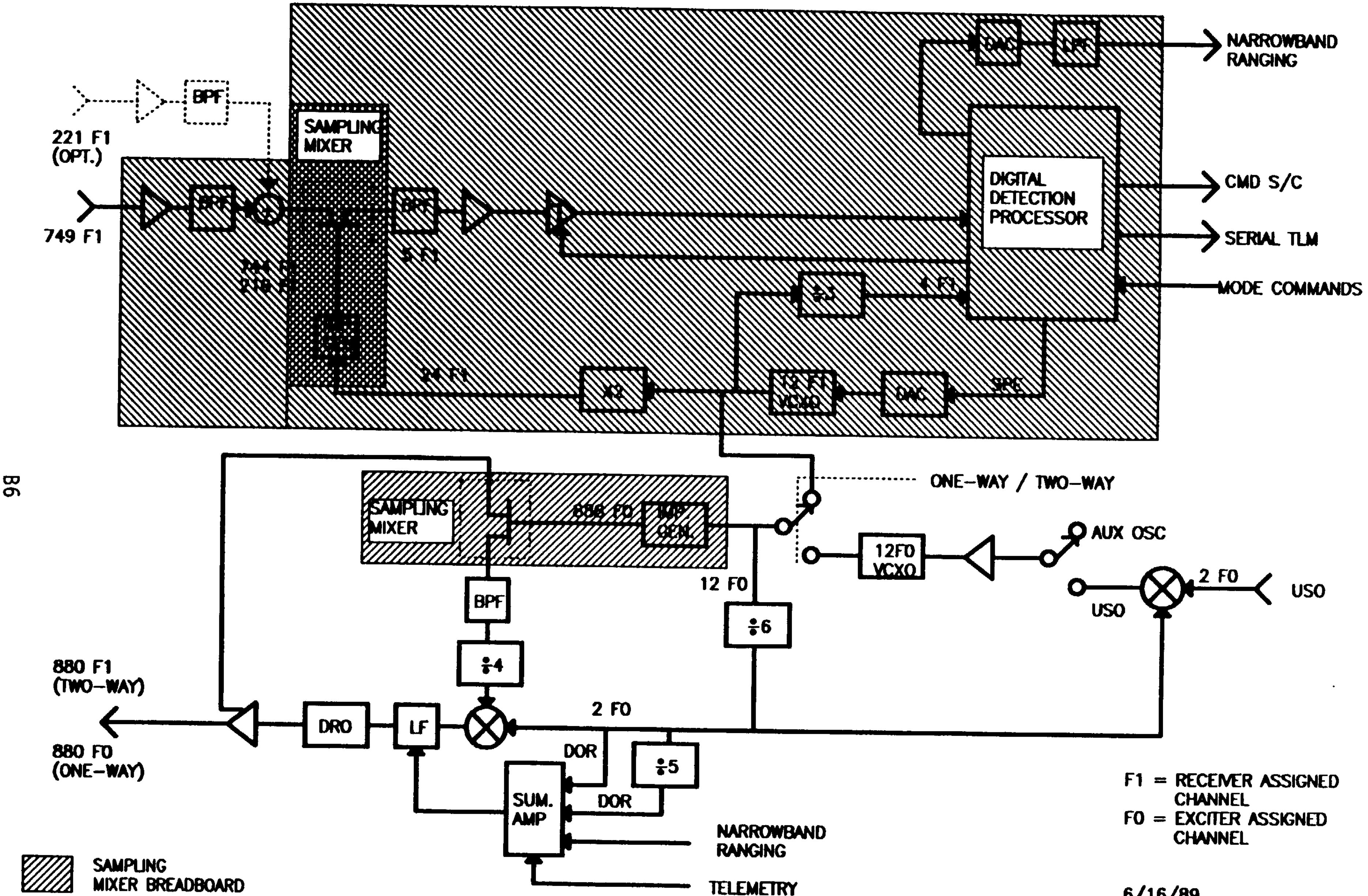
For exciter shown in Figure 1 nf =888F1. The loss is solely a function of τ. s s s
Let

$$n := 74$$

$$f_s := 114.84 \cdot 10^{6}$$

$$\tau_0 := 1 \cdot 10^{-12}$$

$$i := 1 \,..100$$



F1 = RECEIVER ASSIGNED CHANNEL
F0 = EXCITER ASSIGNED CHANNEL

6/16/89

FIGURE 1
CRAF X-BAND TRANSPONDER – OPTION 3
FREQUENCY SCHEME

Neglecting the loss due to the τf_s term in L1

$$L1_i := 20 \cdot \log\left[\frac{\sin\left[n \cdot \pi \cdot f_s \cdot \tau_0 \cdot i\right]}{n \cdot \pi \cdot f_s \cdot \tau_0 \cdot i}\right]$$

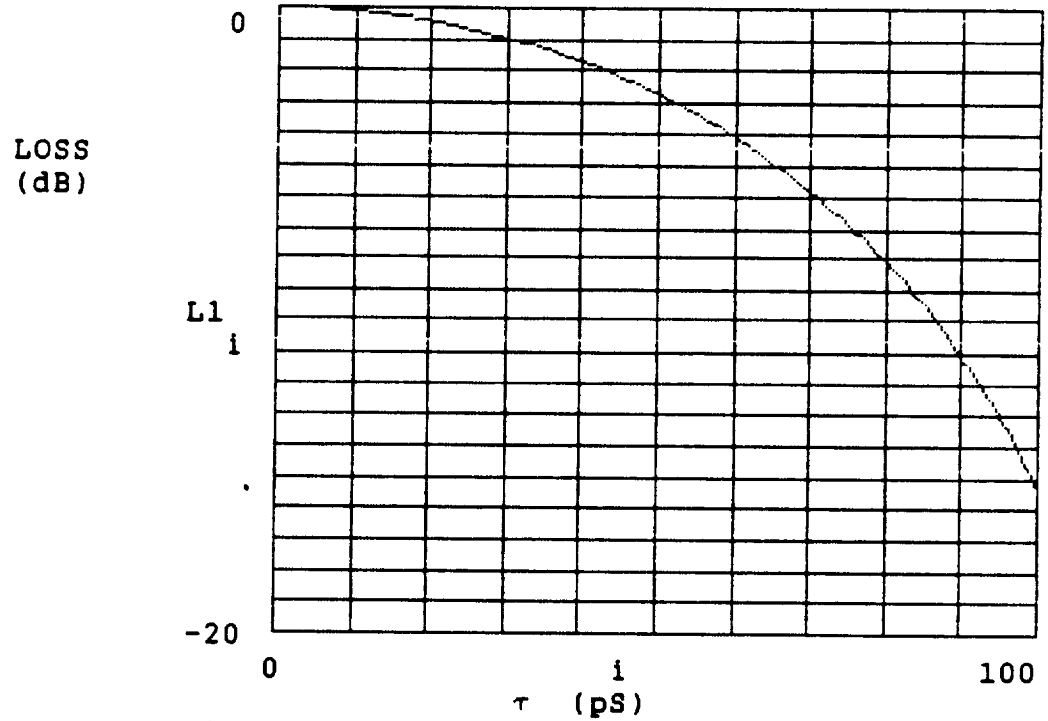


Figure 2 - Exciter First Term Loss vs Pulse Width

A similar situation exists for the receiver in Figure 1, where:

$n := 31$

$f_s := 229.68 \cdot 10^{6}$

$\tau_0 := 1 \cdot 10^{-12}$

$i := 1 \,..100$

$$L1_i := 20 \cdot \log\left[\frac{\sin\left[n \cdot \pi \cdot f_s \cdot \tau_0 \cdot i\right]}{n \cdot \pi \cdot f_s \cdot \tau_0 \cdot i}\right]$$

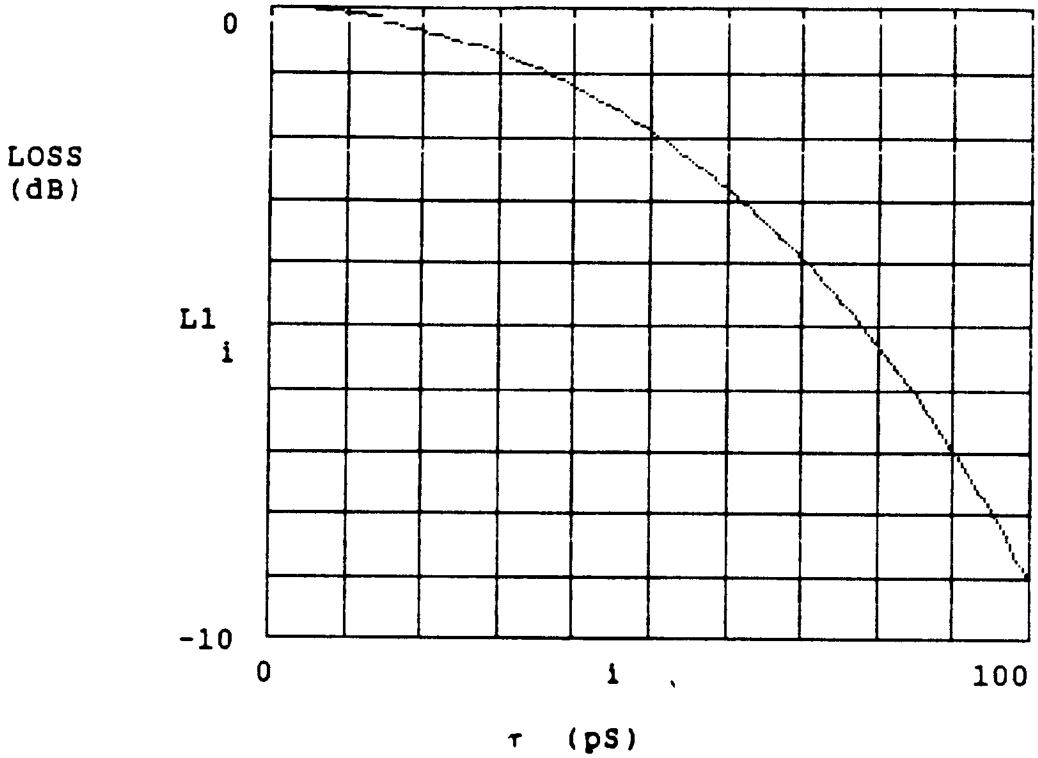


Figure 3 - Receiver First Term Loss vs Pulse Width

An L-band sampling mixer was breadboarded and tested recently on a Motorola IR&D. The theoretical loss of this circuit is considered for comparison. For this L-band sampling mixer:

$$n := 19$$

$$f_s := 81.92 \, 10^{6}$$

$$\tau_0 := 1 \cdot 10^{-12}$$

$$i := 1 \, .. 300$$

$$L1_i := 20 \log\left[\frac{\sin\left[n \cdot \pi \cdot f_s \cdot \tau_0 \cdot i\right]}{n \, \pi \cdot f_s \cdot \tau_0 \cdot i}\right]$$

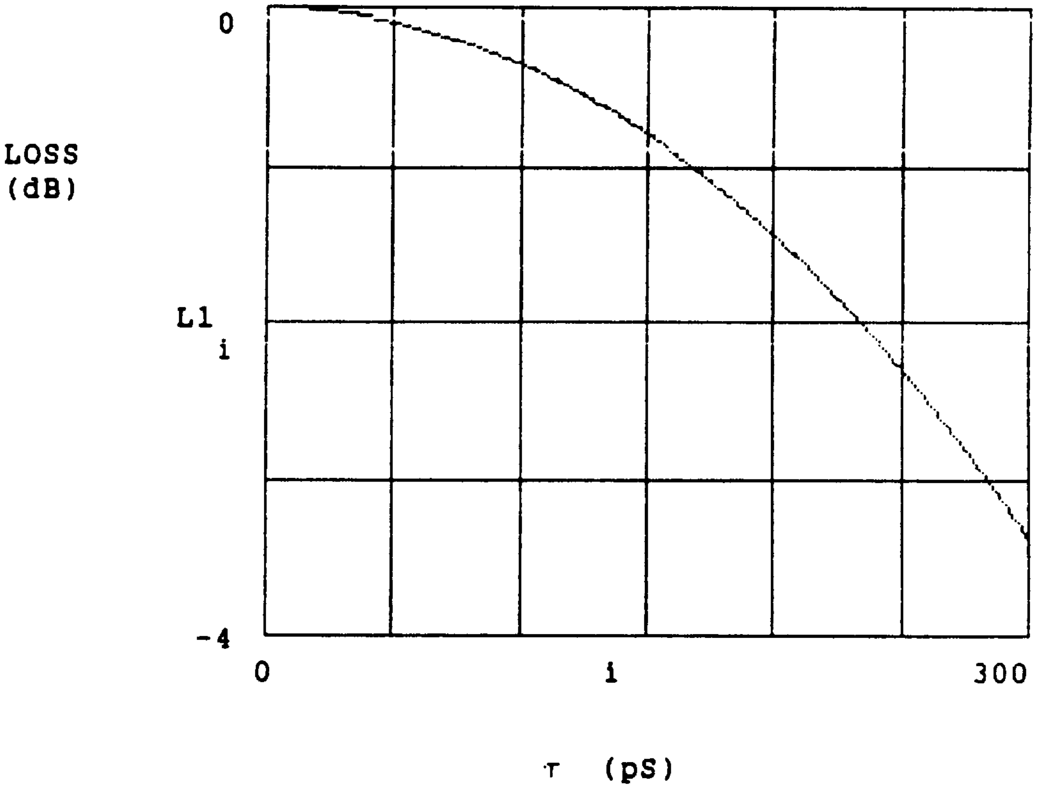


Figure 4
L-Band Sampling Mixer First Term Loss vs Pulse Width

The second ideal sampling mixer loss term was shown to be:

$$L2=\frac{1}{\tau f_s}\;\frac{\sin(\pi f/f_s)}{\pi f/f_s}$$

This is the response of an ideal hold, where f_s and τ are as defined before and f is the IF frequency. For the exciter shown in Figure 1.

$$\tau_0 := 20 \cdot 10^{-12}$$

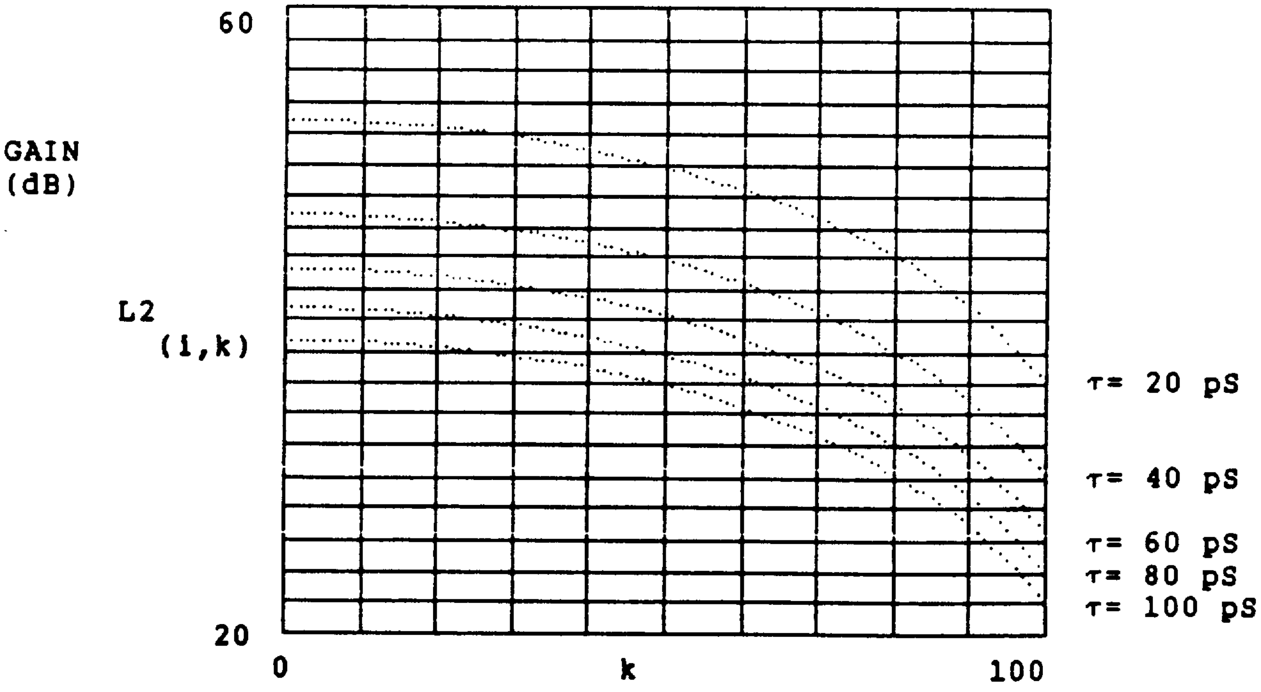
$$f_s := 114.84 \cdot 10^{6}$$

$$i := 1 \,..5$$

$$k := 1 \,..100$$

$$f_0 := 1 \cdot 10^{6}$$

$$L2_{i,k} := 20 \cdot \log\left[\frac{\sin\left[\pi \cdot k \cdot \frac{f_0}{f_s}\right]}{i \cdot \tau_0 \cdot k \cdot f_0 \cdot \pi}\right]$$



IF FREQUENCY (MHz)
Figure 5 - Exciter Second Term Loss vs Pulse Width

Thus the hold provides gain to counteract the sampler loss. Expressing the cascaded loss of the sampler and hold in one equation:

$$L=L1*L2= \tau f_s \frac{\sin(n\pi f_s \tau)}{n\pi f_s \tau} \quad \frac{\sin(\pi f/f_s)}{\pi f/f_s} \quad \frac{1}{\tau f_s}$$

This is the loss at $(f_R$ +/- $nf_s)$. For the exciter shown in Figure 1 :

$$n := 74$$

$$f_s := 114.84 \cdot 10^{6}$$

$$i := 1 \,..5$$

$$\tau_0 := 20 \cdot 10^{-12}$$

$$k := 1 \,..200$$

$$f_0 := 1 \cdot 10^{6}$$

$$L_{i,k} := 20 \cdot \log\left[\sin\left[n \cdot \pi \cdot f_s \cdot i \cdot \tau_0\right] \cdot \frac{\sin\left[\pi \cdot k \cdot \frac{f_0}{f_s}\right]}{n \cdot \pi^{2} \cdot k \cdot f_0 \cdot i \cdot \tau_0}\right]$$

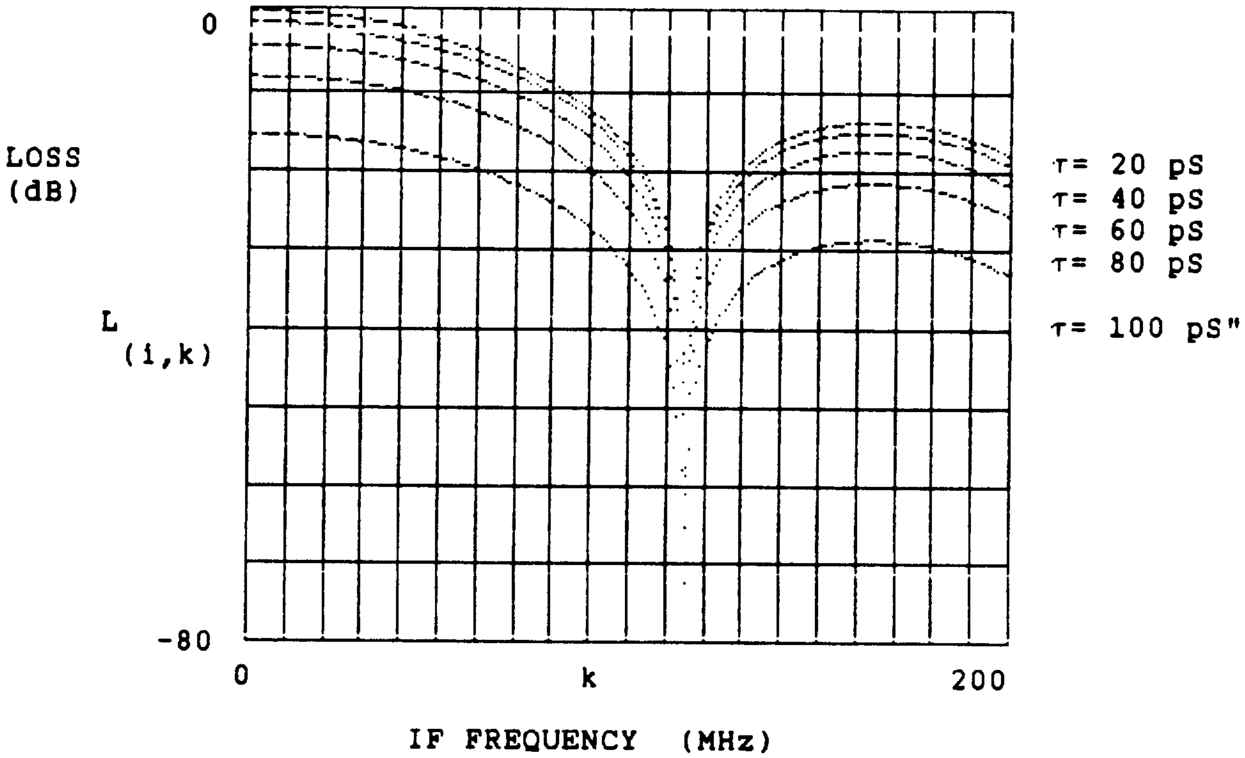


Figure 6 - Exciter Sampling Mixer Loss for n=74

The 8F1 IF has too much loss due to its proximity to f_s. This can be remedied by setting f_s=24F1 and n=32.

$$f_s := 229.68 \cdot 10^6$$

$$n := 32$$

$$L_{i,k} := 20 \cdot \log\left[\sin\left[n \cdot \pi \cdot f_s \cdot i \cdot \tau_0\right] \cdot \frac{\sin\left[\pi \cdot k \cdot \frac{f_0}{f_s}\right]}{n \cdot \pi^2 \cdot k \cdot f_0 \cdot i \cdot \tau_0}\right]$$

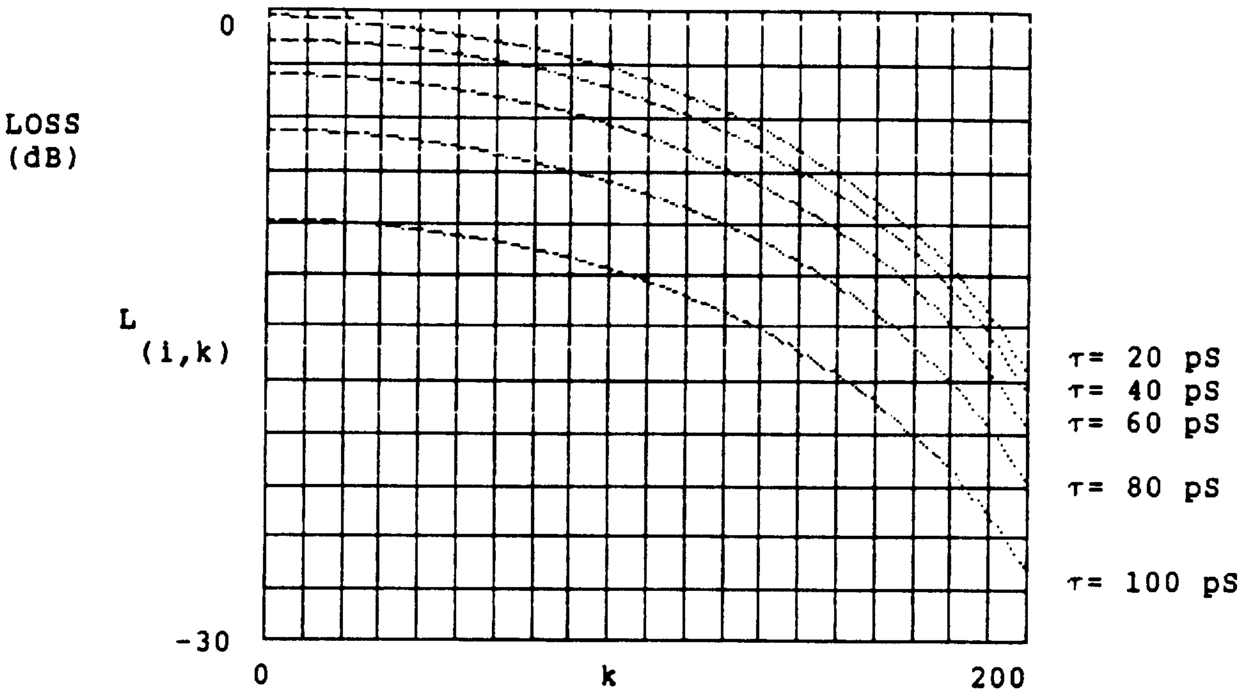


Figure 7 - Exciter Sampling Mixer Loss for n=32

Thus a 60 ns gate time is required of the sampler for a 5 dB loss. Looking more closely at the 8F1 +/-2F1 response for the phase modulator application:

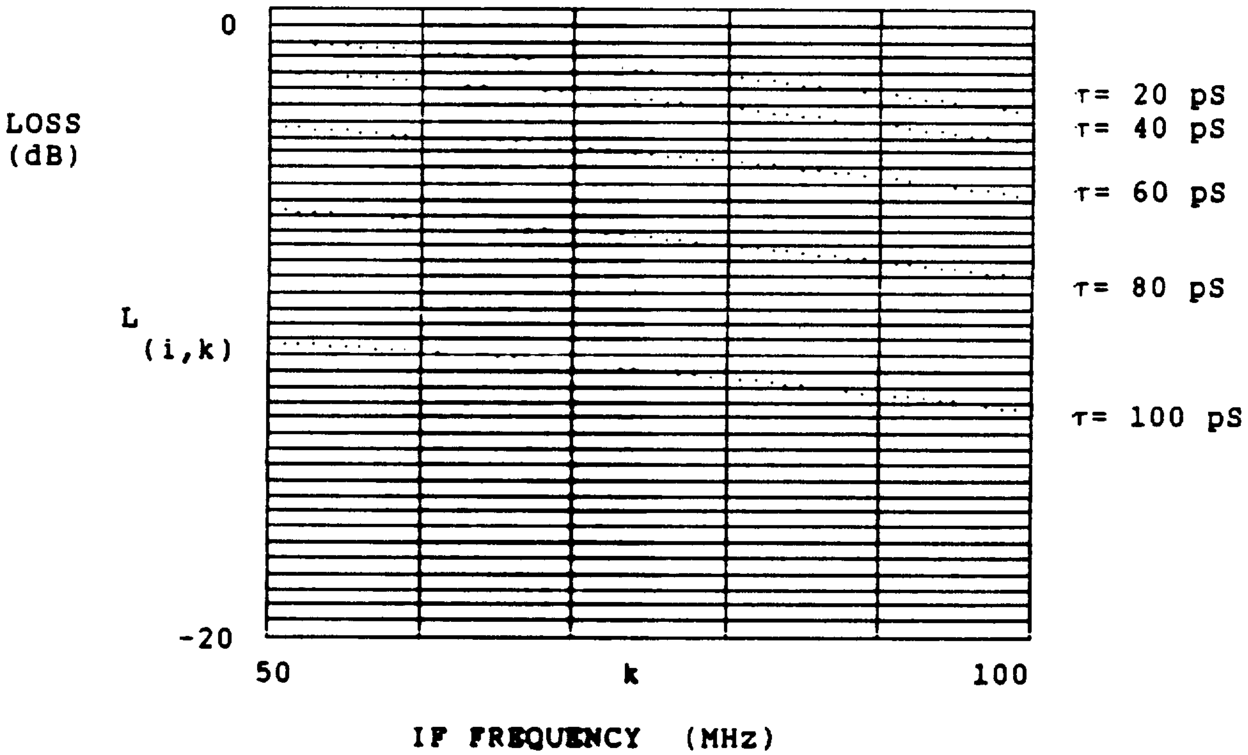


Figure 8 - Exciter Sampling Mixer Loss for n=32 at 8F1+/-2F1

This plot indicates about 1.5 dBpp over 8F1 +/- 2F1 which may be adequate for the modulation response of the phase modulator, as phase modulation is performed within the PLL in Figure 1. A flatter response would be more desireable however.
For the receiver application in Figure 1 :

$$n := 31$$

$$f_s := 229.68 \cdot 10^{6}$$

$$i := 1 \,..5$$

$$\tau_0 := 20 \cdot 10^{-12}$$

$$k := 1 \,..200$$

$$f_0 := 1 \cdot 10^{6}$$

$$L_{i,k} := 20 \cdot \log\left[\sin\left[n \cdot \pi \cdot f_s \cdot i \cdot \tau_0\right] \cdot \frac{\sin\left[\pi \cdot k \cdot \frac{f_0}{f_s}\right]}{n \cdot \pi^2 \cdot k \cdot f_0 \cdot i \cdot \tau_0}\right]$$

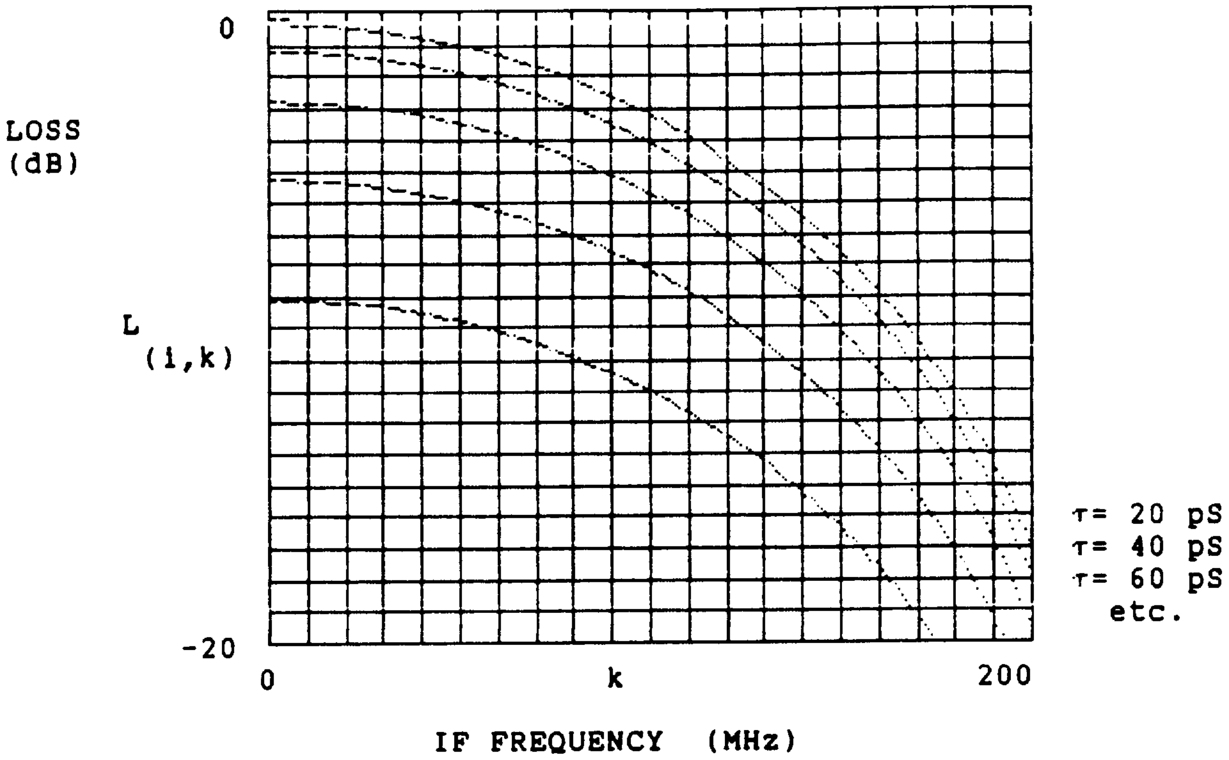


Figure 9 - Receiver Sampling Mixer Loss for n=31

The flatness is adequate for the 5F1 IF of Figure 1.

Computing the L-band IR&D sampling mixer response for a check:

$n := 19$

$f_s := 81.92 \cdot 10^6$

$i := 1 \,..3$

$\tau_0 := 100 \cdot 10^{-12}$

$k := 1 \,..50$

$f_0 := 1 \cdot 10^6$

$$L_{i,k} := 20 \cdot \log\left[\sin\left[n \cdot \pi \cdot f_s \cdot i \cdot \tau_0\right] \cdot \frac{\sin\left[\pi \cdot k \cdot \frac{f_0}{f_s}\right]}{n \cdot \pi^2 \cdot k \cdot f_0 \cdot i \cdot \tau_0}\right]$$

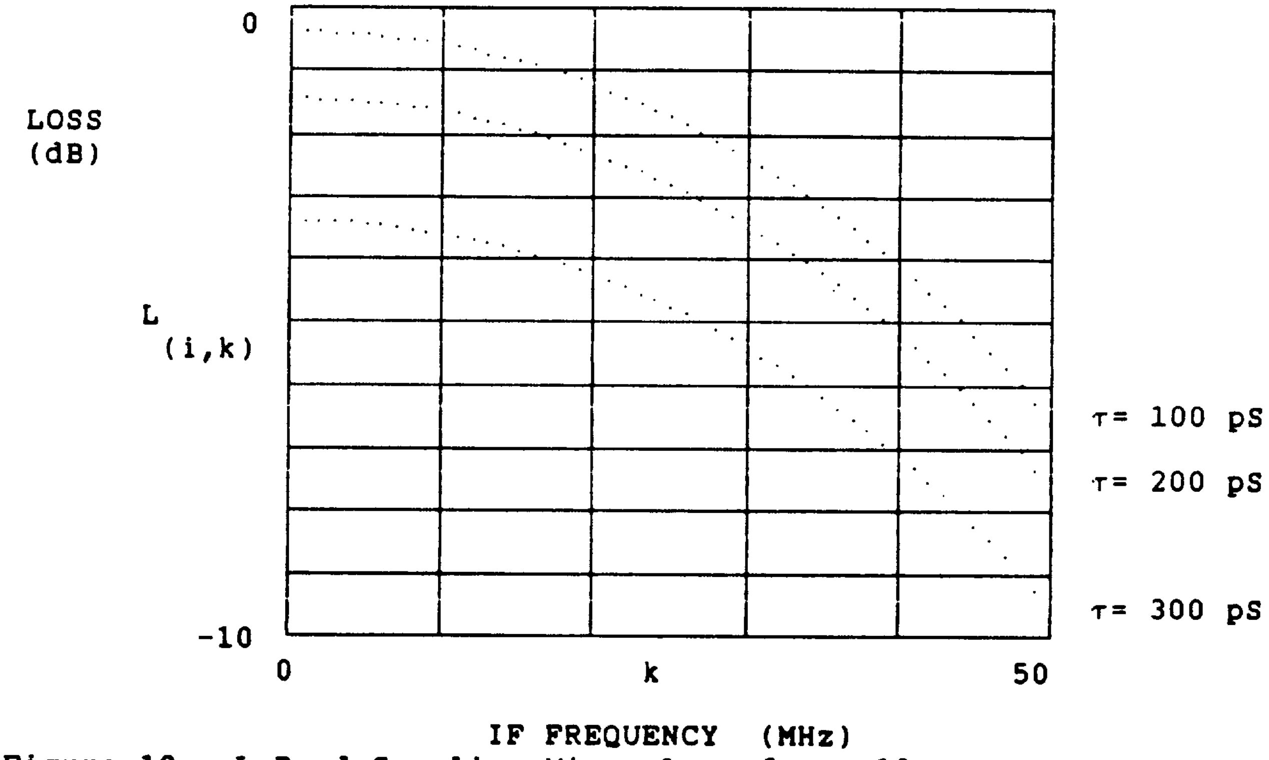


Figure 10 - L-Band Sampling Mixer Loss for n=19

Thus, the L-band IR&D sampling mixer could achieve a 35 Mhz 3dB bandwidth but measured data shows 2.5 Mhz is the best performance achieved to date. Techniques to further flatten the IF frequency response are discussed in section C.

C. IMPROVING IF RESPONSE

The IF response of the sampling mixer generally follows a sin x / x shape as shown in the previous section. Circuit realizations of the sampling mixer can be made to behave as second - order systems as shown in [3]. The frequency response of the second - order system can be used to counteract the sin x / x response and provide a flat IF response if the natural frequency and damping factor are chosen correctly. Consider the response of the circuit in Figure 11 below to the unit step function u(t) from t = 0 to t = τ.

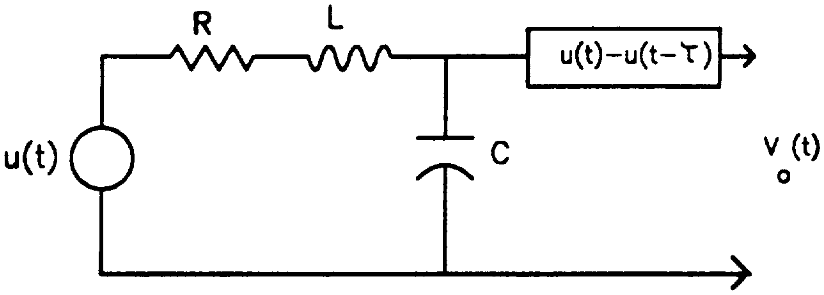


Figure 11 - Second - Order System

The output frequency response of this circuit to an input step function of amplitude 1 is

$$V_o(s) = \frac{\frac{1}{sC}}{R + sL + \frac{1}{sC}} \frac{1}{s} = \frac{\frac{1}{LC}}{s^2 + \left[\frac{R}{L}\right]s + \frac{1}{LC}} \frac{1}{s}$$

Define $\omega_c = \frac{1}{\sqrt{LC}}$ = natural frequency; β = damping factor

$$V_o(s) = \frac{\omega_c^2}{s^2 + \left[\frac{R}{L}\right]s + \omega_c^2} \frac{1}{s} = \frac{\omega_c^2}{s^2 + 2\beta\left[\omega_c\right]s + \omega_c^2} \frac{1}{s}$$

Then $2\beta\left[\omega_c\right]s = \frac{R}{L}$ and $\beta = \frac{R}{2}\sqrt{\frac{C}{L}}$

The time domain response is found by taking the inverse Laplace transform of $V_o(s)$

$$V_o(t) = 1 - \left[\frac{1}{\sqrt{1 - \beta^2}}\right] e^{-\beta\left[\omega_c\right]t} \sin\left[\omega_c \left[\sqrt{1 - \beta^2}\right]t + \phi\right]$$

In general $V_o(t)$ is a damped sinusoidal response as shown in Figure 12.

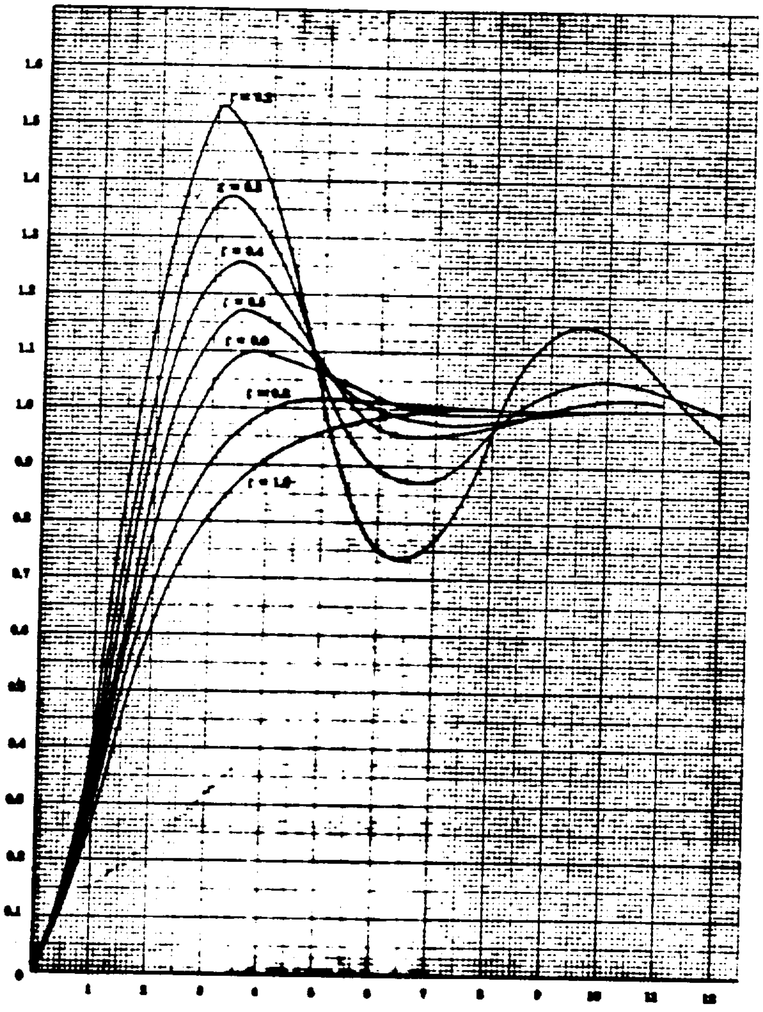


Figure 12 - Response of Second - Order System

We want to solve for β and ω_c such that $V_o(t) = 1$ at $t = \tau$.

Setting $V(\tau) = 0$ in the previous equation we find

$$\omega_c \left[\sqrt{1 - \beta^2}\right]\tau \quad + \mathrm{acos}(\beta) = n\pi$$

The "zero" we want is at n=1, therefore

$$\omega_c = \frac{\pi - \mathrm{acos}(\beta)}{\tau\left[\sqrt{1 - \beta^2}\right]}$$

This equation for ω_c and the one derived previously for β can be used as design equations to flatten the sampling mixer IF response as shown in section II.

II. CIRCUIT ANALYSIS MODEL / PERFORMANCE

In this section the circuit analysis model for a series GaAs FET sampling mixer is developed and performance based on this model is predicted. Predicted performance for an L-band sampling mixer is compared with measured data. Analysis methodology is also discussed.

A. APPROACH

The actual design approach and methodology used to develop the sampling mixer model differed significantly from that proposed. At the time the proposal was written it was planned to breadboard a series GaAs FET sampling mixer very similar to that described in [4]. This implementation had been found in a literature search during "A Transponder Study" and was considered to be the best candidate approach for several reasons. The authors in [4] suggested that the use of a GaAs FET as a sampling switch could reduce the power required to drive the switch over conventional diode-implemented switch configurations due to the high input impedance of the GaAs FET gate. Also it was shown in [4] that the series GaAs FET configuration had much less conversion loss than the diode implementations.

After the proposal for this project had been submitted, development work on a Motorola IR&D, L-band sampling mixer encountered considerable difficulty in obtaining IF bandwidths greater than two MHz. The L-band sampling mixer was an approach similar to [4] and did achieve a conversion loss of less than 6 dB for IF frequencies less than one MHz using a moderate step-recovery diode (SRD) drive power of +10 dBm. Work on the IR&D sampling mixer ceased without solving the IF bandwidth problem prior to the start of this project, the narrow IF response being adequate for the application.

An X/S-band sampling mixer with approximately 100 MHz of IF bandwidth is necessary to implement the block diagram in Figure 1, which was considered to be the most likely implementation of the CRAF/CASSINI transponder at the time the development of the sampling mixer on this contract was conducted. It therefore seemed prudent to find the cause of the narrow IF bandwidth of the existing L-band design before proceeding with a similar but more difficult X/S-band design. This approach would provide an extra breadboard iteration as the existing L-band breadboard was immediately available for test, providing useful "hands on" familiarity with the circuit's operation. For these reasons it was decided to model the L-band circuit first before proceeding with the X/S sampling mixer development. As work progressed it became clear that L-band was near the upper frequency limit for a breadboard composed of conventional discrete components such as CDR12 chip capacitors, M55342 chip resistors and packaged GaAs FETs due to the sampling mixer's high susceptability to stray capacitances and inductances. Thus, L-band was as close to the X/S-band implementation as could be breadboarded with easily changeable discrete components.

The time required to model the L-band sampling mixer and correct the narrow IF bandwidth problem consumed considerably more time than planned due to analysis methodology problems discussed in the next section.

B. METHODOLOGY

The GaAs FET selected for the X/S sampling mixer breadboard was the NEC 710, the same GaAs FET used in the L-band implementation. This selection was based primarily on three reasons. Curtis model [5] parameters were available from NEC so measurement of these parameters was not required. Recent articles in the technical literature showed the NEC 710 to have the required switching speed for an X-Band sampling mixer [6]. The L-Band circuit was implemented with the NEC 710.

The non-linear nature of the sampling mixer dictated the use of SPICE based circuit analysis. Three programs using the SPICE core were tried in an effort to reduce the long run times required to obtain one cycle of the IF frequency so that conversion loss could be determined.

Microwave Spice on an Apollo computer was the first program to be evaluated. This program had extremely long run times as only one time step is allowed throughout the run. The sampling mixer LO is a series of narrow pulses followed by long "off" times. Many small time increments are necessary to adequately define the pulse, however, much longer time increments can be tolerated in the "off" time to model the decay of the voltage on the hold capacitor. Only one copy of this program was available which had to be shared with several projects.

HSPICE on an IBM mainframe computer was evaluated next. HSPICE allows different time steps, but requires many program lines to define the series of time intervals for one cycle of the IF frequency. Off line printing of graphs was also required, reducing the efficiency of this analysis approach.

The selected methodology approach was to use an evaluation copy of PSPICE on an IBM PS 2, model 80 computer with a math coprocessor. PSPICE allows two different time steps in a repeating interval. The program automatically adjusts step size according to the amount of activity in the simulation resulting in 10 to 20 times speed improvement over using a single step size.

Even with using the PSPICE approach, approximately 30 minutes of run time was required to obtain a single frequency sweep (7 frequencies) of the IF output. No automatic sweeping or optimization of componentswas possible as with linear circuit analysis programs. Also, manual plotting of the results was required.

GaAs FET models for high speed switching applications are still being debated in the literature. The case of Vds=0 (the sampling mixer application) may not produce very accurate results as stated in various technical articles [7]. In our simulations Cgs and Cgd were set to zero to obtain results that best matched the breadboard measurements.

Modelling the dynamic output impedance of the SRD drive circuit required a significant amount of unplanned effort.

The problems described previously considerably slowed the analysis of the L-Band circuit. However the frequency response and conversion loss of the selected model circuit were close to the measured data.

C. L-BAND IMPLEMENTATION

The selected L-Band circuit model is shown in Figure 13. Values for the parasitic capacitances and inductances of C4, C5, C6, C7, LD, LS, and LD were estimated for the physical layout of the L-Band breadboard. CHOLD and RL1 are measured values for the FET probe used to take the breadboard data. VIN and R3 represent the 50 ohm input RF signal. VPL and R50 represent the SRD LO source. In this model the LO drive signal is a rectangular pulse of 212 pS duration which drives the gate to Vgs= +0.4 volts to provide complete switching of the FET without driving the diodes in the gate/source and gate/drain regions into significant foward conduction. This models the actual triangular pulse from the SRD impulse generator which is clipped by these diodes to form the switching squarewave. No significant forward conduction occurs in the actual circuit as the output impedance of the SRD impulse generator is much higher than the 50 ohms used in this simulation. The PSPICE circuit file is shown in Table 1. Figure 14 showns the measured, simulated and theoretical circuit performance. As can be seen, the measured and simulated IF flatness falls short of the theoretical sin x /x response.

Using the equations derived in section I, part C for ω_C and β, values of LS+LD = 4.18nH and CHOLD = 1.836pF were obtained for τ = 212pS and β = 0.5. The value of β was selected to provide a good compromise between rise time and bandwidth of the second - order system. The resulting simulated IF response is shown in Figure 15 using the circuit file of Table 2. The response is significantly flatter than that shown in Figure 14. Other simulations showed the amount of peaking in the IF response can be changed by varying β in much the same manner as a traditional second - order system.

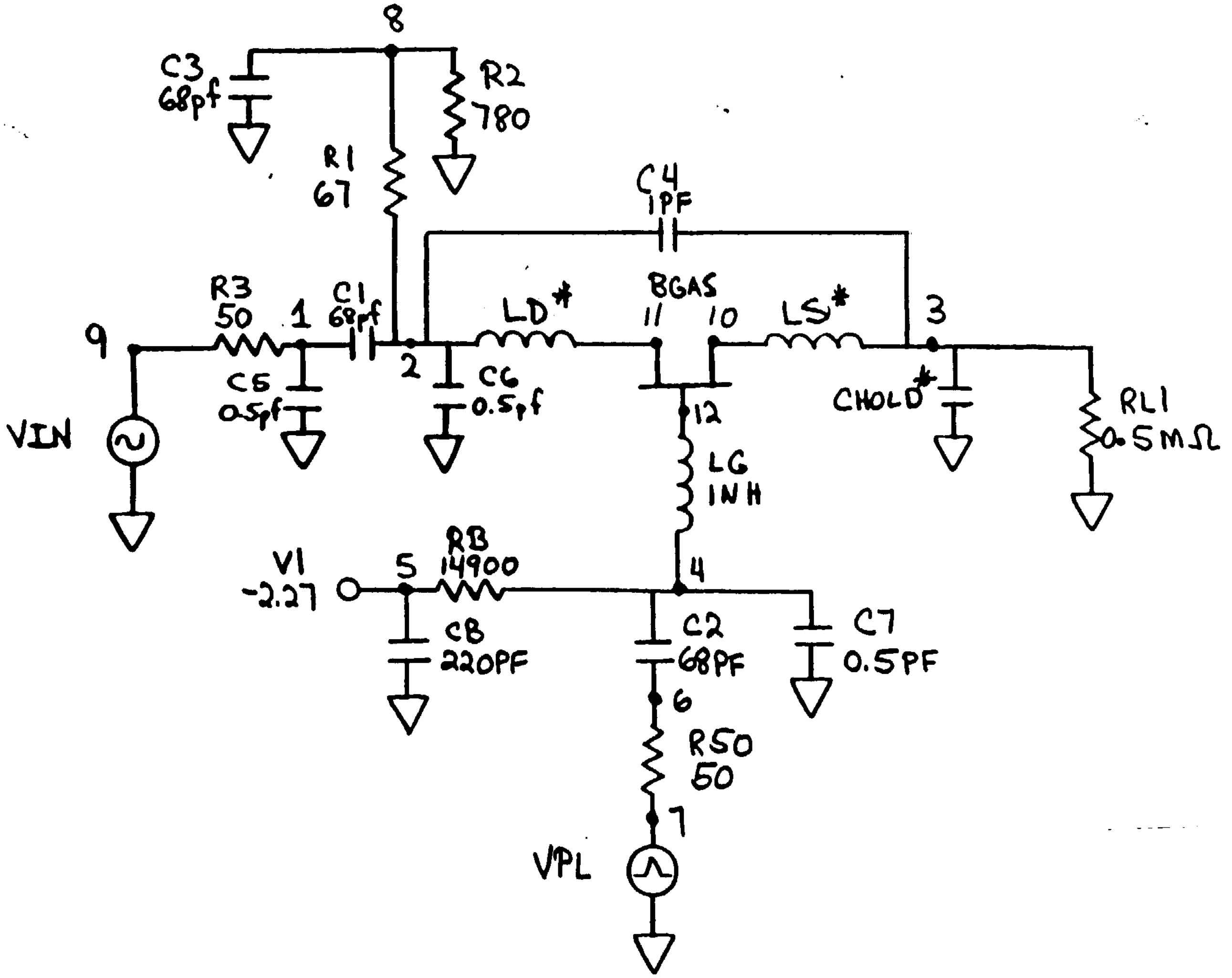


Fig. 13 L-Band Circuit Model

```
******* 09/15/89 ******* Evaluation PSpice (Jan. 1988) ******* 13:42:30 *****


 ****        Table 1.    L-Band Circuit Description

******************************************************************************



C1 1 2 68PF
R1 2 8 67
C3 8 0 68PF
R2 8 0 780
R3 9 1 50
BGAS 11 12 10 NE7101
RB 4 5 14900
CB 5 0 220PF
C2 6 4 68PF
C4 2 3 1PF
LD 10 3 1NH
LS 11 2 1NH
LG 12 4 1NH
C5 1 0 .5PF
C6 2 0 .5PF
C7 4 0 .5PF
R50 6 7 50
CHOLD 3 0 5.8PF
RL1 3 0 .5MEG
V1 5 0 DC  -2.27
VPL 7 0 PULSE(0 2.67 0 1PS 1PS 210PS 20NS)
VIN 9 0 SIN(0 .032 1.551E9 0 0 0)
.TRAN 1NS 1010NS 0
.FOUR 1E6 V(3)
.MODEL NE7101 GASFET(VTO=-1.0 ALPHA=4.5 BETA=.055 LAMBDA=.12
+RG=2.0 N=1.5
+RD=2.37 RS=3.7 CDS=.5PF IS=1.88E-10)
.OPTIONS ITL5=0
.END
```

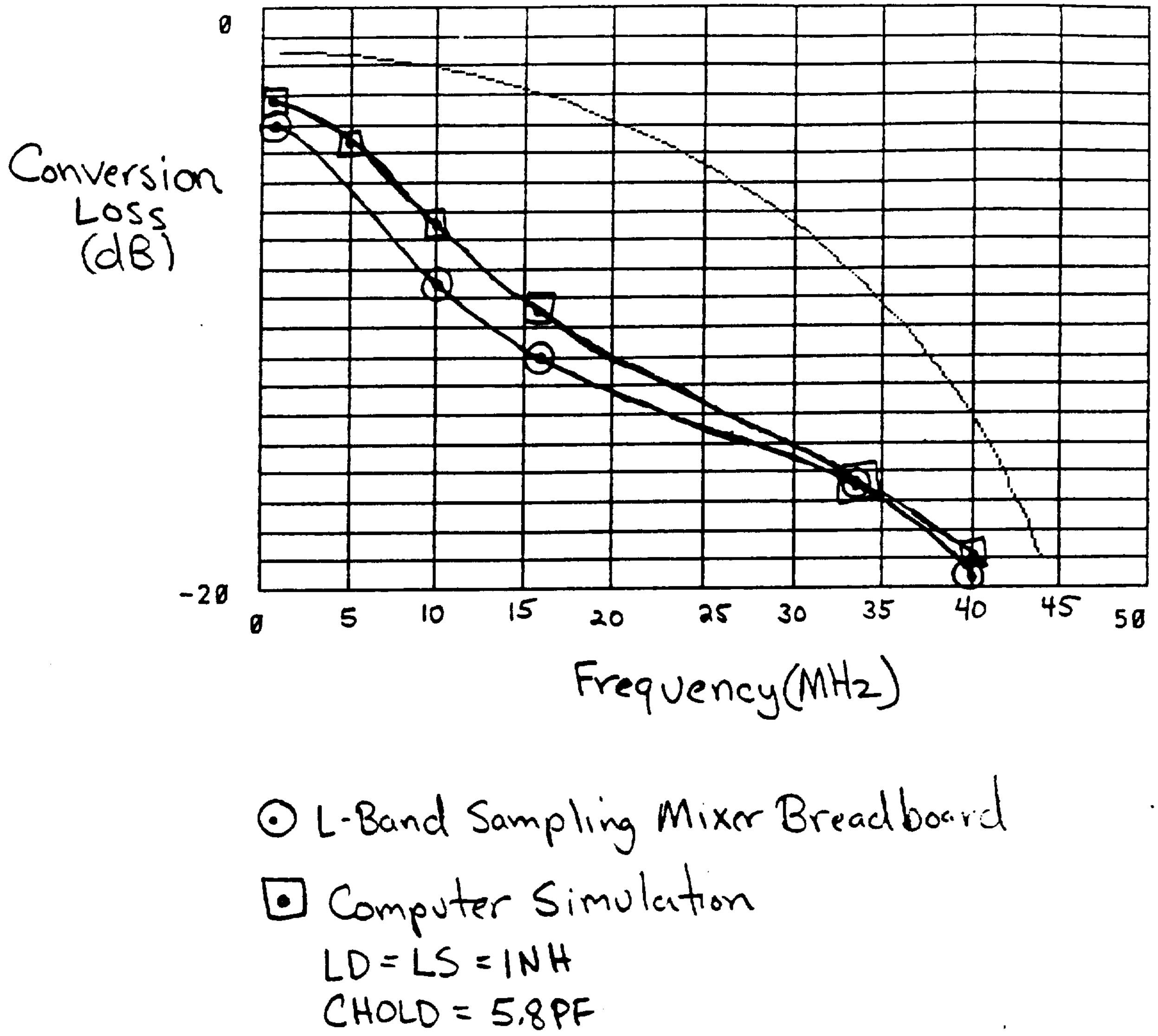


Fig. 14 L-Band Circuit Performance

G 1232

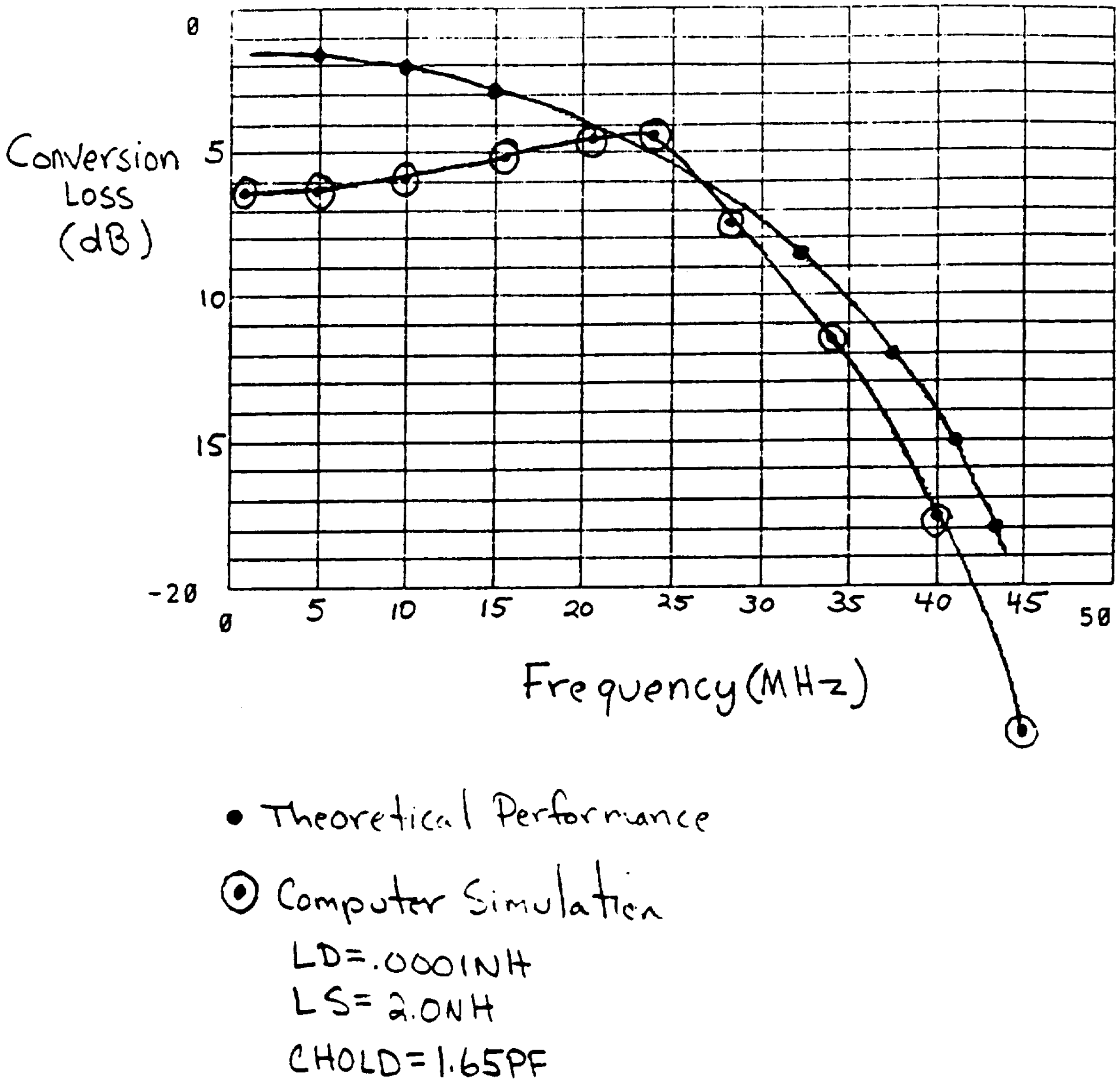


Fig. 15 Improved L-Band Circuit

```
******* 09/15/89 ******* Evaluation PSpice (Jan. 1988) ******* 08:59:06 ****


 ****     Table 2.    Improved L-Band Circuit Description

******************************************************************************


C1 1 2 68PF
R1 2 8 67
C3 8 0 68PF
R2 8 0 780
R3 9 1 50
BGAS 11 12 10 NE7101
RB 4 5 14900
CB 5 0 220PF
C2 6 4 68PF
C4 2 3 1PF
LS 10 3 2.0NH
LD 11 2 .0001NH
LG 12 4 1NH
C5 1 0 .5PF
C6 2 0 .5PF
C7 4 0 .5PF
R50 6 7 50
CHOLD 3 0 1.65PF
RL1 3 0 .5MEG
V1 5 0 DC  -2.27
VPL 7 0 PULSE(0 2.67 0 1PS 1PS 210PS 20NS)
VIN 9 0 SIN(0 .032 1.551E9 0 0 0)
.TRAN 1NS 1010NS 0
.FOUR 1E6 V(3)
.MODEL NE7101 GASFET(VTO=-1.0 ALPHA=4.5 BETA=.055 LAMBDA=.12
+RG=2.0 N=1.5
+RD=2.37 RS=3.7 CDS=.5PF IS=1.88E-10)
.OPTIONS ITL5=0
.END
```

D. X-BAND IMPLEMENTATION

The first simulated X-Band configuration is shown in Figure 16, and its circuit file is listed in Table 3. Using the equations derived in section I, part C, values of LD = 1.18nH and CHOLD = 0.521pF were obtained for τ = 60 pS and β = 0.5. It was desired to simulate the performance of a circuit with no parasitic inductances or capacitances as a baseline for the actual breadboard implementation. The simulated versus theoretical responses are shown in Figure 17. Again, the simulated IF passband is fairly flat as in the L-Band case.

At the time this effort was terminated, investigations into what realization to breadboard had just begun. The major problems to be overcome were determined to be:

1. Minimization of parasitic inductances and capacitances.
2. Realization of a tunable network to align the IF passband to be flat.
3. Realization of a high impedance load (R > 10Kohms).

A first effort to deal with the stray capacitance of a load is shown if Figure 18. The load capacitance of 1.5pF is tuned out by LHOLD providing the bandpass IF response of Figure 19. Since a lowpass IF response is not required for the receiver and exciter of Figure 1, this approach could be used to allievate problems 2 and 3 above.

III. CONCLUSIONS AND RECOMMENDATIONS

The preceeding work indicates that building a manufacturable X/S sampling mixer is possible if good solutions to the above three problems can be found.

Minimization of parasitic inductances and capacitances to the point necessary to implement a practical circuit would likely involve using MIC or MMIC technology. A MIC approach is recommended due to its significantly smaller development cost and alignment possibilities. Model inaccuracies in the design phase would be offset by selecting components on a breadboard sampling mixer MIC. Production units could be aligned by changing the lengths (inductances) of bond wires in series with the hold capacitor. The hold capacitor could be varied in discrete steps by implementing it as several square pads connected in parallel as required in alignment to obtain the flattest response. Realization of a high impedance load is probably best implemented using a GaAs FET source follower after the hold capacitor. A GaAs FET has very low gate to source and gate to drain capacitances which are much smaller than the required CHOLD.

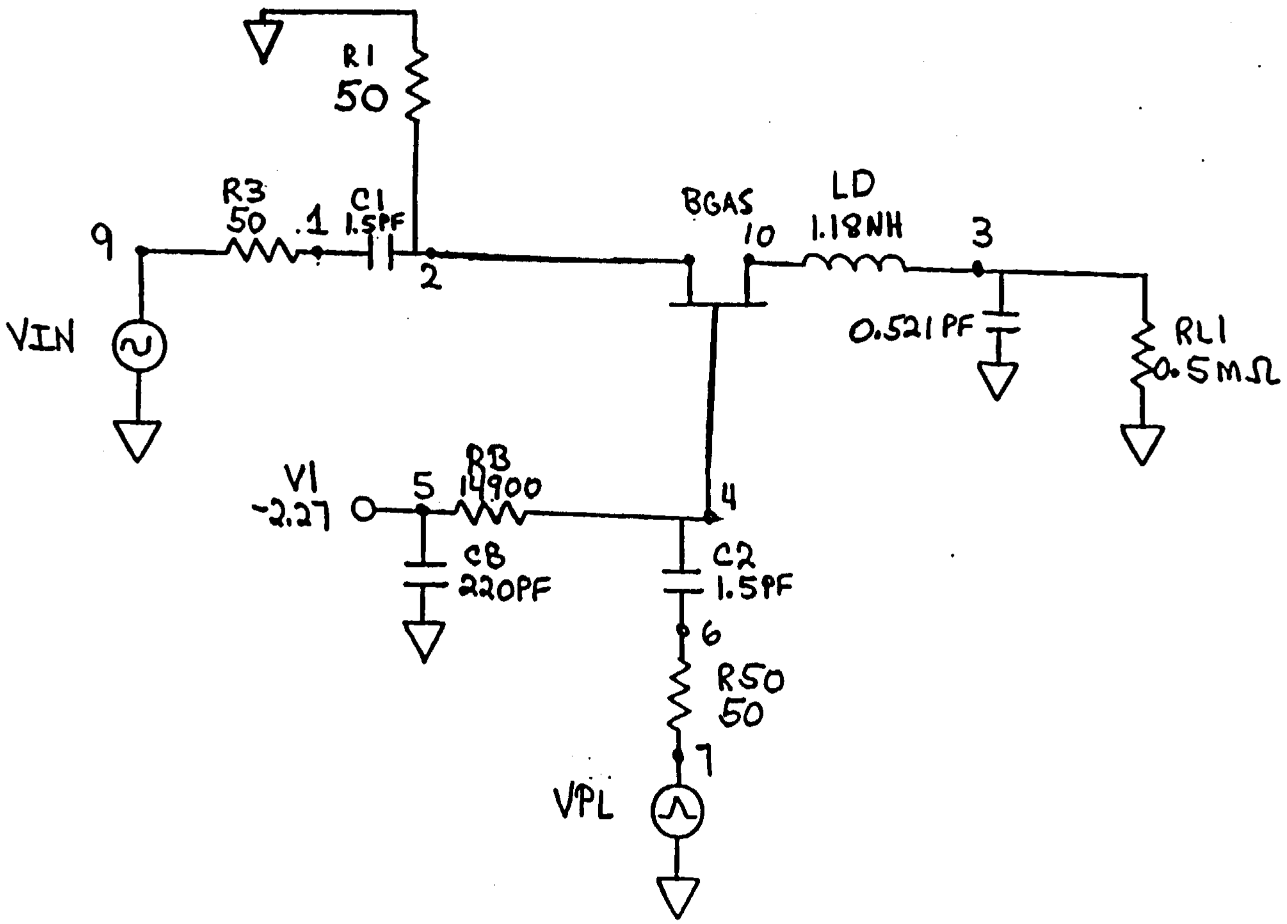


Fig. 16 X-Band Circuit Model

Table 3. X-Band Circuit Description

```
 ****

*************************************************************************************



C1 1 2 1.5PF
R1 2 0 50
R3 9 1 50
BGAS 2 4 10 NE7101
RB 4 5 14900
CB 5 0 220PF
C2 6 4 1.5PF
*C4 2 3 1PF
LD 10 3 1.18NH
*LS 11 2 1NH
*LG 12 4 1NH
*C5 1 0 .5PF
*C6 2 0 .5PF
*C7 4 0 .5PF
R50 6 7 50
CHOLD 3 0 .521PF
RL1 3 0 .5MEG
V1 5 0 DC -2.27
VPL 7 0 PULSE(0 2.67 0 1PS 1PS 60PS 4.3392504NS)
VIN 9 0 SIN(0 .032 8.503772725E9 0 0 0)
.TRAN 1NS 44NS 0
.FOUR 23.045454E6 V(3)
.MODEL NE7101 GASFET(VTO=-1.0 ALPHA=4.5 BETA=.055 LAMBDA=.120
+RD=2.37 RS=3.7 RG=2.0 IS=1.88E-10 N=1.5)
.OPTIONS ITL5=0
.END
```

Circuit Components with * are not in the circuit analysis

The benefits of developing a sampling mixer MIC go far beyond deep space transponder applications. A MIC sampling mixer developed for use at X-Band could be used for other down-conversion applications having different but lower RF frequencies. No redesign or realignment of the MIC for different LO frequencies should be required if the SRD impulse generator circuitry is kept external to the hybrid. Longer pulse widths (acceptable at lower RF input frequencies) could also be accomodated without redesign or realignment if a damping factor β of 0.5 or greater is used in the original design. The sampling mixer mayhave been aligned to provide a first zero crossing in the rise time (see Figure 3) at a 60 pS pulse width. However, longer pulse widths which could be used at lower RF frequencies will simply follow the time response of Figure 12. The maximum conversion loss change will be less than 1.5 dB due to sampling in a peak or valley of the time response for $\beta > 0.5$.

In conclusion Motorola recommends that a sampling mixer MIC be developed to include as wide a range of NASA applications as possible, including deep space and TDRSS transponders. The possibility of the LO source being derived from digital circuitry using various sampling strategies could yield many potential new applications for this part.

Motorola would like to thank N. Mysore of JPL for his assistance in this project.

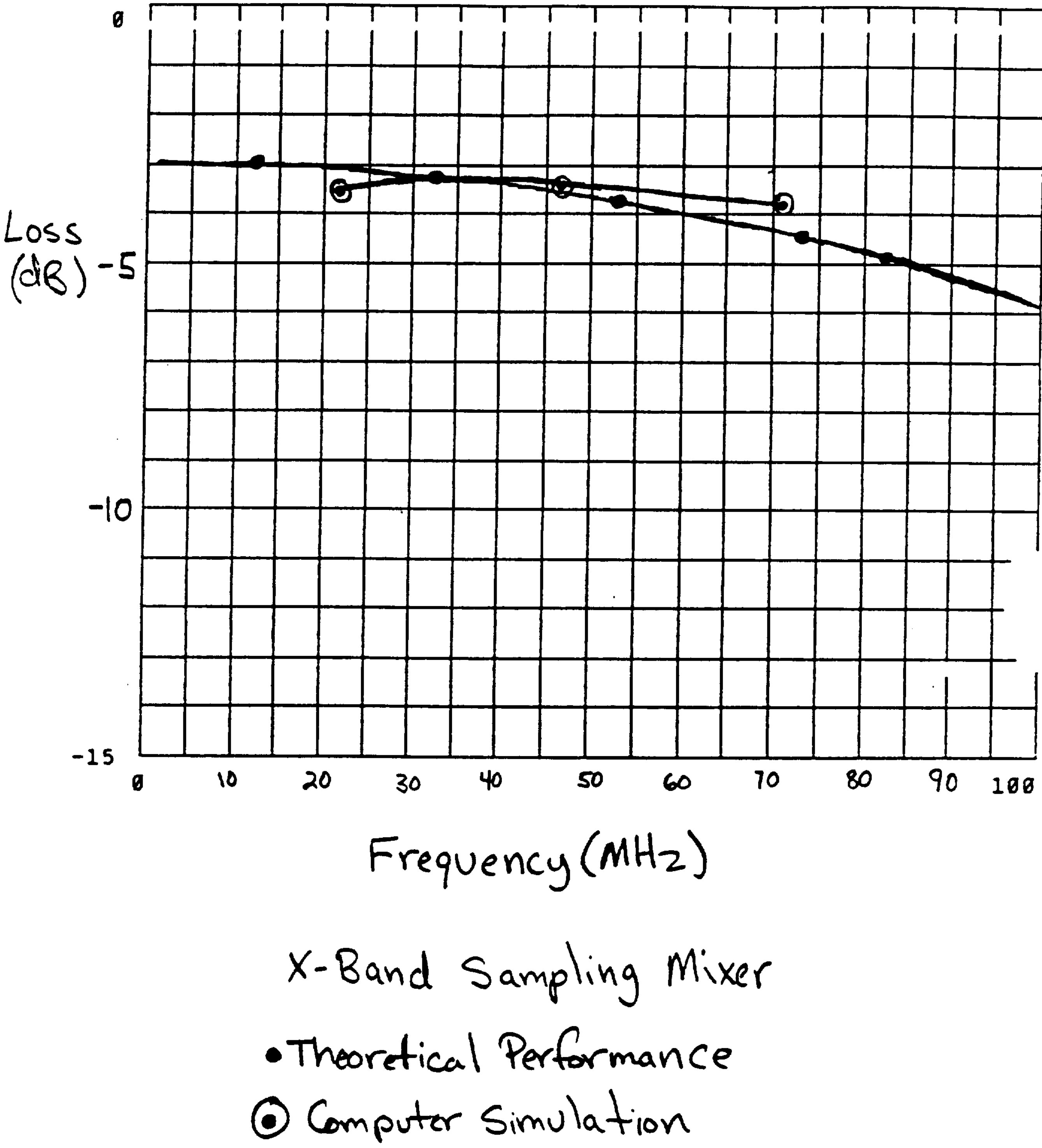


Fig. 17 X-Band Circuit Performance

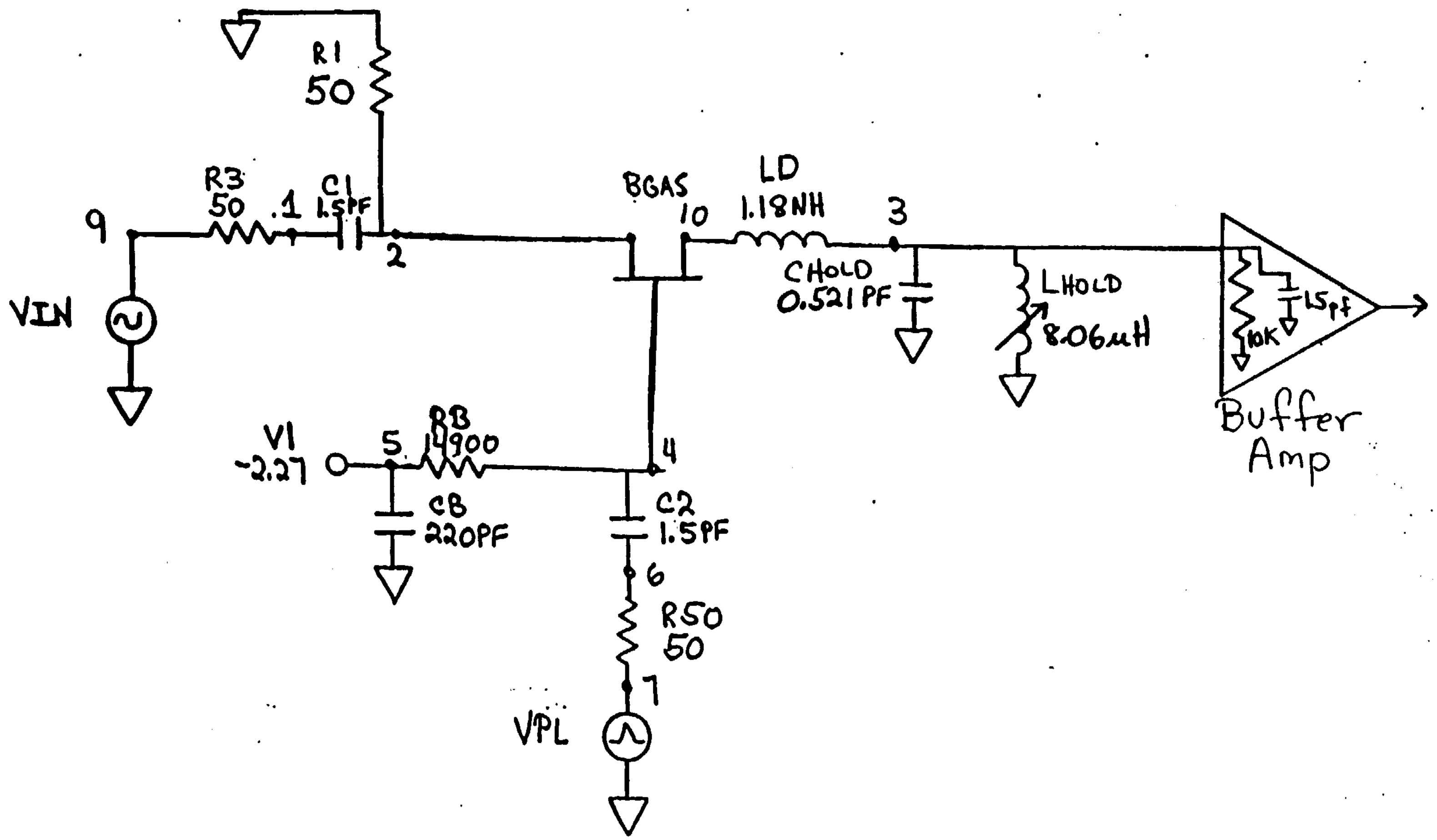


Fig. 18 Bandpass X-Band Sampling Mixer

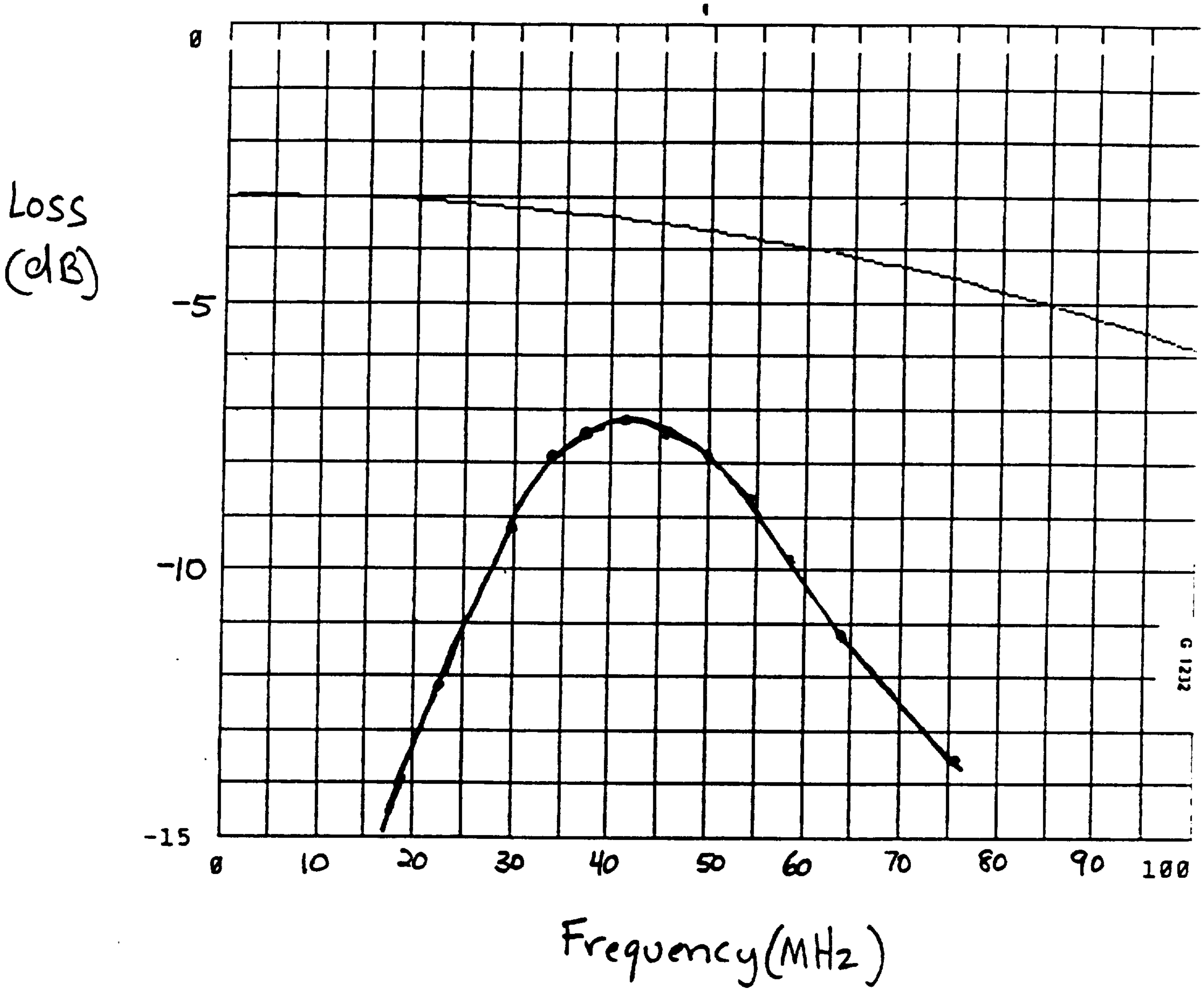


Fig. 19   Bandpass X-Band Sampling Mixer Performance

BIBLIOGRAPHY

[1] W.D. Faulkner and E.V. Mestre, "Subharmonic Sampling for the Measurement of Short Term Stability of MicrowaveOscillators," IEEE Trans. Instrumentation and Measurement, Vol. IM-32, no. 1, March 1983.

[2] M. Schwarz, W.R. Bennett and S. Stein, Communication Systems and Techniques, McGraw Hill, 1966.

[3] W.M. Grove, "Sampling for Oscilloscopes and other RF Systems: Dc Through X-Band," IEEE Trans. Microwave Theory Tech., Vol.MTT-14, no. 12, Dec. 1966.

[4] T. Takano et. al., "Novel GaAs FET Phase Detector Operable to Ka-Band," IEEE MTT-S Digest, 1984.

[5] W.R. Curtice, "A MESFET Model For Use in the Design of GaAs Integrated Circuits", IEEE Trans. Microwave Theory Tech., Vol. MTT-28, 448-456(1980).

[6] A. Ouslimani et. al., "Large-Signal Model of Picosecond FET's and Measurement of the Step Response", IEEE Trans. Microwave Theory Tech. Vol 37, no. 9, Sept. 1989.

[7] C. Kermarrec et. al., "Accurately Model Unbiased FETs for Monolithic Switches", Microwaves and RF, June 1989.

APPENDIX A

Simulation Source Code

```
 **********************************************************************
   This program simulates the JPL advanced transponder.              *
 * It was written for the Microsoft C compiler, version 5.0.          *
 *                                                                    *
 *  Author: Mark Frank.                                               *
 *                                                                    *
 *  File: sim_bpd.c                                                   *
 *                                                                    *
 *  Revision History:                                                 *
 *    1)  9/30/89 - Program started                                   *
 *    2)  10/11/89 -                                                  *
 *        a) Added logical, no_mod to ensure modulation is zero       *
 *           when mod indices are small (i.e., zero).                 *
 *        b) Take the accumulated phases modulo two*pi every jcount to *
 *           ensure accumulated phases do not overflow.               *
 *    3)  10/17/89 -                                                  *
 *        a) Make prnt_data an input variable                         *
 *    4)  10/18/89 -                                                  *
 *        a) Make statistic counters long integers                    *
 *        b) Corrected error in input SNR calculation                 *
 *    5)  10/25/89 -                                                  *
 *        a) Make # of DAC bits an input variable                     *
 *    6)  10/26/89 -                                                  *
 *        a) Allow the sign of ranging filter output to be used       *
 *           as ranging output - user selectable.                     *
 *    7)  10/30/89 -                                                  *
 *        a) Corrected error made in revision 5)                      *
 *    8)  11/15/89 -                                                  *
 *        a) Redirect stderr error messages to 'errors.dat'           *
 *                                                                    *
 *  Functions called:                                                 *
 *    pll_params()  : finds PLL parameters                            *
 *    sig_noise()   : finds signal amplitude for given SNR and BW     *
 *    bpd_sig()     : finds signal level out of 1 bit sampler         *
 *    nrand()       : returns zero mean, sigma=1, normal distributed r.v.*
 *    iir_filt()    : returns recursively filtered signal             *
 *    input_dbl()   : user input function returns double              *
 *    input_lng()   : user input function returns long                *
 *    int_pow()     : returns integer to integer power                *
 *    get_dbl_spc(): returns pointer to block of memory for double array*
 *    snr_calc()    : calculates snr from mean and variance           *
 *    savdbl()      : saves double array to disk to be read by MATLAB *
 *                                                                    *
 *  Macros called:                                                    *
 *    SGN(a)            : returns -1 if a<0, returns 1 if a>0         *
 **********************************************************************/

/****************************************
 * Include files:                       *
 ****************************************/
#include <constant.h>                    /* contains constants, eg, PI */
#include <math.h>                        /* Microsoft math functions   */
#include <stdio.h>                       /* Microsoft i/o functions    */
#include <time.h>                        /* Microsoft time of day fns. */
#include <process.h>                     /* Microsoft stream process fns */
#include <macros.h>                      /* Macro definitions          */
#include <gps_tap.h>                     /* PN generator definitions   */


/****************************************
```

```
 * Program control constants:          *
 ***************************************/
#define DAC_FILT_ON 1                     /* 1 for DAC filter                 */


/***************************************
 * Frequency constants:                *
 ***************************************/
#define F_1             9.56558642e06   /* Receiver assigned channel freq */
#define CAR_MULT        5               /* Carrier freq = CAR_MULT*F_1    */
#define VCO_MULT        4               /* VCO freq = VCO_MULT*F_1        */
#define F_0             (CAR_MULT*F_1)  /* Carrier frequency              */
#define F_VCO           (VCO_MULT*F_1)  /* Nominal VCO frequency          */
#define ICOUNT          4               /* Simul sampling to actual sampling */
#define T_I             (1./(ICOUNT*F_VCO))/* Initial Simulation time step */
#define F_C             16.e03          /* Command frequency              */
#define F_R             1.e06           /* Ranging frequency              */
#define IF_BAND         (ICOUNT*F_VCO)/2 /* IF bandwidth in Hz, (i.e., the */
                                        /* Nyquist rate)                  */


/***************************************
 * DDP constants:                      *
 ***************************************/
#define AGC_REF         10000           /* AGC reference value            */
#define K1_SHIFT        10              /* # of bits to shift K1 output   */
#define INT_SHIFT       10              /* # of bits to shift interpolator*/
                                        /* output                         */
#define INT_MASK        1023            /* Mask for lower INT_SHIFT bits  */


/*****************************************
 * Filter constants:                     *
 * Note: butterworth filter coefficients *
 *       obtained from MATLAB:           *
 *        [b,a] = butter(2,[w1 w2]);     *
 *        where w1 = (F_0-BW_IF/2)/F_I/2 *
 *              w2 = (F_0+BW_IF/2)/F_I/2 *
 *****************************************/
#define NP_IF           2               /* # of poles in IF BPF           */
#define NP_IF2          4               /* 2*NP_IF                        */
#define BW_IF           5.e06           /* two-sided IF filter bandwidth  */
double a_if[2*NP_IF+1] = {1.,
                          1.4277885823307,
                          2.2233519046009,
                          1.2339387148671,
                          0.7480557307791};
double b_if[2*NP_IF+1] = {0.00917437395786,
                          0.,
                         -0.01834874791572,
                          0.,
                          0.00917437395786};
double xs[2*NP_IF]     = {0., 0., 0., 0.};
double ys[2*NP_IF]     = {0., 0., 0., 0.};

/***************************************
 * Command LPF:                        *
 ***************************************/
#define NP_CMD          4               /* # of poles in command BPF      */
#define BW_CMD          2.00e03         /* two-sided CMD filter bandwidth */
```

```
double a_cmd[NP_CMD+1] = {1.,
                          -3.78032699055480,
                           5.36478799712946,
                          -3.38711134961993,
                           0.80269521883904};
double b_cmd[NP_CMD+1] = {0.02804737110451e-4,
                          0.11218948442249e-4,
                          0.16828422661597e-4,
                          0.11218948442693e-4,
                          0.02804737110007e-4};
double xc[NP_CMD]      = {0., 0., 0., 0.};
double yc[NP_CMD]      = {0., 0., 0., 0.};

/****************************************
 * Ranging LPF:                         *
 ****************************************/
#define NP_RAN          4               /* # of poles in Ranging LPF    */
#define BW_RAN          20e03           /* Two-sided Ranging filter bw */
double a_ran[NP_RAN+1] = {1.,
                         -3.93134345850892,
                          5.79637742149559,
                         -3.79867774172945,
                          0.93364423931869};
double b_ran[NP_RAN+1] = {0.02878599469902e-6,
                          0.11514397835199e-6,
                          0.17271596952639e-6,
                          0.11514397790791e-6,
                          0.02878599503209e-6};
double xr[NP_RAN]      = {0., 0., 0., 0.};
double yr[NP_RAN]      = {0., 0., 0., 0.};

/****************************************
 * DAC LPF:                             *
 ****************************************/
#define NP_DAC          2               /* # of poles in interp DAC LPF  */
#define BW_DAC          100.e03         /* One-sided DAC filter bandwidth */
double a_dac[NP_DAC+1] = {1.,          -1.814679857, 0.830452347};
double b_dac[NP_DAC+1] = {0.0039431226, 0.007886245, 0.003943123};
double xd[NP_DAC]      = {0., 0.};
double yd[NP_DAC]      = {0., 0.};


/****************************************
 * Global variables:                    *
 ****************************************/
FILE *output,*inf,*errstream;           /* output and input files       */
extern long seed;                       /* random number generator seed */



/**************************************************************************
 *                                                                        *
 *                          MAIN PROGRAM                                  *
 *                                                                        *
 **************************************************************************/
main(argc,argv)
char *argv[];                           /* argv[0] = output data file */
int  argc;                              /* argv[1] = input data file  */
{
  time_t ltime;                             /* Current time            */

 /*************************************
  * Define variables:                 *
  *************************************/
   int    i,n,l,m,j,k,ii;                   /* Loop counters                  */
   int    ncount,lcount,mcount,jcount,kcount; /* Accumulator count values */
   int    start_stat;                       /* take statistics at this point */
   int    adc,samp_cnt;                     /* a/d output, mux counter       */
   int    k0_sel,k1_sel;                    /* Carrier loop gain selects     */
   int    dac_bits;                         /* # of D/A bits                 */
 /* Logicals: */
   int    print_n;                          /* 1 to print after ncount, else jcount*
   int    rd_file;                          /* 1 to read input from disk     */
   int    prnt_data;                        /* 1 to write de-bug data to disk */
   int    range_msb;                        /* 1 to output MSB of range only */
   int    interp_on;                        /* 1 for DAC interpolator        */
   int    agc_on;                           /* 1 for AGC circuitry           */
   int    no_mod;                           /* 1 if no phase modulation      */
   int    step_phase,step_freq;             /* 1 to put in phase/freq step   */
   int    stepped;                          /* phase got stepped             */
   int    q_sign;                           /* 1 for feed back + acc_n_q     */
                                            /* 0 for feed back - acc_n_q     */

   long   acc_n_i,acc_n_q,acc_l_q;          /* Accumulated values            */
   long   acc_j_q,agc_acc,car_spe_acc,int_acc;
   long   cad_out;                          7* Coherent Amp. Detect output   */
   long   car_spe;                          /* Loop error                    */
   long   dac_in;                           /* DAC input value               */
   long   num_cad_samp,num_cmd_samp;        /* Number of statistic samples   */
   long   num_range_samp,num_phase_samp;
   long   ltemp;                            /* Temporary variable            */

   double sig,psn0_db,if_snr,if_band;       /* Input signal, SNRs            */
   double phase_cmd,phase_range;            /* Accumulated signal phases     */
   double phase_car,phase_tot,phase_vco;
   double ref_cmd_phase,ref_range_phase;    /* Output mixers' phases         */
   double range_ref,cmd_ref;
   double ct,rt;                            /* Command and ranging subcarrier */
   double beta_c,beta_r;                    /* Modulation indices            */
   double omeg_cmd,omeg_range,omeg_car;     /* Signal frequencies in radians */
   double f_vco,f_vco_off;                  /* VCO frequencies in Hz         */
   double t_i;                              /* Simulation time step          */
   double omeg_cmd_t,omeg_range_t;          /* Signal frequencies * (time step) */
   double omeg_car_t;
   double f_d;                              /* Doppler frequency             */
   double ts_car,wn_car,zeta_car,bl_car;    /* Carrier loop constants        */
   double dac_gain,vco_gain,gain;           /* Carrier loop gain constants   */
   double det_gain,k0_gain,k1_gain,int_gain;
   double dac_out;                          /* DAC output value              */
 /* Statistical variables: */
   double range_mix,cmd_mix;                /* Range and command mixer output*/
   double mean_cad,mean_quad,mean_cmd;      /* Output mean values            */
   double mean_range,mean_phase;
   double var_cad,var_quad,var_cmd,var_range; /* Outuput variances           */
   double var_phase;
   double std_cad,std_quad,std_cmd,std_range; /* Output std deviations       */
   double std_phase;
   double snr_cad,snr_cmd,snr_range;        /* Output SNRs                   */
   double snr_phase;
  double dtemp;


/***************************************
 * Functions (defined above):          *
 ***************************************/
  void   pll_params();
  void   snr_calc(),savdbl();
  long   input_lng();
  long   int_pow();
  double nrand(),iir_filt(),input_dbl();
  double bpd_sig(),sig_noise();
  double *get_dbl_spc();


/***************************************
 * Arrays:                             *
 ***************************************/
  double *temp0,*temp1;

/***************************************
 * Redirect error messages to disk:    *
 ***************************************/
  errstream = freopen("errors.dat","w",stderr);


/***************************************
 * Check location of input and output *
 *  files:                             *
 ***************************************/
  if(argc == 1)
  {
    output   = fopen("jpl_sim.dat","w");
    inf      = stdin;
    rd_file = input_lng(inf,"Get input parameters from disk(1/0) : ");
    if(rd_file)
      inf = fopen("jpl_in0.dat","r");
    else
      inf = stdin;
  }
  else
  {
    output = fopen(argv[1],"w");
    if(argc < 3)
    {
      rd_file = 0;
      inf     = stdin;
    }
    else
    {
      rd_file = 1;
      inf     = fopen(argv[2],"r");
      if(inf == NULL)
      {
        printf("Input file does not exit\n");
        exit(1);
      }
    }
  }
  ***************************************
  * Input simulation parameters:        *
  ***************************************/
   psn0_db        = input_dbl(inf,"Signal-to-noise density in dB          : ");
   seed           = input_lng(inf,"Random number generator seed           : ");
   phase_car      = input_dbl(inf,"Initial carrier phase in radians       : ");
   beta_c         = input_dbl(inf,"Command modulation index in radians    : ");
   ref_cmd_phase  = input_dbl(inf,"Reference command phase in radians     : ");
   beta_r         = input_dbl(inf,"Range modulation index in radians      : ");
   ref_range_phase = input_dbl(inf,"Reference range phase in radians      : ");
   f_d            = input_dbl(inf,"Doppler frequency in kHz               : ");
     f_d *= KHZ;
   ncount         = input_lng(inf,"NCOUNT (I and Q accumulators)          : ");
   lcount         = input_lng(inf,"LCOUNT (ranging accumulator)           : ");
   mcount         = input_lng(inf,"MCOUNT (interpolator divisor)          : ");
   jcount         = input_lng(inf,"JCOUNT (I accumulator)                 : ");
   kcount         = input_lng(inf,"Total number of jcount samples         : ");
   start_stat     = input_lng(inf,"Statistic start (jcount samples)       : ");
   step_phase     = input_lng(inf,"Step input phase    (1 = T, 0 = F)     : ");
   interp_on      = input_lng(inf,"Interpolator on     (1 = T, 0 = F)     : ");
   agc_on         = input_lng(inf,"AGC circuits on     (1 = T, 0 = F)     : ");
   range_msb      = input_lng(inf,"Ranging MSB only    (1 = T, 0 = F)     : ");
   print_n        = input_lng(inf,"Print after ncount  (1 = T, 0 = F)     : ");
   prnt_data      = input_lng(inf,"Print phase step    (1 = T, 0 = F)     : ");
   q_sign         = input_lng(inf,"Sign of feedback (q_sign: 1,0)         : ");
   k0_sel         = input_lng(inf,"Carrier loop gain, k0                  : ");
   k1_sel         = input_lng(inf,"Carrier loop gain, k1                  : ");
   dac_bits       = input_lng(inf,"# of D/A bits                          : ");
   vco_gain       = input_dbl(inf,"VCO gain constant rad/s/volt           : ");

/***************************************
 * Get space for arrays:               *
 ***************************************/
  if(prnt_data)
  {
    if(print_n)
    {
      temp0 = get_dbl_spc(jcount*kcount);
      temp1 = get_dbl_spc(jcount*kcount);
    }
    else
    {
      temp0 = get_dbl_spc(kcount);
      temp1 = get_dbl_spc(kcount);
    }
    if(temp1 == NULL)
      exit(1);
  }


/***************************************
 * Calculate Input signal level:       *
 ***************************************/
  if_band = IF_BAND;
  if_snr  = sig_noise(psn0_db,if_band);


/***************************************
 * Calculate DAC constants:            *
```

```
  ***************************************/
  dac_gain = 1./int_pow(2,(dac_bits-1));


 /****************************************
  * Calculate PLL Parameters:             *
  * Notes:                                *
  *   1) Detector gain = sqrt(2*P) = A;   *
  *   2) Loop gain = (detector gain)*     *
  *      (vco_gain)*(dac_gain)*           *
  *      (interpolator gain)*             *
  *      (accumulator gain)*(shift gain)  *
  *   3) If k0_sel or k1_sel < 0          *
  *      -> divide by these gains         *
  ****************************************/
  ts_car   = ncount/F_1/2;
  if(agc_on)
    det_gain = (double)AGC_REF/jcount;
  else
  {
    det_gain = bpd_sig(psn0_db,if_band,BW_IF/2.,1);
    det_gain *= ncount;
  }
  k0_gain  = (double)k0_sel;
  if(k1_sel > 0)
    k1_gain  = (double)k1_sel/int_pow(2,K1_SHIFT);
  else
    k1_gain  = (double)k1_sel*int_pow(2,K1_SHIFT);
  int_gain   = 1/((double)int_pow(2,INT_SHIFT));
  if(interp_on)
    gain     = det_gain*vco_gain*dac_gain*int_gain;
  else
  {
    vco_gain *= int_gain;
    gain     = det_gain*vco_gain*dac_gain;
  }
  pll_params(gain,k0_gain,k1_gain,ts_car,&wn_car,\
             &zeta_car,&bl_car);


/***************************************
 * Output simulation parameters:      *
 ***************************************/
  time(&ltime);
  fprintf(output,"The time is                              : %s\n",ctime(&ltime));
  fprintf(output,"         Input Simulation Parameters\n");
  fprintf(output,"Signal-to-noise density in dB           : %f\n",psn0_db);
  fprintf(output,"Random number generator seed            : %ld\n",seed);
  fprintf(output,"Initial carrier phase in radians        : %f\n",phase_car);
  fprintf(output,"Command modulation index in radians: %f\n",beta_c);
  fprintf(output,"Reference command phase in radians      : %f\n",ref_cmd_phase);
  fprintf(output,"Range modulation index in radians       : %f\n",beta_r);
  fprintf(output,"Reference range phase in radians        : %f\n",ref_range_phase);
  fprintf(output,"Doppler frequency in Hz                 : %f\n",f_d);
  fprintf(output,"NCOUNT (I and Q accumulators)           : %d\n",ncount);
  fprintf(output,"LCOUNT (ranging accumulator)            : %d\n",lcount);
  fprintf(output,"MCOUNT (interpolator divisor)           : %d\n",mcount);
  fprintf(output,"JCOUNT (I accumulator)                  : %d\n",jcount);
  fprintf(output,"Total number of jcount samples          : %d\n",kcount);
  fprintf(output,"Statistic start (jcount samples)        : %d\n",start_stat);
  fprintf(output,"Phase step input   (1 = True)       : %d\n",step_phase);
  fprintf(output,"Interpolator on    (1 = True)       : %d\n",interp_on);
  fprintf(output,"AGC circuits on    (1 = True)       : %d\n",agc_on);
  fprintf(output,"Ranging MSB only   (1 = True)       : %d\n",range_msb);
  fprintf(output,"Print after ncount (1 = True)       : %d\n",print_n);
  fprintf(output,"Print phase step   (1 = True)       : %d\n",prnt_data);
  fprintf(output,"Sign of feedback   (q_sign: 1,0)    : %d\n",q_sign);
  fprintf(output,"Carrier loop gain, k0_sel           : %d\n",k0_sel);
  fprintf(output,"Carrier loop gain, k1_sel           : %d\n",k1_sel);
  fprintf(output,"# of D/A bits                       : %d\n",dac_bits);
  fprintf(output,"VCO gain constant rad/s/volt        : %f\n\n",vco_gain);

  fprintf(output,"         Secondary Parameters\n");
  fprintf(output,"Carrier loop parameters             : wn = %f, zeta   = %f\n",w
  fprintf(output,"                                      bl = %f, ts_car = %f\n",b
  fprintf(output,"                                      det gain  = %f\n",det_gai
  fprintf(output,"                                      loop gain = %f\n\n",gain)
  fprintf(output,"         Output Parameters\n\n");
  fprintf(output,"I\t Q\t AGC\t SPE\t f VCO\t\t cmd out\t range err \n");


/***************************************
 * Initialization:                     *
 ***************************************/
  t_i          = T_I;                            /* Simulation time step*/

  omeg_cmd     = TWOPI*F_C;                      /* Signal frequencies */
  omeg_range   = TWOPI*F_R;
  omeg_car     = TWOPI*(F_0+f_d);
  f_vco        = f_vco_off  = F_VCO;
  vco_gain     /= TWOPI;                         /* Change vco gain to   */
                                                 /* Hz per volt          */

  omeg_cmd_t   = omeg_cmd * t_i;                 /* Mult by time step to */
  omeg_range_t = omeg_range * t_i;               /* save on calculations */
  omeg_car_t   = omeg_car * t_i;

  no_mod       = (fabs(beta_c)<.01) && (fabs(beta_r)<.01);
  step_freq    = (f_d > 1.);

/***************************************
 * Init agc accumulator output near    *
 *  operating point:                   *
 ***************************************/
  agc_acc   = AGC_REF/jcount;
  dtemp     = ncount*bpd_sig(psn0_db,if_band,BW_IF/3.2,1);
  if( (int)dtemp > 0)
    agc_acc       /= (int)dtemp;


/***************************************
 * Zero out variables:                 *
 ***************************************/
  phase_cmd     = phase_range = cmd_mix = range_mix = 0.;
  samp_cnt      = 0;
  num_cad_samp  = num_cmd_samp = num_range_samp = num_phase_samp = 0;
  acc_n_i       = acc_n_q = cad_out = acc_j_q = acc_l_q = 0;
  mean_cad      = mean_quad = mean_cmd = mean_range = mean_phase = 0.;
  var_cad       = var_quad  = var_cmd  = var_range  = var_phase = 0.;
  car_spe       = car_spe_acc   = 0;
  stepped       = 0;

/****************************************
 * Data loop:                           *
 ****************************************/
  ii = m  = l  = 0;

  for(k=0; k<kcount; k++)                                /* kkkkkkkkkk */
  {
   for(j=0; j<jcount; j++)                             /* jjjjjjjjjj */
   {
    for(n=0; n<2*ncount; n++,m++)                    /* nlmnlmnlmn */
    {
      for(i=0; i<ICOUNT; i++)                      /* iiiiiiiiii */
      {

/****************************************
 * Generate input signal:               *
 ****************************************/
        if(no_mod)
          phase_tot = phase_car;
        else
        {
          ct        = cos(phase_cmd);              /* Command modulation */
          rt        = cos(phase_range);            /* Range modulation   */
          phase_tot = phase_car + beta_c*ct + beta_r*rt;
        }
        sig       = if_snr*cos(phase_tot);


/****************************************
 * Add noise:                           *
 ****************************************/
        sig += nrand();


/****************************************
 * Bandpass digital filter:             *
 ****************************************/
        sig = iir_filt(xs,ys,sig,a_if,b_if,NP_IF2);


/****************************************
 * Update accumulated phases:           *
 ****************************************/
        phase_cmd   += omeg_cmd_t;
        phase_range += omeg_range_t;
        phase_car   += omeg_car_t;


      } /* end loop on i */                       /* iiiiiiiiii */

/****************************************
 * 1 bit sample of signal:              *
 ****************************************/
      adc = SGN(sig);


/****************************************
 * Mux and invert sign:                 *
 * Note: if samp_cnt = 0,1  adc = adc *
 *         samp_cnt = 2,3  adc = -adc*
 **************************************/
      switch(samp_cnt)
      {
        case 0:
          acc_n_i += adc;
          samp_cnt = 1;
          break;
        case 1:
          acc_n_q += adc;
          acc_l_q += adc;
          l++;
          samp_cnt = 2;
          break;
        case 2:
          acc_n_i -= adc;
          samp_cnt = 3;
          break;
        case 3:
          acc_n_q -= adc;
          acc_l_q -= adc;
          l++;
          samp_cnt = 0;
          break;
        default:                             /* shouldn't get here */
          samp_cnt = 0;
          break;
      }


/***************************************
 * 1) Check on DAC interpolator:       *
 * 2) Update VCO output:               *
 **************************************/
      if(m >= mcount)
      {
        if(interp_on)
        {
          if(int_acc > 0)
            int_acc     = car_spe + (int_acc & INT_MASK);
          else
            int_acc     = car_spe - ((-int_acc & INT_MASK));
          dac_in        = SIGN_SHIFT_R(int_acc,INT_SHIFT);
        }
        else
        {
          int_acc     = car_spe;
          dac_in      = int_acc;
        }
        dac_out       = dac_gain * dac_in;
        if(DAC_FILT_ON)
          dac_out     = iir_filt(xd,yd,dac_out,a_dac,b_dac,NP_DAC);
        f_vco         = dac_out*vco_gain + f_vco_off;
        t_i           = 1./(ICOUNT*f_vco);           /* update sim time step */
        omeg_cmd_t    = omeg_cmd * t_i;              /* Mult by time step to */
        omeg_range_t  = omeg_range * t_i;            /* save on calculations */
        omeg_car_t    = omeg_car * t_i;

        m             = -1;                          /* reset m */
      }


/****************************************
 * 1) Check on ranging accumulator,     *
 * 2) Mix ranging with reference, LPF,  *
 * 3) Take ranging statistics:          *
 ****************************************/
       if(l > (lcount-1))
       {
          range_ref   = cos(phase_range + ref_range_phase);
          if(range_msb)
            acc_l_q   = SGN(acc_l_q);
          range_mix   = acc_l_q*range_ref;
          range_mix   = iir_filt(xr,yr,range_mix,a_ran,b_ran,NP_RAN);
          if(num_cad_samp > 0)                   /* start statistics ? */
          {                                      /* mix to b/b and LPF: */
            mean_range += range_mix;
            var_range  += range_mix*range_mix;
            num_range_samp++;
          }
          acc_l_q = 0;
          l = 0;                                    /* reset l */
       }

    } /* end loop on n */                           /* nlmnlmnlmn */


/***************************************
 * 1) Accumulate I and Q channels,     *
 * 2) AGC outputs if requested:        *
 ***************************************/
    if(agc_on)
    {
      cad_out += acc_n_i*agc_acc;
      acc_j_q += acc_n_q*agc_acc;
    }
    else
    {
      cad_out += acc_n_i;
      acc_j_q += acc_n_q;
    }


/****************************************
 * 1) Mix command with reference, LPF, *
 * 2) Take command statistics:         *
 ****************************************/
    cmd_ref = cos(phase_cmd + ref_cmd_phase);
    cmd_mix = acc_n_q*agc_acc*cmd_ref;
    cmd_mix = iir_filt(xc,yc,cmd_mix,a_cmd,b_cmd,NP_CMD);
    if(num_cad_samp > 0)                                /* Take statistics? */
    {
      mean_cmd += cmd_mix;
      var_cmd  += cmd_mix*cmd_mix;
      num_cmd_samp++;
    }


/***************************************
```

```
 * Take VCO phase statistics:          *
 ***************************************/
   phase_vco = fmod(phase_car,TWOPI);
   if(num_cad_samp > 0)                                /* Take statistics? */
   {
     mean_phase += phase_vco;
     var_phase  += phase_vco*phase_vco;
     num_phase_samp++;
   }


/***************************************
 * Output Simulation variables:        *
 ***************************************/
   if(print_n || (j == (jcount-2)) )
   {
     fprintf(output,"%ld\t %ld\t %ld\t ",acc_n_i,acc_n_q,agc_acc);
     fprintf(output,"%ld\t %f\t ",car_spe,phase_vco);
     fprintf(output,"%f\t %f\n",cmd_mix,range_mix);
     if(prnt_data)
     {
       if(step_phase||step_freq)
       {
         temp0[ii]   = phase_vco;
         temp1[ii++] = f_vco;
       }
       else
       {
         temp0[ii]   = acc_n_q;
         temp1[ii++] = phase_cmd;
       }
     }
   }


/***************************************
 * Carrier loop filter:                *
 * Note: If k1_sel or k0_sel < 0,      *
 *       divide by their magnitudes.   *
 * Note: q_sign selects the sign of    *
 *       the feedback.                 *
 ***************************************/
   if(k1_sel > 0)
     car_spe_acc += acc_n_q*k1_sel;
   else
     car_spe_acc -= acc_n_q/k1_sel;
   ltemp = SIGN_SHIFT_R(car_spe_acc,K1_SHIFT);
   if(k0_sel > 0)
     car_spe       = ltemp + acc_n_q*k0_sel;
   else
     car_spe       = ltemp - acc_n_q/k0_sel;

   if(!q_sign)
     car_spe = -car_spe;


/***************************************
 * Dump ncount accumulators:           *
 ***************************************/
   acc_n_i = acc_n_q = 0;
   } /* end loop on j */                                /* jjjjjjjjjj */

/***************************************
 * Print out to inform user of progress*
 ***************************************/
   printf("k = %d\n",k);


/**************************************
 * Take accumulated phases modulo     *
 * 2*pi to prevent overflow           *
 **************************************/
   phase_car   = fmod(phase_car,TWOPI);
   phase_cmd   = fmod(phase_cmd,TWOPI);
   phase_range = fmod(phase_range,TWOPI);


/**************************************
 * 1) Take CAD statistics,            *
 * 2) Take quadrature statistics      *
 * 3) Input phase step if requested:  *
 **************************************/
   if(k > start_stat)
   {
      mean_cad  += cad_out;
      var_cad   += cad_out*cad_out;
      mean_quad += acc_j_q;
      var_quad  += acc_j_q*acc_j_q;
      num_cad_samp++;
      if(step_phase && !stepped)
      {
        phase_car += PI/4;
        stepped    = 1;
      }
   }


/**************************************
 * Update AGC accumulator:            *
 **************************************/
   if(agc_on)
     agc_acc += AGC_REF - cad_out;


/**************************************
 * Dump jcount accumulators:          *
 **************************************/
   cad_out =  acc_j_q  =  0;


  } /* end loop on k */                                   /* kkkkkkkkkk */


/**************************************
 * Compute SNRs:                      *
 **************************************/
  if(num_cad_samp>1)
  {
    snr_calc(mean_cad,var_cad,num_cad_samp,&mean_cad,&std_cad,&snr_cad);

    snr_calc(mean_quad,var_quad,num_cad_samp,&mean_quad,&std_quad,&dtemp);

    snr_calc(mean_phase,var_phase,num_phase_samp,&mean_phase,&std_phase, \
             &snr_phase);

    snr_calc(mean_cmd,var_cmd,num_cmd_samp,&mean_cmd,&std_cmd,&snr_cmd);

    snr_calc(mean_range,var_range,num_range_samp,&mean_range,&std_range, \
             &snr_range);
  }

/**************************************
 * Output results to disk:            *
 **************************************/
  fprintf(output,"\n");
  time(&ltime);
  fprintf(output,"The time is                         : %s\n",ctime(&ltime));

  fprintf(output,"Mean of CAD                         : %f\n",mean_cad);
  fprintf(output,"Standard deviation of CAD           : %f\n",std_cad);
  fprintf(output,"SNR of CAD                          : %f\n\n",snr_cad);

  fprintf(output,"Mean of quad                        : %f\n",mean_quad);
  fprintf(output,"Standard deviation of quad          : %f\n\n",std_quad);

  fprintf(output,"Mean of Phase                       : %f\n",mean_phase);
  fprintf(output,"Standard deviation of Phase         : %f\n",std_phase);
  fprintf(output,"SNR of Phase                        : %f\n\n",snr_phase);

  fprintf(output,"Mean of CMD                         : %f\n",mean_cmd);
  fprintf(output,"Standard deviation of CMD           : %f\n",std_cmd);
  fprintf(output,"SNR of CMD                          : %f\n\n",snr_cmd);

  fprintf(output,"Mean of ranging                     : %f\n",mean_range);
  fprintf(output,"Standard deviation of ranging       : %f\n",std_range);
  fprintf(output,"SNR of ranging                      : %f\n",snr_range);


/**************************************
 * Close read and write files         *
 **************************************/
  if(rd_file)
    fclose(inf);
  fclose(output);

} /* end main */
/**********************************************************************
 *
 * long input_lng(inf,prompt)
 *
 *   file pointer input variables
 *   ----------------------------
 *   inf = pointer to input file
 *
 *   char input variables
 *   --------------------
 *   *prompt = pointer to prompt string
 *
 * This routine prompts the user for an long input variable
 ********************************************************************/
long input_lng(inf,prompt)
FILE *inf;
char *prompt;
{
   long response;
   float temp;

   printf("%s",prompt);
   fscanf(inf,"%f",&temp);

   response = temp;
   printf("%ld\n",response);

   return response;
}
/**********************************************************************
 *
 * double input_dbl(inf,prompt)
 *
 *   file pointer input variables
 *   ----------------------------
 *   inf = pointer to input file
 *
 *   char input variables
 *   --------------------
 *   *prompt = pointer to prompt string
 *
 * This routine prompts the user for a double input variable
 ********************************************************************/
double input_dbl(inf,prompt)
FILE *inf;
char *prompt;
{
   double response;
   float temp;

   printf("%s",prompt);
   fscanf(inf,"%f",&temp);
   printf("%f\n",temp);

   response = temp;
   return response;
}
/************************************************************
 *
 *  double iir_filt(x,y,xin,a,b,n)
 *
 *    double input/output variables
 *    -----------------------------
 *    x[n] = input sequence
 *    y[n] = output sequence
 *
 *    double input variables
 *    ----------------------
 *    a[n+1] = denominator coefficients, a[0] = 1.
 *    b[n+1] = numerator coefficients
 *    xin    = current input sample
 *
 * This routine returns the current output of the IIR filter as
 * well as updating the arrays of old output and input values.
 ****************************************************************/
double iir_filt(x,y,xin,a,b,n)
double *x,*y,xin,*a,*b;
int n;
{
   double yout;
   int    i;

   yout = b[0]*xin;
   for(i=n; i>1 ;i--)
   {
     yout += b[i]*x[i-1] - a[i]*y[i-1];
     x[i-1] = x[i-2];
     y[i-1] = y[i-2];
   }
   yout += b[1]*x[0] - a[1]*y[0];
   x[0] = xin;
   y[0] = yout;

   return yout;
}
/*********************************************************************
 *
 *  void pll_params(k,k1,k2,t_s,omega_n,zeta,bl)
 *
 *    double input variables
 *    ----------------------
 *    k = detector gain*VCO gain
 *    k1 = loop filter proportional gain
 *    k2 = loop filter integrator gain
 *    t_s = sample time
 *
 *    double output variables
 *    -----------------------
 *    omega_n = closed loop natural frequency
 *    zeta    = closed loop damping constant
 *    bl      = closed loop one-sided noise bandwidth in Hz
 *
 *  This routine calculates the closed loop parameters of the digital
 *  phase-locked loop.  The loop parameters are based on D. Boman's
 *  thesis page 16.  The noise bandwidth is taken from R. Ziemer and
 *  R. Peterson, Digital Communications and Spread Spectrum Systems,
 *  Table 5-1.
 *******************************************************************/
void pll_params(k,k1,k2,t_s,omega_n,zeta,bl)
double k,k1,k2,t_s;
double *omega_n,*zeta,*bl;
{
   double temp;
   if(k1 < 0.)
     k1 = -1./k1;
   if(k2 < 0.)
     k2 = -1./k2;
   temp = k*k2;
   *omega_n = sqrt(temp/t_s);
   *zeta = k*k1/2./(*omega_n);
   temp = *zeta + 0.25/(*zeta);
   *bl  = 0.5*(*omega_n)*temp;

 /************************************************************
  *
  *   double sig_noise(snr_db,bw)
  *
  *     input double variables
  *     ----------------------
  *     snr_db = input signal/noise density in dB/Hz
  *     bw     = if bandwidth = Nyquist frequency in Hz
  *
  * This routine returns the signal level given the input SNR density.
  * It is assumed that the noise to be generated has
  *    sigma = N0*bw = 1
  * Also the input SNR is given by
  *    SNR  = A**2/(2*N0*bw)
  *
  ************************************************************/
double sig_noise(snr_db,bw)
double snr_db,bw;
{
  double dtemp,signal;

  dtemp = pow(10.,snr_db/10.);
  signal = sqrt(2.*dtemp/bw);

  return signal;
}
/************************************************************
 *
 *  void snr_calc(mean,var,n,&smean,&std,&snr)
 *
 *   double input variables
 *   ----------------------
 *   mean = sum of samples
 *   var  = sum of samples**2
 *
 *   long input variables
 *   --------------------
 *   n  = total # of samples
 *
 *   double output variables
 *   -----------------------
 *   *smean = sample mean
 *   *std   = standard deviation of sample mean
 *   *snr   = signal-to-noise in dB
 *
 *  This routine calculates the SNR using the sample mean.
 ************************************************************/
void snr_calc(mean,var,n,smean,std,snr)
double mean,var;
long   n;
double *smean,*std,*snr;
{
  double temp;

  temp    = n*var - mean*mean;
  *std    = sqrt(fabs(temp/n/(n-1)));
  *smean  = mean/n;
  if(*std > 0.)
```

```
     *snr  = 20.*(log10(fabs(*smean)/(*std)));
   else
     *snr  = 0.;
 }
 /***************************************************************
  *
  * double bpd_sig(psn0,if_band,bw,ideal)
  *
  *   double input variables
  *   ----------------------
  *   psn0 = input signal to noise density in dB-Hz
  *   if_band = one sided IF bandwidth in Hz
  *   bw   = one sided band pass pre-filter bandwidth in Hz
  *
  *   int input variables
  *   -------------------
  *   ideal = 1 for ideal LPF
  *         = 0 for one pole LPF
  *
  *  This routine calculates the signal level out of
  *  a hard limiter following a band pass filter.
  *  This routine assumes that n_0*if_band = 1;
 *******************************************************************/
double bpd_sig(psn0,if_band,bw,ideal)
double psn0,if_band,bw;
int    ideal;
{
  double s,n_0,sigma2,sigma,mean_bpd;
  double qx();

/*
 * Convert decibels to absolute values
 */
  psn0 /= 10.;
  psn0 = pow(10.,psn0);

/*
 * Find signal amplitude assuming n_0*if_band = 1:
 */
  s = 2.*psn0/if_band;
  s = sqrt(s);
  n_0 = 1./if_band;

/*
 * Find noise out of BPF:
 */
  if(ideal)
    sigma2 = n_0*bw*2.;
  else
    sigma2 = n_0*bw*PI;
  sigma = sqrt(sigma2);

/*
 * Now find mean signal value out of limiter
 */
  mean_bpd = 1. - 2.*qx(s/sigma);
  return  mean_bpd;

}
/***************************************************************
```

```
 *
 * double nrand()
 *
 *   implicit long input variables
 *   ----------------------------
 *   seed = initial seed
 *
 *  This routine returns a normally distributed random variable N(0,1)
 *  from the uniform random number generator urand(), using the
 *  the algorithm described in [FOR77] G.E. Forsythe, M.A. Malcolm,
 *  and C.B. Moler, Computer Methods for Mathematical Computations,
 *  Englewood Cliffs: Prentice-Hall, p. 247, 1977.
 *
 *****************************************************************/
double nrand()
{
  double u1,u2,v1,v2,s,ln_s;
  double sqrt_lns,x;
  double urand();

  s = 2.;
  while(s > 1.)
  {
    u1 = urand();
    u2 = urand();
    v1 = u1 + u1 - 1;
    v2 = u2 + u2 - 1;
    s  = v1*v1 + v2*v2;
  }
  ln_s = log(s);
  sqrt_lns = sqrt(-(ln_s+ln_s)/s);
  x = v1*sqrt_lns;
  return x;
}
/****************************************************************
 *
 * double urand()
 *
 *   implicit long input variables
 *   ----------------------------
 *   seed = initialize seed the first time this routine is
 *           called.
 *
 *  This routine generates a random number in [0,1] based on
 *  the algorithm given in [PAR88] S.K. Park and K.W. Miller,
 *  "Random numver generators: good ones are hard to find,"
 *  Comm. ACM, vol. 32, no. 10, pp. 1192-1201, Oct. 1988.
 *
 *****************************************************************/
#define A    16807
#define M    2147483647
#define Q    127773                    /* M div A */
#define R    2836                      /* m mod A */
double urand()
{
  long   lo,hi,test;
  double rand;

  hi   = seed/Q;
  lo   = seed%Q;
  test = A*lo - R*hi;

  if(test > 0)
    seed = test;
  else
    seed = test + M;
  rand = (double)seed/M;
  return rand;
}
/*******************************************************************
 *
 * double qx(x)
 *
 * double input variables
 * --------------------
 * x = real argument
 *
 * this routine finds the q function for a real argument, by
 * calling the routines qx0(small arg), qx1(moderate arg),
 * qx2(large arg).
 *
 ****************************************************************/
#define XLARGE 4
double qx(x)
double x;
{
    double y;
    double qx1(),qx2();

    int    neg;

    neg = (x < 0.);
    if (neg )
       x = -x;
    if (x < XLARGE)
      y = qx1(x);
    else
      y = qx2(x);
    if(neg)
      y = 1. - y;

    return y;
}
/*******************************************************************
 * double qx1(x)
 *
 * double input variables
 * --------------------
 * x = real argument
 *
 * this routine finds the q function for moderate values of x,
 * see: digital communications and spread spectrum systems, by
 * ziemer and peterson, p. 714.
 *
 ****************************************************************/
#define P  0.2316419
#define B1 0.319381530
#define B2 -0.356563782
#define B3 1.781477937
#define B4 -1.821255978
```

```
  define B5 1.330274429

 double qx1(x)
 double x;
 {
     double t,z,z2,x2,y;

     x2 = x*x;
     z = exp(-x2/2.)/SQRT2PI;
     t = 1./(1 + P*x);

     y = z*(t*(B1 + t*(B2 + t*(B3 + t*(B4 + t*B5)))));
     return y;
 }
 /*******************************************************************
  * double qx2(x)
  *
  * real input variables
  * --------------------
  * x = real argument
  *
  * this routine finds the q function for large values of x,
  * see: digital communications and spread spectrum systems, by
  * ziemer and peterson, p. 714.
  *
  ****************************************************************/
#define XL 10.
double qx2(x)
double x;
{
    double x2,z,y;

    if(x > XL)
      y = 0.;
    else
    {
      x2 = x*x;
      z = exp(-x2/2.)/SQRT2PI;

      y = z*(1.-1./x2 + 3./(x2*x2) - 15./(x2*x2*x2))/x;
    }
    return y;
}
/******************************************************************
 *   long int_pow(i,n)
 *
 *   int input variables
 *   ------------------
 *   i = argument
 *   n = power
 *
 *  This function returns i to the nth power.
 ****************************************************************/
long int_pow(i,n)
int i,n;
{
  long pow;
  pow = 1;
  while(n>0)
  {
    pow *= i;
    n--;
  }
  return pow;
}
/****************************************************************
    Header file that contains commonly used constants.

    4/14/89

PURPOSE:
    Defines commonly used constants, such as pi.

DEPENDENCIES:
    none

USAGE:
    invoke with preprocessor directive:
    #include "constant.h"
          or
    #include <constant.h>
    near the beginning of the program.

**************************************************************/
#define LINT_ARGS 1
#define UINT    unsigned int
#define ULONG   unsigned long
#define RAD_DEG 0.01745329                        /* PI/180.    */
#define PI      3.141592654
#define PI2     1.570796327                       /* PI/2       */
#define PI4     0.7853981635                      /* PI/4       */
#define PI34    2.356194491                       /* 3*PI/4     */
#define SQRTPI  1.772453851                       /* sqrt(pi)   */
#define SQRT2   1.414213562                       /* sqrt(2)    */
#define SQRT3   1.732050808                       /* sqrt(3)    */
#define SQRT2PI 2.506628275                       /* sqrt(2*pi) */
#define ONESQRT2 .7071067814                      /* 1/sqrt(2)  */
#define LOG2    .6931471806                       /* ln(2)      */
#define TWOPI   6.283185307                       /* 2*pi       */
#define TWODPI  .6366197723                       /* 2/pi       */
#define MHZ     1.e06
#define KHZ     1.e03
/************************************************************
    Header file having Macro definitions

    9/28/89

PURPOSE:
    Performs in-line a number of useful chores.

DEPENDENCIES:
    none

USAGE:
    invoke with preprocessor directive:
    #include "macros.h"
          or
    #include <macros.h>
    near the beginning of the program.
```

```
 *************************************************************/

/* signum functions */

#define SGN(a)    ( ((a)>=0) ? 1 : -1 )
#define SGN01(a)  ( ((a)>=0) ? 1 :  0 )


/* Signed shift      */

#define SIGN_SHIFT_R(IN,SHIFT) ( (IN >= 0) ? (IN >> SHIFT) : -((-IN >> SHIFT)) )

/* Inc modulo        */

#define INC_MOD(a,modlen)  a++; a%= modlen
```