AN ULTRA HIGH RESOLUTION FMCW RADAR

ALON BAS

BSc (ENGINEERING) UNIVERSITY OF CAPE TOWN

Thesis submitted to the Department of Electrical and Electronic Engineering of the University of Cape Town in partial fulfillment of the requirements for the Degree of MSc (Eng).

DECLARATION.

I declare that this dissertation is my own unaided work. It is being submitted in partial fulfillment of the requirements for the degree of Master of Science in Engineering at the University of Cape Town. It has not been submitted before for any degree or examination at any other university.

signature removed

(Signature of Candidate.)

15 day of July 1992.

SYNOPSIS.

There is a great need for real-time non-intrusive measurements in industry. A short-range radar system can be used to make these measurements. A standard requirement for these type of applications is high resolution. This is a standard problem in radar. Using classical signal processing techniques, the range resolution is proportional to the bandwidth of the transmitted signal. This poses a serious problem in radar as very large bandwidths are required - typically 150GHz for 1mm range resolution.

Alternative techniques have been sought which do not rely on large transmitted bandwidths, but which rely on large signal-to-noise ratio (SNR). Such techniques exist in modern spectral analysis eg. auto-regressive techniques. These techniques model the data. In other words, they assume a-priori information.

Linear frequency-modulated continuous-wave (FMCW) radar was utilized, since a pulsed radar would require very precise time measurements due to the short range (a few ns). The FMCW radar would have to be very linear for the modelling process to work properly. The frequency domain measurement of the received system data would then be proportional to range.

An FMCW radar system was built and tested. The modern signal processing techniques were found to work well when injected with sinusoidal signals from signal generators. The hardware was also found to perform satisfactorily. However, amplitude modulation was observed in the mixing process and subsequently, the modelling process did not perform satisfactorily when interfaced to the hardware. Due to the amplitude modulation problem, two closely-spaced targets disrupted the high resolution properties of the modelling process. Nevertheless, a single target could be resolved within a resolution bin of better than 1cm. A solution is proposed in chapter eight, however, it is out of the scope of this thesis.

ACKNOWLEDGEMENTS.

The author would like to thank Professor B.J. Downing for his guidance and support throughout this project. The author would also like to thank Professor M.R. Innes for his advice and help.

The financial support from the Council for Scientific and Industrial Research was greatly appreciated.

Finally, thanks to De Beers Diamond Research Laboratories for their financial support.

TABLE OF CONTENTS.

LIST OF ILLUSTRATIONS

FIGURES.

TABLES:

1. INTRODUCTION

Radars have traditionally been used for detecting targets at long ranges. This allowed large leeways in terms of system parameters and requirements. Due to the enormous need for short-range, non-intrusive radars in industry [1.1, 1.2, and 1.3] the topic of high-resolution radar was investigated.

Usually, short-range radar (around 1m) require measurements of closely spaced target returns (1mm to 1cm) [1.4]. This requirement for industrial radars creates serious resolution problems for traditional radar implementations. Using classical signal processing techniques, the resolution is proportional to the bandwidth of the transmitted signal [1.5, 1.6, etc]. Thus, for a 1mm resolution:

$$B = \frac{c}{2.dR} = 150GHz.$$

This is an enormous bandwidth.

1.1 RADAR SELECTION

There are many types of radars, but they can be categorized into two types: pulsed and continuous wave (CW) radars [1.7].

1.1.1 PULSED RADARS

Pulsed radars measure the time taken for the transmitted signal to return after being reflected by the target ("travel time"). For pulsed radars, the relative time between two target returns, 1mm apart are:

$$dT = \frac{2.dR}{c} = 6.67\ ps.$$

To resolve timings in the pico-second range is not technologically feasible, even with pulse compression techniques. This demonstrates the serious problems in resolving closely-spaced targets.

For pulsed radars, signal paths over the range of 1m, take:

$$T = \frac{2.R}{c} = 6.67 \text{ ns.}$$

To just accurately measure a single target over this range is extremely difficult, requiring very precise timing. The equipment required for accurate timing is very expensive and pulsed radars have high power requirements.

1.1.2 CW RADARS

CW radars can be categorized into frequency modulated continuous wave (FMCW) radars, Doppler radars, and multiple frequency continuous wave radars. The FMCW radars can be further classified by the different modulation types: sinusoidal and linear frequency modulation.

CW Doppler radars measure velocity, and are therefore unsuitable.

Multiple frequency CW radars measure the relative phases of the transmitted and return signals. More than one frequency is used to increase the unambiguous range. This technique classically also requires large transmitted bandwidths. The problem with this system is that an I and Q channel are required to measure the phase, making a working system expensive and difficult to achieve over such a wide bandwidth.

FMCW radars transmit continuously varying frequencies. The range is obtained by measuring the difference in frequency between the transmitted and received signals. There are similar problems to the above systems where wide bandwidths are required. However, with the FMCW system, the Q channel can be dispensed with which greatly simplifies the system. Also, there are no severe problems with measurement accuracy because the "travel time" of the FMCW system is scaled by the bandwidth (see the derivation in section 2.1). The disadvantages are that in

linear FMCW radar, the oscillator has to be very linear over the frequency sweep [1.8].

The low power consumption and cost effectiveness of CW systems make them more attractive over short-range than pulsed systems.

FMCW radar was chosen because of the above advantages over the other systems.

1.2 RADAR SIGNAL PROCESSING

Signal processing of radar returns is an integral part of radar systems. The classical method of getting the frequency domain response is to take a Fourier transform of the time domain response [1.7]. The Fourier transform limits the resolution due to the windowing problem. Subsequently, the resolution of the system becomes proportional to the bandwidth of the transmitted signal. As previously mentioned, a wide bandwidth is needed to obtain high target resolution, making this technique unviable.

Classically, to obtain wide bandwidths, a switched system of wideband oscillators is necessary, which is prohibitively expensive. Instead of using a wide bandwidth, it was decided to approach the problem directly by using modern signal-processing techniques [1.9 and 1.10] to obtain the frequency response rather than using the classical approach with its associated problems. A-priori information is assumed, thereby decreasing the number of degrees of freedom. A model is created which assumes a certain type of data eg. sinusoids in white noise, and the parameters are estimated. This technique does not have the windowing problems associated with the direct Fourier transform approach, and the resolution becomes dependent on the signal-to-noise ratio (SNR). Few data points are required, making the system very practical.

1.3 THESIS DEFINITION AND COURSE

The aim of this thesis is to investigate the topic of ultra high-resolution radar using the modern spectral estimators, and to develop an experimental, inexpensive ultra high-resolution radar. The system must operate in real-time ie. make an analysis in less than 5ms. The complete system can be seen in figure 1.1.

Fig. 1.1 Complete System.

The following chapter discusses the FMCW radar principals. The next chapter then describes the hardware design implementation of the FMCW radar. Classical and modern signal processing will then be presented. Thereafter, radar and signal processing simulations and results will be displayed. Hardware and signal processing software integration will be considered and finally, the results, conclusions and future recommendations will be discussed.

1.4 REFERENCES

[1.1] CLARRICOATS, P. **Portable Radar For The Detection Of Buried Objects.**, Radar Conference Paper 1977.

[1.2] YAMAURA, I. and HIDAKA, T. **The Double-Swept-Frequency Locating Reflectometer.**, TRANS. IEEE, MTT-23, (3), 1975, 316-317.

[1.3] DELAUDER, D., BALANIS, C. and KAZOVSKY, L. **Microcomputer-Based Signal Processor For Short-Range FM Radar.**, TRANS. IEEE, IM-35, (1), 1986, 70-77.

[1.4] DYBDAL, R.B., HURLBUT, K.H., and MORI, T.T. **High-Resolution Instrumentation Radar.**, TRANS. IEEE, IM-36, (1), 1987, 110-114.

[1.5] SKOLNIK, M. **Introduction To Radar Systems.** (2nd ed.)., McGraw-Hill, Japan.

[1.6] HYMANS, A.J., LAIT, J. **Analysis Of A Frequency-Modulated Continuous-Wave Ranging System.** PROC. IEE., VOL. 107B, 1960, 365-372.

[1.7] HOVANESSIAN, S.A. **Radar System Design And Analysis.** Artech, 1984.

[1.8] MENSA, D.L. **High Resolution Radar Imaging.** ARTECH, 1982.

[1.9] KAY, S.M. **Modern Spectral Estimation: Theory And Application.** PRENTICE-HALL, 1987.

[1.10] MARPLE, S.L. **Digital Spectral Analysis With Applications.** PRENTICE-HALL, 1987.

2. FMCW RADAR PRINCIPALS

2.1 THEORY OF OPERATION

The system is a development of the classical FMCW radar see figure 2.1. A tunable microwave oscillator is linearly frequency modulated. The signal is transmitted and returns after a period - the propagation delay. Whilst this occurs, the oscillator has changed its frequency. Since the oscillator is being linearly modulated, the change in frequency of the oscillator relative to the transmitted signal is proportional to the propagation delay. The propagation delay is then proportional to range - see figure 2.2. The relative frequency change is obtained by heterodyning a portion of the transmitted signal with the received signal [2.1]. The output frequency or beat frequency is derived as follows:

propagation delay:

$$\tau = \frac{2 * R}{c}$$

c = speed of light
$= 3.0 * 10^8$ [m/s]
R = range [m]

rate of change of frequency:

$$f = \frac{\delta f}{(T_{mod} / 2)}$$

$\delta f = f' - f''$
T_{mod} = modulation period
$f_{mod} = 1 / T_{mod}$

beat frequency:

$$f_b = f * \dot{\tau}$$

$$f_b = \frac{4 * R * \delta f}{c * T_{mod}}$$

$$f_b = \frac{4 * R * fmod * \delta f}{c} \quad \text{.... equation 2.1}$$

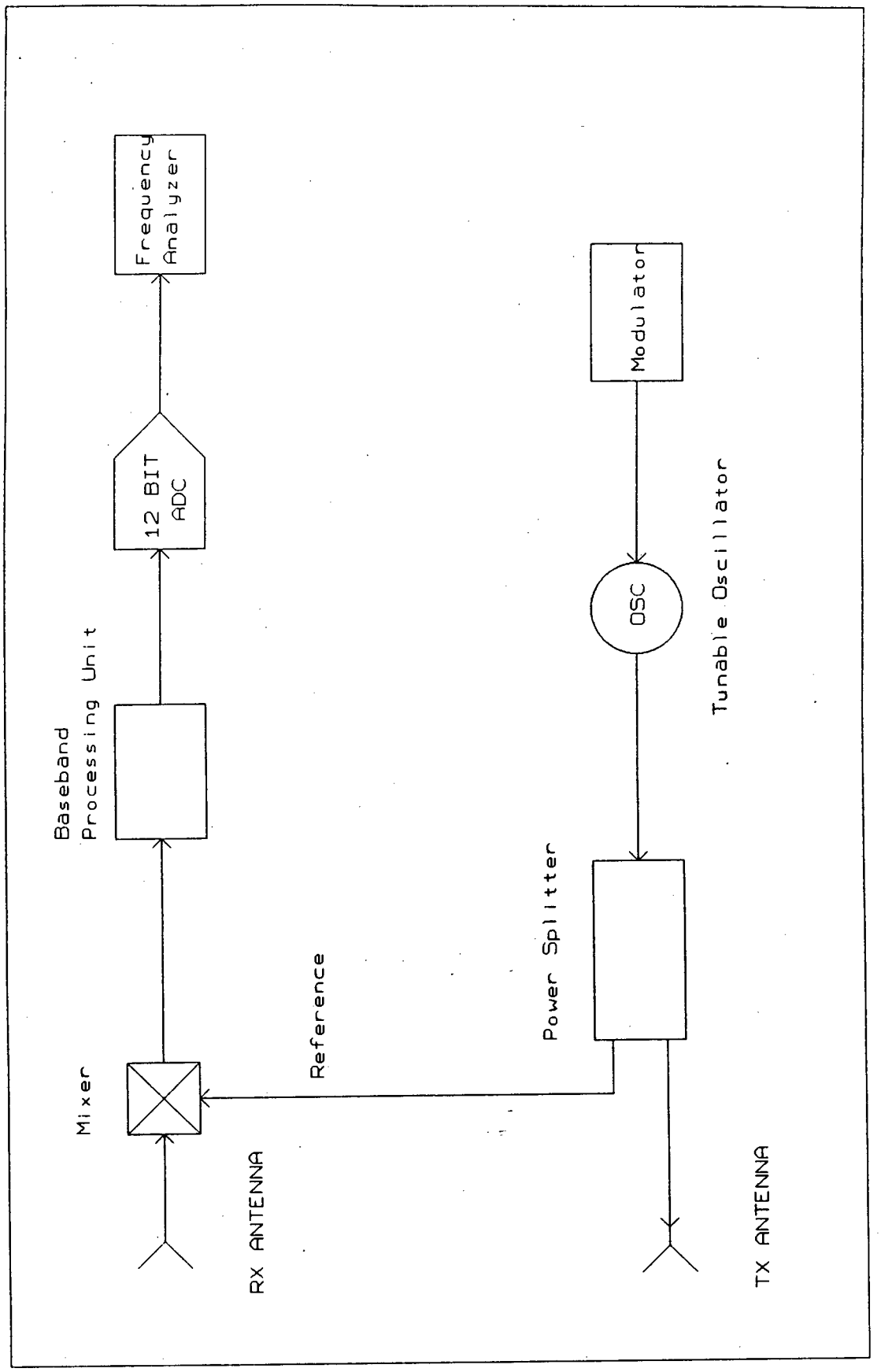
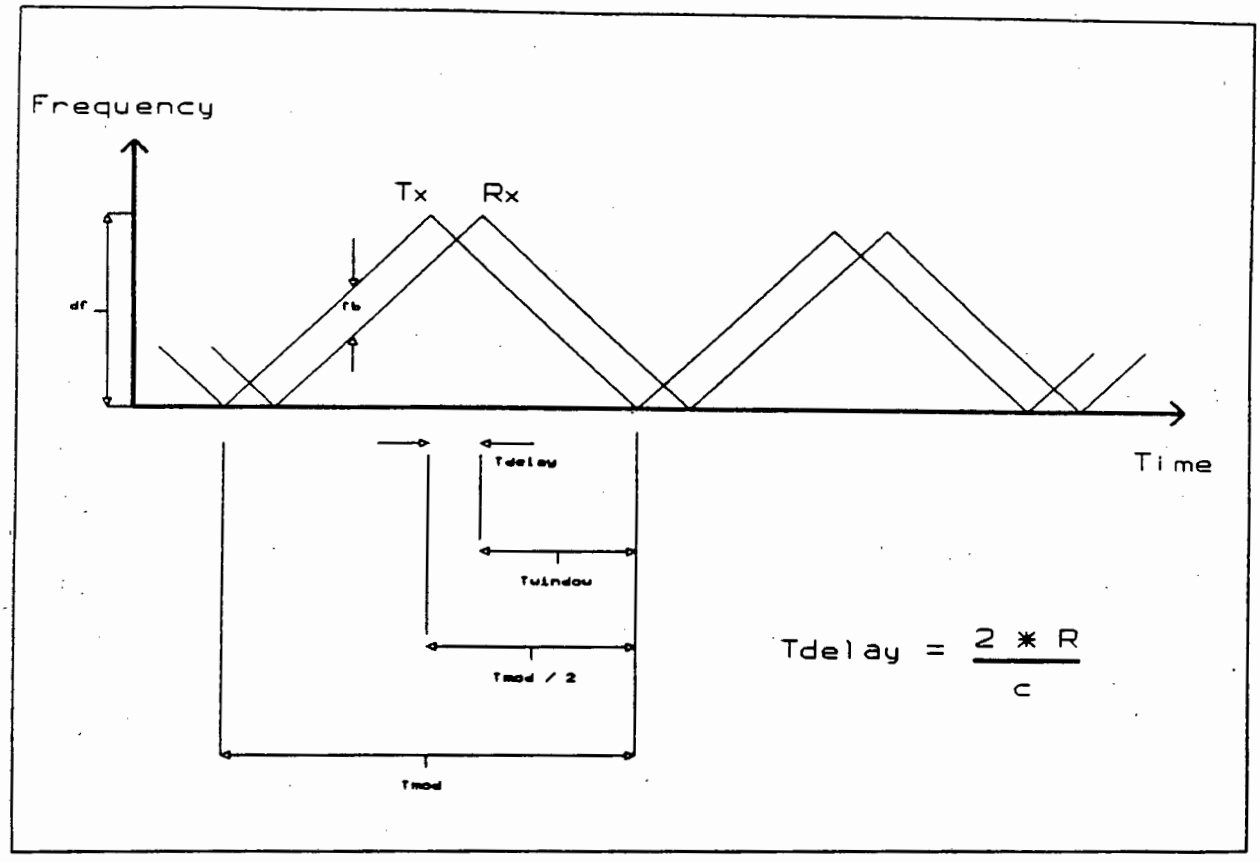


Fig. 2.1 Classical FMCW Radar.



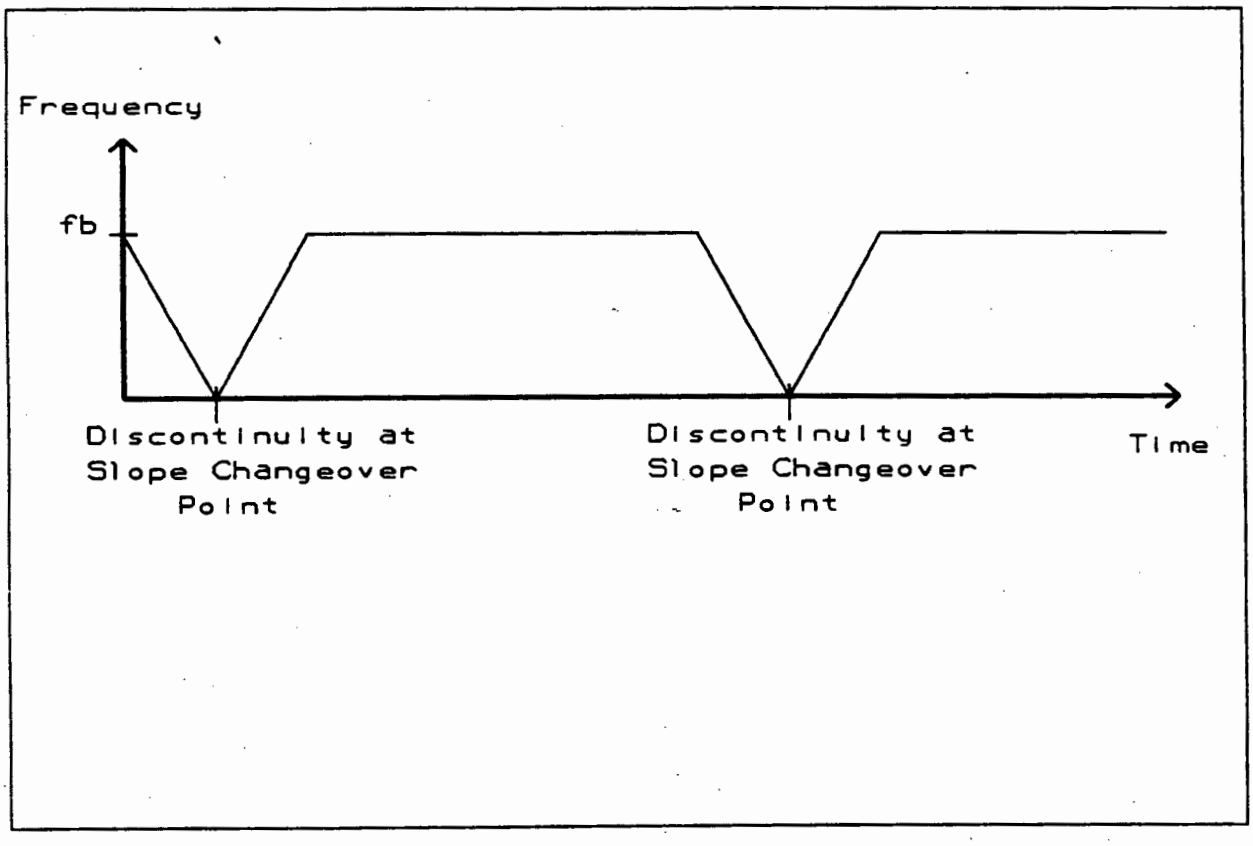


Fig. 2.2 FMCW Principals.

A more rigorous derivation is provided in Appendix A. An FFT of the transmitted waveform is shown in figure 2.3a. The simulation for it is available in Appendix B. The transmitted waveform can be seen to exhibit the Fresnel spectrum. An analytical derivation is available in [2.2]. Also, an FFT of the received beat frequency is shown in figure 2.3b. The simulation for it is available in Appendix C.

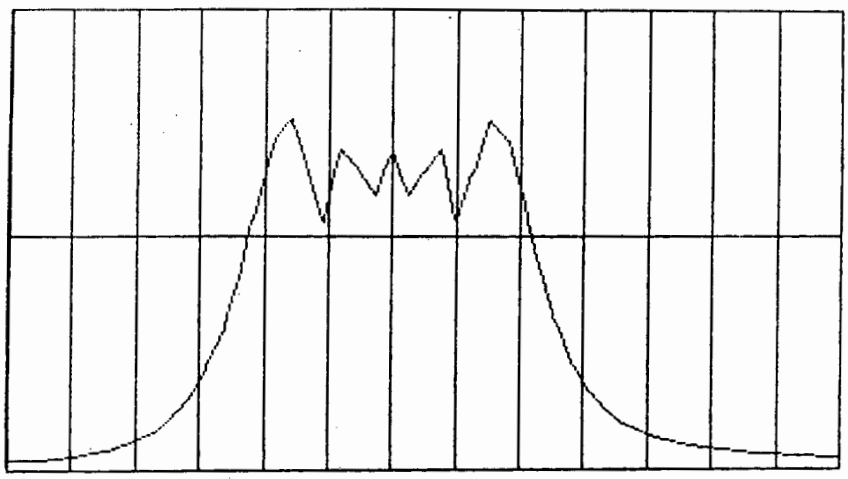


Fig. 2.3. (a) FFT of the Transmit Waveform

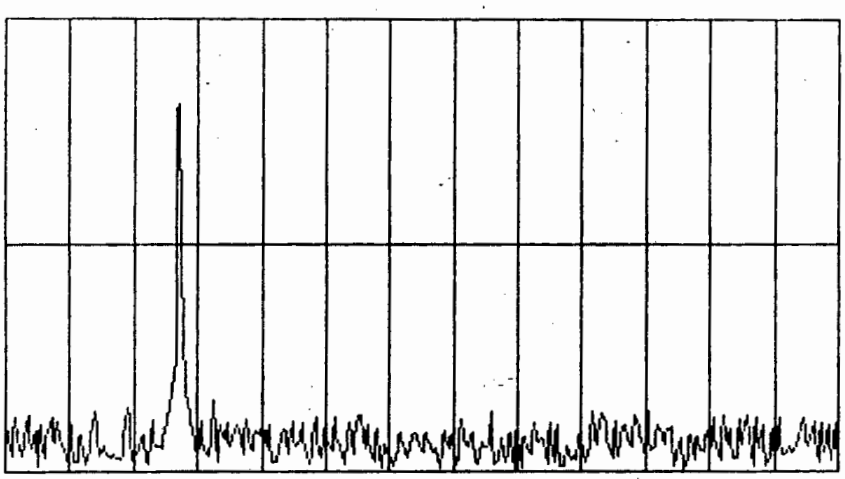


Fig. 2.3. (b) FFT of the Received Beat Frequency.

Since the oscillator has a limited tuning range, the modulation waveform has to have periodicity. Sinusoidal modulation was not chosen due to its need for averaging [2.1] which would degrade the high range resolution. Triangular modulation was chosen for its ease of linearization.

In choosing the system parameters, a practical modulation rate had to be taken into account. Three factors controlled the modulation rate:

a) The maximum modulation or sweep rate of the tunable oscillator:
Maximum modulation rates are:
Varactor tuned : 100MHz.
YIG : 5kHz.

b) The practical limitations on the modulator hardware:
(max. DAC speed : 1MHz).

c) The required beat frequency for the required range:
12 kHz for 2.0m.

The beat frequency range was chosen to start from 10kHz which is well away from the 1/f noise.

A suitable varactor tuned Gunn oscillator was available and the peak frequency deviation δf was chosen to be 500MHz. This was the maximum bandwidth of the tunable oscillator. It was chosen to give the maximum beat frequency per range mm. This oscillator had a poor tuning linearity and later a YIG tuned oscillator with a much more linear tuning characteristic was obtained and used.

The chosen parameters for the FMCW system were:

operating range :	2.0m to 2.5m.
f :	500 MHz.
f_{mod} :	0.9 kHz.
required f_b :	12 kHz to 15 kHz.

2.2 HARDWARE DESCRIPTION

The modulator generates a periodic triangular waveform and is fundamental to the FMCW radar as it provides the reference marks for the propagation delays. It controls the start frequency and the end frequency of the oscillator's sweep.

If the oscillator's frequency-voltage characteristic were linear, the modulator's output would be perfectly triangular. However, since this is not the case with practical tunable oscillators, the modulator has to have a waveform which takes into account the non-linear characteristic of the oscillator. This effectively linearizes the tunable oscillator. Together, the modulator and the tunable oscillator form a linear FMCW transmitter.

A portion of the transmitted signal is reflected off the target and is captured by the receive antenna. A power-splitter is used to couple the transmitter signal to be used as a reference for the received signal as shown in figure 2.1. The two signals are heterodyned using a balanced mixer, yielding a beat frequency at the output port of the mixer.

The higher order mixer products are filtered by the inherent low-pass filter characteristic of the balanced mixer. The received signal is very small, and subsequently, a large amount of amplification is needed.

More precise filtering is required to minimize the effects of leakage, multipath, unwanted clutter outside the required range, and the DC products. This is achieved with the baseband processing module.

The signal then goes to the signal processing block which consists of a PC with a high speed analogue to digital convertor (ADC). The transformation from the time domain to the frequency domain is done by software in the PC.

2.3 RADAR EQUATION

The return power and signal to noise ratio were calculated using the standard free-space radar equation [2.3]. The following parameters for the experimental system were:

Centre frequency of tunable oscillator : f = 10.25GHz.

Antenna efficiency: ß = 0.7

The effective area of the antenna: $A_e = 0.01\ m^2$

The antenna gain was then calculated to be:

$$G = \frac{4.\pi.A_e.ß}{l^2} \qquad \text{where } c = 3.0 * 10^8 \text{ m/s}$$

$$l = \frac{c}{f} = 0.029 \text{ m}$$

$$= 102.7$$

$$\equiv 20 \text{ dB}$$

The antenna 3dB beamwidth:

$$BW = \left[\frac{180}{\pi}\right] * \left[\frac{l}{D}\right] = 16.5° \qquad \text{where } D = \text{aperture diameter}$$

D = 10cm.

Assuming the target is a point reflector, the power received back is:

$$P_{rx} = \frac{P_{tx}.G^2.l^2.\sigma}{(4.\pi)^3.R^4.L}$$

where:

the transmitted power: $P_{tx} = 20mW$

$\equiv$ +13dBm

the wavelength: $l = 0.029m$

greatest range to the target: $R = 2.5\ m$

the assumed system losses: $L = 10dB$

σ is the radar cross section: $\sigma = 0.7 * 10^{-3} m^2$

σ is assumed to be $\pi.r^2$ (resonance region) for an experimental iron ball target of 3cm diameter [2.4] - see figure 2.4.

$$P_{rx} = 1.60 * 10^{-10} W$$

The mean square thermal noise power is:

$$P_N = F.k.T.B$$

where:

the system noise figure: $F = 1\ dB$

Boltzman's constant: $k = 1.38 * 10^{-23}$

temperature: $T = 290\ K$

system bandwidth: $B = 10kHz$

Note that the implemented bandwidth was chosen for versatility to be 10kHz **not** the required range bandwidth of 3kHz.

$$P_N = 6.34 * 10^{-17} W$$

The signal to noise ratio (SNR) is:

$$SNR = 10.\log\left[\frac{P_{rx}}{P_N}\right]$$

$$= 64 \text{ dB}$$

The expected SNR is very large. This is ideal for the future development of the system, where very large SNRs are required.

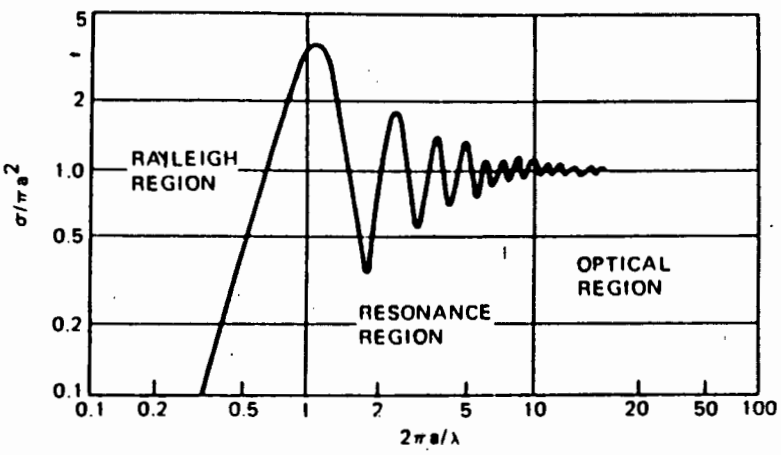


Fig. 2.4. Radar Cross-Section of Sphere as a Function of Wavelength.

The output signal voltage from the mixer is then:

$$V_{rx} = \sqrt{(P_{rx} * Z_{mixer})}$$

$$= \sqrt{(1.60 * 10^{-10} * 50\Omega)}$$

$$= 89\mu V \text{ RMS.}$$

The required voltage gain for 5V peak at the output is:

$$G_{total} = 5.0 / (89 * 10^{-6} * \sqrt{2})$$

$$G_{total} = 39528$$

The effective output noise voltage from the mixer is then:

$$V_{noise} = \sqrt{(4 * P_N * Z_{mixer})}$$

$$= \sqrt{(4 * 6.34 * 10^{-17} * 50\Omega)}$$

$$= 112.6nV \text{ RMS.}$$

It is important to note that in the system implementation, lengths of SMA cable are used to interconnect the microwave devices. The effective path length is affected by the cable lengths. The longer the cable, the longer the effective path length, and hence the higher the return beat frequency. This must be taken into account in using the FMCW radar equation. Also, since there is a shorter effective free-space distance travelled by the electromagnetic radiation, the distance from the target to the antennae must be used in the radar equation - see figure 2.5.

From figure 2.5, it can be seen that the leakage signal will be at a frequency determined by a path length of 25cm. The leakage path length is five times greater than the valid return path length. Therefore, the leakage signal is 28dB greater than the return signal. The filters in the intermediate-frequency (IF) stage, and the baseband filtering stage reduce the leakage by about 30dB. The excellent isolation of the bistatic system also reduces the leakage signal so that the nett effect is that the leakage signal is well below that of the valid radar returns.

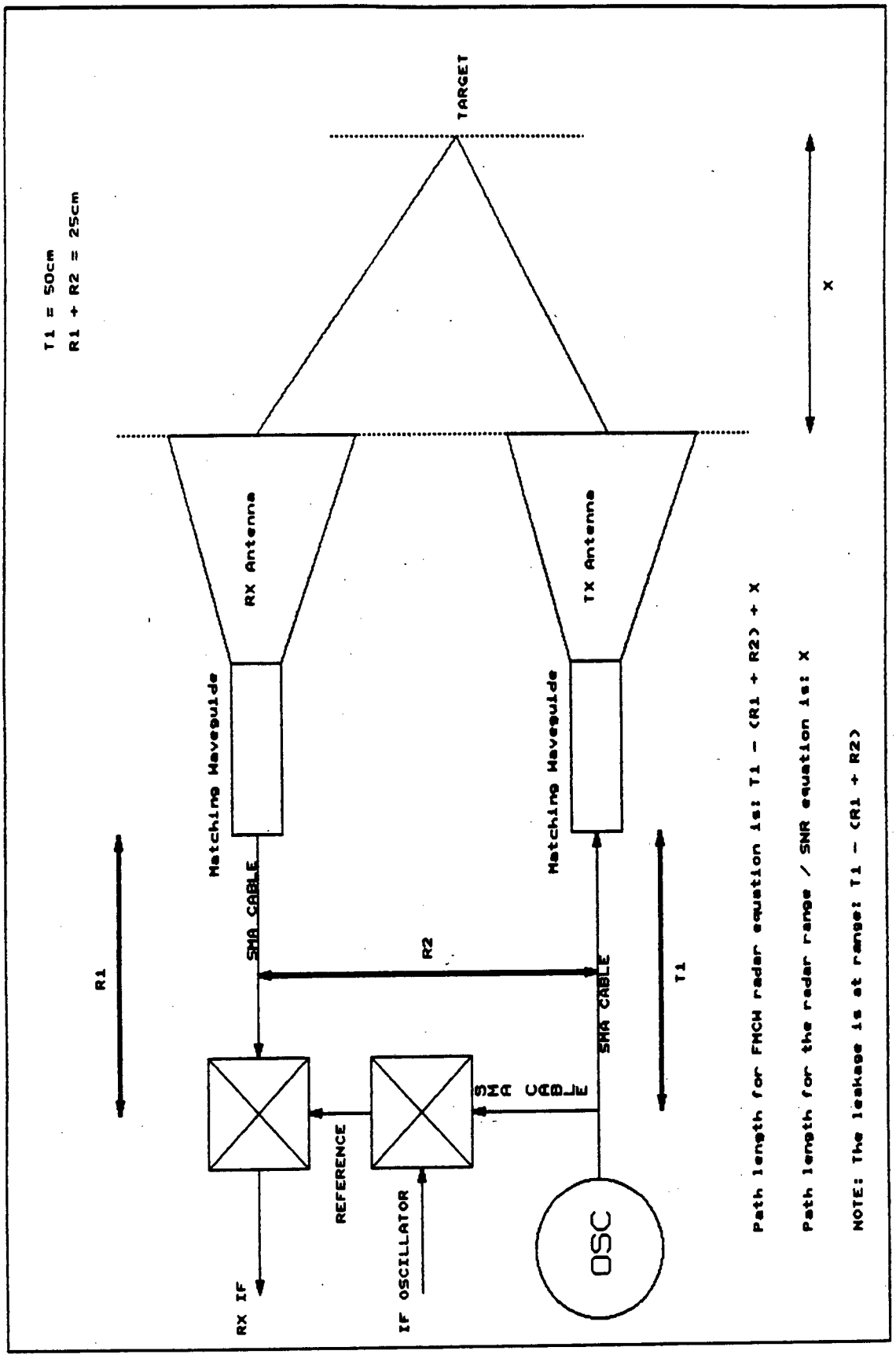


Fig. 2.5. Effective Path Lengths in the Actual Implementation.

2.4 REFERENCES

[2.1] HOVANESSIAN, S.A. **Radar System Design And Analysis.** Artech, 1984.

[2.2] BRYANT, G.H. **Principals Of Microwave Measurements.** IEE ELECTRICAL MEASUREMENT SERIES 5, PETER PEREGRINUS, 1988.

[2.3] SKOLNIK, M. **Introduction To Radar Systems.** (2nd ed.)., McGraw-Hill, Japan.

[2.4] SANDER, K.F. **Microwave Components And Systems.**, Addison-Wesley, 1987.

3. HARDWARE DEVELOPMENT

3.1 SYSTEM OVERVIEW

The system layout can be seen in figure 3.1. A bistatic system was chosen due to its superior leakage isolation [3.1] to monostatic systems. An isolation of better than 45dB was measured. A single conversion system with an intermediate frequency (IF) of 10.8MHz was utilized due to its superior low-noise performance over the direct down-conversion system. Another advantage of having an intermediate frequency system is that it has a higher dynamic range which is advantageous for targets with widely varying radar cross-sections.

The modulator provides the periodic modulation signal for the tunable oscillator. A solid-state oscillator was chosen for this thesis. This oscillator can be YIG or varactor tuned. Since the tunable oscillator is not linear, the modulator will have to take into account the non-linearity. YIG tuned oscillators are more linear than varactor tuned oscillators. Both types were tried and the YIG tuned option was eventually opted for.

The oscillator was chosen to work at X-band due to the practical size and availability of the microwave equipment. Also the high frequency means that the wavelength (3cm) is close to that of the expected experimental targets (3cm radii spheres), thus improving the received signal strength. If smaller targets are used, the radar cross-section will be in the Rayleigh region [3.2], and the received power will drop off sharply as the target size decreases. A 20dB coaxial isolator was connected directly after the oscillator to prevent frequency pulling due to the imperfect antenna match. The oscillator power was +16dBm.

A 3dB Wilkinson power splitter was used to split the power. Wilkinson splitters provide about 20dB isolation at the input port. Half of the oscillator power (+13dBm) from the first output port of the Wilkinson splitter is transmitted through the transmit antenna. Square horn antennae were chosen for their practical sizes. Additional 30dB isolators were connected to the transmit and receive antennae. The rest of the power from the second output port is used as a reference R_1 to heterodyne the received signal.

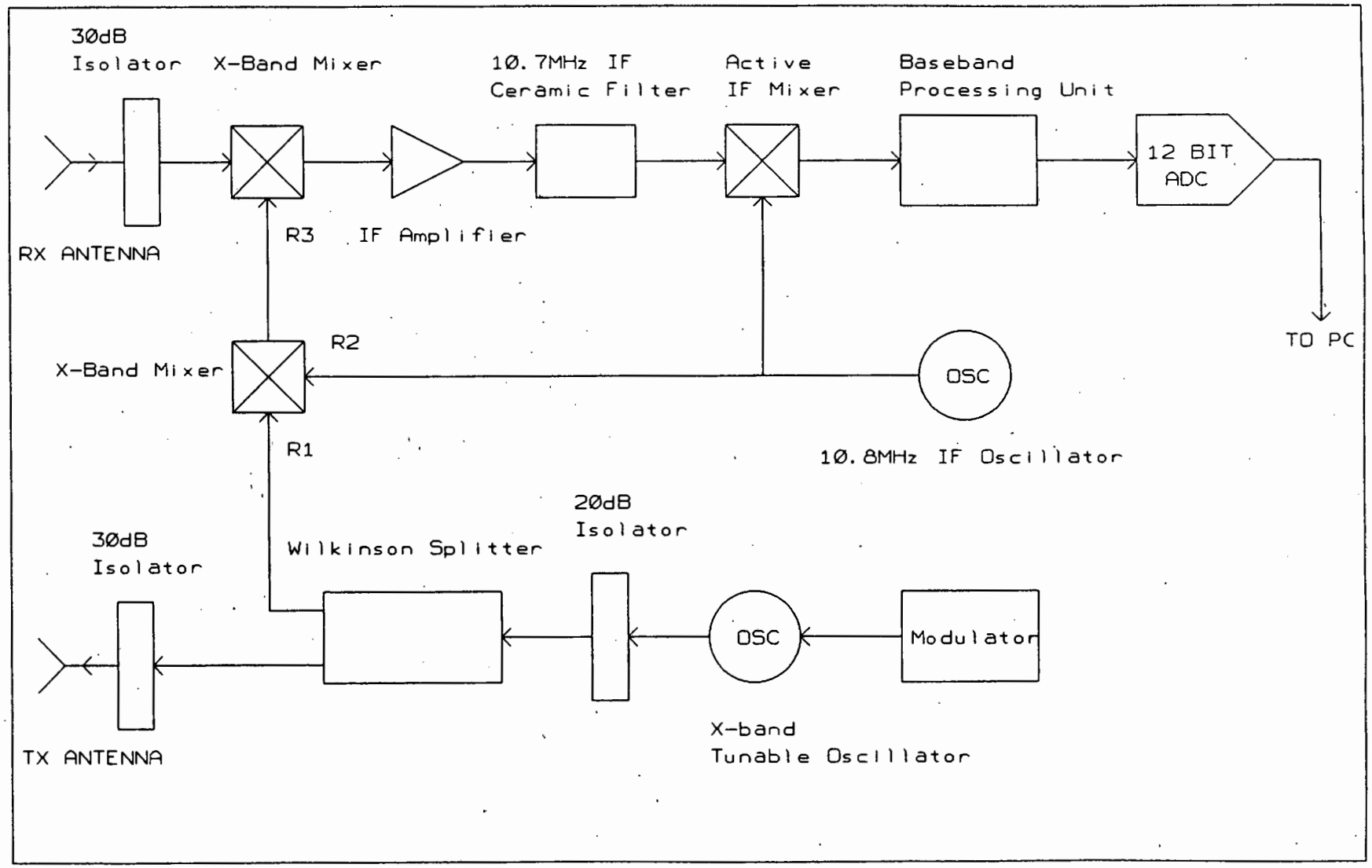


Fig. 3.1. System Layout.

The reference signal R_3 is obtained by mixing the reference signal R_1 with the 10.8MHz IF reference R_2. The mixer is a single balanced X-band mixer with 50Ω port impedances and an up-conversion loss of 6dB. The 10.8MHz IF reference is generated with a crystal oscillator with a power output of +10dBm. The output power of the up-converted reference is +7dBm. The IF signal is obtained by mixing the received signal with the reference R_3. The down-conversion loss is 6dB when pumped with a +10dBm reference, however a loss of 10dB is incurred due to the lower pumping power of +7dBm. The mixers have a further 20dB isolation at their ports.

The IF signal is amplified by an IF amplifier with a gain of 36dB. The signal is bandpass filtered using a 10.7 MHz ceramic filter and mixed down to DC using a double-balanced mixer.

The final filtering stage consists of gain trimming, final bandpass filtering and range filtering.

After the analogue processing, the signal is digitized and digital signal processing is performed.

Instantaneously, the system can be seen as follows:

Neglecting amplitudes and phases:

The oscillator generates:

$$TX(t) = \cos(w_{tx}.t)$$

The received waveform contains the beat frequency:

$$RX(t) = \cos(w_{rx}.t) = \cos(w_{tx}.t \pm w_b.t)$$

The reference signals R_1, R_2, R_3:

$$R_1(t) = \cos(w_{tx}.t)$$

where $w_{tx} = 2.\pi.(10 \text{ to } 10.5\text{GHz})$

$$R_2(t) = \cos(w_{IF}.t)$$

$$\text{where } w_{IF} = 2.\pi.(10.8\text{MHz})$$

$$R_3(t) = \tfrac{1}{2} * \left[\cos(w_{tx}.t + w_{IF}.t) + \cos(w_{tx}.t - w_{IF}.t)\right]$$

The IF signal:

$$IF(t) = RX(t) * R_3(t)$$

$$IF(t) = \tfrac{1}{2}.\cos(w_{tx}.t \pm w_b.t) * \left[\cos(w_{tx}.t + w_{IF}.t) + \cos(w_{tx}.t - w_{IF}.t)\right]$$

$$IF(t) = \tfrac{1}{2} * \left[\tfrac{1}{2}.\cos(2.w_{tx}.t + w_{IF}.t \pm w_b.t) + \tfrac{1}{2}.\cos(w_{IF}.t \pm w_b.t) + \tfrac{1}{2}.\cos(2.w_{tx}.t - w_{IF}.t \pm w_b.t) + \tfrac{1}{2}.\cos(-w_{IF}.t \pm w_b.t)\right]$$

The $2.w_{tx}$ terms are filtered out by the IF stage leaving:

$$IF(t) = {}^1/_4 * \left[\cos(w_{IF}.t \pm w_b.t) + \cos(-w_{IF}.t \pm w_b.t)\right]$$

Then the final down-conversion to baseband:

$$B(t) = R_2(t) * IF(t)$$

$$B(t) = \cos(w_{IF}.t) * IF(t)$$

$$B(t) = {}^1/_4 * \left[\cos(2.w_{IF}.t \pm w_b.t) + \cos(\pm w_b.t)\right]$$

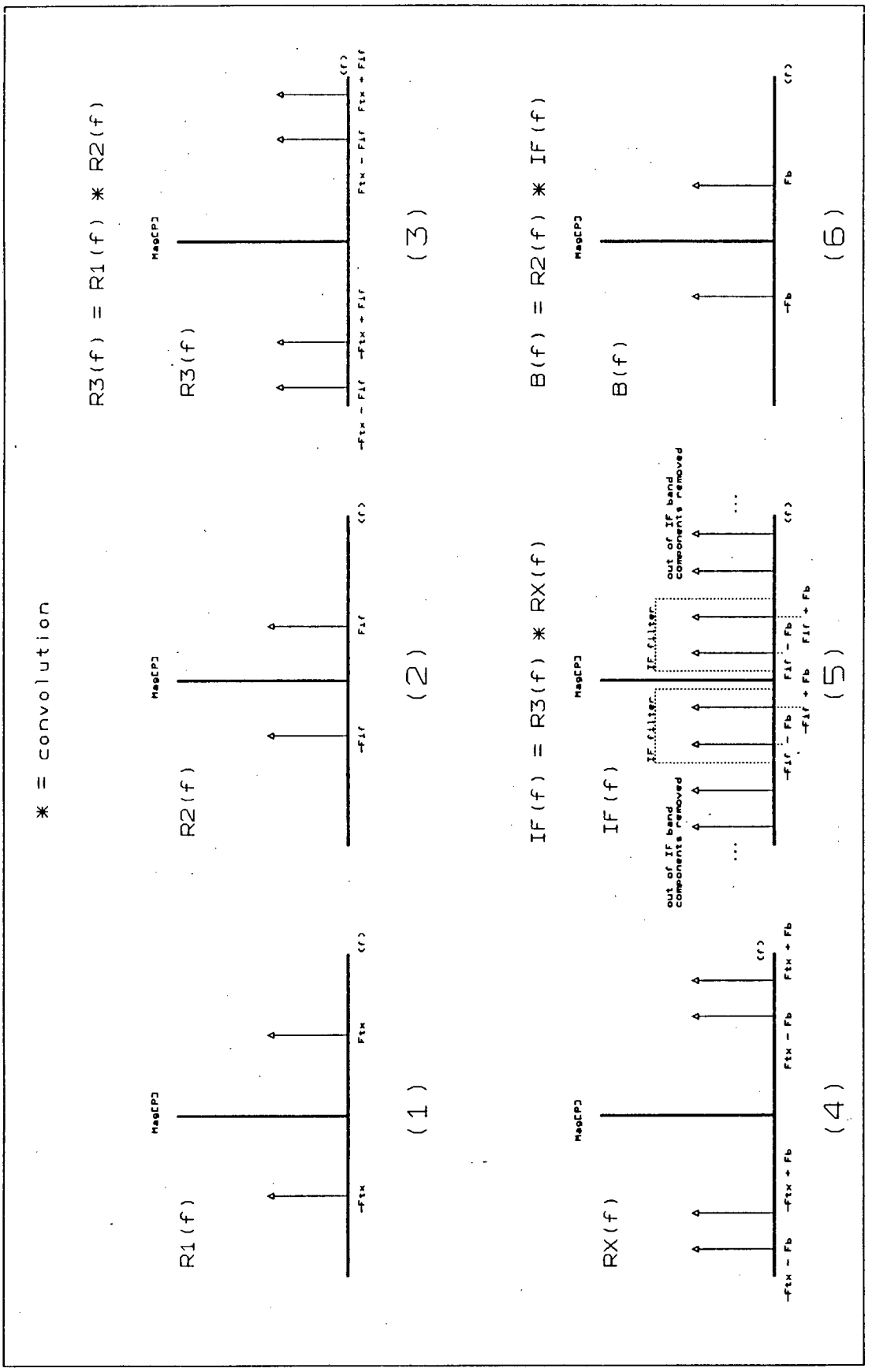


Fig. 3.2. Magnitude Frequency Response of the System.

The $2.w_{IF}$ term is filtered out by the baseband filters giving:

$$B(t) = {}^{1}/_{4} * \cos(\pm w_b.t)$$

For the Magnitude frequency response [3.3], see figure 3.2.

3.2 MIXERS

The type of mixer used was a single balanced X-band mixer with 50Ω port impedances. The up-conversion and down-conversion loss with 10dB pumping power was 6dB. With 5dB pumping power, a down-conversion loss of 10dB was obtained. The mixer is self-biased. It has three ports:

a) local oscillator port,
b) receive port, and
c) IF port.

The local oscillator port is, ideally, driven to saturation. The mixer can therefore be modelled by a switching function [3.4] - see figure 3.3.

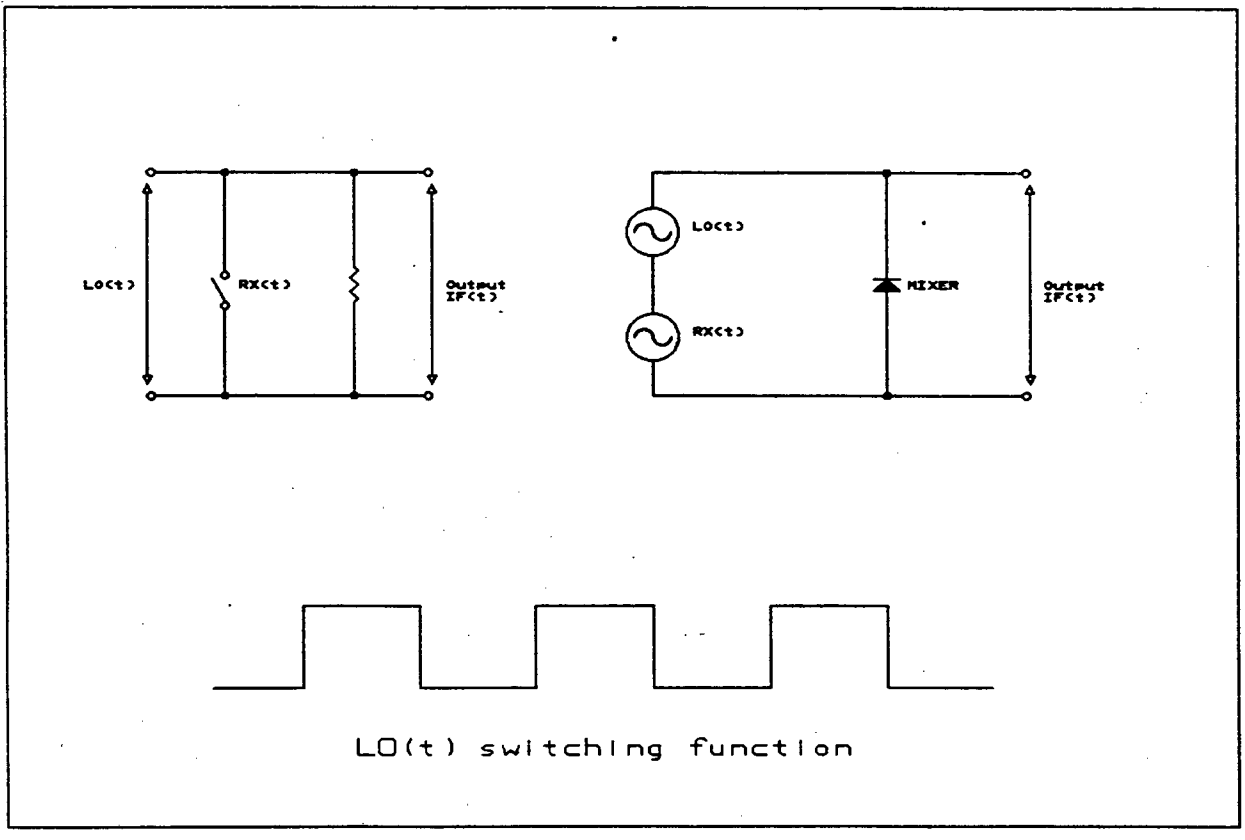


Fig. 3.3. Mixer Model.

The model can be mathematically described as follows:

The local oscillator signal which saturates the mixer can be represented by the Fourier series of a square wave [3.3]:

$$LO(t) = k_0 + k_1.\cos(w_{LO}.t) + k_3.\cos(3.w_{LO}.t) + ..$$

The received signal is:

$$RX(t) = R.\cos(w_{rx}.t)$$

When the received signal is fed to the mixer, it is effectively multiplied by the switching function:

$$IF(t) = RX(t) * LO(t)$$

$$= R.\left[k_0.\cos(w_{rx}.t) + k_1.\cos(w_{rx}.t).\cos(w_{LO}.t) + k_3.\cos(w_{rx}.t).\cos(3.w_{LO}.t) + ...\right]$$

Since the first, third, and subsequent terms are out of the IF frequency range, they are filtered out. The second term can be expressed as a sum and difference of the two frequencies:

$$IF(t)_{filtered} = OP(t) = R.k_1.\cos(w_{rx}.t).\cos(w_{LO}.t)$$

$$OP(t) = {}^1/_2.A.\left[\cos[(w_{rx} - w_{LO}).t] + \cos[(w_{rx} - w_{LO}).t]\right]$$

3.3 TUNABLE OSCILLATOR

Two types of oscillator were tested. The first type was a varactor tuned Gunn diode oscillator, and the second type an yttrium-iron-garnet (YIG) tuned oscillator. The Gunn oscillator was found to be unsuitable due to its low Q, and its inherent non-linear tuning characteristic. The YIG tuned oscillator was deemed suitable due to its highly linear tuning characteristic. The specifications and theory for both types of oscillator will be discussed.

Table 3.1 Gunn Oscillator Specifications.

Plessey Gunn Oscillator Specifications @ 25°C.	
TUNING: RF Centre Frequency: Mechanical: Electronic: Linearity: Frequency stability:	 10.20GHz 500MHz 500MHz 40% 2.0 MHz / °C max
RF Power vs. temperature and tuning voltage:	 6dB max.
Frequency pushing:	100MHz / V
Supply Requirements Gunn voltage: Max. operating current: Tuning voltage:	 +10V DC 500mA +0V to +20V DC
RF output power:	20mW (+13dBm)

An X-band varactor tuned Gunn diode oscillator was utilized. The specifications can be seen in table 3.1. The Gunn diode is mounted along with a varactor diode in a resonant cavity. The Gunn diode oscillates when a regulated supply is applied to its terminals. The resonant frequency of the system is dependent on the cavity size, and the capacitance of the varactor diode. The cavity size is varied by adjusting tuning screws. The capacitance is varied by adjusting the bias voltage across the varactor diode. The higher the voltage, the lower the capacitance, and hence the higher the output frequency. The tuning screws set the frequency offset of the tuning range. These are factory preset. The tuning range is just greater than 500MHz. The voltage tuning range is from 0 to 20V. The corresponding frequencies were 9.95GHz to

10.45GHz. The microwave power is coupled out of a standard X-band microwave flange. Figure 3.4 shows the tuning characteristic of the Gunn source.

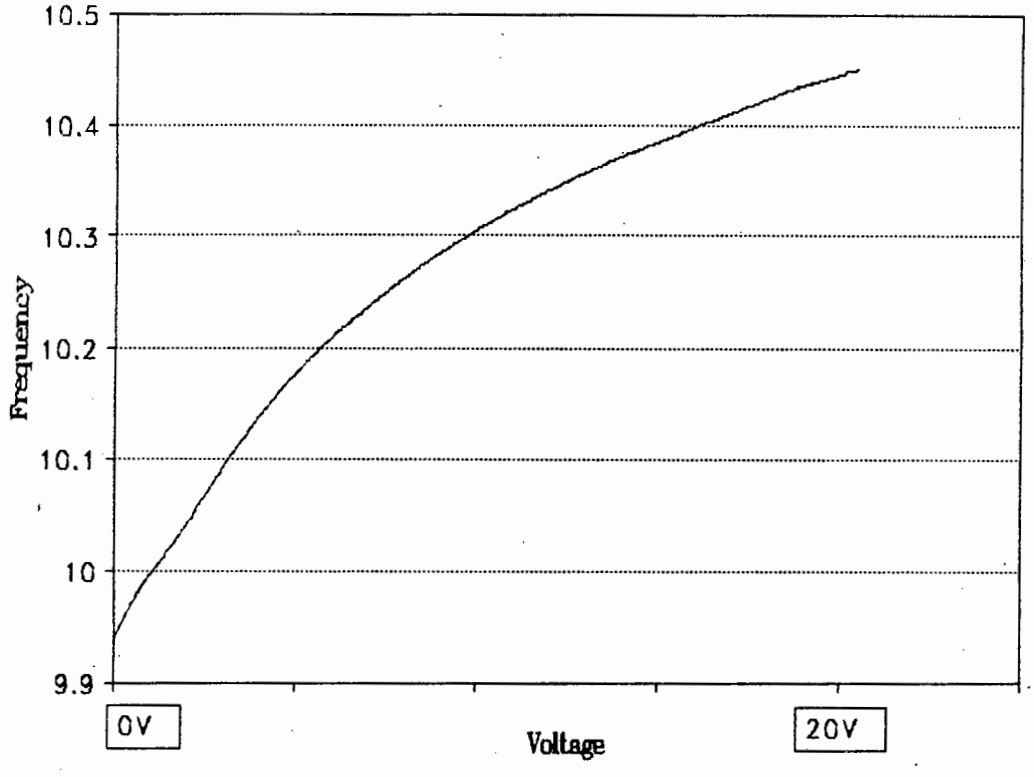


Fig. 3.4. Gunn Tuning Characteristic.

An X-band YIG tuned oscillator was utilized. The specifications can be seen in table 3.2. The following description is extracted from Bryant: "An active element in this oscillator provides a negative resistance in a YIG resonator consisting of a sphere of yttrium-iron-garnet in a DC magnetic field. The active element is usually a bipolar transistor or a GaAs FET. Yttrium-iron-garnets are a family of ferrites doped to resonate at microwave frequencies when immersed in a suitable magnetic field. Elemental dipoles of the material, originating in the precession of electron spin about the DC magnetic field, are aligned in the direction of that field, with precession frequency at the natural magnetic resonance of the garnet. When an RF field is imposed perpendicularly to the DC magnetic field at the resonance frequency, an absorption or reflection occurs depending on the configuration of the circuits external to the garnet.

Table 3.2. YIG Tuned Oscillator Specifications.

Avantek AV-74010 YIG Tuned Oscillator Specifications @ 0-65°C.	
TUNING: Electronic: Linearity: Frequency stability:	 6.5 GHz < 0.05% 20MHz / °C over T.
RF Power Variation vs. temperature and tuning voltage:	 < 6dB over 6GHz < 1dB over 1GHz
Frequency pushing:	20MHz / mA
Supply Requirements Supply voltage Heater voltage Main tuning coil current:	 +15V DC - 5V DC + 20V - 28V DC 450 mA - 503 mA (9.5GHz - 10.5GHz)
RF output power:	40mW (+16dBm)

The electron spins precess about the magnetic field, H_0, at the gyromagnetic frequency given by:

$$f_0 = \Gamma * H_0 \qquad \text{where } \Gamma = 2.8\text{MHz.s}^{-1}\text{.Oe.}$$

Since f_0 is linearly related to the magnetic field, the oscillator can be tuned by varying H_0. A heater is usually built into the oscillator to reduce temperature sensitivity."

The YIG tuned oscillator need not be linearized as for the varactor tuned oscillator since it it already extremely linear. Therefore it can be modulated by a triangular waveform.

3.4 MODULATOR

3.4.1 Circuit Description

This circuit block provides the modulation signal for the tunable oscillator. Since the tunable oscillator does not have a linear transfer characteristic (voltage to frequency), an inverse transfer map of the non-linear transfer characteristic is required. Thus, the modulator output will not be a pure triangular waveform. The modulator can therefore be broken up into two sections:

a) Modulator oscillator
b) Linearizer.

The linearizer was implemented digitally. The modulator circuit can be further broken down into a number of sections - see figure 3.5. The first section is the modulator oscillator. The second section is the address counter which points to the next point in the transfer map. The third section is the transfer map itself, an EPROM. Lastly, the digital transfer point is converted to an analogue voltage by a DAC.

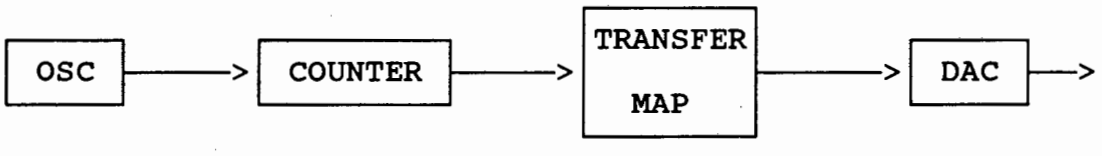


Fig. 3.5. Modulator Components.

Table 3.3. Modulator Specifications.

Modulator Specifications.	
Maximum modulation frequency:	1.95 kHz
Nominal modulation frequency:	0.90 kHz
Maximum output voltage:	20 V
Minimum output voltage:	0.0 V
Resolution amplitude:	4.88 mV
Supply Requirements:	± 27 V

3.4.2 Circuit Details

The circuit diagram for the modulator can be seen in figure 3.6.

The oscillator is based on the standard CMOS crystal oscillator configuration. A CMOS inverter gate U2a is biased to $^1/_2$ supply voltage with R1 so that it acts as an amplifier. Note the CMOS inverter gate has a 180° phase shift. The crystal and capacitors C1, C2 form a 180° phase shift network. The nett combination of the inverting amplifier U2a and the phase shift network yields a 360° phase shift, which causes the system to oscillate at the resonant frequency of the crystal. Since the amplitude is not controlled, it is clipped by the non-linear gain characteristic of the gate and as a result, the oscillator produces a square wave. The crystal's resonant frequency was chosen to be 3.686MHz. The oscillator output is then buffered by inverting gate U2b. The rest of the CMOS packages U2c to U2f are not used in the system and have their inputs grounded to prevent them from affecting U2a, U2b.

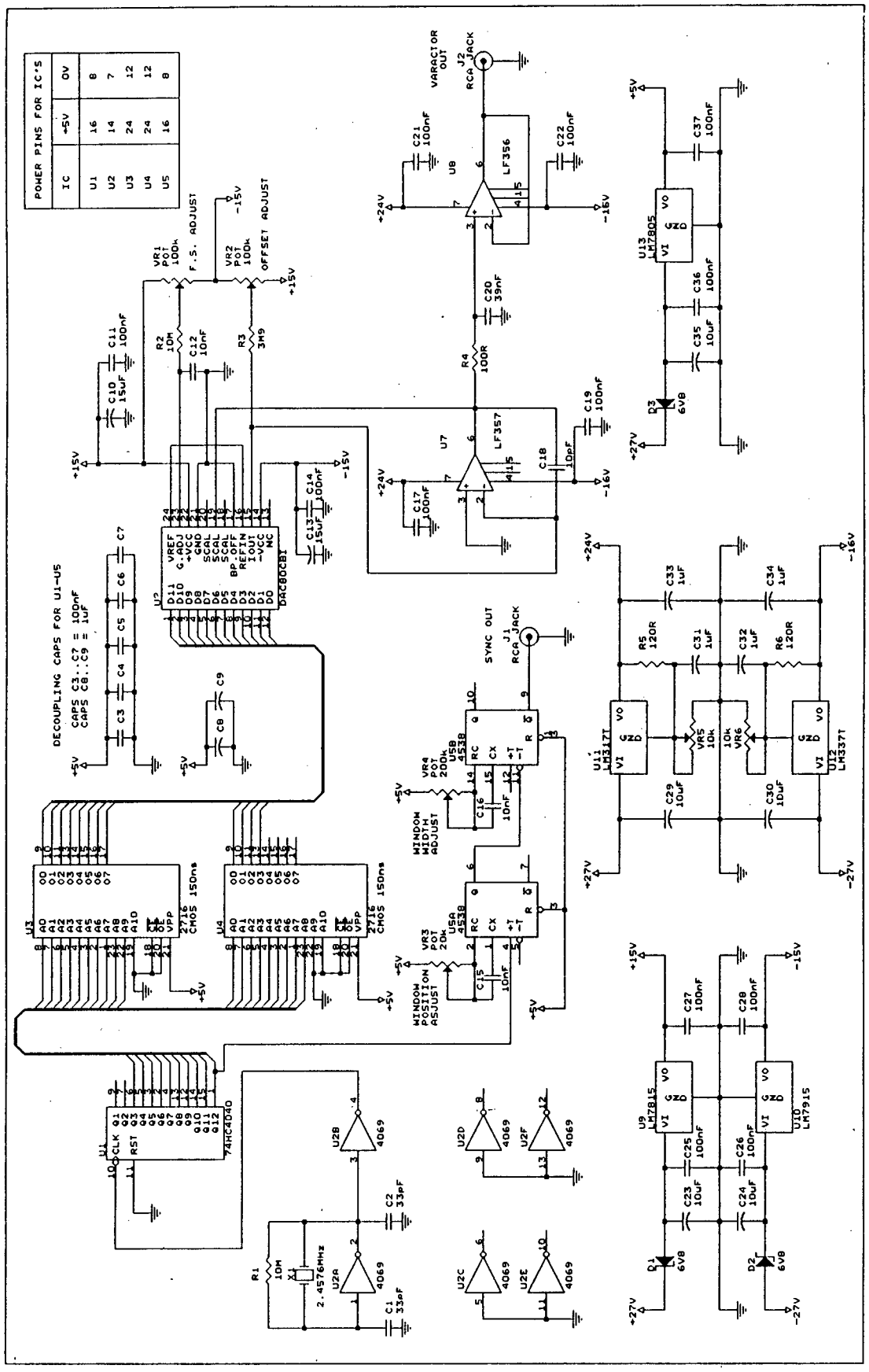


Fig. 3.6. Modulator Circuit Diagram.

The counter, U1, consists of a 74HCT4040 12 bit binary ripple counter. Ten of the bits are used to give a 1024 cycle modulation waveform with a frequency of 900Hz. Half of this ie. 512 cycles provide the up-ramp, whilst the next 512 cycles provide the down-ramp. The first two address bits Q1 and Q2 of U1 are not used which results in a divide by 4 action to reduce the frequency to the required value. The counter is a high speed CMOS version which minimizes glitch pulse widths at the clocking times. This would otherwise get through to the output of the DAC. The next ten address bits are passed on to two 2716 - 150nS CMOS EPROMS - U3, U4. These EPROMS have 2048 address locations. The ten address bits are connected to the address busses of both EPROMS and the remaining unused address bits of the EPROMS are grounded. The OE, CE pins of the EPROMS are also grounded to permanently enable them. The eight data bits of U3 and the lower data nibble (4 bits) of U4 are combined to yield a 12 bit binary value. These twelve bits are connected to the digital-to-analogue converter (DAC), U6.

VR1 and VR2 adjust the full scale amplitude and the amplitude offset respectively. The DAC provides a 0 to 2mA output current which drives the virtual earth of inverting, summing op-amp U7. The internal feedback resistor is connected to the output of U7. It is built into the DAC for temperature tracking. There are three choices of feedback resistors inside U6, however the one chosen was that which yielded the 20V range. Filter capacitor, C18 was chosen so that the breakpoint occurred at the effective clocking rate ie. the crystal frequency divided by four (922kHz). This capacitor smooths the DAC output. The output of U7 is low-pass filtered by R4 and C20 to provide further smoothing and eliminate digital noise at the output. Its high frequency breakpoint is placed at 41kHz. The output of the low-pass filter is then buffered by the op-amp, U8, which is in voltage follower configuration. U7 and U8 are low-noise, low-offset voltage op-amps. They also have low temperature coefficients. U7 has a high slew rate and a wide gain-bandwidth product. The settling time of the DAC / op-amp combination to 0.01% of the final value is about 1μs. This implies a maximum output rate of 1MHz. The output from the modulator with the linearization map can be seen in figure 3.7.

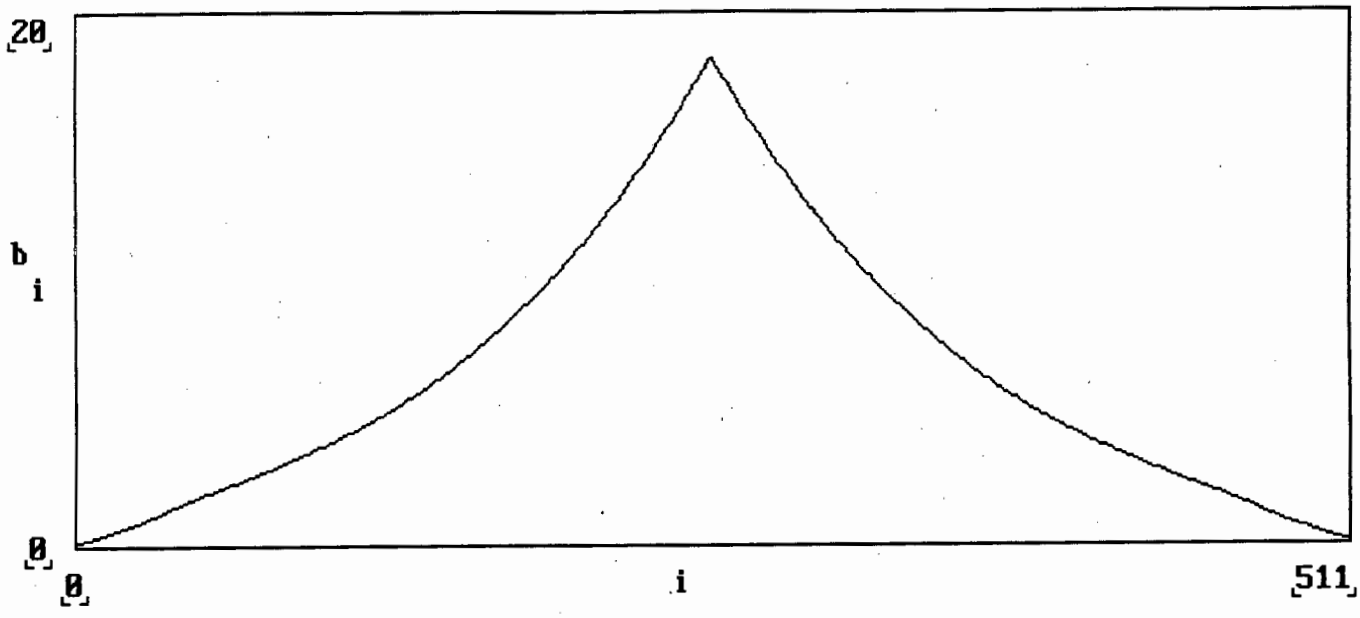


Fig. 3.7. Modulator Output.

The filters of the base-band filtering stage (see later) cause the received signal to be phase-shifted. The ADC has to sample the signal at the periods between the discontinuities. This means that a synchronizing circuit is required to align the ADC sampling period with the phase-shifted, received signal. Two monostables, U5a,b are used for this purpose. The first monostable, U5a, is connected to the tenth effective bit of the counter, so that at the slope change-over points, the monostable is triggered. The first monostable aligns the sampling window start-timing. It can be varied up to 200μs. The second monostable, U5b, is connected to the output of U5a, and is used to set up the sampling window duration. It can be varied up to 2ms. The period of the modulation half-cycle, ie. up-slope or down-slope, is 556μs. The output of U5b, SYNC OUT, is connected to the ADC's sample enable pin.

All the I.C.s have decoupling capacitors.

The power supply regulation circuitry is placed inside the modulator module to protect it from inadvertent mis-connection.

Five supplies are required by the modulator:

a) ± 15V for the DAC.
b) + 5V for the digital circuitry.
c) + 24V
- 16V
for the op-amps.

a) ±15V Supply

This supply was made up using two standard, fixed voltage regulators, the positive voltage regulator U9, and the negative voltage regulator, U10. Zener diodes, D1 and D2 are used to reduce the voltage drop across U9 and U10. This reduces the power dissipation in them and allows them to be used without heatsinks.

b) +5V Supply

This regulator, U13, is a standard positive 5V voltage regulator and also has a zener diode, D3, to reduce the power dissipation. It does, however, require a heatsink due to the relatively high current consumption of the digital circuitry.

c) +24V, -16V Supply

These supplies were necessary due to the high voltage required out of the op-amps. The maximum supply voltage for these op-amps is ± 22V due to the inability to reach the supply rails within up to 3V. A higher supply voltage is required, ie. 23V minimum, since the maximum required output is 20V. Since the output required is above 0V, one solution is to increase the positive supply and decrease the negative supply. This causes a larger offset voltage, but it can be adjusted by using the offset-adjust preset, VR2. Standard adjustable positive and negative voltage regulators, U11 and U12, were used. No heatsinking was required. Note that modulator modifications for the YIG tuned oscillator are contained in appendix U.

3.4.3 Linearization Process

3.4.3.1 Theory

The linearization process is fundamental to the resolution of the radar. To derive the range accuracy formula, we start with the FMCW formula:

$$f_b = \frac{4 * R * fmod * \delta f}{c}$$

Therefore:

$$R = \frac{c * f_b}{4 * f_{mod} * \delta f}$$

Thus the differential of the dependent variable is:

$$\delta R = \left| \frac{dR}{df_b} \right| . Df_b + \left| \frac{dR}{d(\delta f)} \right| . D(\delta f) + \left| \frac{dR}{d(f_{mod})} \right| . D(f_{mod})$$

... equation 3.1

where d represents the partial derivative;
D represents the increment.

The last term of equation 3.1 is effectively 0, because a crystal oscillator is used. So neglecting the last term, we get:

$$\delta R = \left| \frac{dR}{df_b} \right| . Df_b + \left| \frac{dR}{d(\delta f)} \right| . D(\delta f)$$

$$\delta R = \left| \frac{c}{4 * f_{mod} * \delta f} \right| . Df_b + \left| \frac{- c * f_b}{4 * f_{mod} * \delta f^2} \right| . D(\delta f)$$

... equation 3.2

The first term of equation 3.2 can be considered to be the phase-noise error of the oscillator, whilst the second term can be considered to be the linearity error. Substituting in values of:

$$f_b = 10kHz$$
$$f_{mod} = 900Hz$$
$$\delta f = 500MHz$$
$$Df_b = 10Hz$$
$$D(\delta f) = 500kHz,$$

$$\delta R = 1.66mm + 1.66mm$$
$$= 3.33mm \text{ accuracy.}$$

Equation 3.2 represents a perfectly linear transfer characteristic - see figure 3.8.

Equation 3.2 is, however, not correct in that it does not model the true characteristic of the oscillator. Thus, a piecewise model must be implemented, with many linear sections - see figure 3.9.

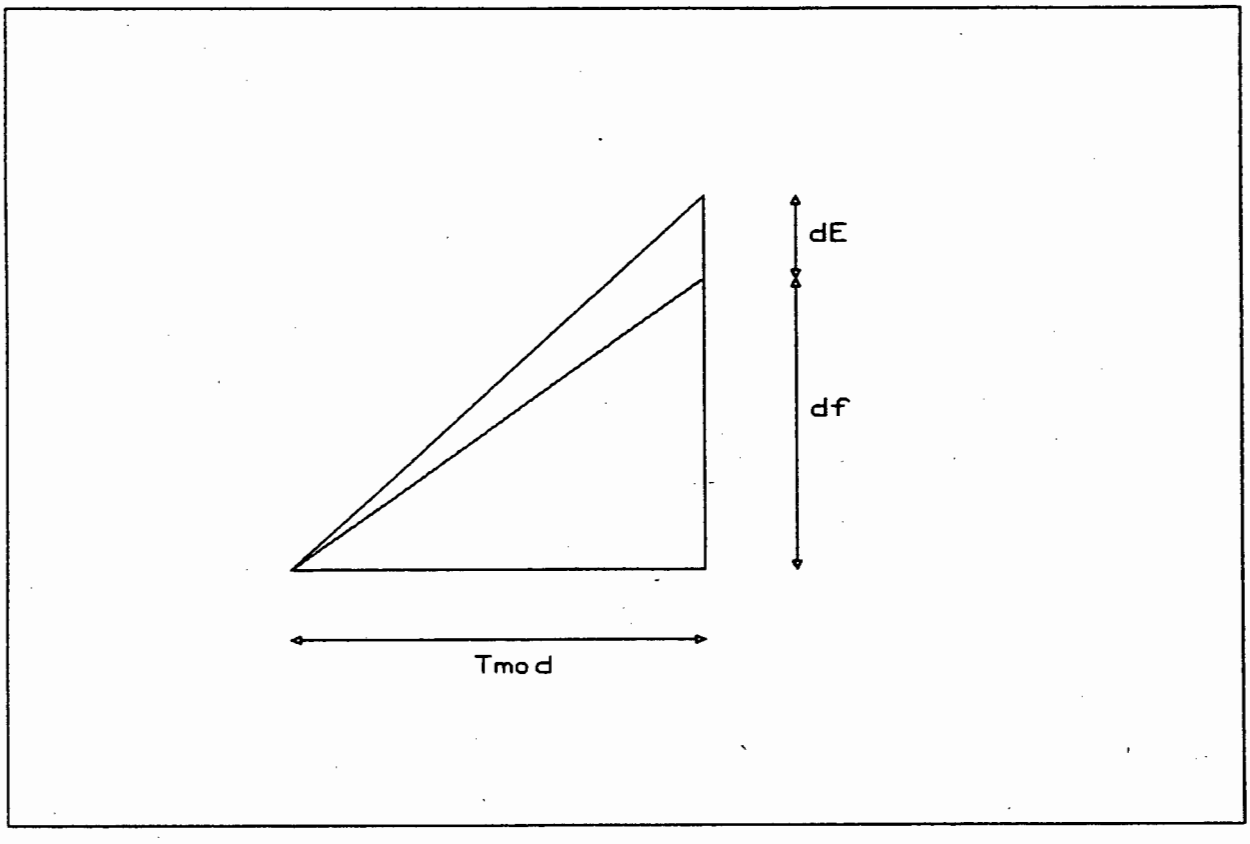


Fig. 3.8. Representation of Equation 3.2.

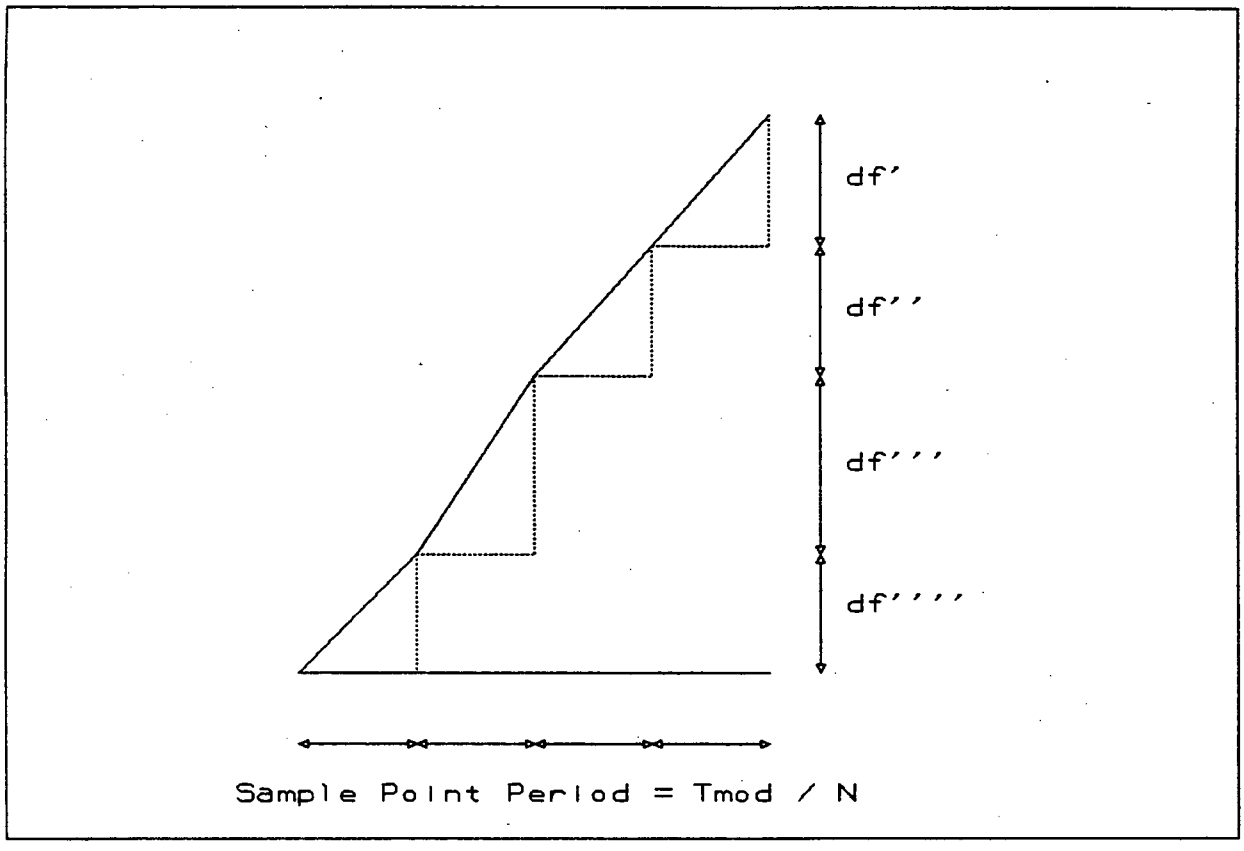


Fig. 3.9. Piecewise Representation.

For each piecewise section, equation 3.2 must be applied. This modelling approach is applicable if the piecewise steps are small enough. The first problem is that the linearity term of equation 3.2 implies that the smaller the steps, the smaller the error allowed to obtain a sufficiently linear transfer characteristic. This means that if relatively large steps are utilized, the system does not have as stringent requirements on frequency accuracy than a more accurate model with smaller piecewise steps.

If we apply equation 3.2 to smaller steps ($^1/_{50}$ of 500MHz), we get:

f_b = 10kHz
f_{mod} = 45kHz (900Hz * 50)
δf = 10MHz
Df_b = 10Hz
$D(\delta f)$ = 120kHz,

$$\delta R = 1.66mm + 20.0mm$$
$$= 21.7mm \text{ accuracy.}$$

The linearity term is more dominant, and is necessary for high resolution. It determines the smearing of the beat frequency.

The DAC of the modulator can represent up to 4096 voltage steps, so the frequency jumps are approximately:

$$Df = \frac{500MHz}{4096}$$

$$= 122kHz.$$

If we want to have errors less than 15mm then we need a frequency accuracy of 120kHz - see above.

Since the beat frequency is proportional to the rate of change of frequency, it is not the absolute frequency error that influences the accuracy as much as the relative error in frequency between the previous frequency point and the current frequency point - see figure 3.10.

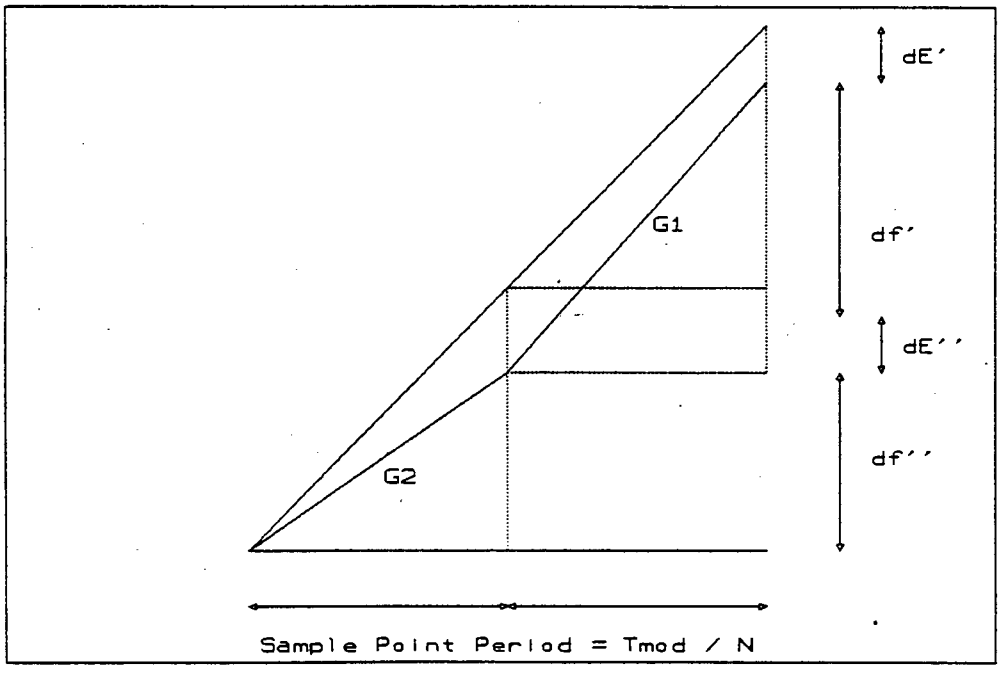


Fig. 3.10. Effect of Absolute and Relative Linearization Errors.

Another important factor is that of curve smoothness. By observing a transmitted stepped-frequency waveform, and a delayed received version in the time-domain, and beating them together, we can see the importance of smoothness - see figure 3.11. As can be seen from figure 3.11, the output of a stepped frequency continuous wave is small, since the beat frequency is only observed for a small fraction of the step period. Thus the voltage waveform driving the modulation input of the oscillator must be smooth-continuous.

From the above observations, it can be seen that a slow temperature drift in the linearization process is only serious if the frequency varies more than the minimum allowable for a particular resolution eg. more than 120kHz between piecewise steps, as in the above example. Note the importance of the relative frequency error. In practice, it was found that varactor tuned oscillators varied up to 50kHz per second at room temperature. YIG tuned oscillators were found to have up to 1kHz variation per second.

The results of the above observations imply that a DAC output cannot be used, unless the output is smoothed enough. What is ideally required is an integrator, which makes the piecewise linear steps at the output. This means that the output must be low-pass (LP) filtered beyond the harmonic which has a minimum DAC resolution effect on the output eg.:

If a triangular wave is to be generated by the modulator for a YIG tuned oscillator, the low-pass filter (LPF) must be placed at:

harmonics of triangular wave are odd and increase in amplitude at the rate of n^2:

$$\text{ie.} \quad (1 / 65^2) = (1 / 4225) < (1 / 4096)$$

(1 / 4096) is the DAC resolution.

So the harmonics beyond the 65th harmonic must be filtered out.

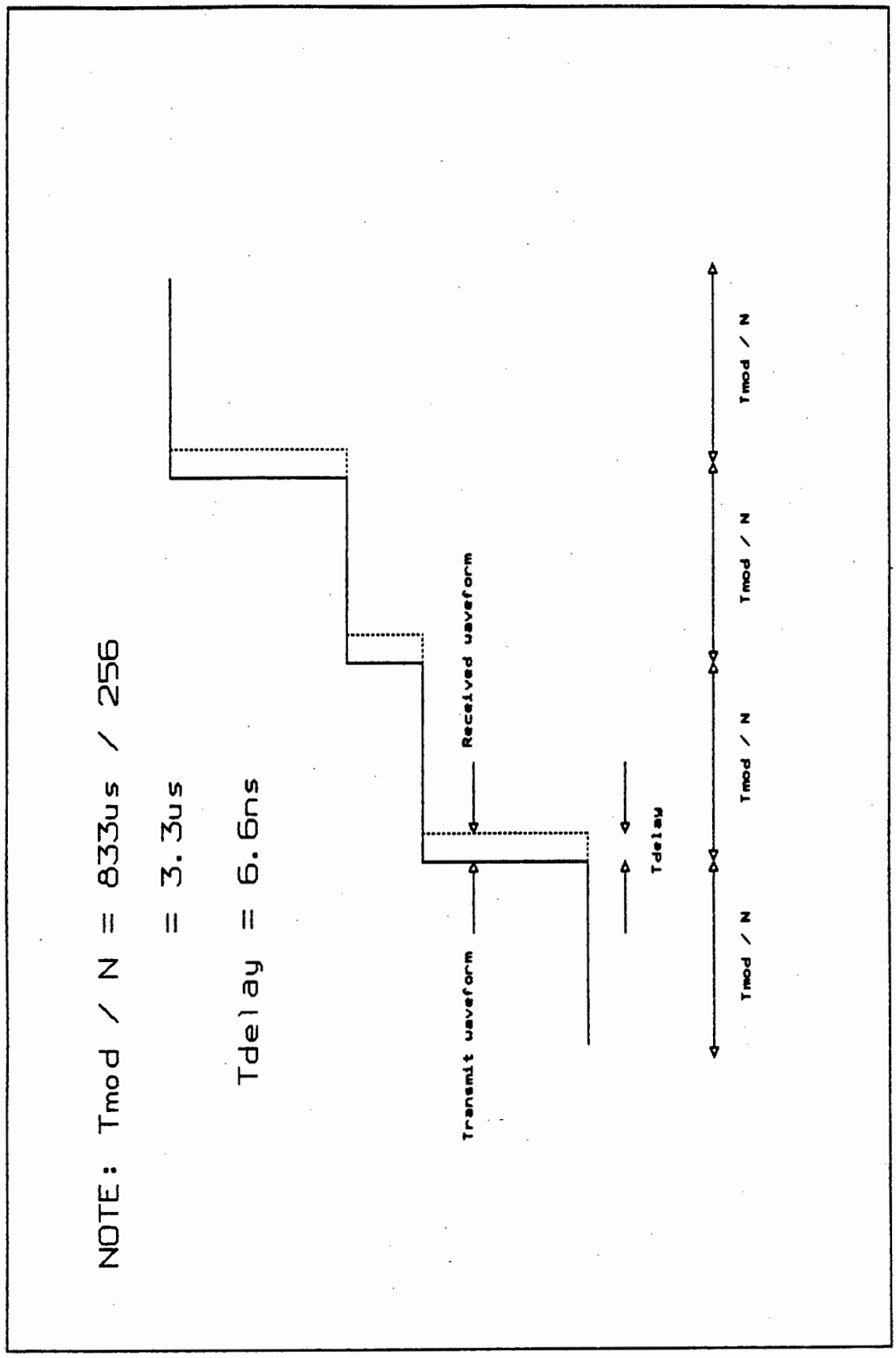


Fig. 3.11. Effect of Utilizing a Stepped Frequency Source.

If fundamental is at 900Hz, then the high frequency breakpoint must be placed at:

$$900Hz * 64 = 57.6kHz.$$

An adjustment factor of $\pi/2$ should be made because of the gradual roll-off of the 1st order RC filter.

$$57.6kHz / (\pi/2) = 36.7kHz$$

If a non-linear modulation waveform is to be used, then the filter must be placed below the step frequency of the DAC ie. 922kHz. It is not easy to calculate the best position for the high frequency breakpoint, since it is non-linear, however, it will have to be determined by measurement. Note that if significant harmonics of the modulation waveform are removed, the transfer characteristic is distorted, leading to errors.

3.4.3.2 Implementation

There are a number of steps required to achieve linearization:

i) Obtain the tunable oscillator's voltage-frequency transfer characteristic,

ii) Calculate the inverse law to linearize the oscillator,

iii) Program the EPROMS in the modulator with the inverse law.

STEP (i)

The tunable oscillator was connected via an SMA cable to an HP5351B microwave frequency counter. A 20dB SMA isolator was connected at the oscillator end to reduce load pulling effects. The HP frequency counter was connected via an HPIB interface to the PC. The modulation voltage input of the tunable oscillator was connected to a programmable power supply in an HP4195A network analyzer. The programmable supply has a 12 bit DAC with a maximum range of 40V. The HP

network analyzer is also connected via an HPIB interface to the PC. The setup is shown in figure 3.12.

A program is then run to obtain the frequencies for the 2048 voltage ranges (0-20V) - see Appendix D for the program GET_LAW. The transfer characteristic can be seen in figure 3.4. The frequency measurements were taken at a slow rate of one point per second due to the frequency counter's sampling limitation. Linearization was found to be inaccurate due to the frequency drift of the oscillator. The inaccuracy is inherent due to the short, thermal time-constant and thermal drift between measurements, especially with low Q varactor tuned oscillators. Higher Q, YIG tuned oscillators have less temperature drift per degree centigrade and an inherently more linear tuning characteristic.

Note: Before the GET_LAW program is run, the oscillator has to warm up. Half an hour minimum is required for the oscillator to reach thermal equilibrium. The frequency can be monitored on the frequency counter before the above procedure to see that the oscillator frequency has stabilized.

STEP (ii)

Another program is then run to obtain the inverse law. A bisection algorithm is used to find the voltage for which the oscillator generates a certain frequency. The transfer map is broken up into two 512 word transfer points. Each word is 12 bits long, however, the LSB is not used. The first 512 words are for the up-slope, and the next 512 words are for the down-slope. The program, LIN_LAW is listed in Appendix E.

STEP (iii)

The final phase is to blow the EPROMS with an EPROM programmer. The data is automatically converted to the correct format for the EPROM programmer by the LIN_LAW program. The transfer map values can be seen in Appendix F.

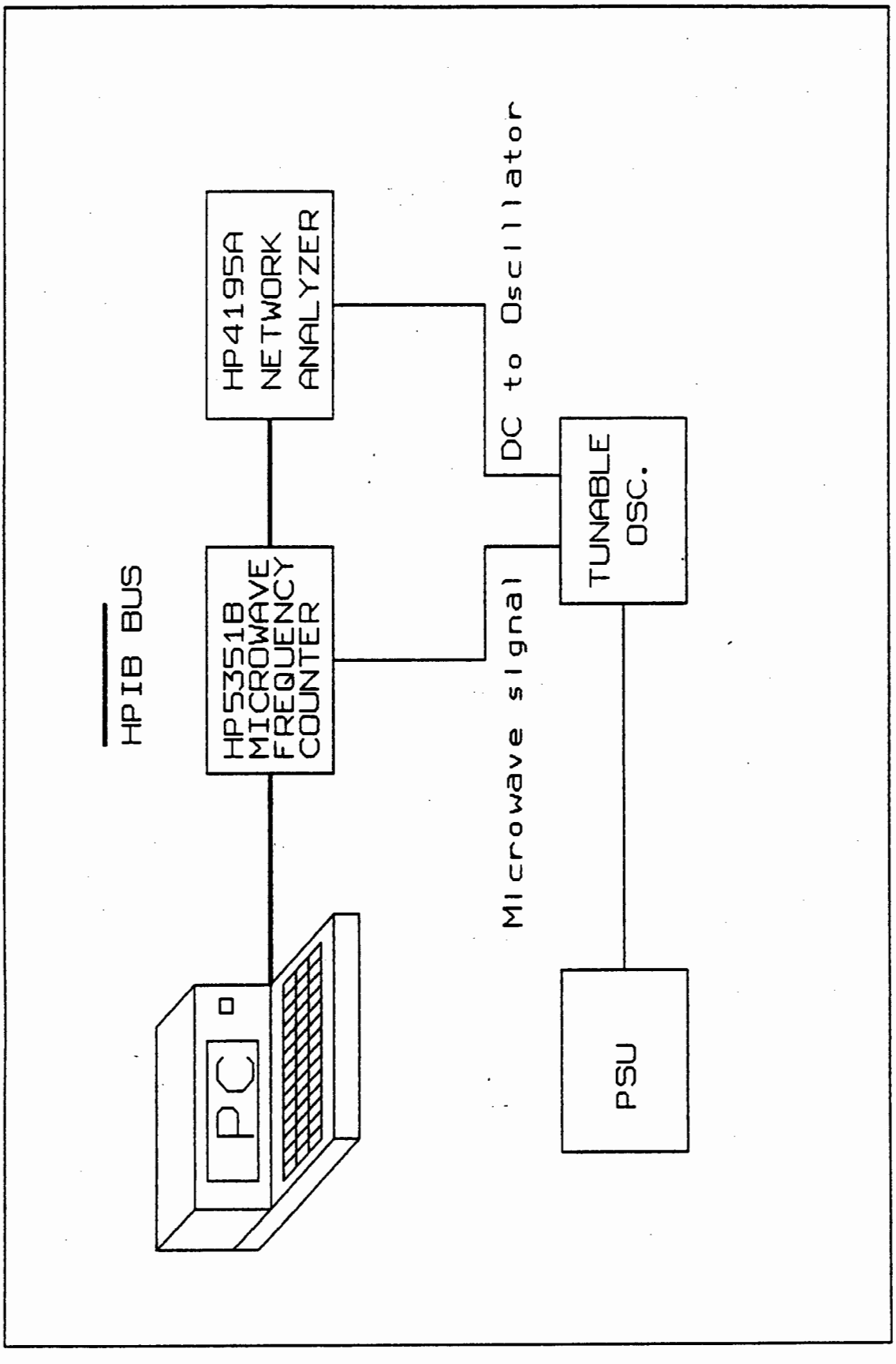


Fig. 3.12. Linearization Setup.

3.5 IF OSCILLATOR

Table 3.4. IF Oscillator Specifications.

IF Oscillator Specifications.	
Crystal frequency:	10.8 MHz
Output A power output:	+ 10 dBm
Output B power output:	+ 0 dBm

The IF oscillator frequency is 10.8MHz. Two outputs are required:

a) One +10dBm output to drive the microwave up-converting mixer,

b) One +0dBm output to drive the IF mixer.

The oscillator is a standard crystal oscillator configuration - see figure 3.13. The crystal, X1, and associated capacitors, C9 to C11, and inductor, L2, form the phase shifting network which in conjunction with the transistor, Q4, yield a 360° phase shift. The gain is chosen to be greater than one. With signal limiting diodes, D1 and D2, and the above circuitry, the circuit oscillates and produces a 300mV peak sinusoid. Capacitor, C12, decouples the supply of the oscillator.

Two obtain the two outputs, the oscillator output is split into two signals:

The **first** signal is sent to an emitter follower based around Q3. The input signal is DC blocked by C8. The emitter resistor, R12, is 47Ω and provides the required matching. R9 and R10 provide the biasing for Q3. The low-frequency (LF) breakpoint, due to C8 and the effective input resistance, is placed at about 318kHz. The input resistance is determined by the parallel combination of the bias resistors, R9, R10, and the referred emitter load.

Referred emitter load: $R_{ref} = R_e * ß = 47 * 400$

$R_{ref} = 18.8k\Omega.$

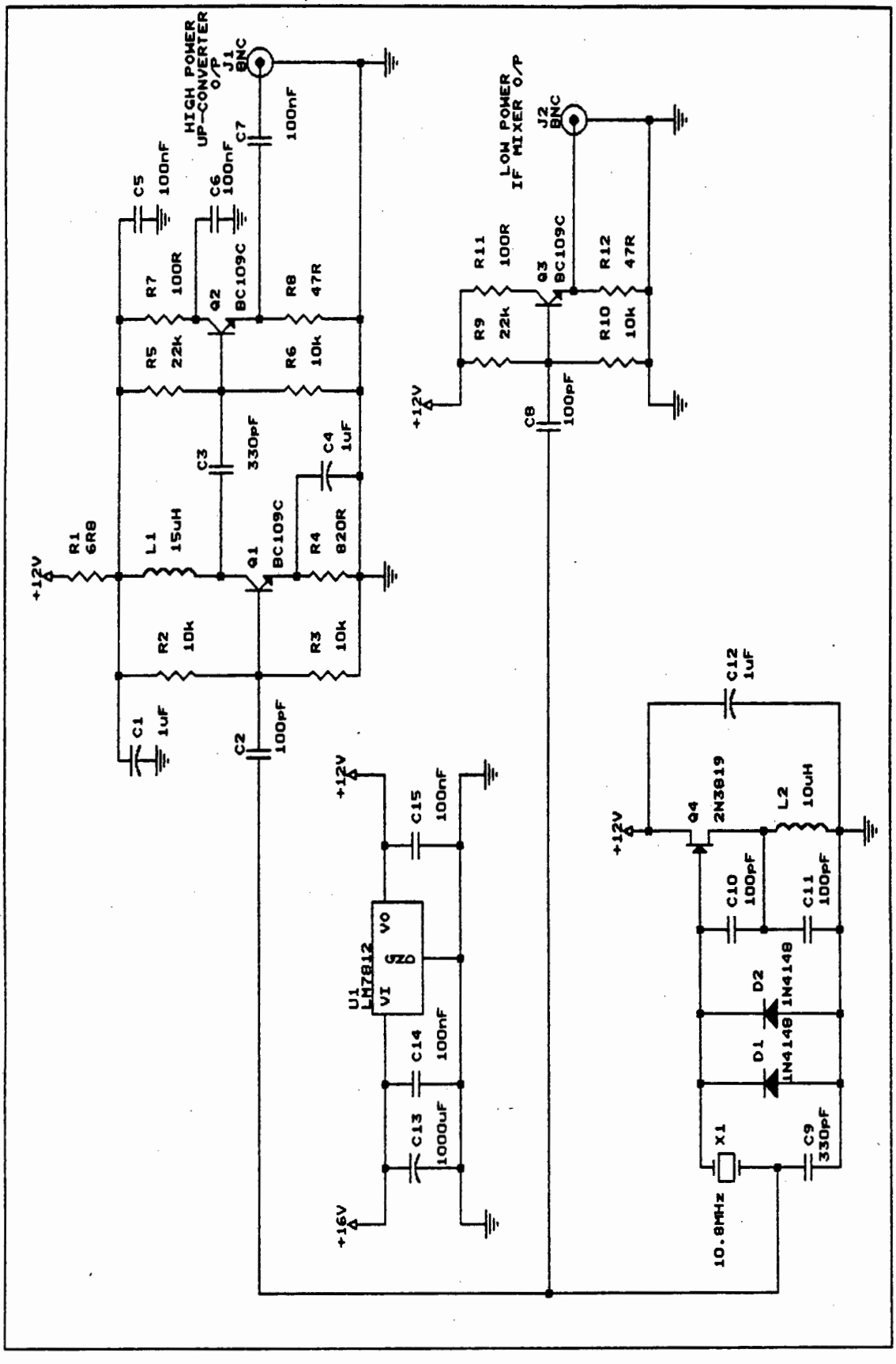


Fig. 3.13. Oscillator Circuit Diagram.

Nett input resistance: $R_{in} = \left[\frac{1}{R9} + \frac{1}{R10} + \frac{1}{R_{ref}} \right]^{-1}$

where R9 = 22kΩ,
R10 = 10kΩ.

$R_{in} = 5k\Omega$

LF breakpoint: $f_{in} = \frac{1}{2 * \pi * C8 * R_{in}}$

$= \frac{1}{2 * \pi * 100pF * 5k\Omega}$

$= 318.5kHz.$

R11 is used to decrease the voltage drop across Q3, and hence decrease its power dissipation. The Quiescent current is about 45mA.

Base bias voltage at Q3:

$$V_b = V^+ * \frac{\left[\frac{R10 * R_{ref}}{R10 + R_{ref}} \right]}{\left[\frac{R10 * R_{ref}}{R10 + R_{ref}} + R9 \right]}$$

$= 2.7V$

where $V^+ = +12V$.

Thus, the emitter voltage at Q3 is:

$$V_e = V_b - 0.6V$$

$$= 2.1V$$

The quiescent current of Q3 is therefore:

$$I_e = \frac{Ve}{R_e}$$

$$= 45mA.$$

The **second** signal is sent to a common emitter configuration amplifier based around Q1. The input signal is DC blocked by C2. The LF breakpoint is also set to 318kHz. The biasing resistors, R2 and R3, form a potential divider which biases Q1 to 6V. Emitter resistor, R4, is chosen to give a 6.6mA emitter current.

Since R4 is high, the referred resistance at the base of Q1 is very high relative to the bias resistors, and therefore its effect is negligible.

The bias voltage at Q1 is therefore: $V_b = V^+ . \left[\frac{R3}{R3 + R2} \right]$

$$= 6V.$$

Nett input resistance at Q1: $R_{in} = \left[\frac{1}{R2} + \frac{1}{R3} \right]$

where $R2 = 10k\Omega$,
$R3 = 10k\Omega$.

$$R_{in} = 5k\Omega$$

LF breakpoint: $$f_{in} = \frac{1}{2 * \pi * C2 * R_{in}}$$

$$= \frac{1}{2 * \pi * 100pF * 5k\Omega}$$

$$= 318.5kHz.$$

Thus, the emitter voltage of Q1 is:

$$V_e = V_b - 0.6V$$

$$= 5.4V$$

The quiescent current of Q1 is therefore:

$$I_e = \frac{V_e}{R_e}$$

$$= 6.6mA.$$

Emitter capacitor, C4, decouples R4 and increases the gain without upsetting the DC bias. Its value is chosen so that its impedance is much lower than R4 at the frequency of interest (10.8MHz). RF choke (peaking inductor), L1, is used to decrease the effect of the Miller capacitance and therefore increase the gain of the amplifier. The value of L1 is chosen to give an impedance of 1kΩ at 10.8 MHz. This allows the best possible AC voltage swing at the collector.

Voltage at collector: $$V_C = Ic.Z_L$$

$$= 6.6mA * 1k\Omega$$

$$= 6.6V$$

where Z_L is the impedance of L1 at 10.8MHz,

$$Z_L = 2 * \pi * L1$$

$$= 1k\Omega$$

The maximum voltage swing at the collector is:

$$V_{swing} = 12V - 6.6V$$

$$= 5.4V \text{ peak.}$$

The nett voltage gain of the amplifier is 3.3X at 10.8MHz or +10dB.

The output of Q1 is fed to another emitter follower, based around Q2, with the same design as the first output (see above). Due to L1, Q1's collector has a DC bias of about 12V, therefore Q1 is AC coupled via C3 to Q2. The value of C3 is chosen such that its impedance at 10.8MHz (48Ω) is much less than nett input impedance of the next stage (5kΩ). The only addition to the previous emitter follower is a decoupling capacitor, C6, which is placed at the collector to improve the follower's response. Capacitor, C7, is used to block any DC from loading this stage. The output voltage is 1V peak. An RC filter based around R1 and C1, is placed at the supply of Q1 and Q2 to reduce RF interference on the supply. The high frequency (HF) breakpoint is set at 23kHz and R1 is set small enough to have negligible voltage drop effects.

HF breakpoint: $$f_{supply} = \frac{1}{2 * \pi * C1 * R1}$$

$$= \frac{1}{2 * \pi * 1\mu F * 6.8\Omega}$$

$$= 23kHz.$$

Both Q2 and Q3 require heatsinks.

A separate positive 12V regulator, U1, was built into this module for additional protection. Its smoothing capacitor and decoupling capacitors are C13, C14 and C15, respectively.

3.6 IF AMPLIFIER, IF FILTER, AND IF MIXER

3.6.1 Specifications

Table 3.5. IF Amp., Filter & Mixer Specifications.

IF Amplifier, Filter and Mixer Specifications.	
IF amplifier bandwidth:	14 MHz (10MHz to 34MHz)
IF amplifier power gain:	28 dB
IF amplifier voltage gain:	66 X
IF amplifier noise floor:	-135 dBm
IF amplifier noise figure:	0.8 dB
IF amplifier maximum input signal: (before clipping)	- 45 dBm 1.2 mVrms
IF filter frequency:	10.8 MHz
IF filter loss:	12 dB
Mixer NF:	0.005 dB
Mixer power gain:	18 dB
Mixer voltage gain:	60 X
Mixer output bandwidth:	26 kHz
Supply Requirements:	+ 15 Volts

The IF amplifier, filter, and mixer are combined into one module.

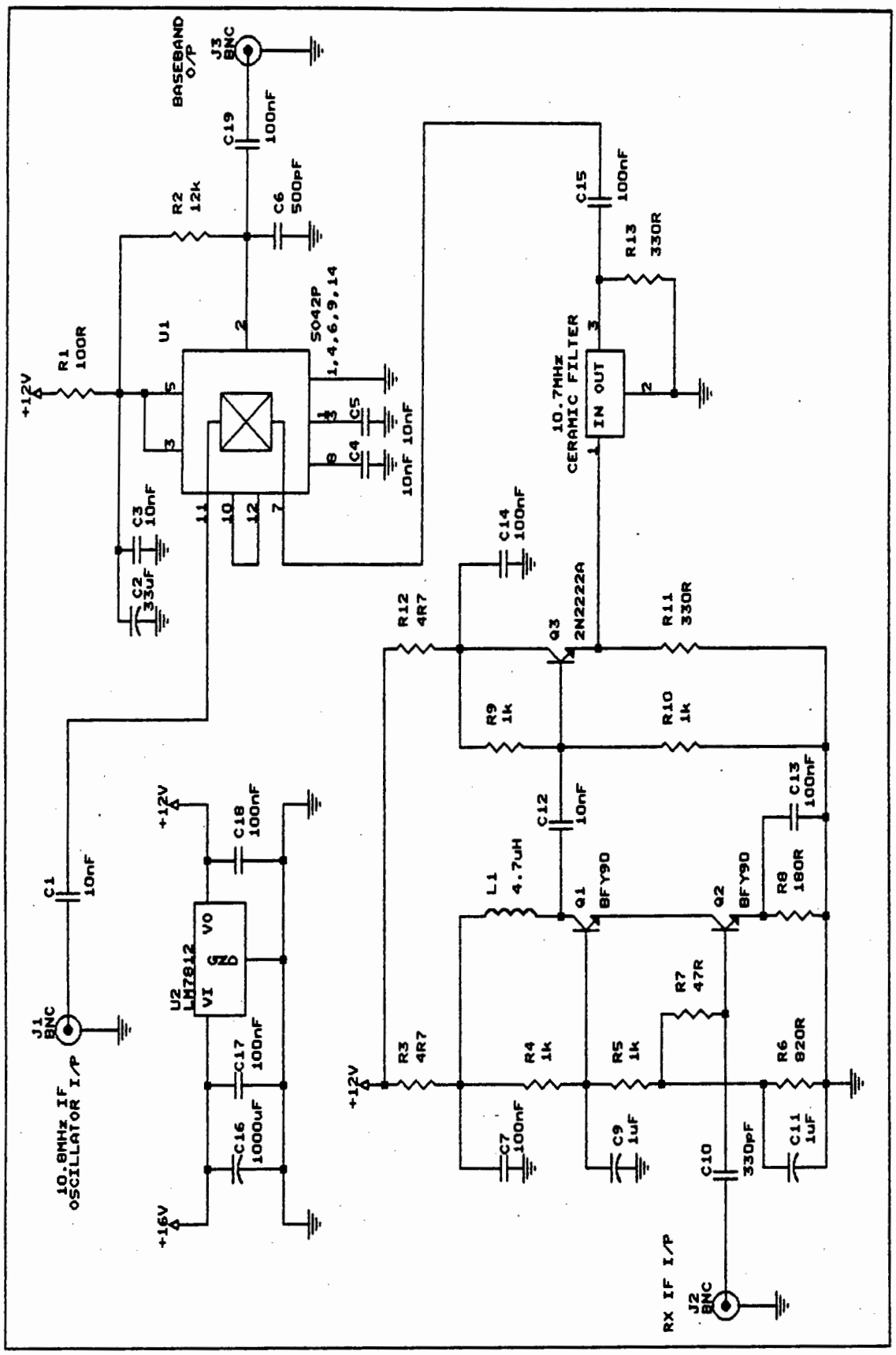


Fig. 3.14. IF Amplifier, IF Filter, and IF Mixer Circuit Diagrams.

3.6.2 IF Amplifier

The IF amplifier is a wide bandwidth cascode design [3.5] - see figure 3.14. The advantage of this design is that it allows high gains to be realized by overcoming the Miller Capacitance problem associated with transistors. The cascode transistors, Q1 and Q2 can be seen in the circuit diagram. Q1 is in common base configuration. The RF input signal is DC blocked by C10, and the LF breakpoint is set to 10.3MHz. Both transistors are biased by the potential divider formed around R4, R5, and R6 and Q2 is biased through R7 so that an A.C. input impedance of 47Ω can be set. Capacitors, C9, C11 are used to A.C. couple the bias points to ground. This is important as Q1 must have a grounded base for the cascode circuit to work and Q2's base must see 47Ω for matching purposes, and also to keep the thermal (Johnson) noise low. The values of C9 and C11 are chosen to have very low impedances at the frequency of interest.

LF breakpoint at input: $f_{in} = \dfrac{1}{2 * \pi * C10 * R7}$

$$= \frac{1}{2 * \pi * 330pF * 47\Omega}$$

$$= 10.27MHz.$$

Q1 base bias: $V_b = V^+ . \left[\dfrac{R5 + R6}{R4 + R5 + R6} \right]$

where R4 = 1kΩ
R5 = 1kΩ
R6 = 820Ω
V^+ = 12V

$$V_b = 7.8V$$

Q2 base bias: $V_b = V^+ . \left[\frac{R6}{R4 + R5 + R6} \right]$

$$V_b = 3.5V$$

Thus, the emitter voltage at Q2 is:

$$V_e = V_b - 0.6V$$

$$= 2.9V$$

The quiescent current is therefore:

$$I_e = \frac{Ve}{R_e} = \frac{Ve}{R8}$$

$$= 16mA.$$

Emitter capacitor, C13 decouples R8 and increases the gain without affecting the DC bias. Its value is chosen so that its impedance is much lower than R8 at the frequency of interest (10.8MHz). The value of RF choke (peaking inductor), L1, is chosen to give an impedance of 319Ω at 10.8 MHz. This allows the best possible AC voltage swing at the collector.

Voltage at collector: $V_c = Ic.Z_L$

$$= 16mA * 319\Omega$$

$$= 5.1V$$

where Z_L is the impedance of L1 at 10.8MHz,

$$Z_L = 2 * \pi * L1$$

$$= 319\Omega$$

The maximum voltage swing at the collector is:

$$V_{swing} = 5.1V$$

The nett voltage gain of the amplifier is 66X at 10.8MHz. The linear power gain is:

$$\left[\frac{P_{out}}{P_{in}}\right] = \left[\frac{V_{out}^2}{R_{out}}\right] \div \left[\frac{V_{in}^2}{R_{in}}\right]$$

$$= \left[\frac{V_{out}^2}{V_{in}^2}\right] * \left[\frac{R_{in}}{R_{out}}\right]$$

$$= 66^2 * \frac{50}{330}$$

$$= 660$$

The logarithmic power gain is:

$$G_{power} = 10 * \log(660) = 28.2dB$$

Due to L1, Q1's collector has a D.C. bias of about 12V. Therefore Q1 is A.C. coupled via C12 to Q3. The emitter resistor of Q3, R11, is 330Ω and provides the required matching. R9 and R10 provide the biasing for Q3. Since R11 is high, the referred resistance at the base of Q1 is very high relative to the bias resistors, and therefore its effect is negligible. The low-frequency (LF) breakpoint, due to C12 and the effective input resistance, is placed at about 3.18kHz. The input resistance is determined by the parallel combination of the bias resistors, R9 and R10.

Nett input resistance: $R_{in} = \left[\frac{1}{R9} + \frac{1}{R10} \right]^{-1}$

where R9 = 10kΩ,
R10 = 10kΩ.
R_{in} = 5kΩ

LF breakpoint: $f_{in} = \frac{1}{2 * \pi * C12 * R_{in}}$

$= \frac{1}{2 * \pi * 10nF * 5k\Omega}$

$= 3.18kHz.$

Base bias voltage at Q3: $V_b = V^+ . \left[\frac{R10}{R9 + R10} \right]$

$= 6.0V$

where $V^+ = +12V.$

Thus, the emitter voltage at Q3 is:

$$V_e = V_b - 0.6V$$

$$= 5.4V$$

The quiescent current of Q3 is therefore:

$$I_e = \frac{Ve}{R_e}$$

$$= 16mA.$$

Decoupling capacitor, C14, is placed at the collector of Q3 to improve its response.

Also, RC filters are placed at the supply to reduce RF interference on the supply of both the cascode circuit and the emitter follower. They are R3, C7, R12, and C14 respectively. The high frequency (HF) breakpoints are set at 339kHz. R3 and R12 are set small enough so as to have negligible voltage drop effects.

HF breakpoint: $$f_{supply} = \frac{1}{2 * \pi * C7 * R3}$$

$$= \frac{1}{2 * \pi * 100nF * 4.7\Omega}$$

$$= 339kHz.$$

The overall bandwidth of the IF amplifier is limited by the LF breakpoint created by C10 and R7, and also by the HF breakpoint created by the parasitic base-emitter capacitance and R7. As previously calculated, the LF breakpoint is placed at 10.3MHz. The high frequency breakpoint is situated at 34MHz.

HF breakpoint: $$f_t = \frac{1}{2 * \pi * C_{be} * r_e} \quad \text{(of Q2)}$$

$$\text{where } r_e = \frac{25}{I_C}$$

$$= 25 / 16mA$$

$$= 1.6\Omega$$

$$ft = 1GHz \text{ for BFY90}$$

Thus: $$C_{be} = \frac{1}{2 * \pi * f_t * r_e}$$

$$= 100pF.$$

Therefore: $$f_{HF} = \frac{1}{2 * \pi * C_{be} * R7}$$

$$= 33.9MHz$$

$$\text{where } R7 = 47\Omega.$$

3.6.3 IF Filter

The IF filter is based around a 10.7MHz ceramic filter. The IF frequency is 10.8 MHz, however, since the filter has a 200kHz bandwidth at the -10dB points, this is not serious. The Q of the ceramic filter is about 107.

$$Q = \frac{f}{B} = \frac{10.7MHz}{100kHz} = 107.$$

The ceramic filter's frequency response can be seen in figure 3.15.

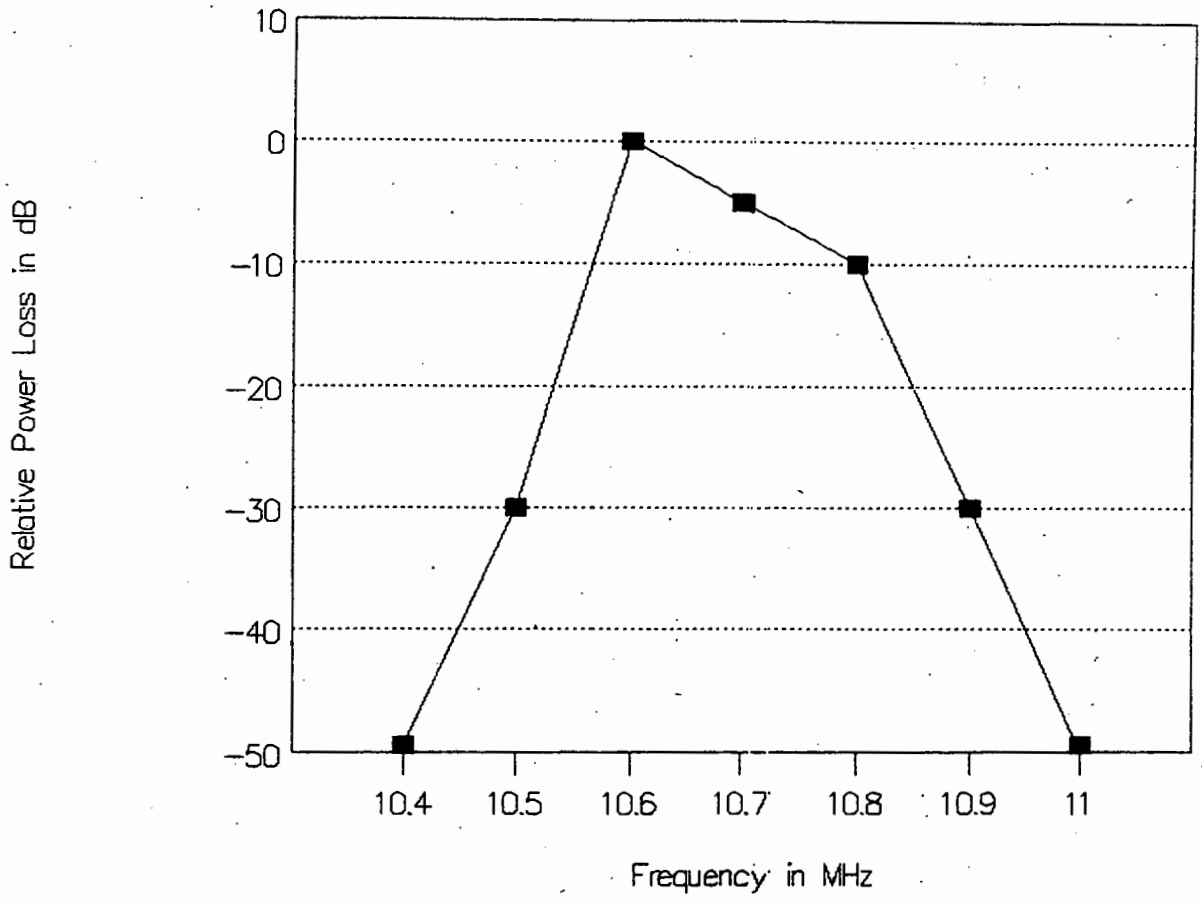


Fig. 3.15. Ceramic Filters Frequency Response.

The filter's input and output impedances are 330Ω, and will have to be accordingly matched. Resistor, R12, is used to provide the matched load for the output of the ceramic filter.

3.6.4 IF Mixer

The IF mixer is an active device. It is based around the Siemens SO42P double balanced mixer I.C., U1. The IF oscillator input is DC blocked by capacitor C1, whose value is chosen so that its impedance is low at the required frequency. The signal input is also DC blocked by capacitor C15. The value of R2 is chosen such that the output is biased to about 6V with the 12V supply. C6 in conjunction with R2 form an RC LPF. The breakpoint is set to 27kHz to only allow the baseband signals to pass through.

HF breakpoint: $$f_{O/P} = \frac{1}{2 * \pi * C6 * R2}$$

$$= \frac{1}{2 * \pi * 500pF * 12k\Omega}$$

$$= 26.5kHz.$$

C19 is used to block DC which could bias the mixer output.

A supply filter, with a breakpoint at 48Hz, is also provided. It consists of R1, C2, and C3.

The nett voltage gain of the mixer is 60 times at 10.8MHz. The linear power gain is:

$$\frac{P_{out}}{P_{in}} = \left[\frac{V_{out}^2}{R_{out}}\right] \div \left[\frac{V_{in}^2}{R_{in}}\right]$$

but, $V_{out} = S * R_{out} * V_{in}$

so, $V_{out} = 60 * Vin$

where S is the conversion transconductance

$$S = 5mS = 1 / R_{in}$$

$$\frac{P_{out}}{P_{in}} = \left[\frac{(S * R_{out} * V_{in})^2}{V_{in}^2}\right] * \left[\frac{R_{in}}{R_{out}}\right]$$

$$= \frac{R_{out}}{R_{in}}$$

$$= \frac{12k\Omega}{200\Omega}$$

$$= 60 \text{ times.}$$

The logarithmic power gain is:

$$G_{power} = 10 * \log(60) = 17.8dB$$

3.6.5 Power Supply

A separate positive 12V regulator, U2, was built into this module for additional protection. Its smoothing capacitor and decoupling capacitors are C16 and C18 respectively.

3.7 BASEBAND AMPLIFIER AND FILTERS

3.7.1 Circuit Description

Table 3.6. Baseband Amplifier & Filters Specifications.

Baseband Amplifier and Filter Specifications.	
Baseband bandwidth: (8.8kHz to 18.8kHz)	10 kHz
Maximum baseband amplifier gain:	52 dB
Baseband amplifier noise floor:	-108 dBV
Maximum signal in before clipping: (on 25X gain)	350 mV pk
Supply Requirements:	±15 V

The baseband amplifier is required to amplify the received signal up to 5V peak. This is a necessary requirement since it is important to use the full dynamic range of the ADC. The initial design catered for received signal amplitudes from small

dielectric targets exhibiting weak reflections. These signals were very small, and in the last design stage the gains were left the same.

The filters are important, since they reduce unwanted signals at the output. They effectively form a bandpass filter (BPF). The lowest frequency of the effective bandpass filter corresponds to the closest range of interest, and the highest frequency to the furthest range of interest. Also included in the filter design is a range filter to keep a constant signal amplitude with range. The main reasons for the filters are:

a) To eliminate the signal caused by direct microwave leakage from antenna to antenna.

b) To reduce the operating bandwidth and hence reduce the signal to noise ratio of the radar system.

c) To act as anti-aliasing filters for the sampling process.

d) To reduce the out-of-range clutter signals.

The gain stages and filters are cascaded to give the required frequency response and gains. The filters are Sallen and Key type filters which have Q = 0.707. Each stage yields a 12 dB per octave roll-off. The gain of each stage has to be set to a certain value to normalize the filter. For two successively cascaded filters, the normalizing gains in each stage is set differently to that of a single stage filter. Each stage must not load the previous stage.

The first stage is a gain stage - see figure 3.16. This has a switchable voltage gain of 16.8X or 5X. The next stage is a 19.2kHz high-pass filter, or the range filter. This filter is used to obtain a +12 dB per octave slope for the range filter. This is required, since:

$$P = k * \frac{1}{R^4}$$

where k = some constant.

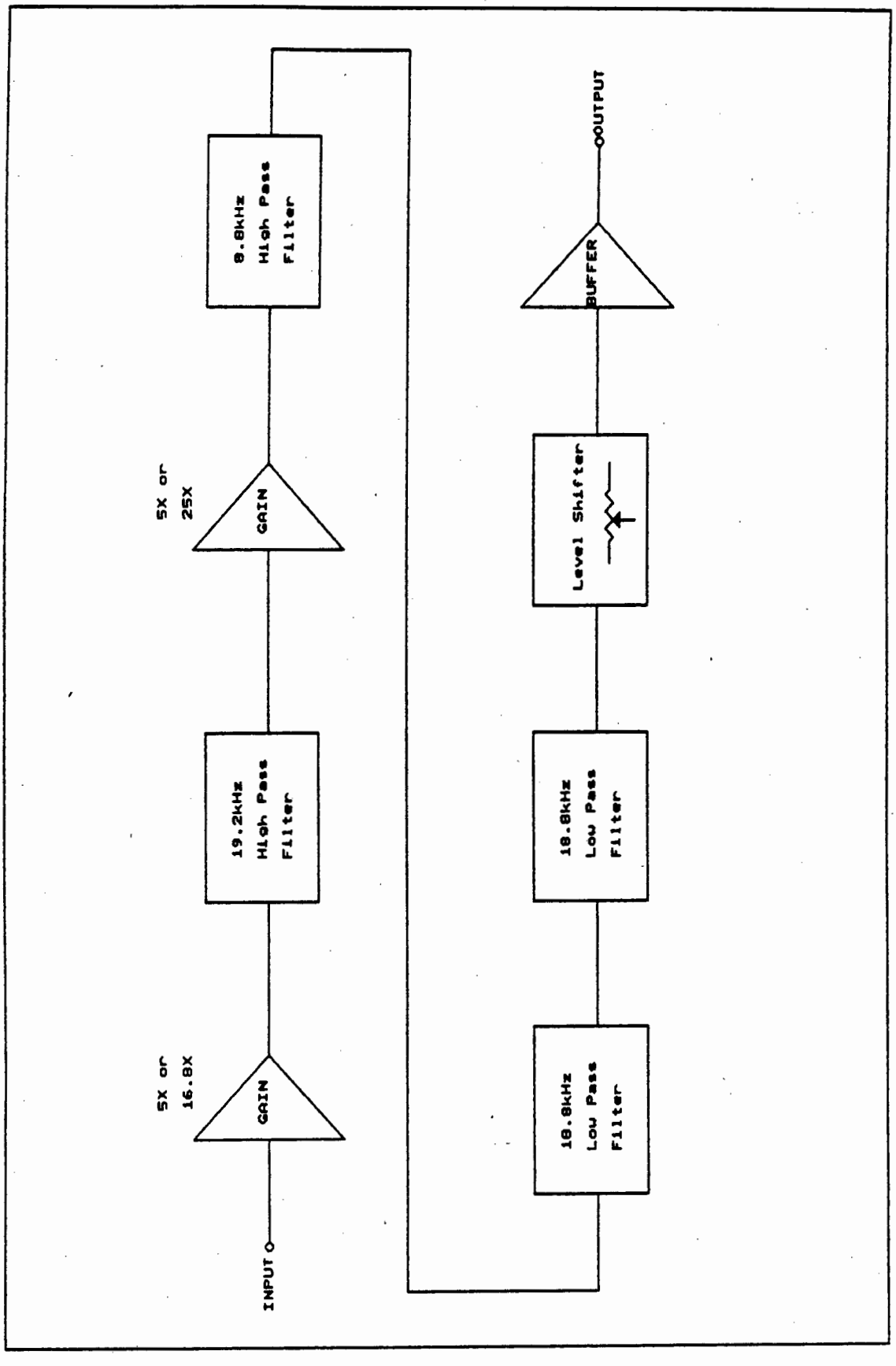


Fig. 3.16. Block Diagram of Baseband Filtering Module.

and: $f = C * R$

where C = some constant.

A twice increase in range yields a twice increase in frequency and a sixteen times decrease in power.

$$10.\log(1/16) = -12 \text{ dB}.$$

The +12 dB per octave filter cancels the -12 dB per octave range effects to yield a constant signal amplitude over the required target range, and corresponding beat frequency range.

The following stage is another gain stage with a switchable voltage gain of 5X, or 25X.

The next filter is an 8.8kHz high-pass filter which defines the lower effective bandpass filter limit, and hence the minimum range wanted. From 8.8kHz to 0Hz, there is a 24dB per octave, LF roll-off. This is due to both high-pass filters - see figure 3.17.

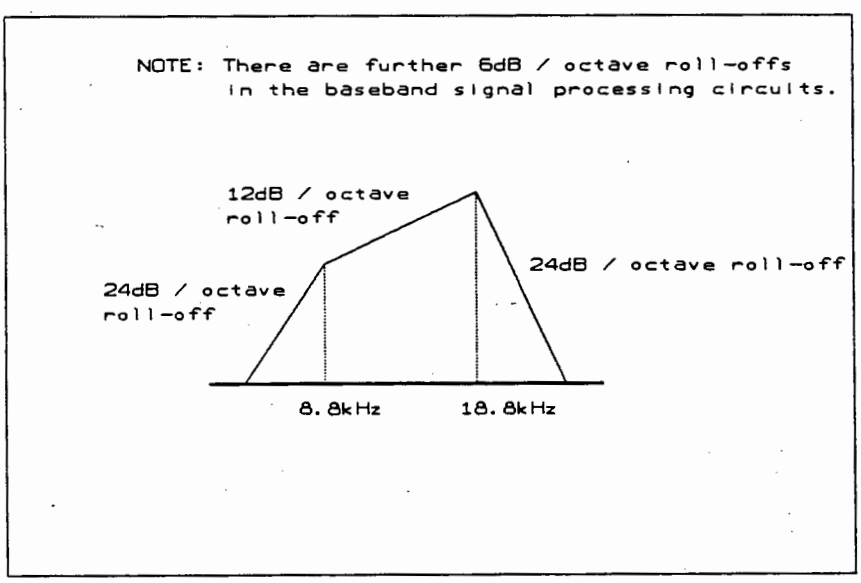


Fig. 3.17. Bandpass Filter Characteristics.

After this, there are two 18.8kHz LP filters. These filters are used to define the upper frequency limit of the effective bandpass filter, and hence the maximum range required. The two filters provide another 24dB per octave, HF roll-off.

Thereafter, a level shifter is employed for the ADC sampling card. This is necessary since the output is AC, and some ADCs only accept uni-polar inputs.

The last stage is a buffer stage which provides a low output driving impedance for the ADC.

In addition to the above filters, further 6dB per octave high-pass and low-pass filters are implemented by RC circuits coupling the successive stages, and the feedback RC circuits of the inverting-configuration gain stages.

3.7.2 Butterworth Filters

The two LPFs and two HPFs used are Butterworth filters [3.6] whose values are calculated by using the formulae that follow. The Butterworth filter is realized with the Sallen and Key configuration, which can be seen in figure 3.18. The cutoff frequency is set by the RC values:

$$f_{3db} = \frac{1}{2 * \pi * C * R}$$

The gain is set by the resistors of the non-inverter configuration op-amp. The gain values are required to normalize the frequency response, and are found in tables - see Appendix G. From these tables, it is seen that the gain of a single stage Butterworth filter is:

$$K = 1.586$$

For two cascaded filters, the gain of the first and second op-amps are set to:

$$K1 = 1.152$$

$$K2 = 2.235$$

The gain formula for the non-inverting configuration is:

$$G_V = \frac{V_{out}}{V_{in}} = \frac{R1}{R2} + 1$$

By inserting the three gain values into the above formula, values for R1 and R2 can be chosen. They are:

For K = 1.586: R1 = 3.3kΩ
R2 = 5.6kΩ.

For K1 = 1.152: R1 = 1kΩ
R2 = 6.8kΩ.

For K2 = 2.235: R1 = 6.8kΩ
R2 = 5.6kΩ.

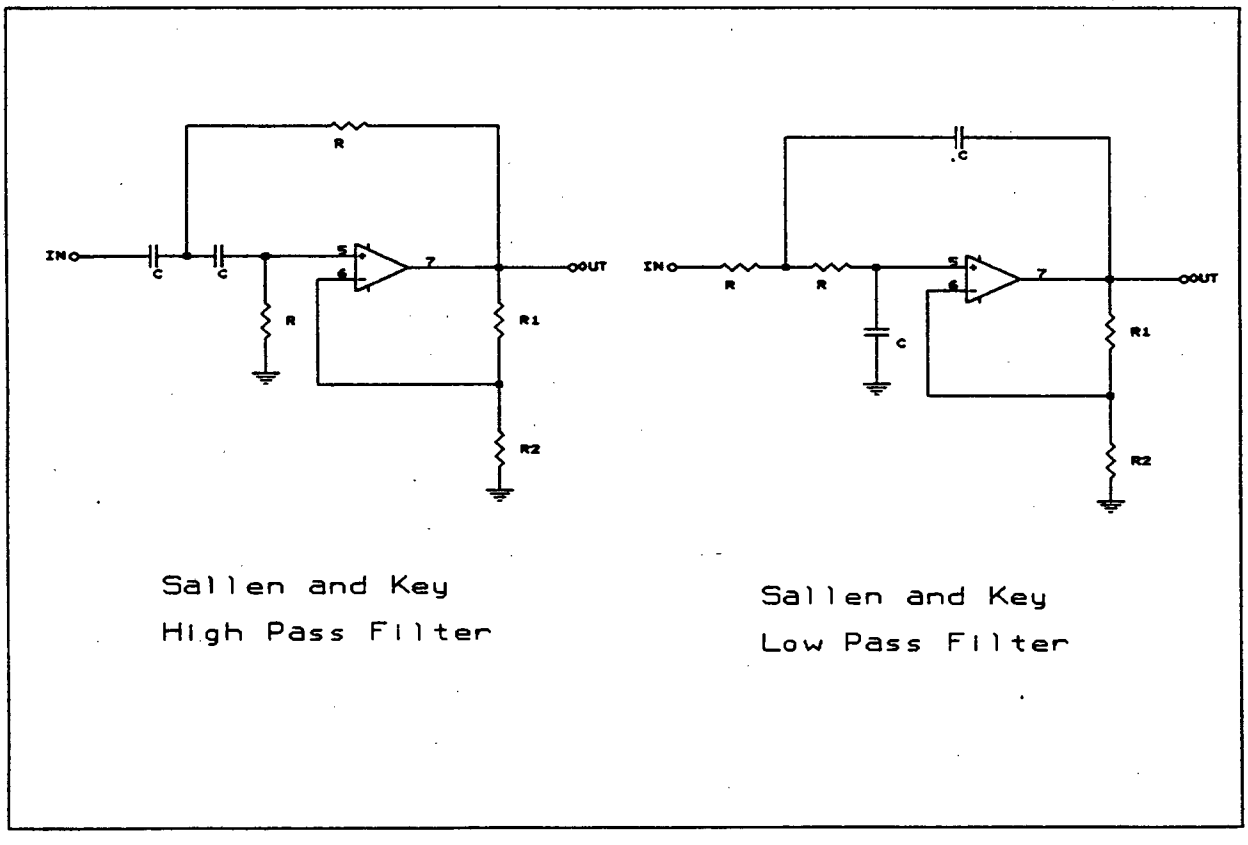


Fig. 3.18. Sallen and Key (a) High Pass, and (b) Low Pass Filters.

3.7.3 Circuit Details

The circuit diagram is shown in figure 3.19. The first gain stage is constructed around U1a. It is in inverting configuration. The gain is:

$$G_v = \frac{V_{out}}{V_{in}} = -\frac{R_f}{R_i}$$

where $R_{f1} = 790k\Omega$ or $R_{f2} = 235k\Omega$, $R_i = 47k\Omega$

$$G_{v1} = 16.8X \equiv 24.5dB$$

$$G_{v2} = 5X \equiv 14dB$$

C1 is used to block DC on the input. The LF breakpoint formed by C1 and R1 is set at 8.7kHz.

LF breakpoint:

$$f_{I/P} = \frac{1}{2 * \pi * C1 * R1}$$

$$= \frac{1}{2 * \pi * 390pF * 47k\Omega}$$

$$= 8.7kHz.$$

The gain is set by adjusting VR1 to R_{f1}, and VR2 to R_{f2}. Jumper, JP1, is used to select the gain. The HF breakpoints are set by the feedback elements, VR1 and C4, and VR2 and C5.

HF breakpoint #1:

$$f = \frac{1}{2 * \pi * C4 * R_{f1}}$$

$$= \frac{1}{2 * \pi * 10pF * 790k\Omega}$$

$$= 20.2\text{kHz}.$$

HF breakpoint #2:

$$f = \frac{1}{2 * \pi * C5 * R_{f2}}$$

$$= \frac{1}{2 * \pi * 33\text{pF} * 235\text{k}\Omega}$$

$$= 20.5\text{kHz}.$$

The next stage is a Butterworth HPF with a cutoff frequency of 19.17kHz. This stage is built around U1b in the Sallen and Key configuration. R4 and R5 are set to give the required gain of 1.586, as described above. The cutoff frequency is set by C8 to C11, and R2 and R3. Note two capacitors are used in parallel to obtain the required value.

The previous stage is AC coupled by C12 to the next stage, a non-inverting gain stage, based around U1c. The voltage gain is jumper selectable to 5X or 25X.

$$G_v = \frac{V_{out}}{V_{in}} = \frac{R6}{VRX} + 1$$

where R6 = 4k7
VR3 = 1175Ω
VR4 = 196Ω

G_{v1} = 5X ≡ 14dB (for VR3)

G_{v2} = 25X ≡ 28dB (for VR4)

The gain is trimmed by VR3, and VR4.

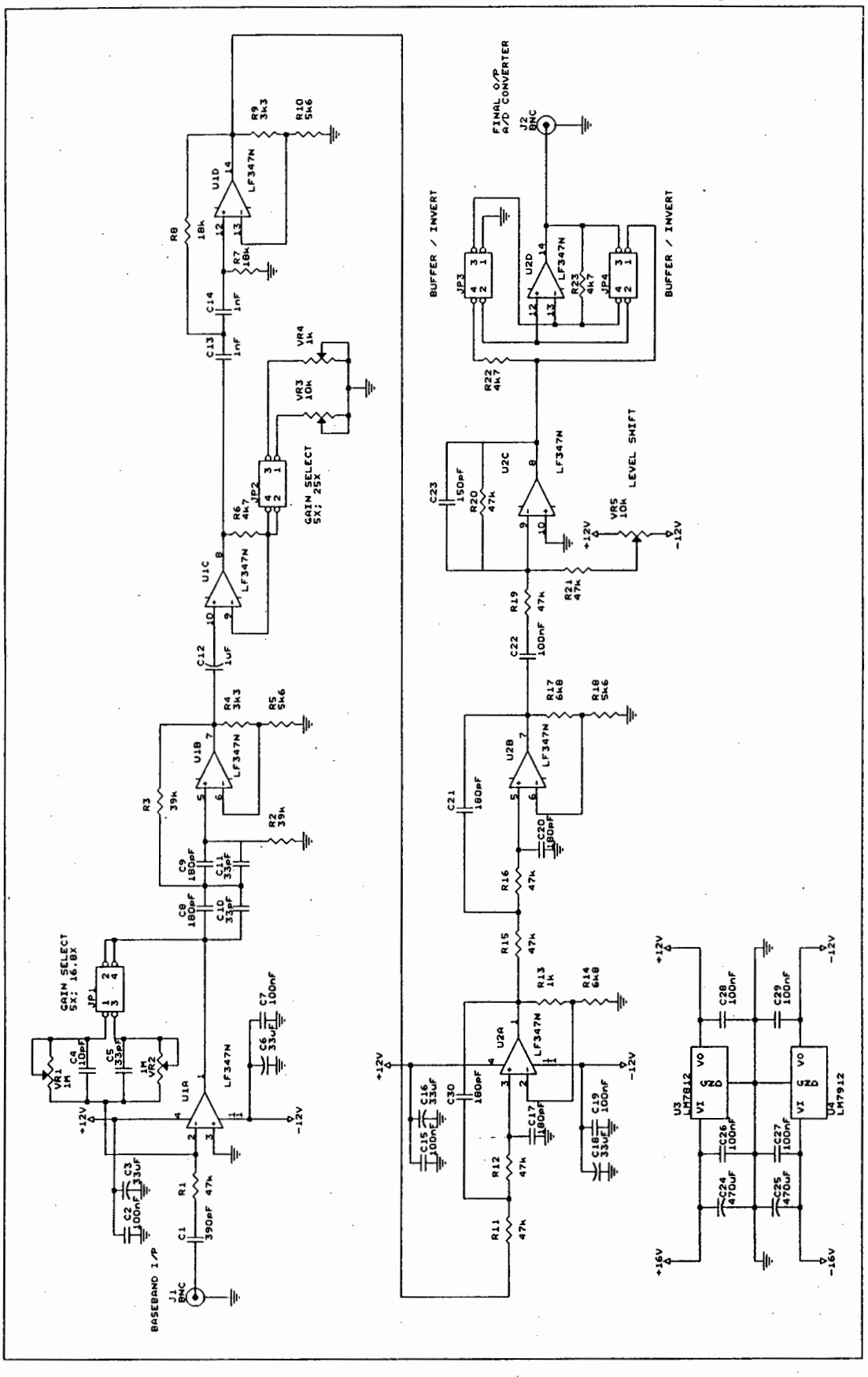


Fig. 3.19. Circuit Diagram of Baseband Processing Module.

The next stage is another Butterworth HPF with a cutoff frequency of 8.8kHz. This stage is built around U1d in the Sallen and Key configuration. R9 and R10 are set to give the required gain of 1.586, as described above. The cutoff frequency is set by C13 and C14, and R7 and R8.

The following two stages consists of Butterworth LPFs with cutoff frequencies of 18.8kHz. They are built around U2a and U2b in the Sallen and Key configuration. R13 and R14 are set to give the required gain of 1.152, as described above. R17 and R18 are also set to give the required gain of 2.235, as described above. The cutoff frequency is set by C17, C30, C20 and C21, and R11, R12, R15 and R16.

The next stage is a level-shifter which is built around inverting configuration op-amp, U2c. The voltage offset is introduced by injecting a current into the summing junction of the op-amp. The current is set by the value of VR5. The gain of the op-amp is set to unity. This stage is A.C. coupled from the previous stage via C22. The LF breakpoint is set to 34Hz by C22 and R19.

LF breakpoint:

$$f = \frac{1}{2 * \pi * C22 * R19}$$

$$= \frac{1}{2 * \pi * 100nF * 47k\Omega}$$

$$= 33.9Hz.$$

The HF breakpoint is set to 22.6kHz by C23 and R20.

HF breakpoint:

$$f = \frac{1}{2 * \pi * C23 * R20}$$

$$= \frac{1}{2 * \pi * 150pF * 47k\Omega}$$

$$= 22.6kHz.$$

The last stage is a buffer stage, based around U2d. The op-amp is in voltage follower mode. The jumper setting allows the op-amp to be set up in inverter mode, however, this facility is not required.

Capacitors, C2, C3, C6, C6, C7, C15, C16, C18, and C19 decouple the supplies of the two op-amp I.C.s. Twelve volt positive and negative voltage regulators, U3 and U4, are used to supply the op-amps. Capacitors, C24 and C25, are used to smooth the supply, and C26 to C29 are decoupling capacitors which are used to stabilize the regulators.

The frequency response of the baseband amplifier and filters module was measured and plotted, and can be seen in figure 3.20.

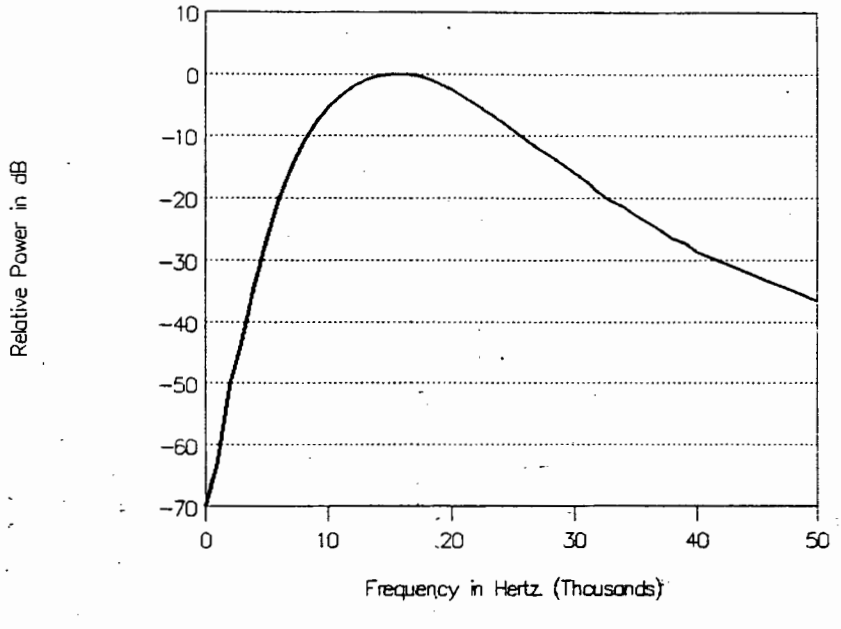


Fig. 3.20. Frequency Response of the Baseband Amplifier and Filter Module.

3.8 POWER SUPPLIES

Table 3.7. Power Supply Specifications.

Power Supply Specifications.	
RF supply:	± 16 V @ 760mA
Modulator Supply:	± 27 V @ 370mA
Oscillator Supply:	+ 10 V @ 1 A

The power supply module can be broken up into three sections:

a) The RF supply,

b) The modulator supply, and

c) The Gunn oscillator supply.

The specifications are shown in table 3.7 above. As can be seen from figure 3.21, each supply is obtained from its own transformer. The 220V mains supply can be broken by switch, SW1. After the mains voltage is stepped down, it is full-wave rectified and smoothed with reservoir capacitors. The smoothing capacitors of all the supplies are C1 to C5.

RF supply output voltage: $V_{RF} = (12V * \sqrt{2}) - 1.2V$

$= 15.8V$

Where the 1.2V is from the two diode drops.

RF supply output current: $P_{in} = P_{out}$

So, $12V * 1A = 15.8V * I_{RF}$

Thus: $I_{RF} = 0.76A$

Modulator supply output voltage: $V_{RF} = (20V * \sqrt{2}) - 1.2V$

$= 27V$

Where the 1.2V is from the two diode drops.

Modulator supply output current: $P_{in} = P_{out}$

So, $20V * 0.5A = 27V * I_{RF}$

Thus: $I_{RF} = 0.37A$

Oscillator supply transformer output:

$V_{RF} = (12V * \sqrt{2}) - 1.2V$

$= 15.8V$

Where the 1.2V is from the two diode drops.

A positive voltage regulator, U1, is then used to obtain the 10V supply required. Capacitors C6 to C8 are used for stability. R1 is the current limiting resistor for 'power-on' L.E.D., D4. VR1 is used to adjust the output voltage to precisely 10.0V.

The output voltage for the regulator is calculated as follows:

$$V_{out} = \frac{1.25}{R3} \cdot \left[R3 + R_{adj}\right]$$

where $R_{adj} = R2 + VR1$

$R3 = 120\Omega$

$R2 = 470\Omega$

So, for $V_{out} = 10V$,

We require: $(R3 + R_{adj}) = 960\Omega$

That is: $R3 = 840\Omega$

And: $VR1 = 370\Omega$

Maximum power dissipation in U1: $P = V_{drop} * I_{max}$

$= (15.8V - 10V) * 1A$

$= 5.8W$

Therefore a heatsink is needed. A different supply voltage is required for the YIG tuned oscillator, which can be obtained by adjusting VR1. **Note** that additional power supplies are required for the YIG tuned oscillator, the details of which can be found in appendix U.

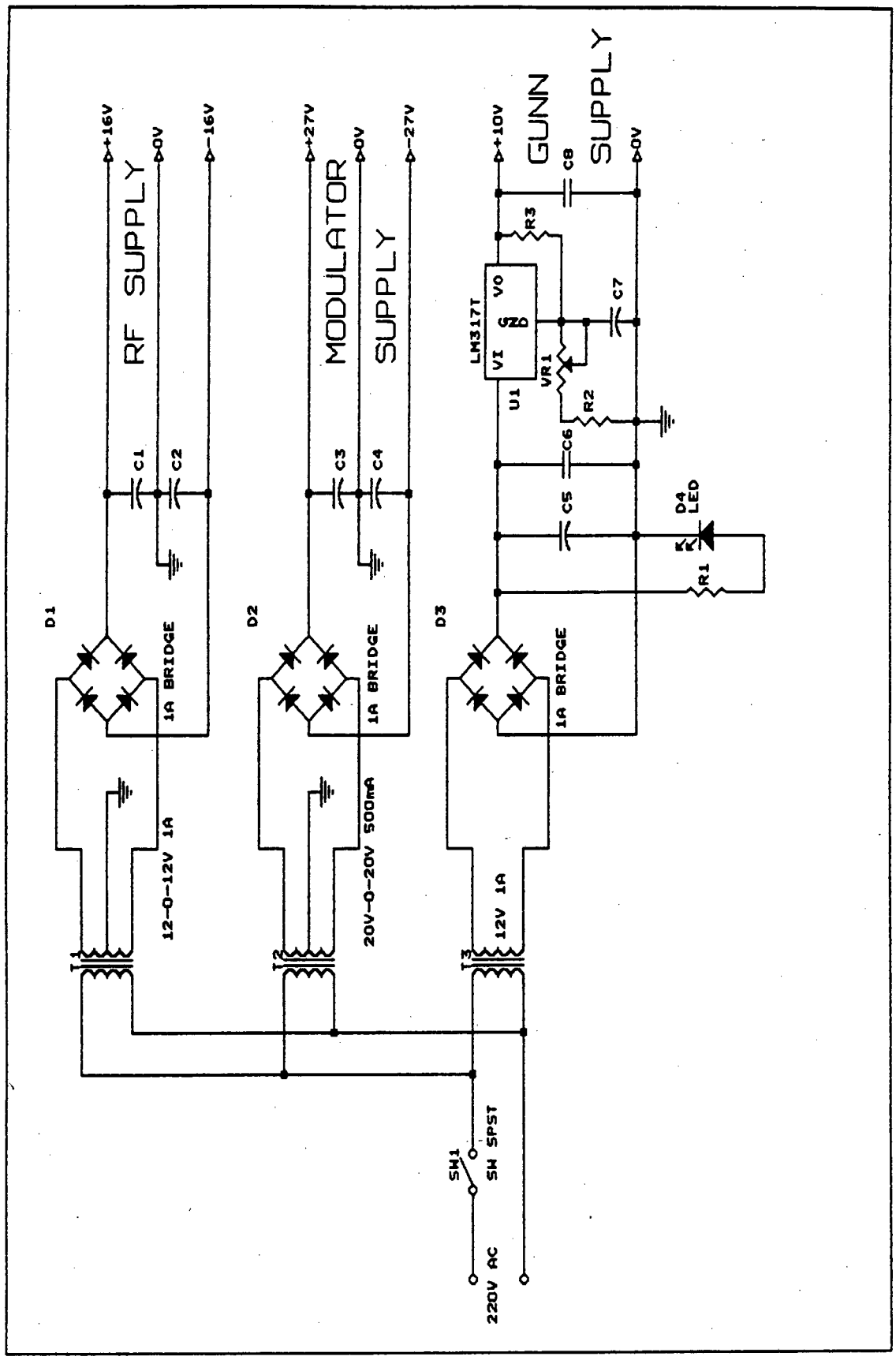


Fig. 3.21. Power Supply Module.

3.9 SYSTEM CONSTRUCTION

The system consists of the microwave components, the RF modules, and the baseband processing module.

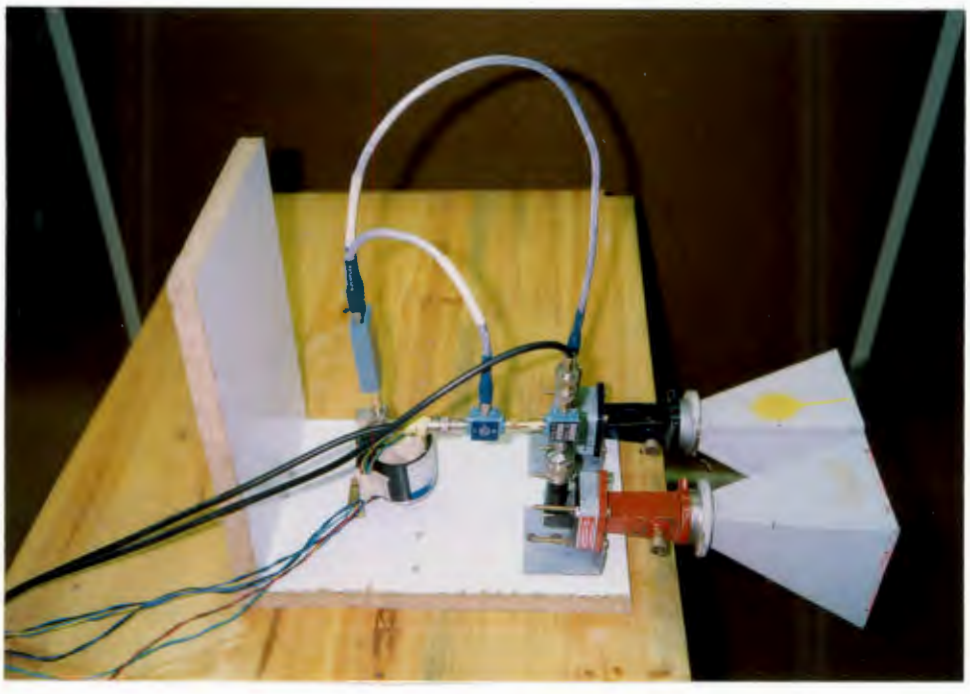


Fig. 3.22. Microwave System Construction.

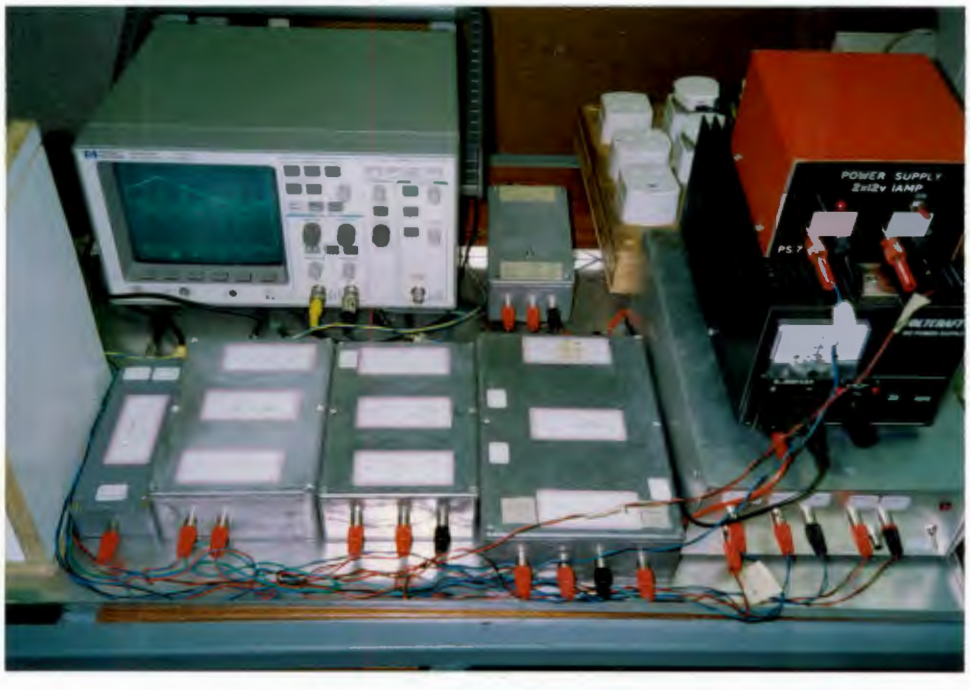


Fig. 3.23. IF and Baseband Modules in Separate Diecast Boxes.

The microwave components were mounted on a right-angled piece of wood - see figure 3.22. The oscillator and antennae were screwed down with self-tapping screws. The antennae were connected via standard square to rectangular waveguide matching sections. The other ends of the matching sections were bolted to 50Ω waveguide to co-axial wire converters. SMA co-axial wire was decided on as the microwave conducting medium, due to its constant waveguide wavelength with varying frequency. This is important because the phase (propagation) velocity changes with frequency. Due to the wide bandwidth used, these variations would become large, causing a variation in beat frequency over the sweep. The length of the SMA microwave cable is important, as it affects the nett path length, and hence the output frequency. The lengths will have to be included for the path length calculations. The two X-band isolators were bolted onto the oscillator to minimize the load pulling effects. These would cause the triangular frequency ramp to be distorted, and hence affect the system. All the microwave co-axial conductor connections were connected with SMA connectors. The IF sections were connected to the microwave components via BNC to SMA converters.

The RF modules were all housed in their own die-cast aluminium boxes to eliminate contamination by HF noise - see figure 3.23. The power supplies to the units were provided through RCA sockets. The RF signals were taken out via BNC sockets. All the RF interconnecting was done through 50Ω RG58 co-axial cable. The construction of the oscillator board (see figure 3.24) and the IF amplifier / filter / mixer boards (see figure 3.25) was done on RF board. RF board has a ground plane which makes it ideal for RF construction.

The modulator and baseband processing modules were housed in die-cast aluminium boxes to eliminate noise problems - see figures 3.26 and 3.27. The power supplies, modulator and SYNC outputs from the modulator unit were interconnected via RCA connectors. The baseband processing unit, however, used BNC sockets for the baseband input and output connections. Both the modulator unit and the baseband processing unit use printed circuit boards for their construction. Both boards utilized a ground plane on the component side to minimize digital feedthrough, and general noise problems. Later modifications done on the modulator unit utilized veroboard for the synchronizing circuit, and RF board for the modulator output modifications.

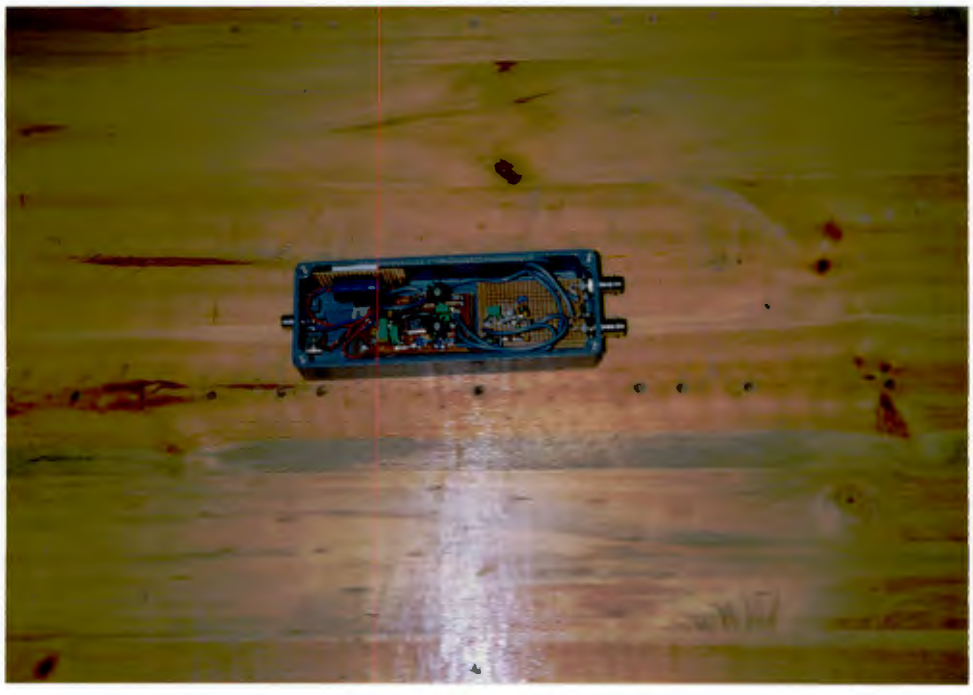


Fig. 3.24. Oscillator Board Construction.

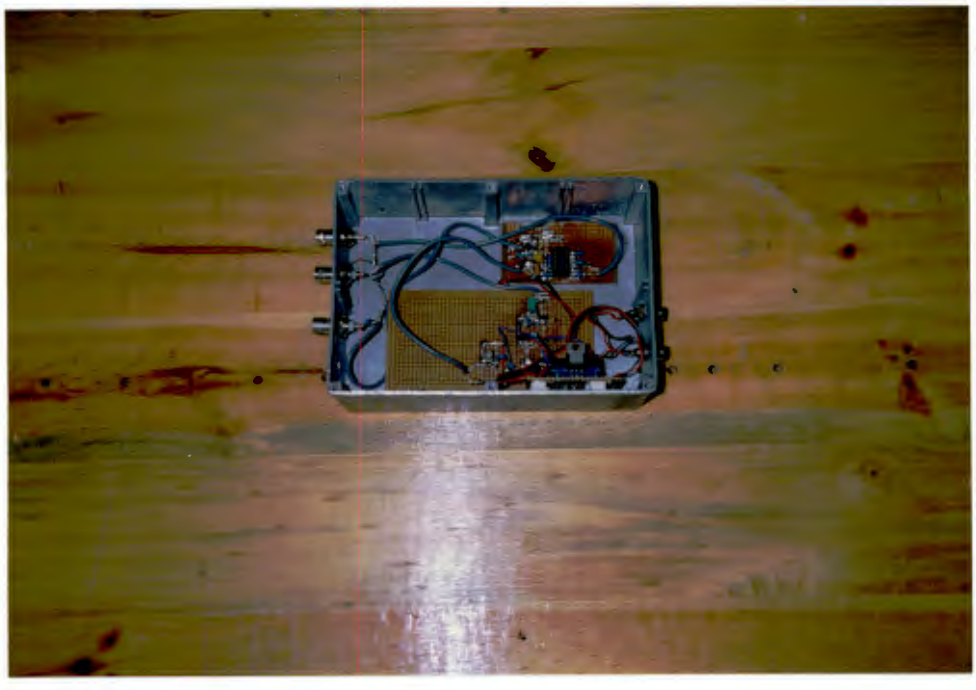


Fig. 3.25. IF Mixer / Filter / Amplifier Construction.

Fig. 3.26. Modulator Construction.

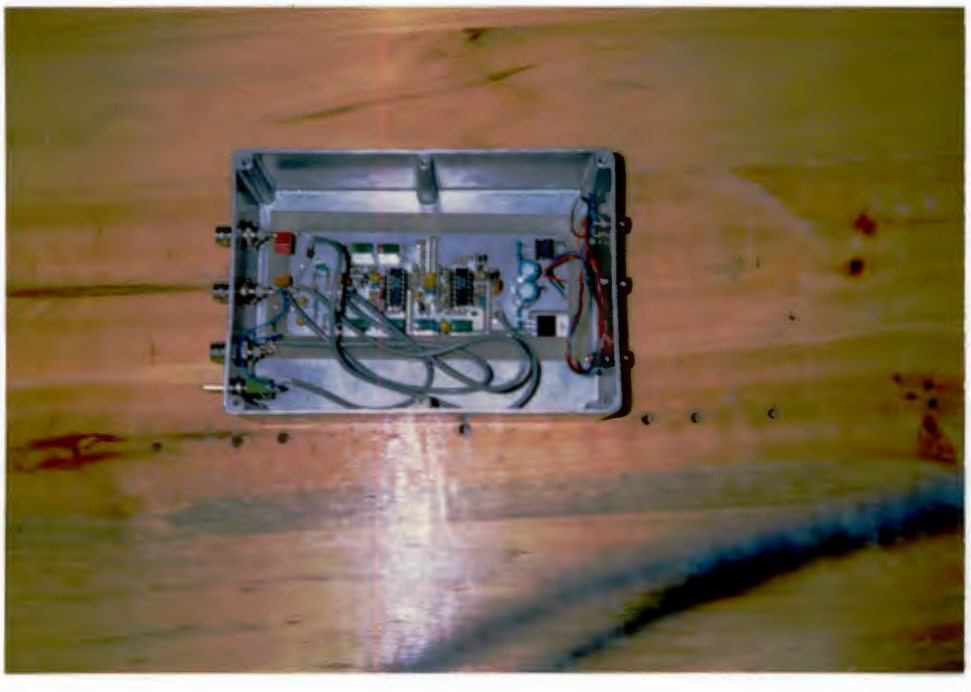


Fig. 3.27. Baseband Processing Unit Construction.

The power supply transformers and rectifying circuits were mounted in a specially constructed aluminium box to eliminate 50Hz interference problems. The supply outputs were connected to RCA sockets - see figure 3.28.

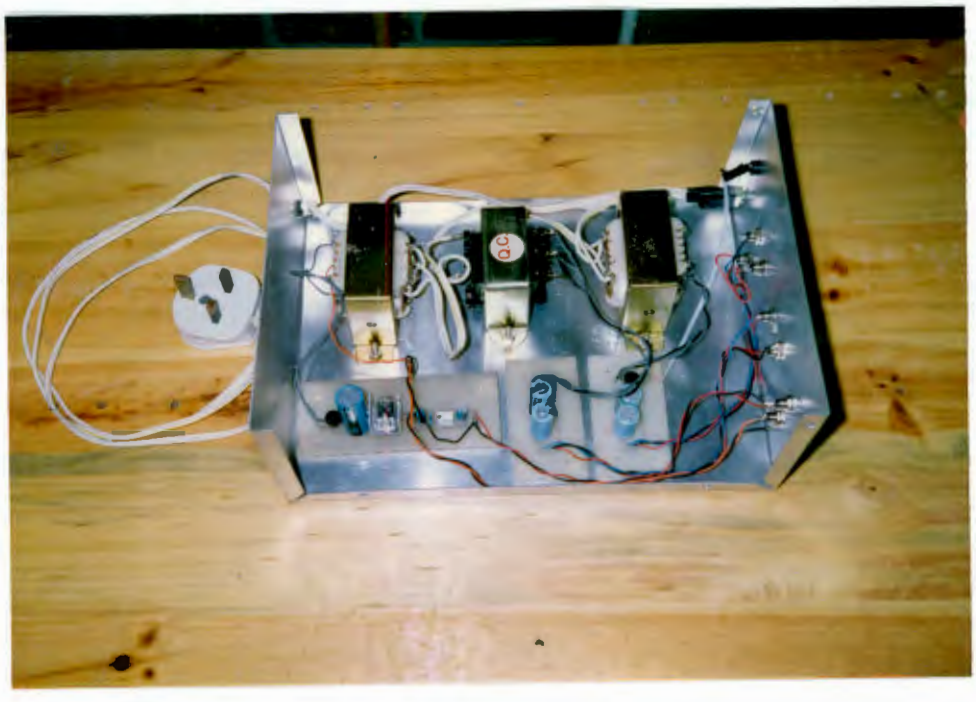


Fig. 3.28. Power Supply Module Construction.

3.10 SYSTEM NOISE ANALYSIS

The noise analysis will be presented and approximate noise measurements performed to verify the noise analysis. Johnson noise (thermal noise) is caused by pure resistances. It is the process which limits the smallest signal detectable in a radar. The system will be broken down into the following subsections (see figure 3.29):

a) Radar noise limitations,

b) IF amplifier noise contribution,

c) IF filter loss,

d) Mixer noise contributions,

e) Baseband amplifiers and filters noise contribution.

Note that all the following noise voltage calculations are RMS.

The signal from the down-conversion mixer at the IF frequency is developed across a matching 50Ω resistance. The radar limiting noise is Johnson noise which is created by pure resistances It therefore follows that purely capacitive or inductive elements contribute no noise, and it will be assumed that the output from the down-converting X-band mixer has a pure resistance of 50Ω. This is a worse-case assumption, since the output of a distributed system is unlikely to be purely resistive.

The noise power as mentioned previously is [3.2]:

$$P_N = k.T.B$$

$$P_N = 4.00 * 10^{-17} W$$

where: Boltzman's constant: $k = 1.38 * 10^{-23}$

temperature: $T = 290\ K$

system bandwidth: $B = 10kHz$

The square noise voltage across the 50Ω load resistance, R_L, is:

$$V_{Nin}^2 = 4 * P_N * R_L$$
$$= 8.00 * 10^{-15}\ V^2$$

The IF amplifier input stage consists of two transistors in cascode configuration and can be modelled as voltage and current noise sources and a noiseless gain stage [3.5 and 3.7]. The 50Ω matching resistor must also be taken into account - see figure 3.30.

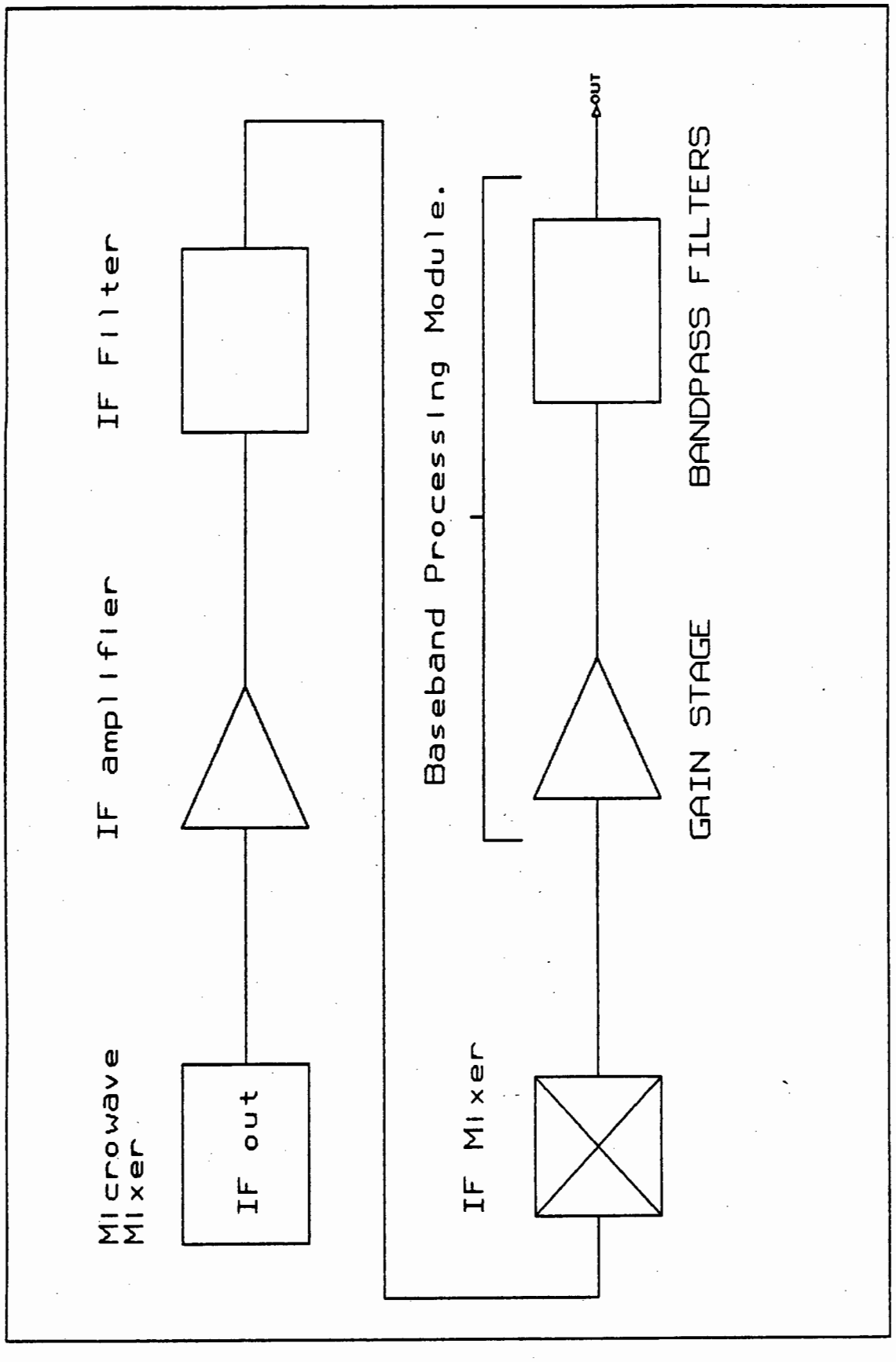


Fig. 3.29. Subsystem Noise Analysis.

The square noise voltage generated by the resistors and the transistor is [3.5]:

$$V_{amp}^2 = 4.k.T.B.R_{match} + V_{trans}^2.B + (i_{trans} * R_S)^2.B$$
$$= 8.00 * 10^{-15} + 1.00 * 10^{-14} + 2.50 * 10^{-17}$$
$$= 1.80 * 10^{-14}\ V^2$$

where V_{trans} = noise voltage of the transistor
= 1nV / $\sqrt{Hz}$.
i_{trans} = noise current of the transistor
= 1pA / $\sqrt{Hz}$.
$R_{match} = R_S = 50\Omega$.

Note that the transistor's noise voltage dominates due to the low input resistance. Also, the voltage and current noise of the second transistor of the cascode configuration is negligible due to the A.C. grounded base.

The total noise voltage at the input of the theoretical, noiseless gain stage is:

$$V_{total}^2 = V_{Nin}^2 + V_{amp}^2$$
$$= 8.00 * 10^{-15} + 1.80 * 10^{-14}$$
$$= 2.60 * 10^{-14}\ V^2$$

$$NF = 10 * \log \left[\frac{V_{total}^2}{V_{Nin}^2} \right]$$

$$= 10 * \log \left[\frac{2.60 * 10^{-14}}{8.00 * 10^{-15}} \right]$$

$$= 5.1dB$$

Note the emitter follower stage of the IF amplifier does not contribute a significant amount of noise to the cascode stage. The square noise voltage generated by the transistor itself is 66^2 smaller than the input noise. The biasing resistors and the emitter resistor of the emitter follower stage yield a square noise voltage approximately 1400 times smaller than the noise voltage at the emitter follower's input. This is due to the large voltage gain (66 times) of the cascode stage:

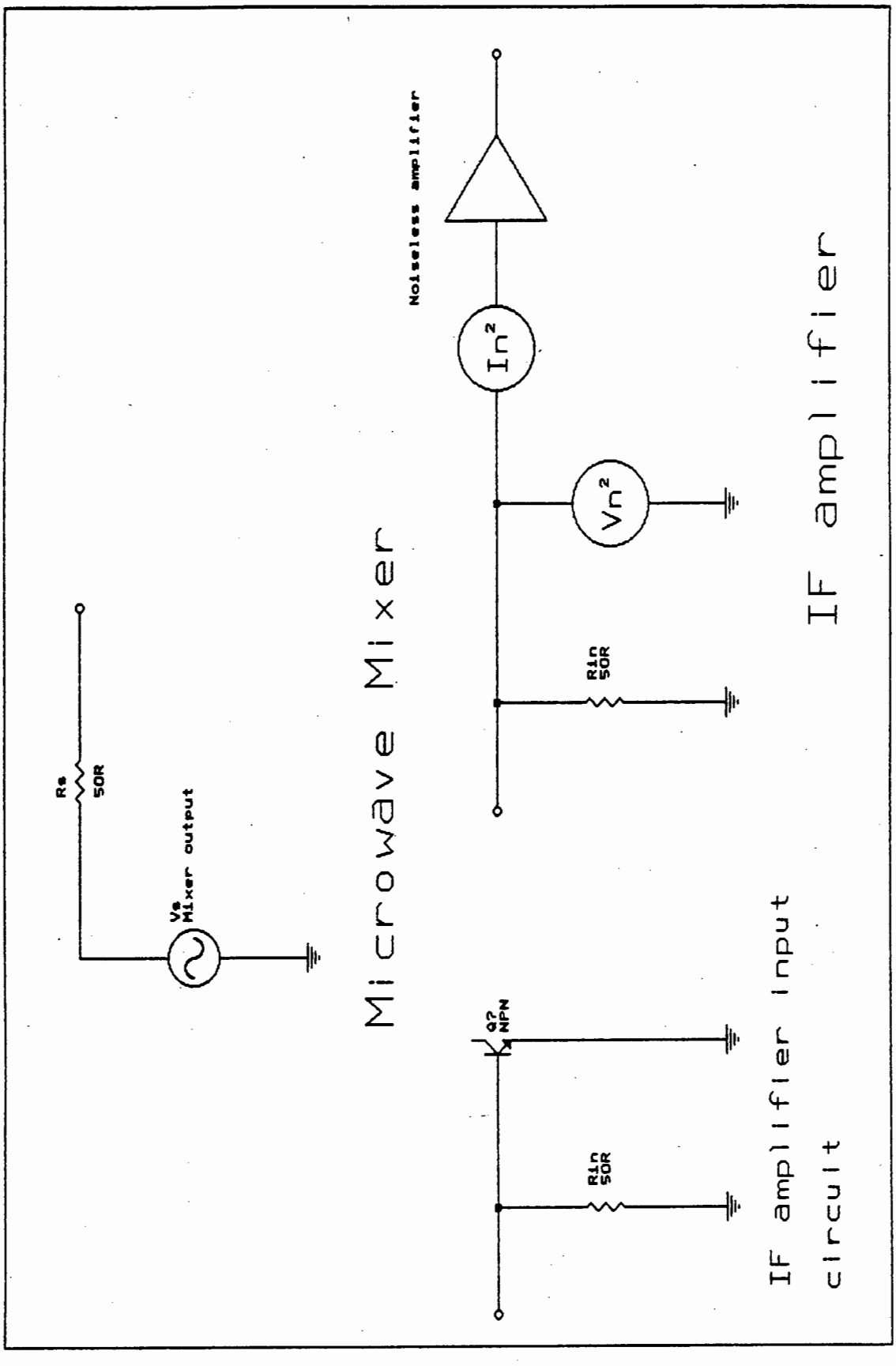


Fig. 3.30. IF Amplifier Input Model.

$$V_{bias}^2 = 4.k.T.B.R_{bias}$$
$$= 8.00 * 10^{-14}\ V^2$$

where R_{bias} = 500Ω and is the Thevenin equivalent of the two bias resistors.

The square noise voltage at the input to the emitter follower is:

$$V_{inf}^2 = G_{amp}^2 * V_{total}^2$$
$$= 66^2 * 2.60 * 10^{-14}$$
$$= 1.13 * 10^{-10}\ V^2$$

The emitter resistor has a smaller value than the equivalent bias resistance and therefore contributes a negligible amount of noise to the system. The IF amplifier noise floor was measured and found to be 40nV or $1.58 * 10^{-15} V^2$. This square noise voltage is about five times smaller than the theoretical value. The measured noise figure is thus [3.5]:

$$NF = 10 * \log \left[\frac{V_{Nin}^2 + V_{measured}^2}{V_{Nin}^2} \right]$$

$$= 10 * \log \left[\frac{8.00 * 10^{-15} + 1.58 * 10^{-15}}{8.00 * 10^{-15}} \right]$$

$$= 0.8dB$$

Despite the fact that the ceramic IF filter has a loss of 12dB, the effective decrease in the SNR is very small due to the relatively high signal levels at its input. Note here that the 330Ω load resistance also contributes a negligible amount of noise to the system.

The mixer has a noise figure (NF) of 2dB when referring to the input 330Ω resistance, however, relative to the system noise, the NF is less than 0.005dB:

$$V_{mixerin}^2 = 4.k.T.B.R_{in}$$

where R_{in} = 200Ω and is the equivalent input resistance of the mixer.

$$= 3.20 * 10^{-14}\ V^2$$

$$NF = 10 * \log \left[\frac{V_{inf}^2 * ML + V_{mixerin}^2}{V_{inf}^2 * ML} \right]$$

$$= 10 * \log \left[\frac{2.828 * 10^{-11}}{2.825 * 10^{-11}} \right]$$

$$= 0.005dB$$

where ML = $^1/_4$ (mixer loss)

The power gain as previously mentioned is 18dB. Note that the output resistor of the mixer also has a square noise voltage 53000 times smaller than the system noise voltage from the mixer output. It therefore has a negligible effect on the system SNR:

$$V_{mixR}^2 = 4.k.T.B.R_{out}$$

$$= 1.92 * 10^{-12}\ V^2$$

where R_{out} = 12kΩ output resistor of the mixer.

The noise at the output of the mixer is:

$$V_{mixout}^2 = G_{mix}^2 * (V_{inf}^2 * ML + V_{mixerin}^2)$$
$$= 60^2 * 2.828 * 10^{-11}$$
$$= 1.02 * 10^{-7}\ V^2$$

The baseband amplifier and filter stages can be subdivided into amplifier and filter op-amp stages. The op-amp stages can also be modelled as voltage and noise current generators and a noiseless gain stage - see figure 3.31.

The square noise voltage of the first inverting op-amp stage can be obtained by the following formula [3.5]:

$$V_{stage1}^2 = V_{opamp}^2.B + R_i^2 * \left[opamp^2.B + \frac{4.k.T.B}{R_f} \right]$$

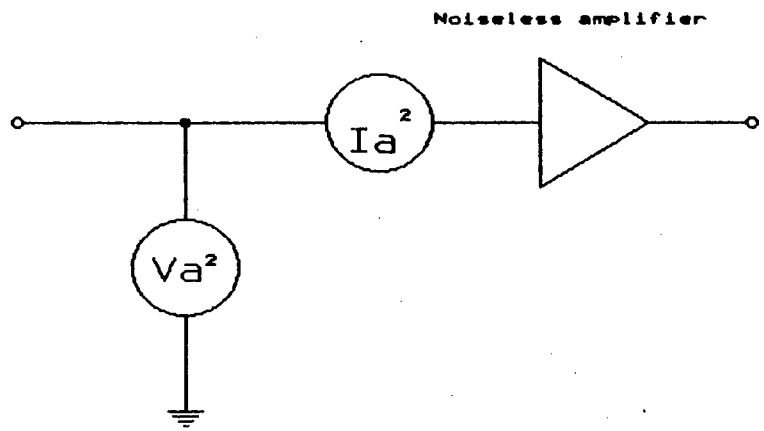
$$= 4.00 * 10^{-12} + 1.51 * 10^{-12}$$

$$= 5.51 * 10^{-12}\ V^2$$

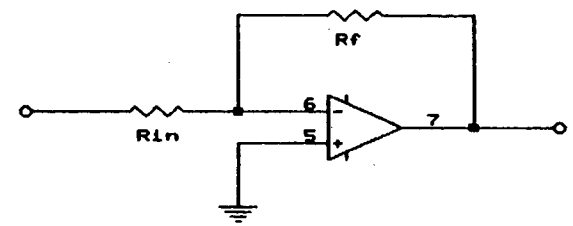
where V_{opamp} = 20nV / $\sqrt{Hz}$ (noise voltage)
i_{opamp} = 0.01 pA / $\sqrt{Hz}$ (noise current)
R_i = 47kΩ (inverting op-amp input resistor)
R_f = 235kΩ(inverting op-amp feedback resistor)

This is 18000 times smaller than the input system noise and so does not affect the system SNR. The following stages will also have negligible effects on the system SNR, since they are gain and filter stages. The noise floor of the entire baseband signal processing module was measured, and was found to be at 4μV or $1.6*10^{-11}V^2$. This is 3 times more than expected, but is still insignificant.

The total system NF in the radar equation is: NF = 0.8dB



Op-amp noise model



Op-amp circuit

$$Ia^2 = In^2 + 4kT / Rf$$

$$Ea^2 = En^2 + Rin^2 . Ia^2$$

In = amplifier noise current per root Hertz.

Vn = amplifier noise voltage per root Hertz.

Fig. 3.31. Op-amp Noise Model.

3.11 REFERENCES

[3.1] BRYANT, G.H. **Principals Of Microwave Measurements.** IEE ELECTRICAL MEASUREMENT SERIES 5, PETER PEREGRINUS, 1988.

[3.2] HOVANESSIAN, S.A. **Radar System Design And Analysis.** Artech, 1984.

[3.3] STREMLER, F.G. **Introduction To Communication Systems.** (2nd ed.), ADDISON-WESLEY, 1982.

[3.4] SANDER, K.F. **Microwave Components And Systems.** ADDISON-WESLEY, 1987.

[3.5] HOROWITZ, P., HILL, W. **The Art Of Electronics.** CAMBRIDGE UNIVERSITY PRESS, 1987.

[3.6] DARYANANI, G. **Principals Of Active Network Synthesis And Design.** JOHN WILEY & SONS, 1976.

[3.7] MILLMAN, J., HALKIAS, C.C. **Integrated Electronics: Analogue And Digital Circuits And Systems.** McGRAW-HILL, 1972.

4. CLASSICAL AND MODERN SIGNAL PROCESSING

4.1 GENERAL

Since we need to resolve a number of targets at different ranges, ie. sinusoids, a multi-frequency spectral estimator is required. Thus, a simple frequency counter cannot be used, as this can only resolve one target (ie. one sinusoid).

Spectral analysis is any signal processing technique that characterizes the frequency content of a measured signal. The Fourier transform is a mathematical basis for relating the time domain of a signal to its frequency domain. Statistics play an important role in spectral estimation, since most signals have noise associated with them. Only an estimate of a signal can be made from a finite set of data.

Some spectral estimators have a better representation of the signal. This is not the most important criterion in this thesis, however, high range-resolution properties are the most important. Resolution is defined as the smallest difference in the measured quantity by which two or more objects can be distinguished. Resolution can be viewed from either a time or a frequency domain approach. The frequency domain approach is used here.

An important concept, is that of bias and variance. The bias of a spectral estimator is the average error in the estimator. Thus, the larger the bias, the less accurate the spectral estimator. The variance is the mean-squared error in the estimator. A fairly smooth curve in a spectral plot shows little variance, whereas a very "spiky" plot with large peaks exhibits large variance. The higher the variance, the better the resolution properties of the spectral estimator, but the less accurate it becomes (the bias is larger).

The type of signals to be analyzed are sampled signals, which are quantized both in time and amplitude. Therefore all the techniques used will be digital signal processing techniques.

The following sections will briefly discuss the traditional methods of spectral estimation, and the modern approach to spectral estimation.

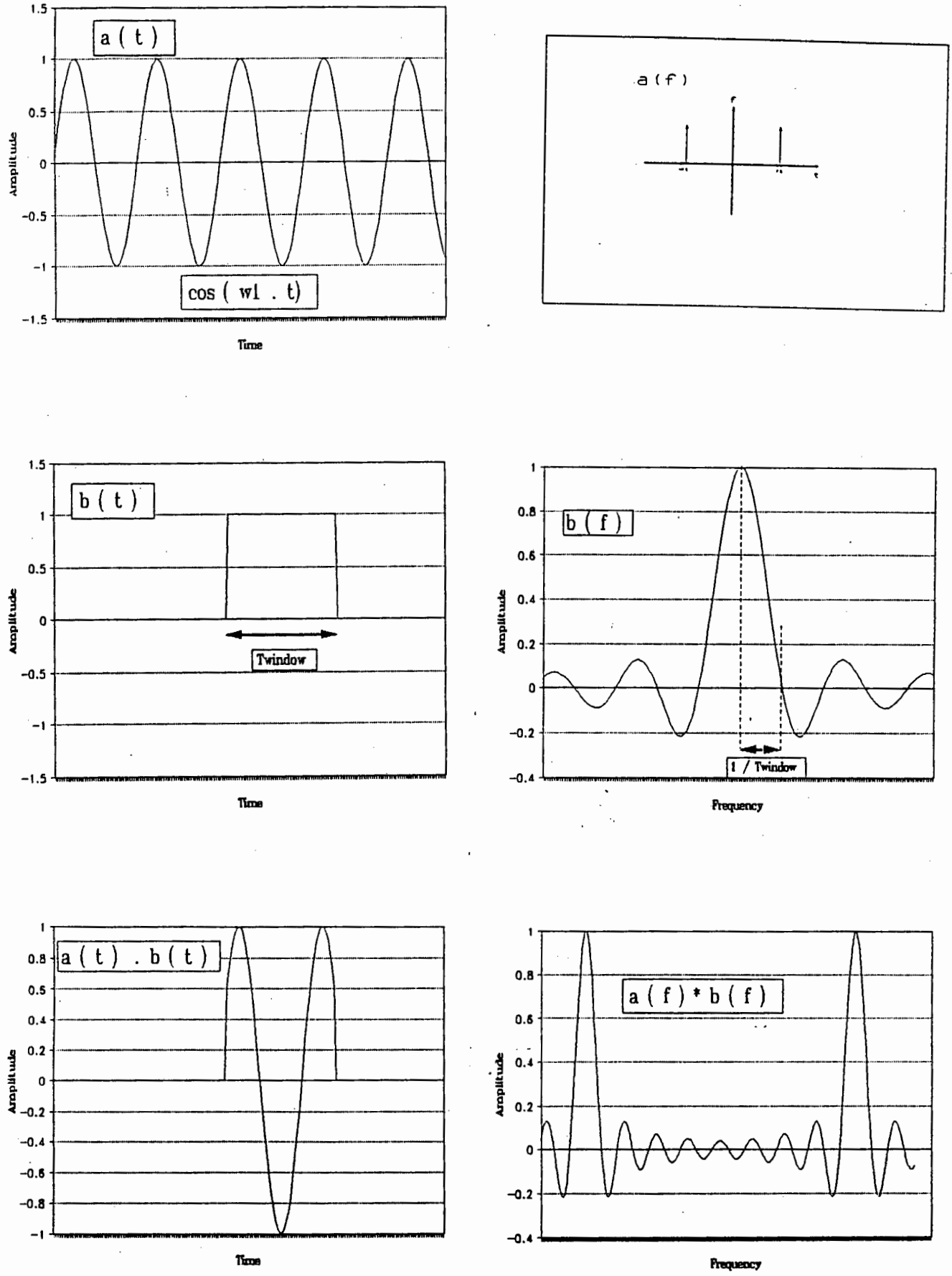


Fig. 4.1. Window Transformations.

Note: The Doppler content in the direction of measurement is zero, since the target is assumed to be moving perpendicular to the radar. Therefore, the Doppler resolution will not be measured.

4.2 CLASSICAL SIGNAL PROCESSING

The standard way of obtaining a frequency estimate of a sample of data is to use the Fourier transform. The primary resolution limit in using the Fourier transform directly is due to the fact that a finite sample of data is observed. This finite sample of data can be seen as a rectangular window of an infinite sequence of data. This rectangular window causes spectral blurring or more precisely, the received sequence in the frequency domain is convolved with a sin(x)/x function [4.1] - see figure 4.1. This can be mathematically stated as:

$$\text{time domain transformed to frequency domain}$$

$$\text{rect}(t/T_{window}) \longleftrightarrow \sin(f/T_{window})/T_{window}$$

There are numerous ways of rigorously defining resolution. One way is Rayleigh's definition - see figure 4.2.

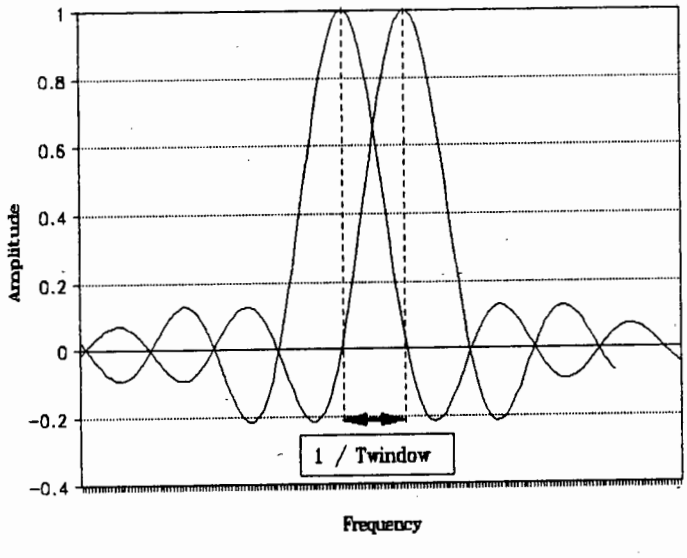


Fig. 4.2. Rayleigh's Resolution Definition.

The resolution is defined as the point at which the peak of the first curve cuts the minimum of the second curve. This is with reference to the square magnitude of the curve. From this it can be seen that the resolution limit is:

$$\delta f = \frac{1}{T_{window}}$$

A more optimistic way of defining the resolution limit is when the two curves intersect at the -3dB point - see figure 4.3.

The resolution limit from this definition is:

$$\left[\frac{\sin(\pi * T_{window} * \delta f')}{(\pi * T_{window} * \delta f')} \right]^2 = {}^1/_2$$

therefore: $\pi * T_{window} * \delta f' = 1.392$

thus: $\delta f' = \frac{0.443}{T_{window}}$

so, the -3dB bandwidth of the sin(x)/x curve is:

$$\delta f = \frac{0.886}{T_{window}}$$

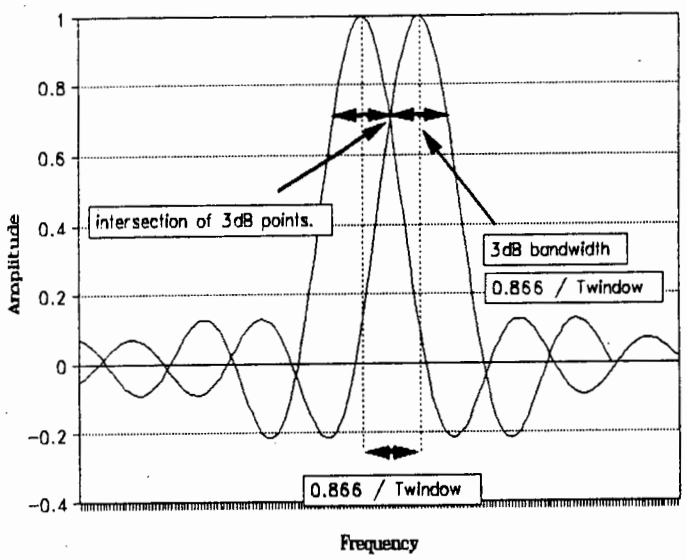


Fig. 4.3. More Optimistic Resolution Definition.

It would seem that one could improve the resolution by simply increasing the time window, however, it will be shown that once the resolution limit is inserted into the FMCW radar equation, the resolution becomes proportional to the transmitted bandwidth of the radar.

From equation 2.1, we can deduce the resolution equation:

$$\delta R = \frac{c}{4 * \delta_p} * \left[\frac{\delta f_b}{f_{mod}} \right]$$

Using the Rayleigh Definition of resolution, the resolution beat frequency is:

$$\delta f = \frac{1}{T_{window}}$$

But, $T_{window} = {}^{1}/_{2} * T_{mod} - \tau$ (see figure 4.4.)

$$\text{where } \tau = \frac{2 * R}{c}$$

However, since $\tau = 6.6ns$,

$$T_{mod} >> \tau$$

Therefore: $T_{window} \approx {}^{1}/_{2} * T_{mod}$

$$\text{So: } \delta R = \frac{c}{4 * \delta p} * \left[\frac{1}{fmod} * \frac{1}{Twindow} \right]$$

$$\delta R \approx \frac{c}{4 * \delta_p} * \left[\frac{1}{f_{mod}} * \frac{2}{T_{mod}} \right]$$

$$\delta R = \frac{c}{2 * \delta_p}$$

However, $B = \delta_p$

Therefore:

$$\delta R = \frac{c}{2 * B}$$

This is just the standard radar resolution formula which states that higher resolution is only obtainable by increasing the transmitted bandwidth.

Note, the above equation is for an electromagnetic wave in free space. The velocity of an electromagnetic wave is affected by the media through which it travels:

$$\delta R = \frac{c / \sqrt{\epsilon}}{2 * B}$$

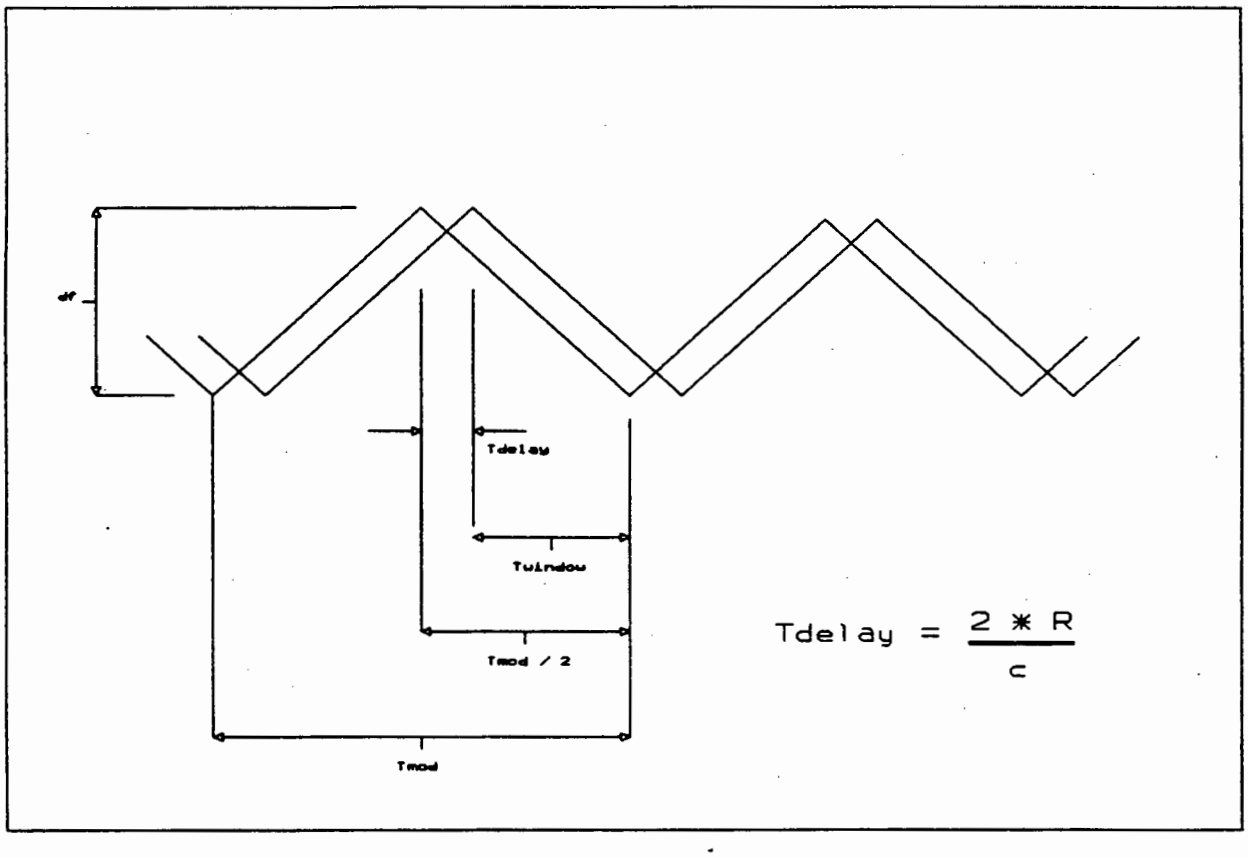


Fig. 4.4. T_{window} Representation.

Various techniques have been developed around the Fourier transform. They are:

a) Periodogram methods,

b) Correlogram methods.

The periodogram method relies on the direct approach by taking the Fourier transform and then squaring the magnitudes to obtain a spectral estimate. The resultant spectrum is then obtained by averaging a number of the spectral estimates.

$$P(f) = \lim_{N\to\infty} E\left[\frac{1}{(2N+1).T} * \left| T. \sum_{n=-N}^{N} x(n).\exp[-j.2.\pi.k.n / N] \right|^2\right]$$

where E is the expectation operator.

The averaging of the Fourier transform data in the periodogram method decreases the bias, but also decreases the variance. Therefore the resolution properties of the Fourier transform are worsened.

The correlogram approach uses the indirect approach by taking the Fourier transform of the autocorrelation sequence.

$$\hat{P}(f) = T * \sum_{m=-L}^{L} \hat{r}_{xx}[m].\exp[-j.2.\pi.f.m.T]$$

Windowing techniques, such as the Bartlett Window, etc, do not improve the resolution. They only reduce sidelobe levels at the expense of slightly reduced resolution [4.1].

All of these approaches have their advantages and disadvantages. Even with many different improvements such as: Wiener filtering, homomorphic deconvolution, and other techniques, the resolution cannot be much improved beyond the $1/T_{window}$ limitation.

4.3 MODERN SIGNAL PROCESSING

There are many modern spectral estimation techniques, some of which are parametric estimators and some non-parametric estimators. It has been found that using a parametric estimator has many advantages over the classical techniques, which are non-parametric [4.2 and 4.3]. Parametric estimators assume certain a-priori information about a process before it is analyzed. This allows a model to be generated which does not assume zero data outside the given sequence, but makes a

more realistic approximation. The window effect is therefore eliminated, giving rise to improved resolution. The resolution then becomes dependent on the signal to noise ratio.

The parametric methods can be broken up into three branches:

a) Autoregressive (AR),

b) Moving Average (MA), and

c) Autoregressive Moving Average (ARMA).

The selection of one of the above models requires some knowledge of the spectral shape of the process. If spectra with sharp peaks, but no nulls are required, then an AR model is appropriate. If spectra with deep nulls, but no sharp peaks are required, then an MA model is appropriate. If the process exhibits sharp peaks and deep nulls, then an ARMA model is required. Sometimes it is advantageous to use an AR model of high order in place of an MA model of lower order, since MA and ARMA models are non-linear and hence cause a severe computational burden.

Since the ARMA model is a general model, we will derive it first and then set the model up for MA only and AR only modelling. The data being analyzed must be wide-sense stationary. A time-series model that approximates many discrete-time deterministic and stochastic processes is represented by the filter, which is described by the following linear difference equation [4.2]:

$$\sum_{k=0}^{p} a[k].x[n-k] = \sum_{k=0}^{q} b[k].u[n-k] \quad \text{... equation 4.3}$$

$$\text{for } n \geq 0$$

where x[n] is the output sequence of a causal filter

u[n] is an input driving sequence.

a[k], and b[k] are the filter coefficients.

Taking the z-transform of equation 4.3, we get:

$$X(z).\sum_{k=0}^{p} a[k].z^{-k} = U(z).\sum_{k=0}^{q} b[k].z^{-k} \quad \text{... equation 4.4}$$

The system transfer function is therefore:

$$H(z)_{ARMA} = \frac{X(z)}{U(z)} = \frac{\sum_{k=0}^{q} b[k].z^{-k}}{\sum_{k=0}^{p} a[k].z^{-k}} = \frac{B(z)}{A(z)}$$

The above is the rational form for which the polynomials are:

$$A(z) = \sum_{k=0}^{p} a[k].z^{-k}$$

$$B(z) = \sum_{k=0}^{q} a[k].z^{-k}$$

Therefore the system can be defined as:

$$H(z)_{ARMA} = \sum_{k=0}^{\infty} h[k].z^{-k}$$

Both polynomials A(z) and B(z) must have all their zeros within the unit circle of the z-plane to guarantee that H(z) is a stable minimum-phase causal filter. By the Wiener-Khintchine theorem:

$$P_{xx}(z) = P_{uu}(z) * |H(z)|^2 \quad .... \quad \text{equation 4.5}$$

$P_{xx}(z)$ is the output power spectral density;

$P_{uu}(z)$ is the input driving power spectral density.

The driving process is assumed to be an additive white Gaussian noise process of zero mean and variance σ_w^2.

Substituting in the σ_w^2 and $z = \exp(j.2.\pi.f.T)$, we get:

$$P_{xx}(f) = \sigma_w^2 * \left| \frac{B(f)}{A(f)} \right|^2$$

where:

$$B(f) = B(z)\Big|_{z = \exp(j.2.\pi.f.T)}$$

$$A(f) = A(z)\Big|_{z = \exp(j.2.\pi.f.T)}$$

Note that the b[k] parameters form the MA portion of the ARMA model, whilst the a[k] parameters form the AR portion of the model.

If we set B[z] = 1, then we get an AR model:

$$H(z)_{AR} = \sum_{k=0}^{p} \left[\frac{1}{a[k].z^{-k}} \right]$$

$$\text{So, } P_{AR} = \frac{\sigma_w^2}{|\ A(f)\ |^2}$$

If we set A[z] = 1, then we get an MA model:

$$H(z)_{MA} = \sum_{k=0}^{q} b[k].z^{-k}$$

$$\text{So,} \quad P_{MA} = \sigma_w^2 \cdot |\, B(f) \,|^2$$

Note that the AR model is an all-pole model, whilst the MA model is an all zero model.

Another important consideration is that an ARMA model can be synthesized with an AR model or MA model of infinite order [4.2]. In practise this is very useful, since for certain functions, an AR model of modest order can be used to generate a model close enough for the task at hand. The computational burden is then released by the use of an AR model [4.2].

The AR model will be concentrated on due to its computational practicality and its similarity to the actual radar spectral content. We can assume a[0] = 1 and b[0] = 1, since any filter gain can be incorporated into σ_w^2. Expanding equation 4.5, we get:

$$P_{xx}(z) = \sigma_w^2 \cdot \frac{B^*(1/z^*).B(z)}{A^*(1/z^*).A(z)}$$

Therefore:

$$P_{xx}(z).A(z) = \sigma_w^2 \cdot H^*(1/z^*).B(z)$$

Taking the inverse z-transform and using the Wiener Khintchine theorem:

$$Z^{-1}\left[Pxx(z).A(z)\right] = \left[\, Z^{-1}\, \sigma_w^2 * H^*(1/z^*).B(z) \right]$$

$$\Rightarrow \qquad r_{xx}[k] * a[k] = \sigma_w^2 \cdot \sum_{l=0}^{q} b[l].h^*[l-k]$$

where $r_{xx}[k]$ is the autocorrelation function of x.

Note: '*' is a convolution operator in the above equation.

$$\Rightarrow \quad \sum_{l=0}^{p} a[l].r_{xx}[k-l] = \sigma_w^2 . \sum_{l=0}^{q} b[l].h^*[l-k]$$

Now, setting $b[l] = \delta[l]$ for an AR process, and setting $h[k] = 0$ for $k < 0$ (by causality) we get:

$$r_{xx}[k] = -\sum_{l=1}^{p} a[l].r_{xx}[k-l] + \sigma_w^2 .h^*[-k]$$

Since $h^*[-k] = 0$ for $k > 0$, and $h^*[0] = 1$, we get:

$$r_{xx} = \begin{cases} -\sum_{l=1}^{p} a[l].r_{xx}[k-l] & \text{for } k \geq 1 \\ -\sum_{l=1}^{p} a[l].r_{xx}[-l] + \sigma_w^2 & \text{for } k = 0 \end{cases}$$

.... equation 4.6

Equations 4.6 are called the Yule-Walker equations. They define a non-linear relationship between the AR process and the autocorrelation function. The first section of equation 4.6 is linear, and can be used to determine the AR parameters. These can be solved by numerically efficient algorithms. Once the filter parameters have been obtained, the power spectral density is obtained by simply **taking the Fourier transform of the filter. Note** that this is not the same as taking the Fourier transform of the sampled data points. The Levinson algorithm is the most efficient algorithm available to solve these equations, however, the data is available in the form of reflection coefficients, and must be converted to the AR parameters. When used in the different methods, the only difference in the usage of the Levinson algorithm is in the constraints applied. The different methods place various constraints on the equations and hence have different characteristics. The AR model

can be seen graphically in figure 4.5 to be equivalent to a tapped filter. The white noise driving process is merely a theoretical concept. The filter is that obtained through the different methods. The observed sequence x[n] is the actual signal from the radar which we want to analyze - see figure 4.6.

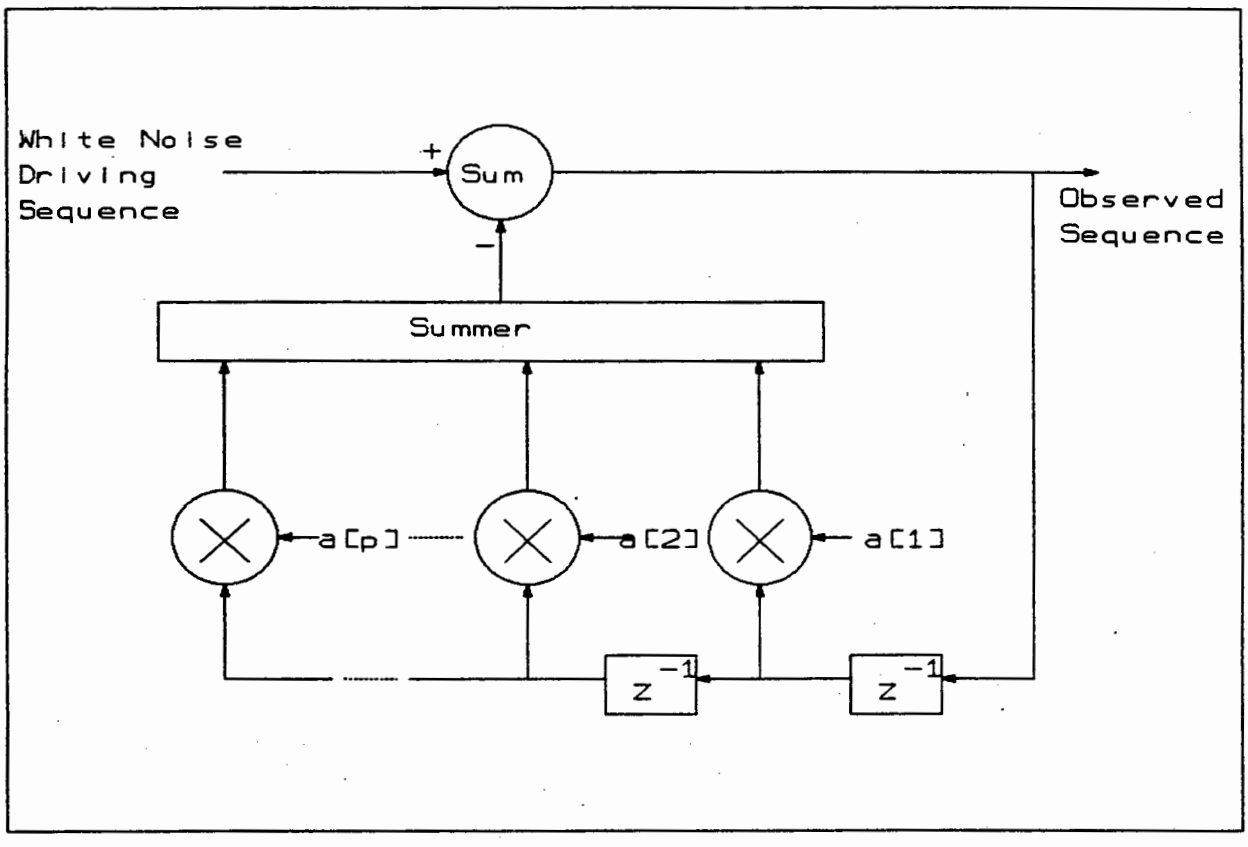


Fig. 4.5. AR Model as a Tapped Filter.

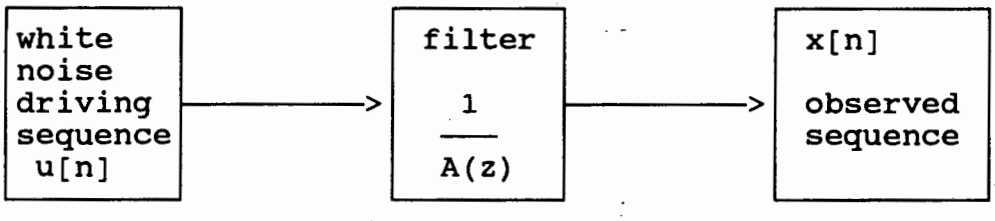


Fig. 4.6. Theoretical AR Process.

The various methods have different properties. Some of the algorithms will be briefly discussed. There are many types of AR spectral estimators, however, the ones most appropriate for this thesis are the Burg and the modified covariance methods. This is because they perform well with minimum computational complexity [4.3]. The Burg method is effectively a maximum entropy spectral estimator. The rationale for choosing the maximum entropy criterion is that it imposes the fewest constraints on the unknown time series, by maximizing its randomness, and thereby producing a minimum bias solution [4.3]. The Burg method is unsuitable for this thesis, since for frequency (sinusoidal) estimation, it has been found to produce false peaks (line splitting) - [4.3]. The method is shown in Appendix H, and the program is listed in Appendix I.

The modified covariance method is effectively an approximate maximum likelihood estimator. It is a frequency estimator and is used to estimate sinusoids in white noise. It is more specific than the Burg method, since the Burg method is a more general spectral estimator. Also, the modified covariance method has **not** been found to exhibit line-splitting [4.3]. For the above reasons, the modified covariance method was the chosen AR method for this thesis. The method is shown in Appendix J, and the program is listed in Appendix K.

There are a few important considerations to note:

The first is that if there is additional noise in the observation process, zeroes are effectively added to the data, and therefore, the AR model does not fit the data properly. This increases the bias of the model ie. the poles are shifted slightly out of place. Thus the higher the SNR, the better the resolution. The modified covariance method has been found to work well when the SNR is greater than 20dB [4.3].

The second factor to note is that ARMA models cannot model purely deterministic sinusoidal signals [4.2]. This is because the deterministic sinusoid is not wide-sense stationary. Thus the phase has to be random to make the sinusoid stationary or white Gaussian noise has to be added.

A resolution formula for AR processes is [4.2]:

$$F = \frac{1.03}{T.p.[SNR.(p+1)]^{0.31}}$$

```
where F = resolution in Hz,
      T = sample interval in seconds,
      p = number of poles in the model,
    SNR = linear signal-to-noise ratio.
```

Non-parametric, high resolution spectral estimators are also available. A special class of non-parametric spectral estimator is the principal component or eigenanalysis estimator. The most successful of these for frequency (sinusoidal) estimation is the MUSIC algorithm [4.2]. The advantage of this algorithm over the modified covariance technique, is that it is more robust, however, it requires n^3 computations compared to the n^2 computations of the modified covariance estimator. Both the MUSIC method and the modified covariance method yield comparable high resolution results, however, despite the robustness of the MUSIC method, the modified covariance method was preferred due to its lower computational overhead.

4.4 REFERENCES

[4.1] MULLIS, C.T., ROBERTS, R.A. **Digital Signal Processing.**, ADDISON-WESLEY, 1987.

[4.2] MARPLE, S.L. **Digital Spectral Analysis With Applications.** PRENTICE-HALL, 1987.

[4.3] KAY, S.M. **Modern Spectral Estimation: Theory And Application.** PRENTICE-HALL, 1987.

[4.4] LACOSS, R.T. **Data Adaptive Spectral Analysis Methods.** GEOPHYSICS, VOL. 36, NO. 4, 1971.

5. RADAR AND SIGNAL PROCESSING SIMULATIONS

5.1 GENERAL

Two sets of simulations were performed:

a) One set was performed on two software simulated sinusoids with varying separations in frequency,

b) The second set was sampled from two signal generators with their outputs summed. The ADC sampling arrangement will be described in the following chapter.

In each simulation, different parameters and methods were experimented with. The Burg method and the modified covariance method was tested. A modified Burg method was experimented with, to eliminate pole-splitting, but the improvements were negligible.

The best number of poles to utilize in the AR models were experimentally found to be $^1/_3$ the number of samples. More than this was found to give better resolution (higher variance), but the system became noisy and low amplitude poles were emphasized. Also, the accuracy of the pole positioning decreased (greater bias). The more data points used, decreased the bias, however, the computational overheads increased. The number of sample points used was 64 and the number of poles used was 21. The theoretical maximum number of poles is half the number of sample points.

5.2 SOFTWARE GENERATED SINUSOIDS

The software simulated sinusoids were generated by two programs. The first was generated by an FMCW radar return simulator; the program can be seen in Appendix L. The output consists of two sinusoids in white noise. A later version generates the output directly without using FMCW radar simulation - see Appendix M. The white noise is added to the summed sinusoids. This is an important step since, as previously mentioned, purely deterministic sinusoids cannot be modelled by AR processes because they are not wide-sense stationary. The noise power was

set so that the SNR was 80dB. The minimum expected SNR from the radar is 60dB, however, the maximum SNR is limited by the ADC resolution, and is 72dB. Once the modelled data was obtained, it was processed by the programs BURG and MODCOV - see Appendices I and K respectively. The Fourier transform of the AR coefficients was then taken to yield the final spectrum. The program that performed the Fourier transform is called CWPRINT - see Appendix N. The outputs were plotted on graphs with the magnitude in logarithmic format. It is important to note that the AR filter coefficients must be zero-padded [5.1] to obtain the required resolution bins in the Fourier transform. The frequency step per resolution frequency bin is determined by the following formula:

$$\delta F_{step} = \frac{N1}{N2} * f_{window}$$

where N1 = number of data points.
N2 = number of zeros padded on plus number of filter parameters.
f_{window} = 1 / period of sampled window.

For: N1 = 64,
N2 = 16384,
f_{window} = 1.8kHz.

δF_{step} = 7Hz.

A number of frequency differences (resolutions) between the two sinusoids were modelled. Frequencies differences from 30Hz up to 4kHz were generated and successfully resolved by both the Burg and the modified covariance methods. The Burg method, however, was demonstrated to exhibit line-splitting, and bias in the frequency estimation. A resolution of 15Hz was attempted, however it was unsuccessful. The frequency differences (resolutions) are shown in table 5.1.

Table 5.1. Software Resolutions Attempted.

FREQUENCY SEPARATION BETWEEN TWO SINUSOIDS
30 Hz
60 Hz
120 Hz
240 Hz
480 Hz
960 Hz
1820 Hz
3640 Hz

The spectral plots of the modified covariance method and the Burg method for all these resolutions are available in Appendix O. A best resolution of 30Hz was obtained in the simulations. The required parameters substituted into the resolution equation 4.8:

$$F = \frac{1.03}{T.p.[SNR.(p+1)]^{0.31}}$$

where F = resolution in Hz,
T = 5μs
p = 21
SNR = 10^8.

F = 12Hz

This demonstrates that the resolution equation is inaccurate by a factor of about 2 to 3 times.

The simulation result of 30Hz is equivalent to 5mm resolution if the FMCW radar parameters were:

f_{mod} = 900Hz
δf = 500MHz
f_b = 30Hz.

5.3 HARDWARE GENERATED SINUSOIDS

The hardware generated sinusoids were obtained by summing two sinusoids from signal generators with white noise from a white noise generator - see figure 5.1.

The program for the sampling is called GET_DAT1 and can be examined in appendix P. A later version, GETDAT2, just samples the two summed sinusoids and generates the white noise itself in software - see Appendix Q. The SNR for the first version was limited to 50dB, due to the white noise generator. The noise power in the second version was set in software so that the SNR was 60dB. As mentioned above, the maximum SNR is limited by the ADC resolution, and is 72dB. Once the sampled data is obtained, it is processed by the program MODCOV - see Appendix K. The Fourier transform of the AR coefficients is then taken to yield the final spectrum. The program that performs the Fourier transform is called CWPRINT - see Appendix N. The outputs are plotted on graphs with the magnitude in logarithmic format.

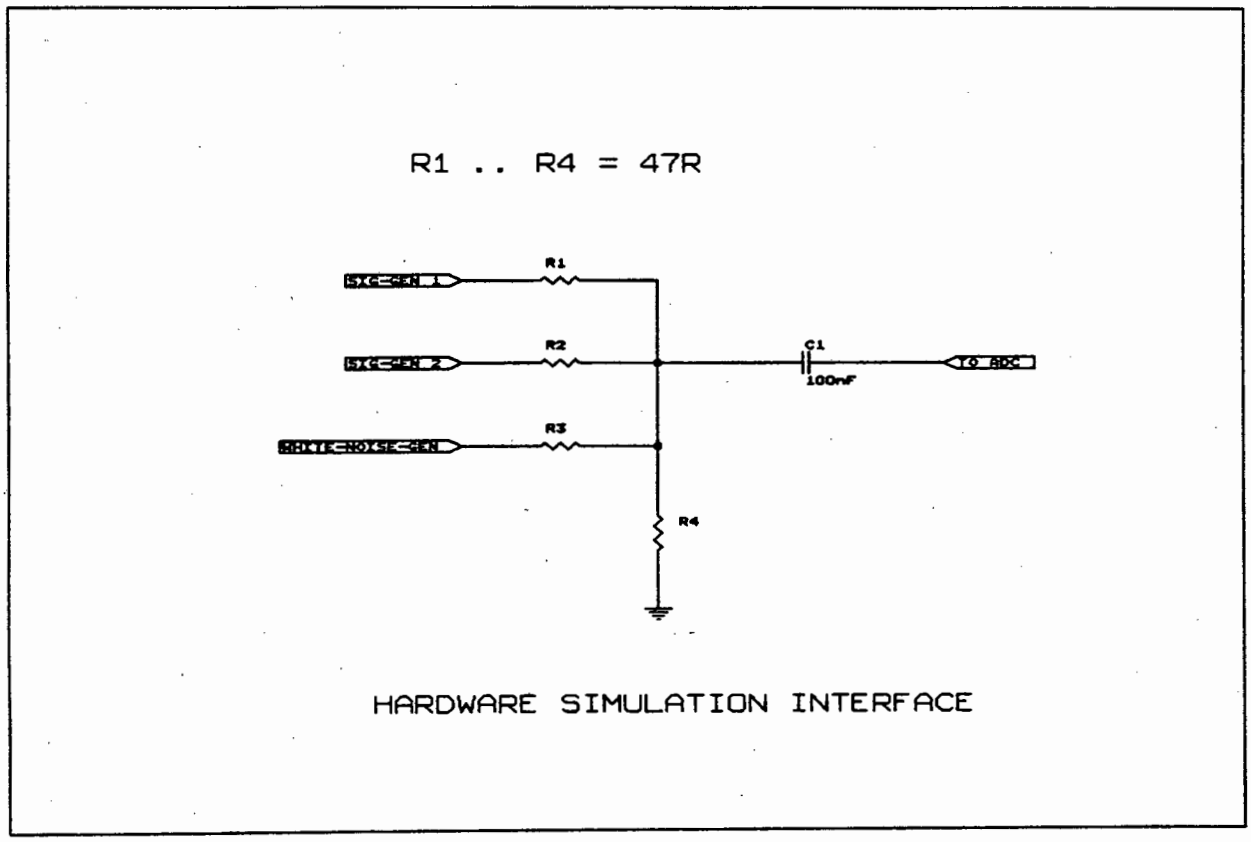


Fig. 5.1. Hardware Sampling Setup.

As before, it is important to note that the AR filter coefficients must be zero-padded to obtain the required resolution bins in the Fourier transform.

Again this demonstrates that the resolution equation is inaccurate by a factor of about 2 times.

The 100Hz resolution result of the hardware generated signal is equivalent to 16.1mm resolution if the FMCW radar parameters were:

$$f_{mod} = 900Hz$$
$$\delta f = 500MHz$$
$$f_b = 100Hz.$$

A best resolution of 80Hz was obtained for the **second version** (software generated noise), and the required parameters substituted into the resolution equation 4.8 are:

$$F = \frac{1.03}{T.p.[SNR.(p+1)]^{0.31}}$$

where F = resolution in Hz,
$T = 5$
$p = 35$
$SNR = 10^6$.

$F = 27Hz$

The resolution formula is inaccurate by a factor of 3 times.

The 80Hz result is equivalent to about 13.3mm resolution if the FMCW radar parameters were:

$$f_{mod} = 900Hz$$
$$\delta f = 500MHz$$
$$f_b = 80Hz.$$

It was observed that sinusoids with power differences of up to about 20dB were still resolvable. Also, the range resolution of the AR technique is over 30 times better than the (c / 2B) resolution limit.

5.4 REFERENCES

[5.1] MULLIS, C.T., ROBERTS, R.A. **Digital Signal Processing.**, ADDISON-WESLEY, 1987.

6. HARDWARE AND SIGNAL PROCESSING SOFTWARE INTEGRATION

6.1 GENERAL

The FMCW radar hardware needs to be interfaced to the signal processing software. The output from the radar hardware is a signal in the form of a $\pm 5V$ signal. An ADC will be required to interface the computer to the radar hardware. A 12-bit PC30D ADC card made by Eagle Electric was used to interface to the computer. The maximum sampling rate of this card is 200kHz. This sampling rate was used to read in the data. Note that this sampling rate is much greater than twice the maximum expected beat frequency and hence fulfills the Nyquist criterion [6.1 and 6.2]. Since 64 samples are required in the window time:

$$T_{window} = \frac{1/2}{f_{mod}} \qquad \text{where } f_{mod} = 900Hz$$

$$= 556\mu s$$

The total sample period is:

$$T_{tot\_samp} = N_{sample} * T_{sample}$$

$$= 64 * 5\mu s$$

$$= 320\mu s$$

This leaves $118\mu s$ "guard time" on either side of the discontinuities.

Since the bandpass processing unit causes phase shifts, the synchronization (SYNC) pulse is necessary to synchronize the sampling time. This is performed with two monostables in the modulator unit. The ADC has an enable sample pin, which is used to enable the sampling. This pin is connected to the SYNC output on the modulator via enable circuitry - see figure 6.1. The enabling circuitry is important, because the enable pin is not edge sensitive. When the program is run, the enable signal from the computer could go low in the middle of the "window" of the SYNC

timing. This would mean that some of the data from the next window would be concatenated with that from the current window. It is thus imperative to implement edge sensitive enabling signals in software.

The software runs in a loop, which consists of the sampling program (DMA_NEW as before), the modified covariance estimator method, and the spectrum plotter (CWPRINT also as before). The modified covariance estimator method was chosen for its computational efficiency and its satisfactory performance as described previously. The program, LOOP_SPEC, is listed in Appendix S.

The calculation time for the modified covariance method with 21 poles and 64 sample points is approximately one second with a maths co-processor on a 16MHz AT. The calculation rate of the above combination is 0.012Mflops per second. If a Motorola 56001 DSP processor board, with its rated 10Mflops per second is plugged into the PC, it will run about 1000 times faster than the PC/maths co-processor combination. This would mean that the estimator calculation would take about 1ms. The Fourier Transform would take another 4ms. These times added to the overheads for determining if a target exists, would yield approximately 8ms calculation time. This timing is ideal for real-time observation of material on a conveyer belt. It is important to note that the dynamic range of the ADC is [6.3]:

$$SNR = 10.\log(2^{2n}) \qquad \text{where } n = \text{number of bits}$$

$$= 10.\log(2^{2.12})$$

$$= 72\text{dB}$$

Therefore, the maximum SNR for the system is 72dB. The quantization noise can be shown to have a white noise Gaussian distribution [6.3]. The variance of such a signal is:

$$\sigma^2 = \frac{q^2}{12} \qquad \text{where } q = \text{quantization level}$$

$$= \frac{[1 \,/\, 4096]^2}{12}$$

$$= 5.10^{-9}.$$

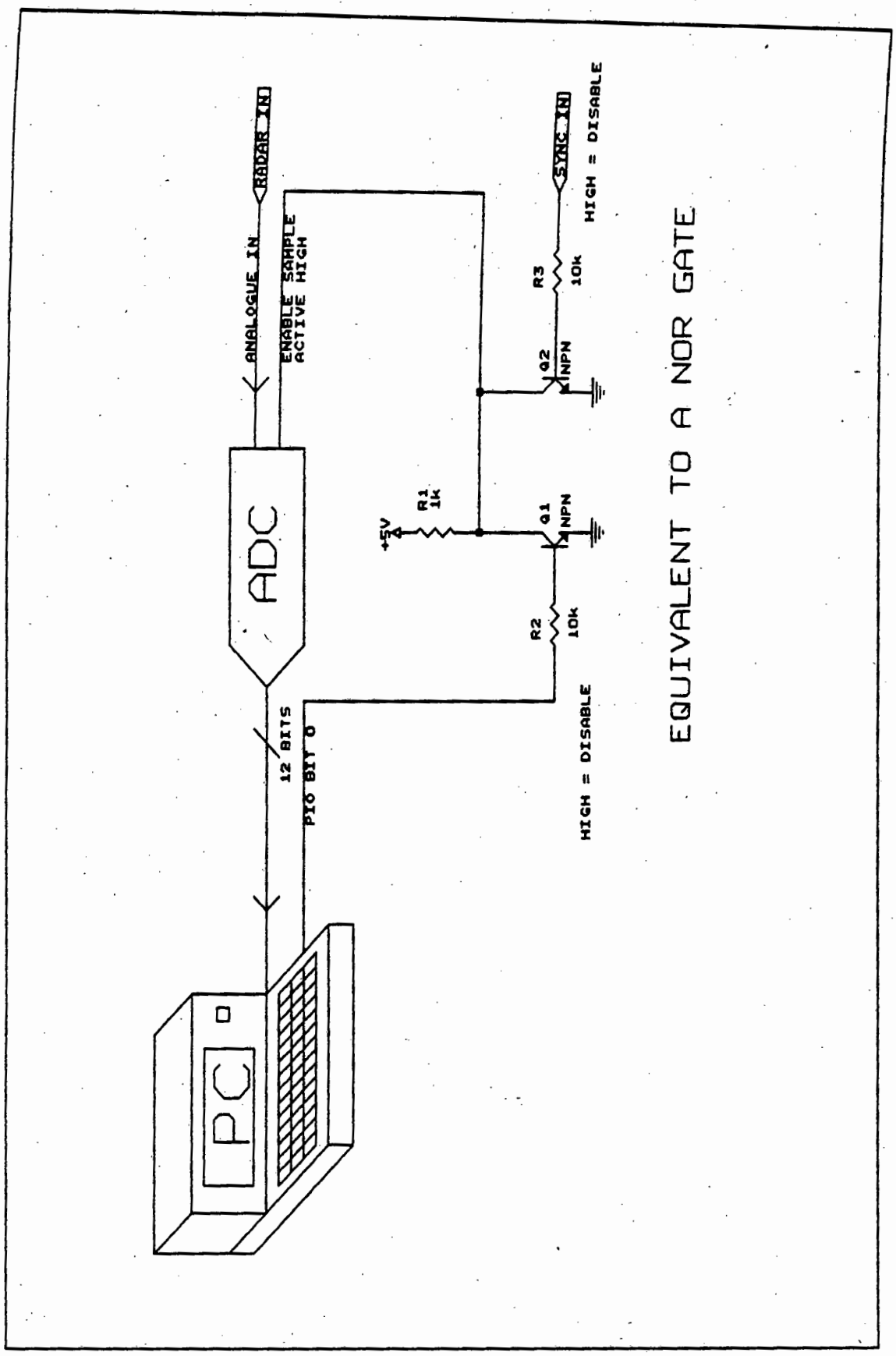


Fig. 6.1. SYNC Circuitry to ADC.

6.2 REFERENCES

[6.1] HAYKIN, S. **Digital Communications**. JOHN WILEY & SONS, 1988.

[6.2] STREMLER, F.G. **Introduction To Communication Systems**. (2nd ed.), ADDISON-WESLEY, 1982.

[6.3] MULLIS, C.T., ROBERTS, R.A. **Digital Signal Processing.**, ADDISON-WESLEY, 1987.

7. RESULTS OF THE SYSTEM INTEGRATION

The final system tests were performed on the roof of the engineering department at UCT. This was done to minimize clutter from walls and other objects. The tests were performed with the targets within the range of the filters. The first targets utilized were made of ping-pong balls wrapped in aluminium foil and hung from a horizontal beam by nylon wires. This was found unsuitable because the targets could not be kept stationary in the wind and hence the radar cross-section (RCS) seen by the radar, varied immensely. Finally, metal poles on stands were used - see figure 7.1. It is important to note that the poles yield a distributed frequency return and hence have to be aligned suitably to minimize the distributed frequency response. The distributed response impairs high resolution properties.

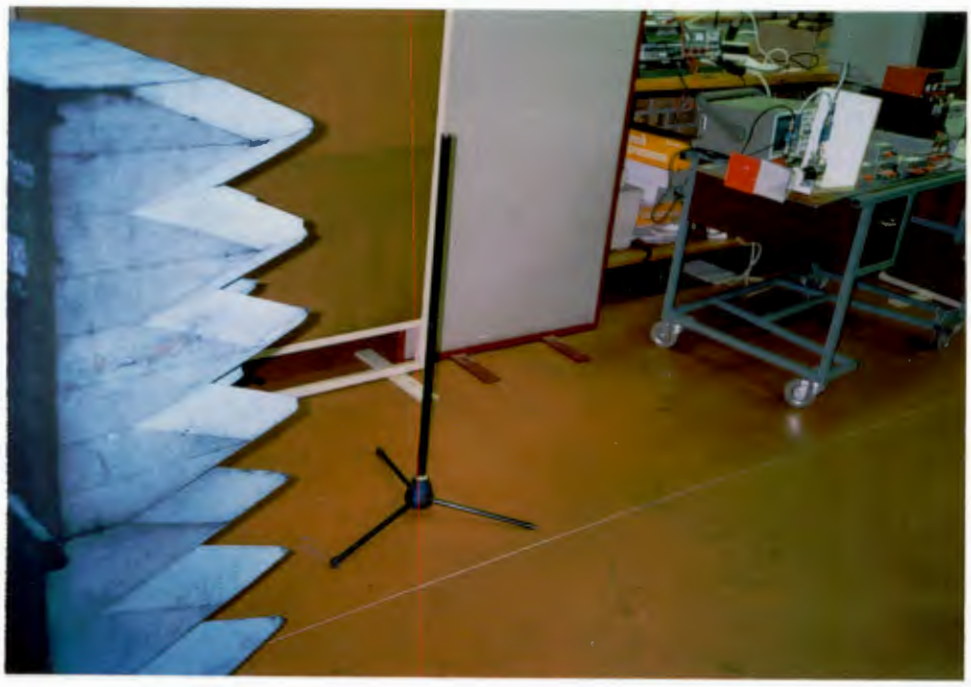


Fig. 7.1. Single Target Setup.

The program, LOOP_SAMPLE, was run and the results observed. Initially one target was set up. Figure 7.2 shows the time domain plot of the single target. The Fourier transform is shown in figure 7.3. The target was identifiable, however two sharp spikes per target were observed. Spikes further on were from obstructions on the roof. Point targets yield sharper spikes than distributed targets. Also, good

oscillator linearity allows sharper spikes. This implies that the YIG tuned oscillator is sufficiently linearized for this thesis.

As the target was moved, the frequency separation of the double spike remained constant. This implied that the multiple spikes were not caused by multipath. The frequency separation was measured and found to be about three times the modulation rate (2700Hz). As the modulation rate was varied, the frequency separation between the spikes varied. The double spikes are probably caused in the mixer by amplitude modulation of the return signal with the modulation waveform.

The modified covariance spectrum yielded similar results, however, the spikes were much higher and exhibited narrower bandwidth than the Fourier transform spectrum - see fig 7.4. As the target was moved, proportional changes were observed on the frequency plot. The relative ranges are shown in table 7.1.

Table 7.1. Hardware Resolutions Attempted.

RELATIVE RANGES ATTEMPTED
0.5 cm
1 cm
2 cm
5 cm
10 cm
20 cm
40 cm

The spectral plots for all these relative ranges are available in Appendix T. Less than 1cm changes were resolved, however, the plots do not show a proportional frequency change with range since it was difficult to align the long targets vertically. Clutter was 15dB down from the return signal and multipath effects were negligible. The system noise was 50dB down from the main spikes.

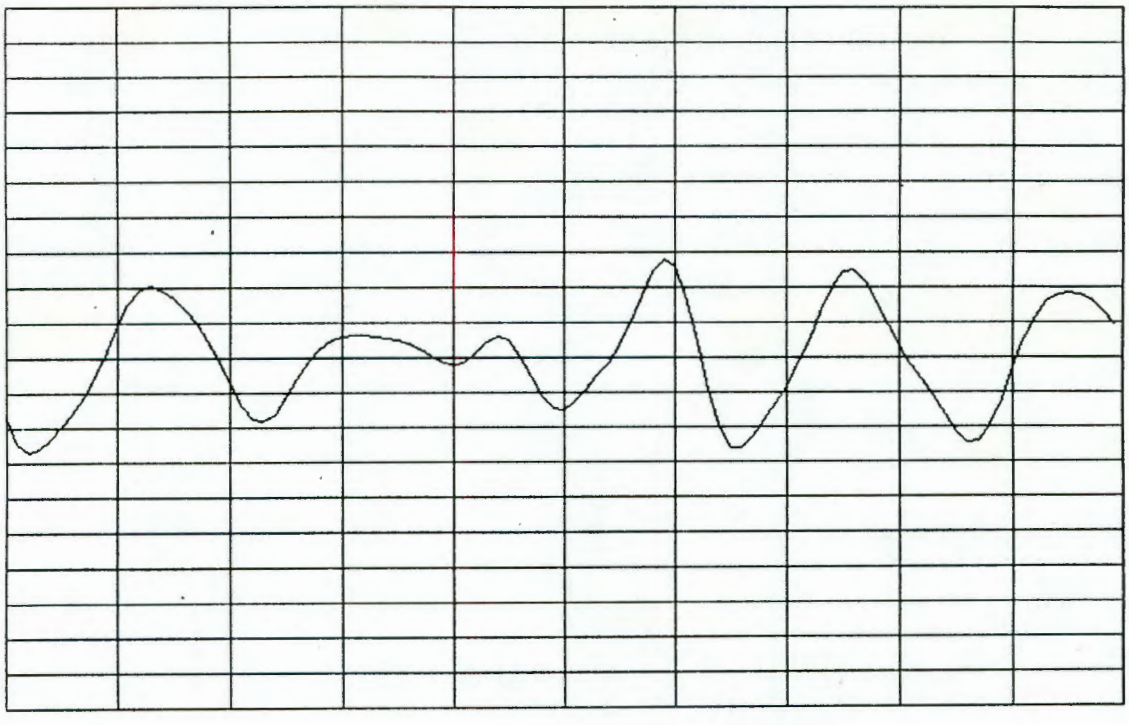


Fig. 7.2. Time Domain Plot of Single Target.

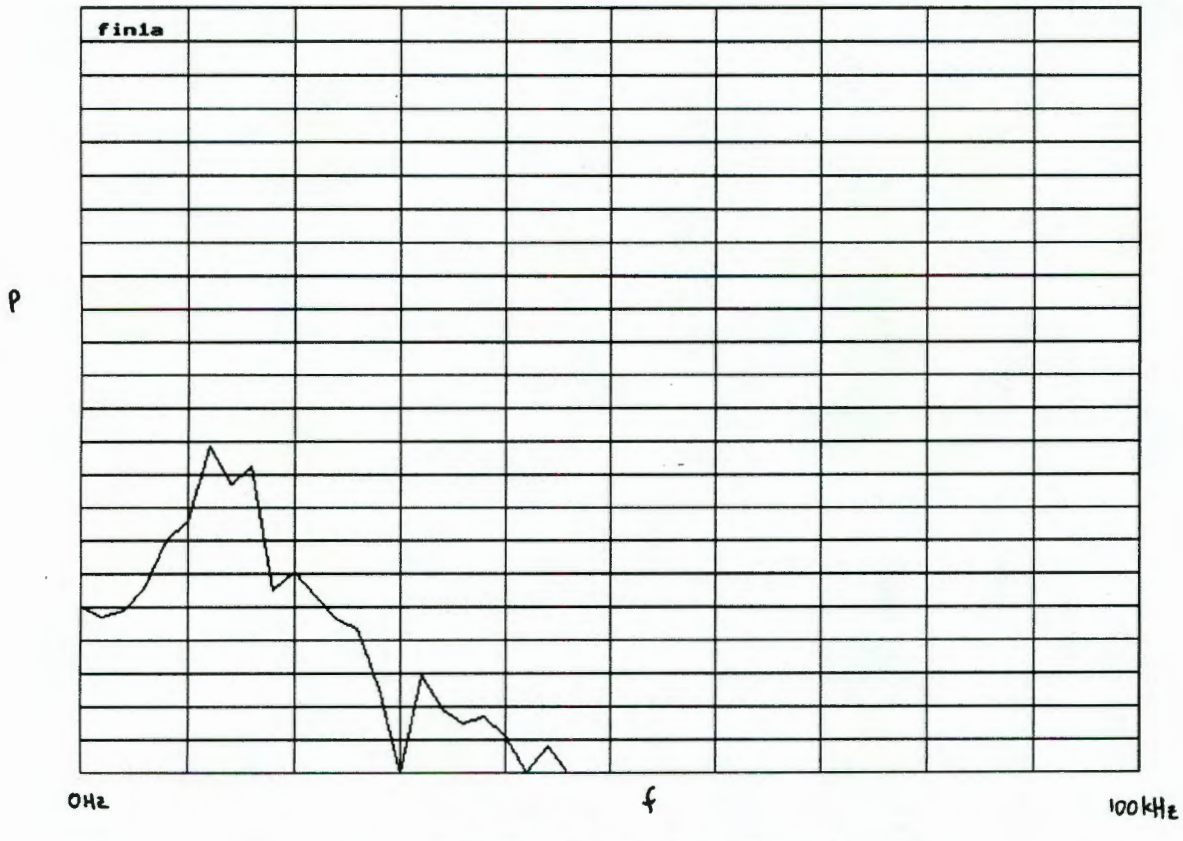


Fig. 7.3. Fourier Transform of Single Target.

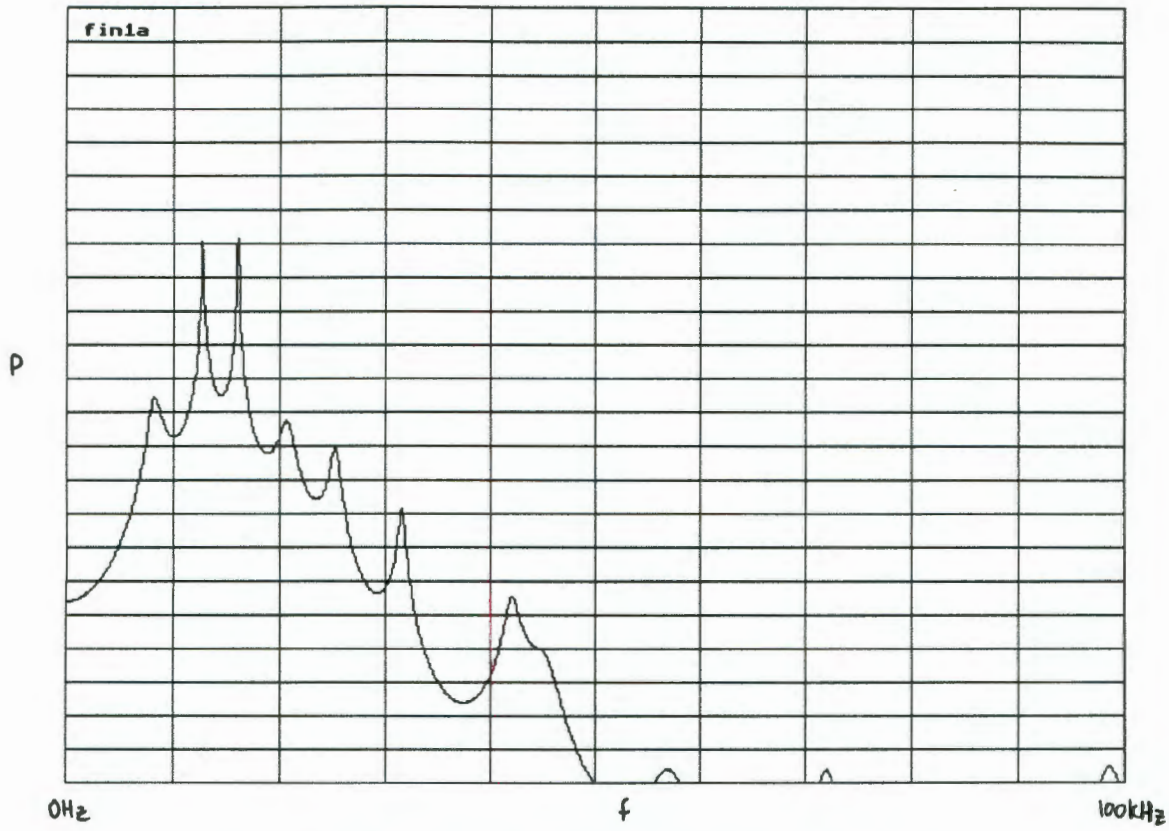


Fig. 7.4. Modified Covariance Spectrum of Single Target.

Two targets were then set up - see fig. 7.5. A time domain graph can be seen in figure 7.6.

Fig. 7.5 Double Target Setup and Mountings.

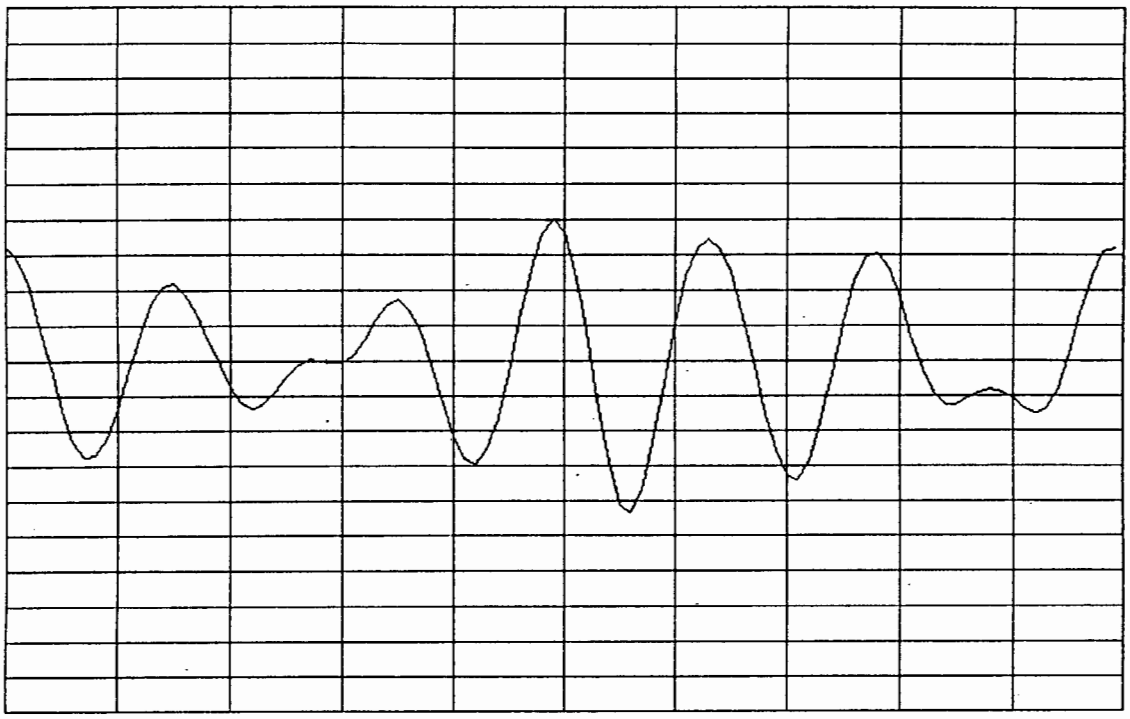


Fig. 7.6. Time Domain Plot for Two Targets.

The Fourier transform of this data can be seen in figure 7.7. Four returns for the two targets were obtained. If the targets were brought closer than the amplitude modulation frequency, then the results became meaningless.

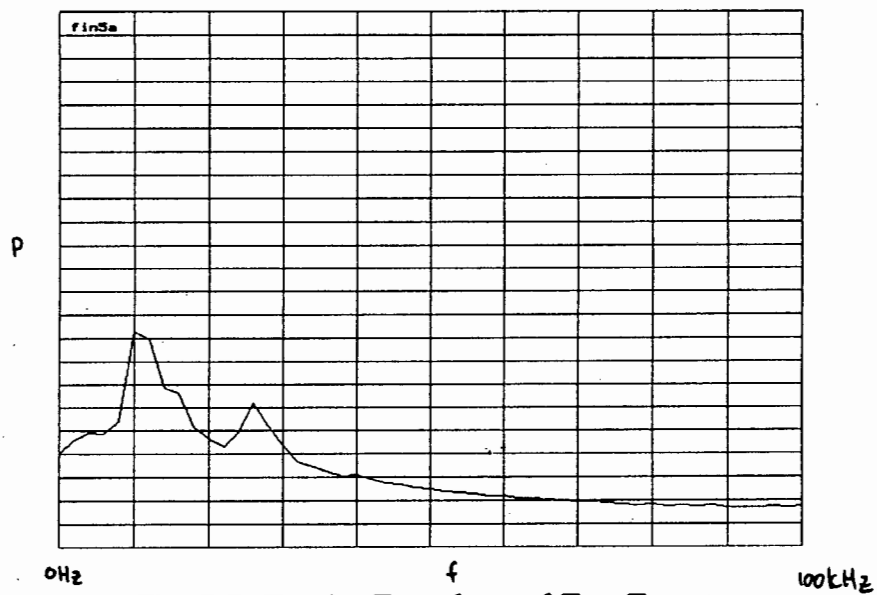


Fig. 7.7. Fourier Transform of Two Targets.

The modified covariance spectrum of the same data can be seen in fig. 7.8. The modified covariance method can be seen to have problems modelling the amplitude modulated waveform. Similar problems were encountered as with the Fourier

transform method. Numerous samples were taken with the targets at various ranges and the results were unsatisfactory.

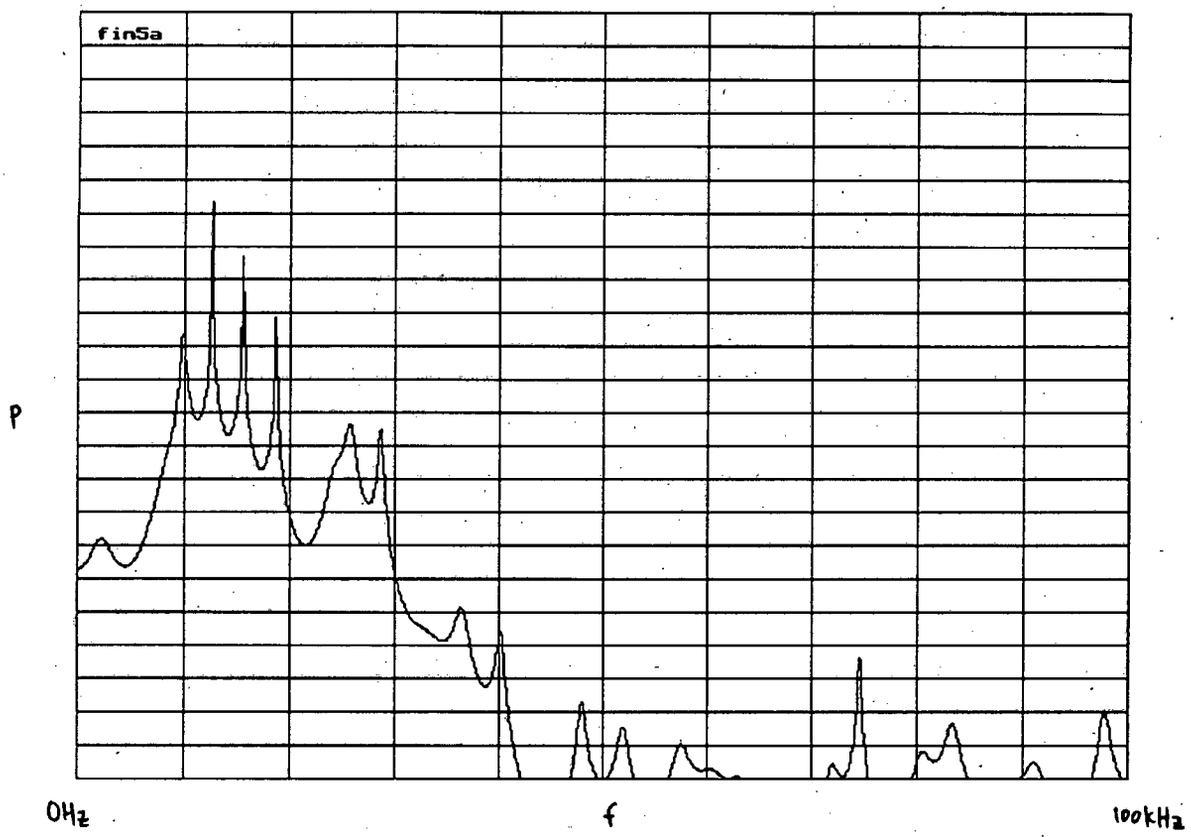


Fig. 7.8. Modified Covariance Spectrum of Two Targets.

9. FUTURE WORK AND RECOMMENDATIONS

Most importantly, a digital bandpass filter needs to be implemented in the time domain to eliminate one of the double spikes. This would mean that the range would be limited to the amplitude-modulation frequency, but this minor problem could be overcome by decreasing the peak frequency deviation. This method would effectively restore the high resolution properties of the modified covariance technique.

Other frequency generation techniques for future investigation are the lock and roll technique [9.1], and the third-order phase-locked delay-line discriminator [9.2 and 9.3]. These techniques can improve curve smoothness, and most importantly, linearity.

A further limitation on resolution is oscillator phase- noise [9.1]. Its effects can be minimized by increasing the modulation rate and / or the peak frequency deviation and hence increase the resolution-frequency per cm.

Multipath and clutter problems can be reduced by using microwave absorptive foam around the test-site. Also, pointing the antennae into the sky would minimize the above problems. Smaller bandwidth antennae could also help to minimize multipath and clutter.

The nett SNR of the system could be increased by increasing the transmitted power. This would improve the resolution of the spectral estimator.

The system could be made to run in real-time if a high speed DSP board is used to do the spectral estimation.

The system worked when the antennae were replaced with a test cable - one spike per reflection was obtained. Thus it would be very useful in time domain reflectometry.

9.1 REFERENCES

[9.1] BRYANT, G.H. **Principals Of Microwave Measurements.** IEE ELECTRICAL MEASUREMENT SERIES 5, PETER PEREGRINUS, 1988.

[9.2] DYBDAL, R.B., HURLBUT, K.H., and MORI, T.T. **High-Resolution Instrumentation Radar.**, TRANS. IEEE, IM-36, (1), 1987, 110-114.

[9.3] MENSA, D.L. **High Resolution Radar Imaging**. ARTECH, 1982.

BIBLIOGRAPHY

BRYANT, G.H. **Principals Of Microwave Measurements.** IEE ELECTRICAL MEASUREMENT SERIES 5, PETER PEREGRINUS, 1988.

CLARRICOATS, P. **Portable Radar For The Detection Of Buried Objects.**, Radar Conference Paper 1977.

DARYANANI, G. **Principals Of Active Network Synthesis And Design.** JOHN WILEY & SONS, 1976.

DELAUDER, D., BALANIS, C. and KAZOVSKY, L. **Microcomputer-Based Signal Processor For Short-Range FM Radar.**, TRANS. IEEE, IM-35, (1), 1986, 70-77.

DYBDAL, R.B., HURLBUT, K.H., and MORI, T.T. **High-Resolution Instrumentation Radar.**, TRANS. IEEE, IM-36, (1), 1987, 110-114.

HAYKIN, S. **Digital Communications.** JOHN WILEY & SONS, 1988.

HOROWITZ, P., HILL, W. **The Art Of Electronics.** CAMBRIDGE UNIVERSITY PRESS, 1987.

HOVANESSIAN, S.A. **Radar System Design And Analysis.** Artech, 1984.

HYMANS, A.J., LAIT, J. **Analysis Of A Frequency-Modulated Continuous-Wave Ranging System.** PROC. IEE., VOL. 107B, 1960, 365-372.

KAY, S.M. **Modern Spectral Estimation: Theory And Application.** PRENTICE-HALL, 1987.

LACOSS, R.T. **Data Adaptive Spectral Analysis Methods.** GEOPHYSICS, VOL. 36, NO. 4, 1971.

MARPLE, S.L. **Digital Spectral Analysis With Applications.** PRENTICE-HALL, 1987.

MENSA, D.L. **High Resolution Radar Imaging.** ARTECH, 1982.

MILLMAN, J., HALKIAS, C.C. **Integrated Electronics: Analogue And Digital Circuits And Systems.** McGRAW-HILL, 1972.

MULLIS, C.T., ROBERTS, R.A. **Digital Signal Processing.**, ADDISON-WESLEY, 1987.

SANDER, K.F. **Microwave Components And Systems.**, Addison-Wesley, 1987.

SKOLNIK, M. **Introduction To Radar Systems.** (2nd ed.)., McGraw-Hill, Japan.

STREMLER, F.G. **Introduction To Communication Systems.** (2nd ed.), ADDISON-WESLEY, 1982.

YAMAURA, I. and HIDAKA, T. **The Double-Swept-Frequency Locating Reflectometer.**, TRANS. IEEE, MTT-23, (3), 1975, 316-317.

APPENDICES

APPENDIX A

RIGOROUS DERIVATION OF SAWTOOTH MODULATED FMCW RADAR EQUATION

Propagation delay: $\tau = \dfrac{2 * R}{c}$

frequency: $f(t) = f_{tx} + \dfrac{\delta f}{T_{mod}} * t = \dfrac{d\phi}{dt}$

phase: $\phi = f_{tx} * t + \dfrac{\delta f}{2 * T_{mod}} * t^2$

modulation factor: $k = \dfrac{2 * \pi * \delta f}{2 * T_{mod}} = \dfrac{\pi * \delta f}{T_{mod}}$

where δf is the peak frequency deviation;
T_{mod} is the period of the modulation waveform.

Neglecting amplitudes which are just scaling factors:

Transmit waveform: $TX(t) = \cos(w_{tx}.t + k.t^2)$

Receive waveform: $RX(t) = \cos(w_{tx}.[t-\tau] + k.[t-\tau]^2 + \phi)$

for $0 \leq t \leq T_{mod}$ because of the periodicity.

After Heterodyning, the beat frequency is:

$$f_b = \tfrac{1}{2} * \cos\{ 2.w_{tx}.t - w_{tx}.\tau + 2.k.t^2 - 2.k.t.\tau + k.\tau^2 + \phi\}$$
$$+ \tfrac{1}{2} * \cos\{ w_{tx}.\tau + 2.k.t.\tau - k.\tau^2 - \phi\}$$

so,

$$f_b = \tfrac{1}{2} * \cos\{ 2.w_{tx}.t + 2.k.t^2 - 2.k.t.\tau + \phi_1\}$$
$$+ \tfrac{1}{2} * \cos\{ 2.k.t.\tau + \phi_2\} \qquad .. (1)$$

The first term of equation (1) is filtered out by the IF stage since it is at twice the transmit frequency (approximately 20 GHz). The second term is simply the beat frequency with a range dependent phase shift. Therefore:

$$f_b = \tfrac{1}{2} * \cos\left[2.\pi * \left[\frac{2 * R}{c} * \frac{\delta f}{T_{mod}}\right] * t + \phi_2\right]$$

For triangular modulation a factor of 2 is included for the T_{mod} definition:

$$f_b = \tfrac{1}{2} * \cos\left[2.\pi * \left[\frac{2 * R}{c} * \frac{2 * \delta f}{T_{mod}}\right] * t + \phi_2\right]$$

APPENDIX B

MATHCAD SIMULATION OF TRANSMITTED WAVEFORM

$$T_{mod} := \frac{1}{1 \cdot 10^{8}}$$

$$N := 2^{9}$$

$$i := 0 \,..\, N - 1$$

$$t_i := \frac{i}{N - 1} \cdot T_{mod}$$

$$\delta f := 1 \cdot 10^{9}$$

$$w_c := 8 \cdot 10^{9}$$

$$R := 1 \quad m$$

$$c := 3 \cdot 10^{8}$$

$$j := \sqrt{-1}$$

$$\beta := \delta f \cdot T_{mod}$$

$$\beta = 10$$

$$Tx_i := \cos\left[w_c \cdot t_i + 2 \cdot \pi \cdot \frac{\delta f}{T_{mod}} \cdot t_i^{2}\right]$$

$$f_b := 4 \cdot R \cdot \frac{\delta f}{c \cdot T_{mod}} \qquad f_b = 1.33333 \cdot 10^{9}$$

$$Rx_i := \cos\left[w_c \cdot \left[t_i + 2 \cdot \frac{R}{c}\right] + 2 \cdot \pi \cdot \frac{\delta f}{T_{mod}} \cdot \left[t_i + 2 \cdot \frac{R}{c}\right]^{2}\right]$$

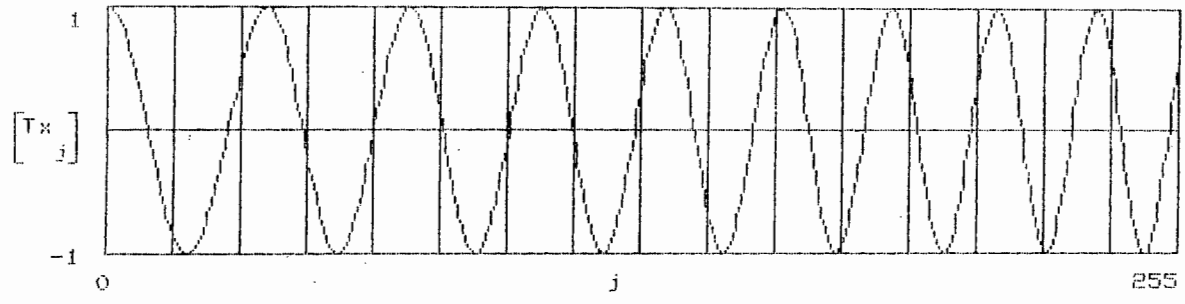
$$TXR_i := Tx_i \cdot Rx_i$$

$$F_1 := fft(Tx)$$

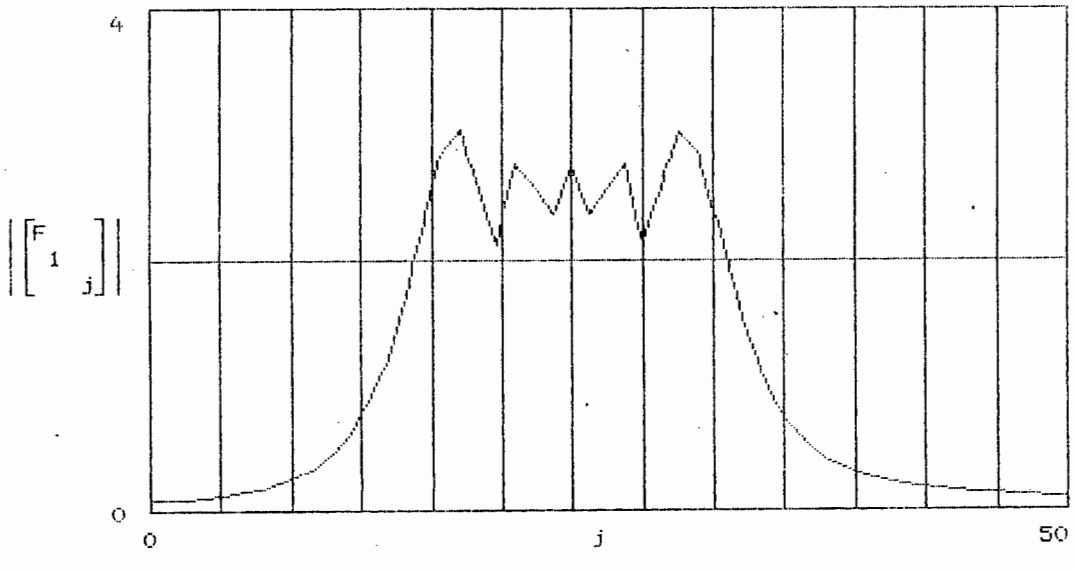
$$F_2 := fft(TXR)$$

$$j := 0 \,..\, \left[\left[\frac{N}{2}\right] - 1\right]$$

TX waveform



TX spectrum



APPENDIX C

MATHCAD SIMULATION OF RECEIVED BEAT FREQUENCY

$$T_{mod} := \frac{1}{10 \cdot 10^{3}}$$

$$N := 2^{9}$$

$$i := 0 \,..\, N - 1$$

$$t_i := \frac{i}{N - 1} \cdot T_{mod}$$

$$\delta f := 4 \cdot 10^{9}$$

$$w_c := 3$$

$$R := 1 \quad m$$

$$c := 3 \cdot 10^{8}$$

$$Tx_i := \cos\left[w_c \cdot t_i + 2 \cdot \pi \cdot \frac{\delta f}{T_{mod}} \cdot t_i^{2}\right]$$

$$Rx_i := \cos\left[w_c \cdot \left[t_i + 2 \cdot \frac{R}{c}\right] + 2 \cdot \pi \cdot \frac{\delta f}{T_{mod}} \cdot \left[t_i + 2 \cdot \frac{R}{c}\right]^{2}\right]$$

$$TXR_i := Tx_i \cdot Rx_i$$

$$F_1 := fft(Tx)$$

$$F_2 := fft(TXR)$$

$$j := 0 \,..\, \left[\left[\frac{N}{2}\right] - 1\right]$$

$$j := \sqrt{-1}$$

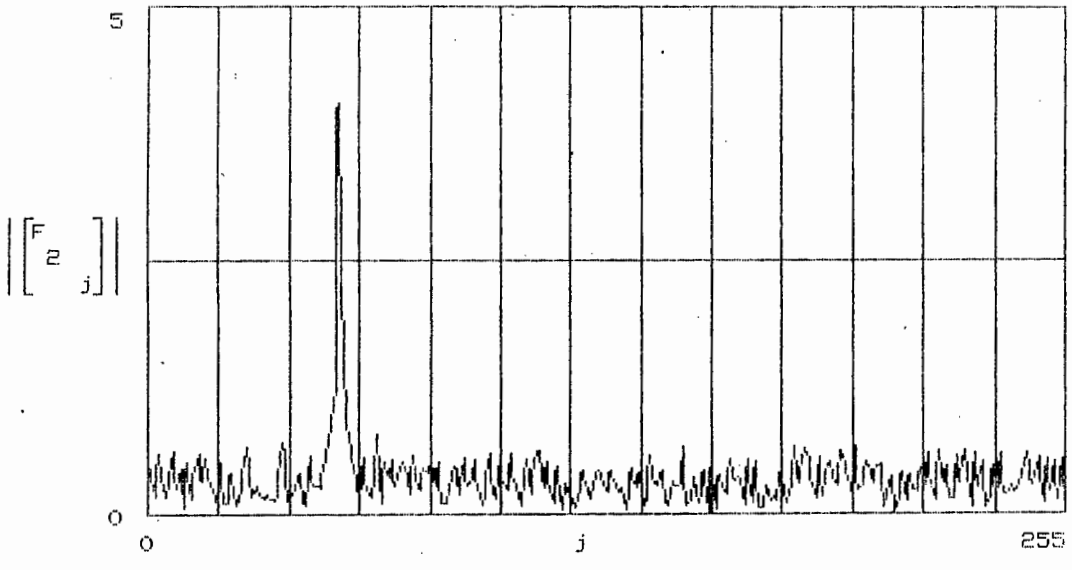
$$\beta := \delta f \cdot T_{mod}$$

$$\beta = 4 \cdot 10^{5}$$

$$f_b := 4 \cdot R \cdot \frac{\delta f}{c \cdot T_{mod}}$$

$$f_b = 5.33333 \cdot 10^{5}$$

FMCW output spectrum



APPENDIX D

LISTING OF PROGRAM GET_LAW

```
PROGRAM get_law;

{ Program written by A. Bas. }

{ This program sets the network analyzer's DAC voltage, and then reads the
  frequency from the microwave frequency counter. The voltage is varied over
  the oscillator tuning range and thus the voltage-frequency characteristic
  is obtained. The network analyzer and the microwave frequency counter are
  connected to the PC via the HPIB interface. }

{$M 32000,0,32000}
{$N+}

uses crt, dos;

var gpout, gpin : text;
    a           : integer;
    v_f_temp    : array [0..2048] of real;
    v,f         : real;


procedure savedisk;

var fname :string;
    n  : integer;
    ff : text;

begin
     writeln;
     writeln('Enter name of disk file: ');
     readln(fname);
     assign(ff,fname);
     rewrite(ff);
     for n := 0 TO 2047 do
         begin
              writeln(ff,v_f_temp[n]);
              writeln(n,')  ',v_f_temp[n]);
         end;
     close(ff);
end;

procedure set_voltage(voltage : real);

var  str1, str2, voltage_str, out_string : string;

begin
     rewrite(GPOUT);
     str1 := 'hpibmode.com';
     str2 := 'HPIB=717';
     exec(str1,str2);
     str(voltage,voltage_str);
     out_string := 'BIAS=' + voltage_str;
     writeln(GPOUT,out_string);
     close(GPOUT);
end;

procedure get_freq(var freq:real);

var  str1, str2, freq_str : string;
     d : integer;

begin
```

APPENDIX E

LISTING OF PROGRAM LIN_LAW

```
        PROGRAM linlaw;

{ Program written by A. Bas. }

{ This program linearizes the voltage-frequency law of the tunable oscillator
  and stores it on disk in EPROM format. }

{$m 32000,0,32000}

uses crt, dos;

var gpout, gpin : text;
    voltage  : real;
    v_f_fin  : array [0..2048] of integer;
    v_f_temp : array [0..2048] of real;

procedure savedisk;

var fname : string;
    n    : integer;
    ff   : file of byte;
    gg   : text;
    l_byte, h_byte : byte;
    v_val : word;

begin
  writeln;
  writeln('Enter name of output linearized file (without ext) : ');
  readln(fname);
  assign(ff,(fname + '.dat'));
  assign(gg,(fname + '.asc'));
  rewrite(ff);
  rewrite(gg);
  for n := 0 TO 511 do
      begin
           v_val := 4095 - v_f_fin[n];
           h_byte := v_val div 256;
           l_byte := v_val mod 256;
           write(ff,l_byte,h_byte);
           writeln(gg,v_f_fin[n]);
           writeln(n,')  ',v_f_fin[n],'  ',l_byte,' ', h_byte);
      end;
  close(ff);
  close(gg);
end;


procedure getdisk;

var fname:string;

n    : integer;
ff   : text;

begin
  writeln;
  writeln('Enter name of varactor file: ');
  readln(fname);
  assign(ff,fname);
  reset(ff);
  for n := 0 TO 1023 do
      begin
```

```
            readln(ff,v_f_temp[n]);
            writeln(n,'  ',v_f_temp[n]);
        end;
  close(ff);
end;

procedure freq_search(f : real; var mid_val : integer);

var  v_start, v_end, SAR : integer;

begin
     v_start := 0;
     v_end := 1023;
     for SAR := 1 to 10 do
         begin
              mid_val := (v_start + v_end) div 2;
              if (f < v_f_temp[mid_val]) then
                 v_end := mid_val
              else v_start := mid_val;
         end;
end;

procedure linearize;

var  f_current, f_start, f_stop : real;
     lin_val , f_num : integer;

begin
     write('Enter START frequency : ');
     readln(f_start);
     write('Enter STOP frequency : ');
     readln(f_stop);

     f_current := f_start;

     for f_num := 0 to 255 do
         begin
              freq_search(f_current, lin_val);
              writeln(f_num,')  linval = ',lin_val);
              v_f_fin[f_num] := lin_val * 4;          {to shift 2 bits}
              v_f_fin[511 - f_num] := lin_val * 4;
              f_current := f_current + (f_stop - f_start) / 255;
         end;
end;

begin
     clrscr;
     getdisk;
     linearize;
     savedisk;
end.
```

APPENDIX F

LINEARIZATION - EPROM TRANSFER MAP VALUES

26	442	934	1730	3038	2742
30	450	946	1750	3066	2718
38	458	954	1766	3094	2694
42	466	966	1786	3126	2666
50	474	978	1802	3154	2642
54	482	990	1818	3182	2618
58	486	998	1838	3210	2590
66	494	1010	1858	3238	2566
70	502	1022	1874	3270	2542
74	510	1034	1894	3298	2518
82	518	1042	1910	3330	2494
86	526	1054	1934	3358	2470
94	530	1066	1950	3390	2446
98	538	1078	1970	3422	2422
106	546	1090	1990	3450	2398
110	554	1102	2010	3482	2374
118	562	1114	2030	3514	2354
122	570	1126	2050	3550	2330
134	578	1138	2070	3578	2306
138	586	1150	2090	3610	2286
146	594	1166	2110	3646	2262
154	602	1178	2130	3678	2238
158	610	1190	2154	3710	2218
166	618	1202	2174	3746	2198
174	626	1214	2198	3746	2174
182	634	1230	2218	3710	2154
186	646	1242	2238	3678	2130
194	654	1258	2262	3646	2110
202	662	1270	2286	3610	2090
210	670	1282	2306	3578	2070
222	678	1298	2330	3550	2050
230	686	1310	2354	3514	2030
238	694	1326	2374	3482	2010
246	702	1338	2398	3450	1990
254	710	1354	2422	3422	1970
262	718	1366	2446	3390	1950
270	726	1382	2470	3358	1934
278	738	1398	2494	3330	1910
286	746	1414	2518	3298	1894
294	754	1426	2542	3270	1874
302	762	1442	2566	3238	1858
314	770	1458	2590	3210	1838
322	778	1470	2618	3182	1818
330	790	1486	2642	3154	1802
338	798	1502	2666	3126	1786
346	806	1518	2694	3094	1766
354	818	1534	2718	3066	1750
362	826	1550	2742	3038	1730
370	834	1566	2770	3010	1714
378	846	1582	2794	2982	1698
382	854	1598	2822	2958	1682
394	862	1614	2846	2926	1662
402	874	1630	2874	2902	1646
406	882	1646	2902	2874	1630
414	894	1662	2926	2846	1614
422	906	1682	2958	2822	1598
430	914	1698	2982	2794	1582
438	926	1714	3010	2770	1566

1550	826	362
1534	818	354
1518	806	346
1502	798	338
1486	790	330
1470	778	322
1458	770	314
1442	762	302
1426	754	294
1414	746	286
1398	738	278
1382	726	270
1366	718	262
1354	710	254
1338	702	246
1326	694	238
1310	686	230
1298	678	222
1282	670	210
1270	662	202
1258	654	194
1242	646	186
1230	634	182
1214	626	174
1202	618	166
1190	610	158
1178	602	154
1166	594	146
1150	586	138
1138	578	134
1126	570	122
1114	562	118
1102	554	110
1090	546	106
1078	538	98
1066	530	94
1054	526	86
1042	518	82
1034	510	74
1022	502	70
1010	494	66
998	486	58
990	482	54
978	474	50
966	466	42
954	458	38
946	450	30
934	442	26
926	438	
914	430	
906	422	
894	414	
882	406	
874	402	
862	394	
854	382	
846	378	
834	370	

APPENDIX G

BUTTERWORTH FILTER GAINS

TABLE 4.2. VCVS LOW-PASS FILTERS

Poles	Butterworth K	Bessel f_n	Bessel K	Chebyshev (0.5dB) f_n	Chebyshev (0.5dB) K	Chebyshev (2.0dB) f_n	Chebyshev (2.0dB) K
2	1.586	1.274	1.268	1.231	1.842	0.907	2.114
4	1.152	1.432	1.084	0.597	1.582	0.471	1.924
	2.235	1.606	1.759	1.031	2.660	0.964	2.782
6	1.068	1.607	1.040	0.396	1.537	0.316	1.891
	1.586	1.692	1.364	0.768	2.448	0.730	2.648
	2.483	1.908	2.023	1.011	2.846	0.983	2.904
8	1.038	1.781	1.024	0.297	1.522	0.238	1.879
	1.337	1.835	1.213	0.599	2.379	0.572	2.605
	1.889	1.956	1.593	0.861	2.711	0.842	2.821
	2.610	2.192	2.184	1.006	2.913	0.990	2.946

APPENDIX H

BURG ALGORITHM (FROM MARPLE)

additive observation noise appears to be less pronounced than for many of the other AR spectral estimators [Swingler 1979A]. Also, the peak location dependence on initial sinusoidal phase [Chen and Stegen 1974] is considerably reduced [Ulrych and Clayton 1976]. Spectral line splitting in which a single sinusoidal component gives rise to two distinct spectral peaks [Fougere et al. 1976, Herring 1980] has never been observed with the modified covariance method [Kay and Marple 1979, Marple 1980]. Illustrations of spectral estimation performance for nonsinusoidal processes are provided in Section 7.10.

7.6 BURG METHOD

In contrast to the autocorrelation, covariance, and modified covariance methods, which estimate the AR parameters directly, the Burg method estimates the reflection coefficients and then uses the Levinson recursion to obtain the AR parameter estimates. The reflection coefficient estimates are obtained by minimizing estimates of the prediction error power for different order predictors in a recursive manner. Specifically, based on the Levinson algorithm (see Figure 6.4), if estimates of the reflection coefficients $\{k_1, k_2, \ldots, k_p\}$ are available, the AR parameters may be estimated as follows:

$$\hat{r}_{xx}[0] = \frac{1}{N}\sum_{n=0}^{N-1} |x[n]|^2$$

$$\hat{a}[1] = k_1$$

$$\hat{\rho}_1 = (1 - |\hat{a}_1[1]|^2)\hat{r}_{xx}[0].$$

For $k = 2, 3, \ldots, p$,

$$\hat{a}_k[i] = \begin{cases} \hat{a}_{k-1}[i] + k_k\hat{a}^*_{k-1}[k-i] & \text{for } i = 1, 2, \ldots, k-1 \\ k_k & \text{for } i = k \end{cases} \tag{7.25}$$

$$\hat{\rho}_k = (1 - |\hat{a}_k[k]|^2)\hat{\rho}_{k-1}. \tag{7.26}$$

The estimates of the AR filter parameters are $\{\hat{a}_p[1], \hat{a}_p[2], \ldots, \hat{a}_p[p]\}$ and the white noise variance estimates is $\hat{\rho}_p$. It remains only to obtain estimates of the reflection coefficients. In deriving the kth reflection coefficient estimate, Burg assumed that the $(k-1)$st order prediction error filter coefficients had already been estimated as $\{\hat{a}_{k-1}[1], \hat{a}_{k-1}[2], \ldots, \hat{a}_{k-1}[k-1]\}$, having been obtained by minimizing the $(k-1)$st order prediction error power. Using the Levinson recursion the coefficients of the kth order prediction error filter depend only on k_k according to (7.25), and hence the kth order prediction error power estimate also depends only on k_k. Burg proposed to estimate k_k by minimizing the *average* of the estimates of the forward and backward prediction error powers. This approach, which preceded the modified covariance method, is a *constrained minimization* of $\hat{\rho}$ as given by (7.18) in contrast to the modified covariance method, which minimizes $\hat{\rho}$ in an unconstrained manner. The constrained or recursive minimization will not in general produce a global minimum. Hence, to obtain the estimate of k_k, we minimize

$$\hat{\rho}_k = \tfrac{1}{2}(\hat{\rho}^f_k + \hat{\rho}^b_k) \tag{7.27}$$

where

$$\hat{\rho}^f_k = \frac{1}{N-k}\sum_{n=k}^{N-1}\left| x[n] + \sum_{i=1}^{k} a_k[i]x[n-i]\right|^2 \tag{7.28}$$

$$\hat{\rho}^b_k = \frac{1}{N-k}\sum_{n=0}^{N-1-k}\left| x[n] + \sum_{i=1}^{k} a^*_k[i]x[n+i]\right|^2 \tag{7.29}$$

and

$$a_k[i] = \begin{cases} \hat{a}_{k-1}[i] + k_k\hat{a}^*_{k-1}[k-i] & \text{for } i = 1, 2, \ldots, k-1 \\ k_k & \text{for } i = k. \end{cases} \tag{7.30}$$

$\hat{\rho}^f_k$ and $\hat{\rho}^b_k$ are functions *only* of k_k since the $(k-1)$st order prediction coefficients are assumed to have already been estimated by minimizing $\hat{\rho}_{k-1}$. Defining estimated forward and backward prediction errors in a similar fashion to those presented in Chapter 6 [see (6.15) and (6.57)],

$$\hat{e}^f_k[n] = x[n] + \sum_{i=1}^{k} a_k[i]x[n-i] \tag{7.31}$$

$$\hat{e}^b_k[n] = x[n-k] + \sum_{i=1}^{k} a^*_k[i]x[n-k+i] \tag{7.32}$$

the forward prediction error power estimate becomes

$$\hat{\rho}^f_k = \frac{1}{N-k}\sum_{n=k}^{N-1} |\hat{e}^f_k[n]|^2 \tag{7.33}$$

while the backward prediction error power estimate becomes

$$\hat{\rho}^b_k = \frac{1}{N-k}\sum_{n=k}^{N-1} |\hat{e}^b_k[n]|^2. \tag{7.34}$$

The lattice filter relations which describe the model order update of the forward and backward prediction error time series as given by (6.58) are applicable here. [Alternatively, substitute (7.30) in (7.31) and (7.32).] These are

$$\hat{e}^f_k[n] = \hat{e}^f_{k-1}[n] + k_k\hat{e}^b_{k-1}[n-1] \tag{7.35}$$

$$\hat{e}^b_k[n] = \hat{e}^b_{k-1}[n-1] + k^*_k\hat{e}^f_{k-1}[n] \tag{7.36}$$

where

$$\hat{e}^f_0[n] = \hat{e}^b_0[n] = x[n].$$

When these relations are substituted into (7.33) and (7.34) and then into (7.27),

the average estimated prediction error power becomes

$$\hat{\rho}_k = \frac{1}{2(N-k)} \sum_{n=k}^{N-1} [\,|\, \hat{e}_{k-1}^f[n] + k_k \hat{e}_{k-1}^b[n-1] \,|^2 + |\, \hat{e}_{k-1}^b[n-1] + k_k^* \hat{e}_{k-1}^f[n] \,|^2\,]. \tag{7.37}$$

Differentiating $\hat{\rho}_k$ with respect to the real and imaginary parts of k_k by using the complex gradient [see (2.79)] and setting the result equal to zero yields

$$\frac{\partial \hat{\rho}_k}{\partial k_k} = \frac{1}{N-k} \sum_{n=k}^{N-1} \{(\hat{e}_{k-1}^f[n] + k_k \hat{e}_{k-1}^b[n-1])\hat{e}_{k-1}^b[n-1]^* + (\hat{e}_{k-1}^b[n-1]^* + k_k \hat{e}_{k-1}^f[n]^*)\hat{e}_{k-1}^f[n]\} = 0.$$

Solving for k_k, we have

$$k_k = \frac{-2 \sum_{n=k}^{N-1} \hat{e}_{k-1}^f[n]\hat{e}_{k-1}^b[n-1]^*}{\sum_{n=k}^{N-1} (\,|\, \hat{e}_{k-1}^f[n] \,|^2 + |\, \hat{e}_{k-1}^b[n-1] \,|^2)} \tag{7.38}$$

which is the Burg method for estimation of the kth reflection coefficient. It can be shown that $|\, \hat{k}_k \,| \leq 1$ (see Problem 7.6), which agrees with the theoretical constraint that a partial correlation coefficient should be less than 1 in magnitude [see (6.42)]. In summary, the Burg method for estimation of the AR parameters is

$$\begin{aligned} \hat{r}_{xx}[0] &= \frac{1}{N} \sum_{n=0}^{N-1} |\, x[n] \,|^2 \\ \hat{\rho}_0 &= \hat{r}_{xx}[0] \\ \hat{e}_0^f[n] &= x[n] \qquad n = 1, 2, \ldots, N-1 \\ \hat{e}_0^b[n] &= x[n] \qquad n = 0, 1, \ldots, N-2. \end{aligned} \tag{7.39}$$

For $k = 1, 2, \ldots, p$,

$$k_k = \frac{-2 \sum_{n=k}^{N-1} \hat{e}_{k-1}^f[n]\hat{e}_{k-1}^b[n-1]^*}{\sum_{n=k}^{N-1} (\,|\, \hat{e}_{k-1}^f[n] \,|^2 + |\, \hat{e}_{k-1}^b[n-1] \,|^2)}$$

$$\hat{\rho}_k = (1 - |\, k_k \,|^2)\hat{\rho}_{k-1} \tag{7.40}$$

$$\hat{a}_k[i] = \begin{cases} \hat{a}_{k-1}[i] + k_k \hat{a}_{k-1}^*[k-i] & \text{for } i = 1, 2, \ldots, k-1 \\ k_k & \text{for } i = k. \end{cases}$$

(If $k = 1$, $\hat{a}_1[1] = \hat{k}_1$.)

$$\begin{aligned} \hat{e}_k^f[n] &= \hat{e}_{k-1}^f[n] + k_k \hat{e}_{k-1}^b[n-1] \qquad n = k+1, k+2, \ldots, N-1 \\ \hat{e}_k^b[n] &= \hat{e}_{k-1}^b[n-1] + k_k^* \hat{e}_{k-1}^f[n] \qquad n = k, k+1, \ldots, N-2 \end{aligned} \tag{7.41}$$

The estimates are given as $\{\hat{a}_p[1], \hat{a}_p[2], \ldots, \hat{a}_p[p], \hat{\rho}_p\}$. A computer program entitled BURG, which is given in Appendix 7D, implements this algorithm.

The Burg method yields estimated poles that are on or inside the unit circle. This is due to the property $|\, \hat{k}_k \,| \leq 1$. To reduce the necessary computation of the Burg algorithm, we can recursively compute the denominator of (7.38) by using the lattice recursions of (7.41). It may be shown (see Problem 7.7) that if DEN(k) denotes the denominator of (7.38), then [Anderson 1978]

$$\text{DEN}(k) = (1 - |\, \hat{k}_{k-1} \,|^2)\,\text{DEN}(k-1) - |\, \hat{e}_{k-1}^f[k-1] \,|^2 - |\, \hat{e}_{k-1}^b[N-1] \,|^2. \tag{7.42}$$

In general, the Burg algorithm produces accurate AR spectral estimates for data which are truly AR [Nuttall 1976]. For sinusoidal data, however, some difficulties have been observed. The Burg algorithm is subject to line splitting [Fougere et al. 1976, Herring 1980] and peak locations are strongly dependent on phase [Chen and Stegen 1974]. To reduce the phase dependence, we can use a modified reflection coefficient estimate,

$$k_k^w = \frac{-2 \sum_{n=k}^{N-1} w_k[n]\hat{e}_{k-1}^f[n]\hat{e}_{k-1}^b[n-1]^*}{\sum_{n=k}^{N-1} w_k[n] (\,|\, \hat{e}_{k-1}^f[n] \,|^2 + |\, \hat{e}_{k-1}^b[n-1] \,|^2)} \tag{7.43}$$

where $w_k[n]$ is a suitably chosen window with nonnegative weights. This modified reflection coefficient estimate, which was originally proposed by Burg [1975], has been found to reduce phase dependence effects [Swingler 1979B, Kaveh and Lippert 1983].

The Burg estimate of the reflection coefficient is only one of a large class of estimates that maintain the minimum-phase property. One that was proposed by Itakura and Saito [1971] replaces the theoretical ensemble average [see (6.42)] by their time averages to yield

$$k_k^l = -\frac{\sum_{n=k}^{N-1} \hat{e}_{k-1}^f[n]\hat{e}_{k-1}^b[n-1]^*}{\sqrt{\sum_{n=k}^{N-1} |\, \hat{e}_{k-1}^f[n] \,|^2 \sum_{n=k}^{N-1} |\, \hat{e}_{k-1}^b[n-1] \,|^2}} \tag{7.44}$$

It can be shown (see Problem 7.8) that

$$|\, k_k^B \,| \leq |\, k_k^l \,| \tag{7.45}$$

where k_k^B denotes the Burg estimate given by (7.38). Many other possible estimators exist [Makhoul 1977], although in practice the differences in spectral estimation performance appear to be minor. Some illustrations of the Burg spectral estimator are given in Section 7.10.

APPENDIX I

LISTING OF PROGRAM TO OBTAIN THE BURG FILTER COEFFICIENTS

(CODED TO PASCAL FROM MARPLE'S FORTRAN PROGRAM)

```
        program AR_Burg;

{ Program from Marple coded by D. Levy from Fortran to Pascal. This program
  calculates the AR coefficients using the Burg (maximum entropy) method.

{$N+}

uses crt,dos;

const  maxorder=512;
       maxsamples=512;
type
  float = double;

  complex = record
                  Re, Im : float;
            end;
  vector1 = array[1..maxsamples] of complex;
            vecptr1 = ^vector1;
  vector2 = array[1..maxorder] of complex;
            vecptr2 = ^vector2;

var
  th1, th2, tm1, tm2, ts1, ts2, ths1, ths2 : word;
  IOcode : word;
  I1 : integer;
  infile : text;          {file of complex;}
  in_name : string;
  outfile : text;         {file of complex;}
  out_name : string;
  IP : integer;           {polynomial order}
  ISTAT : integer;
  N : integer;            {number of data samples}
  X : vecptr1;            {N  samples}
  A : vecptr2;            {IP autoregressive parameters}
  P : float;              {input noise variance = input PSD}

{$I BURGMATH.INC}

procedure BURG(N, IP : integer; X : vecptr1; var P : float ;
               var A : vecptr2; var ISTAT : integer);
var
  EF : vecptr1;           {forward  error}
  EB : vecptr1;           {backward error}
  NUM : complex;          {numerator}
  DEN : float;            {denominator}
  KHALF, KJ : integer;
  TEMP : float;
  SAVE1, SAVE2 : complex;
  J, K : integer;         {loop counters}
  save3C : complex;

const
   zero : complex=(Re:0.0;Im:0.0);

begin
 new(EF); new(EB);
 if (IP = 0) then begin ISTAT:=2; exit  end;    (* *)
 ISTAT:=0;                       {initialization}
 P:=0.0;
 for J:=1 to N do
   begin
```

```
    squaremag(X^[J],save3C);
    P := P + save3C.Re;
  end;
DEN := P*2;
P:=P/N;
for J := 1 to N do
  begin
    EF^[J]:=X^[J];
    EB^[J]:=X^[J];
  end;
  TEMP:=1.0;
  K:=0;

   {Main recursion}
repeat
  inc(K);
  NUM:=zero;
  for J:=succ(K) to N do
   begin
      conjugate(EB^[pred(J)],save3C);
      multiply(EF^[J],save3C,save3C);
      add(NUM,save3C,NUM);
   end;
  DEN:=TEMP*DEN
         -sqr(EF^[K].Re) - sqr(EF^[K].Im)
         -sqr(EB^[N].Re) - sqr(EB^[N].Im);
  divideCR(NUM,-DEN*0.5,SAVE1);
  TEMP:=1.0 - sqr(SAVE1.Re) - sqr(SAVE1.Im);
  P:=P*TEMP;

  if TEMP <= 0.0 then
   begin
      ISTAT:=1;
   end
  else
   begin
      A^[K]:=SAVE1;

      if K<>1 then
       begin
          KHALF := K div 2;
          for J:=1 to KHALF do
           begin
              KJ:=K-J;
              SAVE2:=A^[J];

              conjugate(A^[KJ],save3C);
              multiply(SAVE1,save3C,save3C);
              add(save3C,SAVE2,A^[J]);

              if J <> KJ then
                begin
                  conjugate(SAVE2,save3C);
                  multiply(SAVE1,save3C,save3C);
                  add(A^[KJ],save3C,A^[KJ]);
                end;
           end;
       end;

      if K<IP then
           for J := N downto succ(K) do
             begin
```

```
                SAVE2:=EF^[J];

                multiply(SAVE1,EB^[pred(J)],save3C);
                add(SAVE2,save3C,EF^[J]);

                conjugate(SAVE1,save3C);
                multiply(SAVE2,save3C,save3C);
                add(EB^[pred(J)],save3C,EB^[J]);
              end;
    end; {if ill conditioned}
 until ((K=IP)  or  (ISTAT<>0));
end;  {procedure BURG}

begin;
  new(X); new(A);
  clrscr;
{$I-}
  repeat
      write('Enter name for input file (eg. A:MEASURE.DAT) : ');
      readln(in_name);
      if in_name = '' then
         in_name := 'c:\alonbas\adc1.dat'
      else
          in_name := 'c:\perfect\' + in_name + '.dat';
      writeln(in_name);
      assign(infile,in_name);
      reset(infile);
      IOcode:=IOresult;
      if IOcode<>0 then writeln('Problem! Try again.');
  until IOcode=0;
  writeln;
  I1 := 1;
  repeat
    begin
      readln(infile,X^[I1].Re,X^[I1].Im);
      writeln(X^[I1].Re,'  ',X^[I1].Im);
      inc(I1);
    end;
  until eof(infile);
    N := pred(I1);
    close(infile);
  writeln(N,'  coefficients');

  writeln;
  writeln('Enter order of AR process');
  readln(IP);
  gettime(th1,tm1,ts1,ths1);
  BURG(N,IP,X,P,A,ISTAT);
  gettime(th2,tm2,ts2,ths2);
  writeln(ths2-ths1,' ',ts2-ts1);

  case ISTAT of
    0:begin
       repeat
           write('Enter name for output file (eg. A:AR_COEF.DAT) : ');
           readln(out_name);
           if out_name = '' then
              out_name := 'c:\alonbas\ar1.dat'
           else
               out_name := 'a:\burg\' + out_name + '.dat';
           writeln(out_name);
           assign(outfile,out_name);
```

```
            rewrite(outfile);
            IOcode:=IOresult;
            if IOcode<>0 then writeln('Problem! Try again.');
          until IOcode=0;
          writeln;
          for I1 := 1 to IP do
            begin
              writeln(outfile,A^[I1].Re:25:15,' ',A^[I1].Im:25:15);
              writeln(A^[I1].Re:25:15,' ',A^[I1].Im:25:15);
            end;
            close(outfile);
            writeln('sig2 = ',P);
        end;
      1:writeln('Ill conditioned data');
      2:writeln('AR process must have at least one pole');
    end;
end.
```

```
{ General purpose complex number routines     INCLUDE FILE }

procedure conjugate(A:complex; var C:complex);
begin
  C.Im := -A.Im;
  C.Re :=  A.Re;
end;

procedure multiply(A,B:complex;var C:complex);
begin
  C.Re := A.Re*B.Re - A.Im*B.Im;
  C.Im := A.Re*B.Im + A.Im*B.Re;
end;

procedure multiplyCR(A:complex;B:real;var C:complex);
begin
  C.Re := A.Re*B;
  C.Im := A.Im*B;
end;

procedure add(A,B:complex;var C:complex);
begin
  C.Re := A.Re + B.Re;
  C.Im := A.Im + B.Im;
end;

procedure subtract(A,B:complex;var C:complex);
begin
  C.Re := A.Re - B.Re;
  C.Im := A.Im - B.Im;
end;

procedure squaremag(A:complex; var C:complex);
begin
  C.Re := sqr(A.Re) + sqr(A.Im);
  C.Im := 0;
end;

procedure divide(A,B:complex; var C:complex);
var D:float;
begin
  D    := sqr(B.Re) + sqr(B.Im);
  C.Re := (A.Re*B.Re + A.Im*B.Im)/D;
  C.Im := (A.Im*B.Re - A.Re*B.Im)/D;
end;

procedure divideCR(A:complex; B:real; var C:complex);
begin
  C.Re := A.Re/B;
  C.Im := A.Im/B;
end;
```

APPENDIX J

MODIFIED COVARIANCE ALGORITHM (FROM MARPLE)

described more fully in Chapter 13 are usually based on the covariance method or a variant thereof, the modified covariance method, described in the next section.

The covariance method is identical to the modern version of Prony's method for pole estimation. As originally proposed, Prony's method [Prony 1795] is a technique for identifying the frequencies f_i, damping factors α_i, amplitudes A_i, and phases ϕ_i of real exponential signals, or

$$x[n] = \sum_{i=1}^{p} A_{c_i} \exp\left[(\alpha_i + j2\pi f_i)n\right] \qquad n \geq 0 \tag{7.10}$$

where $A_{c_i} = A_i \exp(j\phi_i)$ is a complex amplitude. For real signals the *modes* $z_i = \exp(\alpha_i + j2\pi f_i)$ and the complex amplitudes A_{c_i} must occur in complex-conjugate pairs. Since the data may be written as

$$x[n] = \sum_{i=1}^{p} A_{c_i} z_i^n \qquad n \geq 0$$

it is well known [Oppenheim and Schafer 1975] that $x[n]$ may be generated by the recursive difference equation (see Problem 7.5)

$$x[n] = -\sum_{k=1}^{p} a[k]x[n-k] \qquad n \geq p \tag{7.11}$$

with appropriate initial conditions $\{x[0], x[1], \ldots, x[p-1]\}$. The signal may also be viewed as the natural response of an all-pole filter with the given initial conditions. The coefficients $a[k]$ are related to the modes or *poles* z_i by

$$\mathcal{A}(z) = 1 + \sum_{k=1}^{p} a[k]z^{-k} = \prod_{i=1}^{p} (1 - z_i z^{-1}). \tag{7.12}$$

If we are given the data $\{x[0], x[1], \ldots, x[2p-1]\}$, the poles may be determined *exactly* by solving the set of linear equations given by (7.11) for $n = p, p+1, \ldots, 2p-1$ to find the $a[k]$'s and then rooting the polynomial of (7.12).

When noise is present the original Prony method performs poorly [Van Blaricum and Mittra 1978] since (7.11) no longer holds. Attempts have been made to extend Prony's method to the case of exponential signals in noise. If $y[n]$ denotes the noise corrupted process and $w[n]$ denotes white observation noise so that

$$y[n] = x[n] + w[n]$$

then (7.11) becomes

$$y[n] - w[n] = -\sum_{k=1}^{p} a[k](y[n-k] - w[n-k])$$

or

$$y[n] = -\sum_{k=1}^{p} a[k]y[n-k] + w[n] + \sum_{k=1}^{p} a[k]w[n-k]. \tag{7.13}$$

Note that $y[n]$ has the form of an ARMA(p, p) process and in fact for a pure sinusoid, a special case of (7.10), in white noise, (7.13) may be considered as the limiting form of an ARMA process (see Section 9.7). However, for exponential signals that are not WSS, no such interpretation is possible. If we now let

$$\epsilon[n] = w[n] + \sum_{k=1}^{p} a[k]w[n-k] \tag{7.14}$$

which may be thought of as an error due to noise, a least squares estimator of the $a[k]$'s is found by minimizing

$$\frac{1}{N-p}\sum_{n=p}^{N-1} |\epsilon[n]|^2 = \frac{1}{N-p}\sum_{n=p}^{N-1} \left| y[n] + \sum_{k=1}^{p} a[k]y[n-k] \right|^2. \tag{7.15}$$

This method has been termed the *extended Prony method*. Note that the minimization of (7.15) is *identical* to that encountered in the covariance method and thus the *extended Prony method and covariance method are identical*. The underlying assumptions, however, are radically different. From the results in Chapter 6 which illustrated the poor performance of AR spectral estimators for sinusoids in noise (see Figure 6.11 for large errors in the pole position estimates), it would be expected that even the extended Prony method would perform poorly in the presence of noise. This has been demonstrated by Kumaresan [1982], although the extended Prony method is clearly superior to the original method. It should also be mentioned that since (7.14) is not a white Gaussian process, the minimization of (7.15) *does not produce* an approximate MLE of the $a[k]$'s. An approximate MLE that is based on the representation of (7.11) is described in Section 13.5 for sinusoids in white Gaussian noise. For exponential signals in white noise, improved Prony-type methods may be found in Kumaresan [1982]. Finally, if the data are noiseless, consisting of p complex sinusoids, the extended Prony method or covariance method will yield the correct pole positions. This is because (7.15) will be zero and hence minimized for the true $a[k]$'s due to (7.11) (see Problem 13.9). Further discussions of Prony's method as well as some computer subroutine implementations may be found in the book by Marple [1987].

Some computer simulation results that illustrate the behavior of the covariance method are included in Section 7.10. They generally indicate that the resolution of the covariance method exceeds that of the autocorrelation method.

7.5 MODIFIED COVARIANCE METHOD

In Chapter 6 it was shown that for an AR(p) process the optimal forward predictor is

$$\hat{x}[n] = -\sum_{k=1}^{p} a[k]x[n-k] \tag{7.16}$$

while the optimal backward predictor is [see (6.57)]

$$\hat{x}[n] = -\sum_{k=1}^{p} a^*[k]x[n+k] \tag{7.17}$$

where the $a[k]$'s are the AR filter parameters. In either case the minimum prediction error power is just the white noise variance σ^2. The modified covariance method estimates the AR parameters by minimizing the *average* of the estimated forward and backward prediction error powers, or

$$\hat{\rho} = \frac{1}{2}(\hat{\rho}^f + \hat{\rho}^b) \tag{7.18}$$

where

$$\begin{aligned}\hat{\rho}^f &= \frac{1}{N-p}\sum_{n=p}^{N-1}\left|x[n] + \sum_{k=1}^{p} a[k]x[n-k]\right|^2 \\ \hat{\rho}^b &= \frac{1}{N-p}\sum_{n=0}^{N-1-p}\left|x[n] + \sum_{k=1}^{p} a^*[k]x[n+k]\right|^2.\end{aligned} \tag{7.19}$$

As in the covariance method the summations are only over the prediction errors that involve observed data samples. Note that an alternative way of viewing this estimator is to recognize that $\hat{\rho}^b$ is the prediction error power estimate obtained by "flipping the data record around" and complex conjugating it (i.e., letting $x'[0] = x^*[N-1]$, $x'[1] = x^*[N-2]$, etc.) and applying a forward predictor to this new data set. In this manner we obtain some "extra" data points and hence more prediction errors over which to average. Note that for any set of $a[k]$'s the forward and backward prediction error estimates will be slightly different due to the range of the summations. This procedure is also equivalent to combining the prediction errors of the forward and backward AR models described in Section 5.8.

To minimize (7.18) we can differentiate $\hat{\rho}$ with respect to the real and imaginary parts of $a[k]$ for $k = 1, 2, \ldots, p$. Alternatively, as shown in Section 2.8, we can take advantage of the complex gradient relationship given by (2.79) to yield

$$\begin{aligned}\frac{\partial\hat{\rho}}{\partial a[l]} &= \frac{1}{N-p}\left[\sum_{n=p}^{N-1}\left(x[n] + \sum_{k=1}^{p} a[k]x[n-k]\right)x^*[n-l]\right. \\ &\quad \left. + \sum_{n=0}^{N-1-p}\left(x^*[n] + \sum_{k=1}^{p} a[k]x^*[n+k]\right)x[n+l]\right] \\ &= 0 \qquad l = 1, 2, \ldots, p.\end{aligned} \tag{7.20}$$

After some simplification this becomes

$$\begin{aligned}&\sum_{k=1}^{p}\hat{a}[k]\left(\sum_{n=p}^{N-1}x[n-k]x^*[n-l] + \sum_{n=0}^{N-1-p}x^*[n+k]x[n+l]\right) \\ &\qquad = -\left(\sum_{n=p}^{N-1}x[n]x^*[n-l] + \sum_{n=0}^{N-1-p}x^*[n]x[n+l]\right)\end{aligned} \tag{7.21}$$

for $l = 1, 2, \ldots, p$. Letting

$$c_{xx}[j,k] = \frac{1}{2(N-p)}\left(\sum_{n=p}^{N-1}x^*[n-j]x[n-k] + \sum_{n=0}^{N-1-p}x[n+j]x^*[n+k]\right) \tag{7.22}$$

(7.21) can be written in the identical matrix form as (7.8), or

$$\begin{bmatrix} c_{xx}[1,1] & c_{xx}[1,2] & \cdots & c_{xx}[1,p] \\ c_{xx}[2,1] & c_{xx}[2,2] & \cdots & c_{xx}[2,p] \\ \vdots & \vdots & \ddots & \vdots \\ c_{xx}[p,1] & c_{xx}[p,2] & \cdots & c_{xx}[p,p] \end{bmatrix}\begin{bmatrix}\hat{a}[1]\\ \hat{a}[2]\\ \vdots \\ \hat{a}[p]\end{bmatrix} = -\begin{bmatrix} c_{xx}[1,0]\\ c_{xx}[2,0]\\ \vdots \\ c_{xx}[p,0]\end{bmatrix}. \tag{7.23}$$

The estimate of the white noise variance is

$$\begin{aligned}\hat{\sigma}^2 = \hat{\rho}_{\text{MIN}} = \frac{1}{2(N-p)}&\left[\sum_{n=p}^{N-1}\left(x[n] + \sum_{k=1}^{p}\hat{a}[k]x[n-k]\right)x^*[n]\right. \\ &\left. + \sum_{n=0}^{N-1-p}\left(x^*[n] + \sum_{k=1}^{p}\hat{a}[k]x^*[n+k]\right)x[n]\right]\end{aligned}$$

where (7.20) has been used, or finally,

$$\hat{\sigma}^2 = c_{xx}[0,0] + \sum_{k=1}^{p}\hat{a}[k]c_{xx}[0,k] \tag{7.24}$$

where $c_{xx}[j,k]$ is defined by (7.22). It is observed that the modified covariance method is identical to the covariance method except for the definition of $c_{xx}[j,k]$, the autocorrelation estimator. The matrix in (7.23) is hermitian ($c_{xx}[k,j] = c_{xx}^*[j,k]$) and positive definite (excluding the pure sinusoid case), so that the Cholesky decomposition may be used to solve the linear equations (see Section 2.7). A computer program entitled COVMCOV, which is given in Appendix 7C, implements the modified covariance method. A more computationally efficient means of solving the equations derived by Marple [1980] takes advantage of the special structure of the equations. As in the covariance method a singular matrix will occur for data consisting of $p-1$ or fewer sinusoids (see Problem 7.4). For p sinusoids the modified covariance method may be used to obtain perfect estimates of the frequencies. In Chapter 13 we present more details on the frequency estimation problem. The modified covariance method does not guarantee a stable all-pole filter, although in practice the estimated poles usually fall inside the unit circle. This method of AR parameter estimation was originally proposed by Nuttall [1976], who termed it the *forward-backward* method and also, independently by Ulrych and Clayton [1976], who called it the *least squares* approach.

The modified covariance method appears to yield statistically stable spectral estimates with high resolution [Nuttall 1976, Kay 1983, Shon and Mehrota 1984]. For data consisting of sinusoids in white noise a number of desirable properties of the estimator have been observed from computer simulations. The usual shifting of the peaks of an AR spectral estimate from the true frequency locations due to

APPENDIX K

LISTING OF PROGRAM TO OBTAIN THE MODIFIED COVARIANCE FILTER CO-EFFICIENTS

(CODED TO PASCAL FROM MARPLE'S FORTRAN PROGRAM)

```
program Modified_covariance;

{ Program from Marple coded from Fortran to Pascal by A. Bas }

{ This program calculates the AR coefficients using the Modified
  covariance method. The number of poles required is entered and the results
  are stored to disc. }

{$N+}

uses crt, dos;

const  maxorder=100;
       maxo1=101;
       maxsamples=512;

type
    float = double;

    complex = record
                 Re,Im : float;
              end;

    data_pnts = array[1..maxsamples] of complex;

    filt_pnts = array [1..maxorder] of complex;

    filt_pnts1 = array[1..maxo1] of complex;

var
   th1, th2, tm1, tm2, ts1, ts2, ths1, ths2 : word;
   IOcode  : word;
   infile  : text;          {file of complex;}
   in_name : string;
   in_name1 : string;
   outfile : text;          {file of complex;}
   out_name : string;
   N : integer;             {number of data samples}
   X : data_pnts;
   ISTAT : integer;
   A : filt_pnts;
   P : float;
   I1 : integer;
   ch : char;
   IP : integer;            {AR order (no. of poles)}


{$I c:\alonbas\BURGMATH.INC}

procedure MCOV(N, IP : integer; X : data_pnts; var P : float;
               var A : filt_pnts; var ISTAT : integer);

var
   C , D : filt_pnts1;
   R : filt_pnts;
   LAMBDA, THETA, XI, PSI, val, val2 : complex;
   EF, EB, C1, C2, C3, C4, SAVE1, SAVE2, SAVE3, SAVE4 : complex;
   M, K, MK : integer;            {loop counters}
   DELTA, GAMMA, R1, R2, R3, R4, R5 : float;

const
```

```
    zero : complex=(Re:0.0;Im:0.0);

begin
    R1 := 0.0;
    for K := 2 to N-1 do
        begin
            squaremag(X[K],val);
            R1 := R1 + 2.0 * val.Re;
        end;

    squaremag(X[1],val);
    R2 := val.Re;

    squaremag(X[N],val);
    R3 := val.Re;

    R4 := 1.0 / (R1 + 2.0*(R2 + R3));
    P := R1 + R2 + R3;
    DELTA := 1.0 - R2 * R4;
    GAMMA := 1.0 - R3 * R4;

    multiply(X[1],X[N],val);
    conjugate(val,val);
    multiplyCR(val,R4,LAMBDA);

    multiplyCR(X[N],R4,C[1]);

    conjugate(X[1],val);
    multiplyCR(val,R4,D[1]);

    ISTAT := 0;
    M := 0;
    if IP = 0 then
        begin
            P := (0.5 * R1 + R2 + R3) / N;
            exit;
        end;

  repeat
    M := M + 1;
    SAVE1 := zero;
    for K := (M+1) to N do
        begin
            conjugate(X[K-M],val);
            multiply(X[K],val,val);
            add(SAVE1,val,SAVE1);
        end;

    multiplyCR(SAVE1,2.0,SAVE1);

    conjugate(SAVE1,R[M]);

    multiply(X[N],D[1],THETA);

    multiply(X[N],C[1],PSI);

    conjugate(X[1],val);
    multiply(val,D[1],XI);

    if (M <> 1) then
       for K := 1 to (M-1) do
           begin
```

```
                multiply(X[N-K],D[K+1],val);
                add(THETA,val,THETA);

                multiply(X[N-K],C[K+1],val);
                add(PSI,val,PSI);

                conjugate(X[K+1],val);
                multiply(val,D[K+1],val);
                add(XI,val,XI);

                conjugate(X[N+1-M+K],val);
                multiply(X[N+1-M],val,val);
                subtract(R[K],val,R[K]);
                conjugate(X[M],val);
                multiply(val,X[M-K],val);
                subtract(R[K],val,R[K]);

                conjugate(R[K],val);
                multiply(val,A[M-K],val);
                add(SAVE1,val,SAVE1);
        end;
divideCR(SAVE1,-P,C1);
A[M] := C1;
squaremag(C1,val);
P := P * (1.0 - val.Re);
if (M<>1) then
   for K := 1 to (M div 2) do
       begin
            MK := M - K;
            SAVE1 := A[K];

            conjugate(A[MK],val);
            multiply(C1,val,val);
            add(SAVE1,val,A[K]);

            if (K <> MK) then
               begin
                    conjugate(SAVE1,val);
                    multiply(C1,val,val);
                    add(A[MK],val,A[MK]);
               end;
       end;

if M = IP then
    begin
         P := 0.5 * P / (N - M);
         exit;
    end;

squaremag(LAMBDA,val);
R1 := 1.0 /(DELTA * GAMMA - val.Re);

conjugate(LAMBDA,val);
multiply(THETA,val,val);
multiplyCR(PSI,DELTA,val2);
add(val,val2,val);
multiplyCR(val,R1,C1);

multiply(PSI,LAMBDA,val);
multiplyCR(THETA,GAMMA,val2);
add(val,val2,val);
multiplyCR(val,R1,C2);
```

```
conjugate(LAMBDA,val);
multiply(XI,val,val);
multiplyCR(THETA,DELTA,val2);
add(val,val2,val);
multiplyCR(val,R1,C3);

multiply(THETA,LAMBDA,val);
multiplyCR(XI,GAMMA,val2);
add(val,val2,val);
multiplyCR(val,R1,C4);

for K := 1 to (((M-1) div 2) + 1) do
    begin
         MK := M + 1 - K;
         conjugate(C[K],SAVE1);
         conjugate(D[K],SAVE2);
         conjugate(C[MK],SAVE3);
         conjugate(D[MK],SAVE4);

         multiply(C1,SAVE3,val);
         multiply(C2,SAVE4,val2);
         add(val,val2,val);
         add(C[K],val,C[K]);

         multiply(C3,SAVE3,val);
         multiply(C4,SAVE4,val2);
         add(val,val2,val);
         add(D[K],val,D[K]);

         if not(K = MK) then
            begin
                 multiply(C1,SAVE1,val);
                 multiply(C2,SAVE2,val2);
                 add(val,val2,val);
                 add(C[MK],val,C[MK]);

                 multiply(C3,SAVE1,val);
                 multiply(C4,SAVE2,val2);
                 add(val,val2,val);
                 add(D[MK],val,D[MK]);
            end;
    end;

squaremag(PSI,val);
R2 := val.Re;

squaremag(THETA,val);
R3 := val.Re;

squaremag(XI,val);
R4 := val.Re;

conjugate(THETA,val);
multiply(LAMBDA,val,val);
multiply(PSI,val,val);
R5 := GAMMA - (R2*DELTA + R3*GAMMA + 2.0*val.Re) * R1;

conjugate(XI,val);
multiply(LAMBDA,val,val);
multiply(THETA,val,val);
R2 := DELTA - (R3*DELTA + R4*GAMMA + 2.0*val.Re) * R1;
```

```
GAMMA := R5;
DELTA := R2;

conjugate(PSI,val);
multiply(C3,val,val);
conjugate(THETA,val2);
multiply(C4,val2,val2);
add(val,val2,val);
add(LAMBDA,val,LAMBDA);

if not(P > 0.0) then
    begin
         ISTAT := 1;
         exit;
    end;

if not((DELTA > 0.0) and (DELTA <= 1.0) and (GAMMA > 0.0) and
        (GAMMA <= 1.0)) then
        begin
             ISTAT := 2;
             exit;
        end;

  R1 := 1.0/P;
  squaremag(LAMBDA,val);
  R2 := 1.0 / (DELTA * GAMMA - val.Re);
  EF := X[M+1];
  EB := X[N-M];
  for K := 1 to M do
      begin
           multiply(A[K],X[M+1-K],val);
           add(EF,val,EF);

           conjugate(A[K],val);
           multiply(val,X[N-M+K],val);
           add(EB,val,EB);
      end;

  multiplyCR(EB,R1,C1);

  conjugate(EF,val);
  multiplyCR(val,R1,C2);

  conjugate(EB,val);
  multiplyCR(val,DELTA,val);
  multiply(EF,LAMBDA,val2);
  add(val,val2,val);
  multiplyCR(val,R2,C3);

  multiply(EB,LAMBDA,val);
  conjugate(val,val);
  multiplyCR(EF,GAMMA,val2);
  add(val,val2,val);
  multiplyCR(val,R2,C4);

  for K := M downto 1 do
      begin
           SAVE1 := A[K];
           multiply(C3,C[K],val);
           multiply(C4,D[K],val2);
           add(val,val2,val);
```

```
                add(SAVE1,val,A[K]);

                multiply(C1,SAVE1,val);
                add(C[K],val,C[K+1]);

                multiply(C2,SAVE1,val);
                add(D[K],val,D[K+1]);
            end;

        C[1] := C1;
        D[1] := C2;
        squaremag(EB,val);
        R3 := val.Re;

        squaremag(EF,val);
        R4 := val.Re;

        multiply(EB,LAMBDA,val);
        multiply(EF,val,val);
        P := P - (R3*DELTA + R4*GAMMA + 2.0*val.Re) * R2;
        DELTA := DELTA - R4 * R1;
        GAMMA := GAMMA - R3 * R1;
        multiply(EF,EB,val);
        conjugate(val,val);
        multiplyCR(val,R1,val);
        add(LAMBDA,val,LAMBDA);
        if not(P > 0.0) then
           begin
                ISTAT := 3;
                exit;
           end;
        if not((DELTA > 0.0) and (DELTA <= 1.0) and (GAMMA > 0.0) and
               (GAMMA <= 1.0)) then
           begin
                ISTAT := 4;
                exit;
           end;
      until false;

end;

begin;
  clrscr;
  repeat
        write('Enter name for input file (eg. A:MEASURE.DAT) : ');
        readln(in_name);
        if in_name = '' then
           in_name1 := 'c:\alonbas\adc1.dat'
        else
            in_name1 := 'a:\new2\adc' + in_name + '.dat';
        writeln(in_name1);
        assign(infile,in_name1);
        reset(infile);
        IOcode:=IOresult;
        if IOcode<>0 then writeln('Problem! Try again.');
  until IOcode=0;
  writeln;
  I1 := 1;
  repeat
        begin
             readln(infile,X[I1].Re);
             X[I1].Im := 0;
```

```
            writeln(X[I1].Re,'  ',X[I1].Im);
            inc(I1);
        end;
  until eof(infile);
  close(infile);
  N := pred(I1);
  writeln(N,'  coefficients');
  writeln;
  writeln('Enter order of AR process');
  readln(IP);
  writeln(IP);
  gettime(th1,tm1,ts1,ths1);
  MCOV(N,IP,X,P,A,ISTAT);
  gettime(th2,tm2,ts2,ths2);
  writeln(ths2-ths1,' ',ts2-ts1);

  case ISTAT of
   0:begin
       writeln(ISTAT,' : Success');
       repeat
           write('Enter name for output file (eg. A:AR_COEF.DAT) : ');
           readln(out_name);
           if out_name = '' then
              out_name := 'c:\alonbas\ar1.dat'
           else
               out_name := 'a:\modcov2\' + in_name + '.dat';
           writeln(out_name);
           assign(outfile,out_name);
           rewrite(outfile);
           IOcode:=IOresult;
           if IOcode<>0 then writeln('Problem! Try again.');
       until IOcode=0;
       writeln;
       for I1 := 1 to IP do
           begin
                writeln(outfile,A[I1].Re:25:15,' ',A[I1].Im:25:15);
                writeln(A[I1].Re:25:15,' ',A[I1].Im:25:15);
           end;
       close(outfile);
       writeln;
       writeln('sig2 = ',P);
       writeln;
       writeln(-10 * ln(P) / 2.3:5:2,' dB');
     end;
    1:writeln(ISTAT,': Ill conditioned data');
    2:writeln(ISTAT,': Ill conditioned data');
    3:writeln(ISTAT,': Ill conditioned data');
    4:writeln(ISTAT,': Ill conditioned data');
  end;
  Ch := readkey;
end.
```

```
      { General purpose complex number routines     INCLUDE FILE }

procedure conjugate(A:complex; var C:complex);
begin
  C.Im := -A.Im;
  C.Re :=  A.Re;
end;

procedure multiply(A,B:complex;var C:complex);
begin
  C.Re := A.Re*B.Re - A.Im*B.Im;
  C.Im := A.Re*B.Im + A.Im*B.Re;
end;

procedure multiplyCR(A:complex;B:real;var C:complex);
begin
  C.Re := A.Re*B;
  C.Im := A.Im*B;
end;

procedure add(A,B:complex;var C:complex);
begin
  C.Re := A.Re + B.Re;
  C.Im := A.Im + B.Im;
end;

procedure subtract(A,B:complex;var C:complex);
begin
  C.Re := A.Re - B.Re;
  C.Im := A.Im - B.Im;
end;

procedure squaremag(A:complex; var C:complex);
begin
  C.Re := sqr(A.Re) + sqr(A.Im);
  C.Im := 0;
end;

procedure divide(A,B:complex; var C:complex);
var D:float;
begin
  D    := sqr(B.Re) + sqr(B.Im);
  C.Re := (A.Re*B.Re + A.Im*B.Im)/D;
  C.Im := (A.Im*B.Re - A.Re*B.Im)/D;
end;

procedure divideCR(A:complex; B:real; var C:complex);
begin
  C.Re := A.Re/B;
  C.Im := A.Im/B;
end;
```

APPENDIX L

LISTING OF FMCW RADAR SIMULATOR PROGRAM (AND WHITE NOISE)

```
program fmcw_gen;

{ Program by A. Bas. }

{ This program generates two complex sinusoids in white gaussian noise.
  The sinusoids are generated via the FMCW equation. The noise power can
  be set up in the program. If real numbers are required then the imaginary
  parts of the output can be set to zero. }

{$N+}

uses dos, crt;

type
    float = double;

    comp = record
              Re,Im : float;
           end;

const   noise_pow = 1e-4;

var
  outfile : text;
  in_name : string;
  pi : float;
  IOcode : word;
  i,N : integer;
  Tmod, fmod, df, wc, c, R1, R2, fb1, fb2, t, TXI, TXQ, RX1, RX2 : float;
  TWOFI1, TWOFI2, TWOFQ1, TWOFQ2 : float;
  TXR : array[0..1024] of comp;

function noise : float;

var sum : float;
    a : integer;

begin
     sum := 0;
     for a := 0 to 12 do
         sum := sum + noise_pow * 2 * (((random(32767) / 32767) - 0.5 ));
     {writeln(sum);}
     noise := sum;
end;

begin
      clrscr;
      pi := 3.141592654;
      N := 70;
      fmod := 1800;
      Tmod := 1/(fmod);
      df := 0.5 * 1e9;
      wc := 10e9;
      c := 3.0 * 1e8;
      writeln('Enter Range 1 , Range 2');
      readln(R1, R2);
      fb1 := 4 * df * R1 / c * fmod;
      fb2 := 4 * df * R2 / c * fmod;
      writeln;
      writeln(fb1:10:2,' Hz   ',fb2:10:2,' Hz');
      writeln;
```

```
    for i := 0 to N-1 do
        begin
            t := i / (N - 1) * Tmod;
            TXI := cos(wc * t + 2 * pi * df * SQR(t) / Tmod);
            TXQ := sin(wc * t + 2 * pi * df * SQR(t) / Tmod);
            RX1 := cos(wc*(t - 2*R1/c) + 2*pi*df/Tmod*SQR(t-2*R1/c));
            RX2 := cos(wc*(t - 2*R2/c) + 2*pi*df/Tmod*SQR(t-2*R2/c));
            TWOFI1 := 0.5 * cos(wc * t + 2 * pi * df * SQR(t) / Tmod +
                      wc*(t - 2*R1/c) + 2*pi*df/Tmod*SQR(t-2*R1/c));
            TWOFI2 := 0.5 * cos(wc * t + 2 * pi * df * SQR(t) / Tmod +
                      wc*(t - 2*R2/c) + 2*pi*df/Tmod*SQR(t-2*R2/c));
            TWOFQ1 := 0.5 * sin(wc * t + 2 * pi * df * SQR(t) / Tmod +
                      wc*(t - 2*R1/c) + 2*pi*df/Tmod*SQR(t-2*R1/c));
            TWOFQ2 := 0.5 * sin(wc * t + 2 * pi * df * SQR(t) / Tmod +
                      wc*(t - 2*R2/c) + 2*pi*df/Tmod*SQR(t-2*R2/c));

            TXR[i].re := TXI * (RX1 + RX2) - TWOFI1 - TWOFI2 + noise;
            TXR[i].im := TXQ * (RX1 + RX2) - TWOFQ1 - TWOFQ2 + noise;
         end;

    write('Enter name for output file (eg. A:MEASURE.DAT) : ');
    readln(in_name);
    assign(outfile,in_name);
    rewrite(outfile);
    IOcode:=IOresult;
    if IOcode<>0 then writeln('Problem! Try again.');
    writeln;
    for i := 0 to N-1 do
        begin
            writeln(outfile,TXR[i].Re:14:10,TXR[i].Im:14:10);
            writeln(i,' )  ',TXR[i].re,'  ',TXR[i].im);
        end;
    close(outfile);
end.
```

APPENDIX M

LISTING OF DIRECT SINUSOID GENERATOR PROGRAM (AND WHITE NOISE)

```
program sine_gen;

{ Program written by A. Bas }

{ This program generates two complex sinusoids in white gaussian noise.
  The sinusoids are generated directly. The noise power can be set up
  in the program. If real numbers are required then the imaginary
  parts of the output can be set to zero. }

uses dos, crt;

type    complex = record
                  re, im : double;
        end;

const   noise_pow = 1e-4;

var chan : text;
    out_name : string;
    a, N : integer;
    dataf, dataf2 : complex;
    d1, d2 : text;
    noise1, noise2, f1, f2 : double;

function noise : double;

var sum : double;
    a : integer;

begin
     sum := 0;
     for a := 1 to 12 do
              sum := sum + noise_pow * 2 * (((random(32767) / 32767) - 0.5 ));
     {writeln(sum);}
     noise := sum;
end;

begin
     clrscr;
     writeln('Enter f1, f2');
     readln(f1, f2);
     writeln;
     write('Enter output file name : ');
     readln(out_name);
     assign(chan, out_name);
     rewrite(chan);
     N := 70;
     for a := 0 to N-1 do
         begin
              dataf.re := cos(a / (N - 1) * 2 * f1 * pi) + cos(a / (N - 1) *
                          2 * f2 * pi) + noise;
              dataf.im := sin(a / (N - 1) * 2 * f1 * pi) + sin(a / (N - 1) *
                          2 * f2 * pi) + noise;
              writeln(chan,dataf.re:14:10,dataf.im:14:10);
         end;
     close(chan);

     reset(chan);
     a := 0;
     while not eof(chan) do
             begin
```

```
                readln(chan,dataf.re,dataf.im);
                writeln(a,') ',dataf.re,' ',dataf.im);
                a := a + 1;
            end;

    close(chan);
end.
```

APPENDIX N

LISTING OF PROGRAM CWPRINT

```
program cwprint;

{ Program in Pascal by A. BAS. }

{ This program calculates and plots the spectrum from the AR coefficients. }

{$N+}

uses dos,crt,graph;

type
    comp = record
              Re,Im :double;
           end;

var
  infile : text;
  in_name : string;
  in_name1 : string;
  pi : double;
  IOcode : word;
  i, p, l : integer;
  A : array[0..50] of comp;
  scale, sumR, sumI, mag, thet, N : double;
  grdriver, grmode : integer;
  R1, R2 : integer;


procedure getdat;

begin
      write('Enter name for input file (eg. arcoef4.dat) : ');
      readln(in_name);
      if in_name = '' then
         begin
              in_name1 := 'c:\alonbas\ar1.dat';
              writeln(in_name);
         end
      else
         in_name1 := 'a:\perfectf\' + in_name + '.dat';
      writeln(in_name1);
      assign(infile, in_name1);
      reset(infile);
      IOcode := IOresult;
      if IOcode <> 0 then writeln('Problem! Try again.');
      writeln;
      p := 1;
      while not(eof(infile)) do
          begin
               readln(infile,A[p].Re,A[p].Im);
               writeln(p,' ',A[p].re,'  ',A[p].im);
               p := p + 1;
          end;
      close(infile);
end;

procedure axes;

var xaxis_pointer,yaxis_pointer :integer;

begin
     setcolor(9);
```

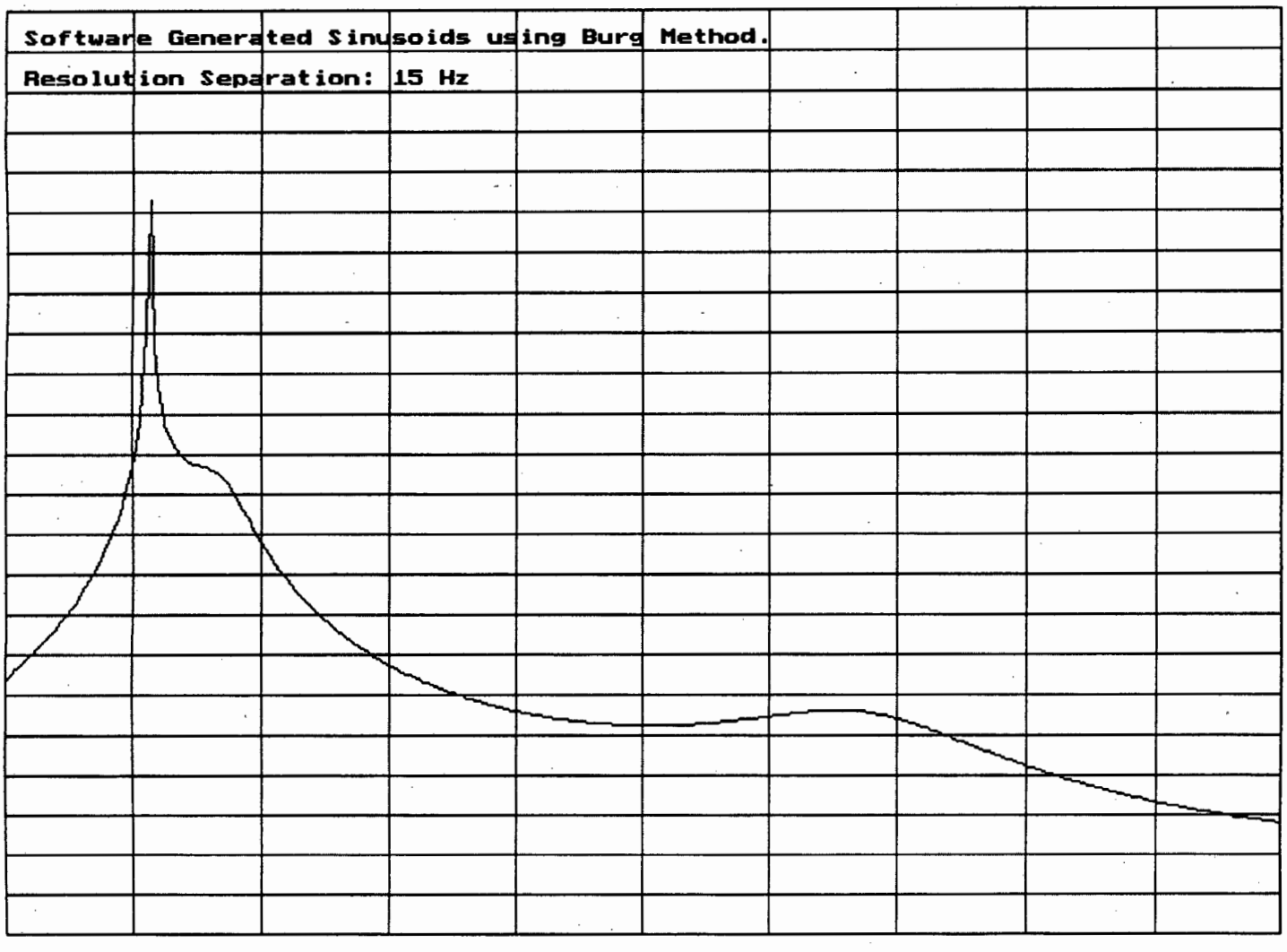
```
      moveto(639,0);
      lineto(0,0);
      lineto(0,460);
      lineto(639,460);
      lineto(639,0);
      moveto(0,0);

      xaxis_pointer := 0;
      repeat
            moveto(xaxis_pointer,0);
            linerel(0,460);
            xaxis_pointer := xaxis_pointer + 64;
      until xaxis_pointer > 639;

      yaxis_pointer := 0;
      moveto(0,0);
      repeat
            moveto(0,yaxis_pointer);
            linerel(639,0);
            yaxis_pointer := yaxis_pointer + 20;
      until yaxis_pointer > 460;

end;

procedure draw_spectrum;

var  graph_label : string;

begin
      pi := 3.141592654;
      N := 8192;
      R1 := 320;
      R2 := 960;
      scale := 1e5;
      writeln(N:5:0,'  ',R1,'  ',R2,'  ',scale:5:0);
      delay(500);
      grdriver := detect;
      initgraph(grdriver,grmode,'');
      setgraphmode(grmode);
      axes;
      graph_label := in_name + ' Hz';
      setcolor(15);
      outtextxy(10,10,graph_label);
      moveto(0,400);


      for i := R1 to R2 do
          begin
               sumR := 1;
               sumI := 0;
               thet := i / N * 2 * pi;
               for l := 1 to 10 do
                 begin
                   sumR := sumR + (A[l].Re * cos(thet*l) +
                                   A[l].Im * sin(thet*l));
                   sumI := sumI + (A[l].Im * cos(thet*l) -
                                   A[l].Re * sin(thet*l));
                 end;
               mag := SQR(sumR) + SQR(sumI);
               lineto(round((i - R1) / (R2 - R1) * 640),
                    200 - round(40.0 * Ln((1.0 / mag) / scale) / ln(10.0)));
          end;
```

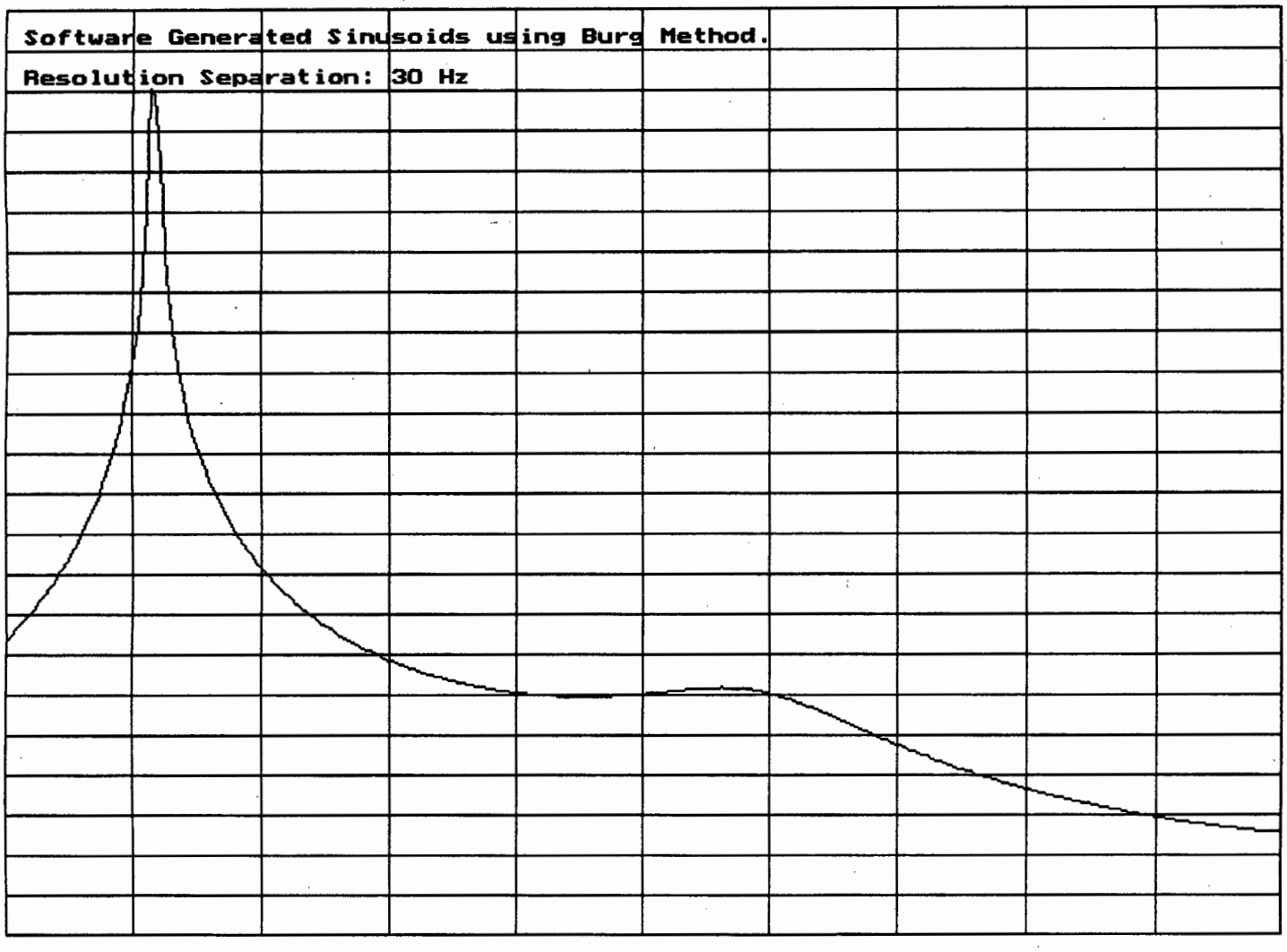
```
          sound(100);
          delay(100);
          nosound;

          repeat
          until keypressed;

          closegraph;
end;

begin
     clrscr;
     getdat;
     draw_spectrum;
end.
```

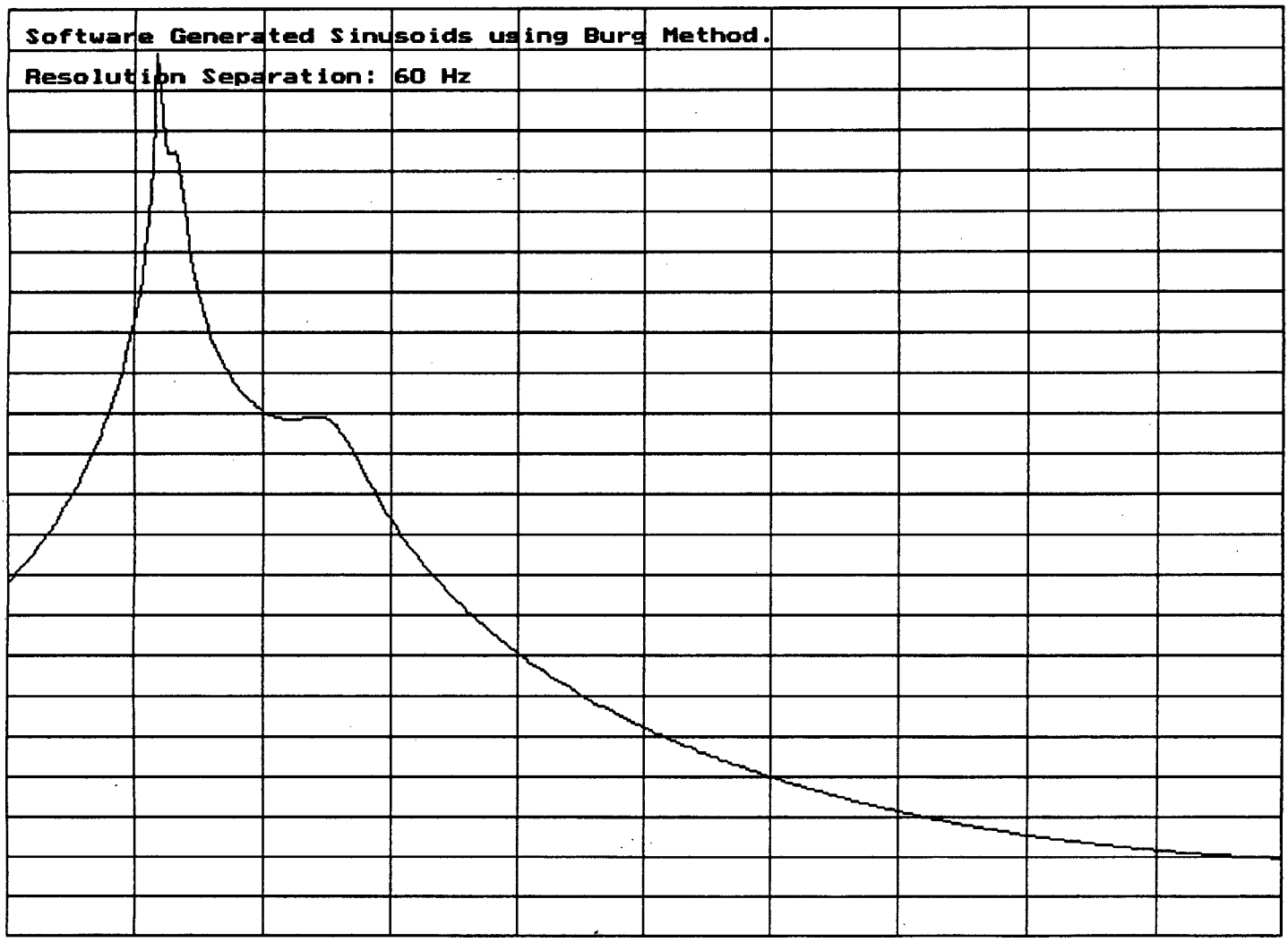
APPENDIX O

BURG AND MODIFIED COVARIANCE SPECTRAL PLOTS FOR VARIOUS RESOLUTION TESTS

BURG SPECTRAL PLOTS FOR VARIOUS RESOLUTION TESTS

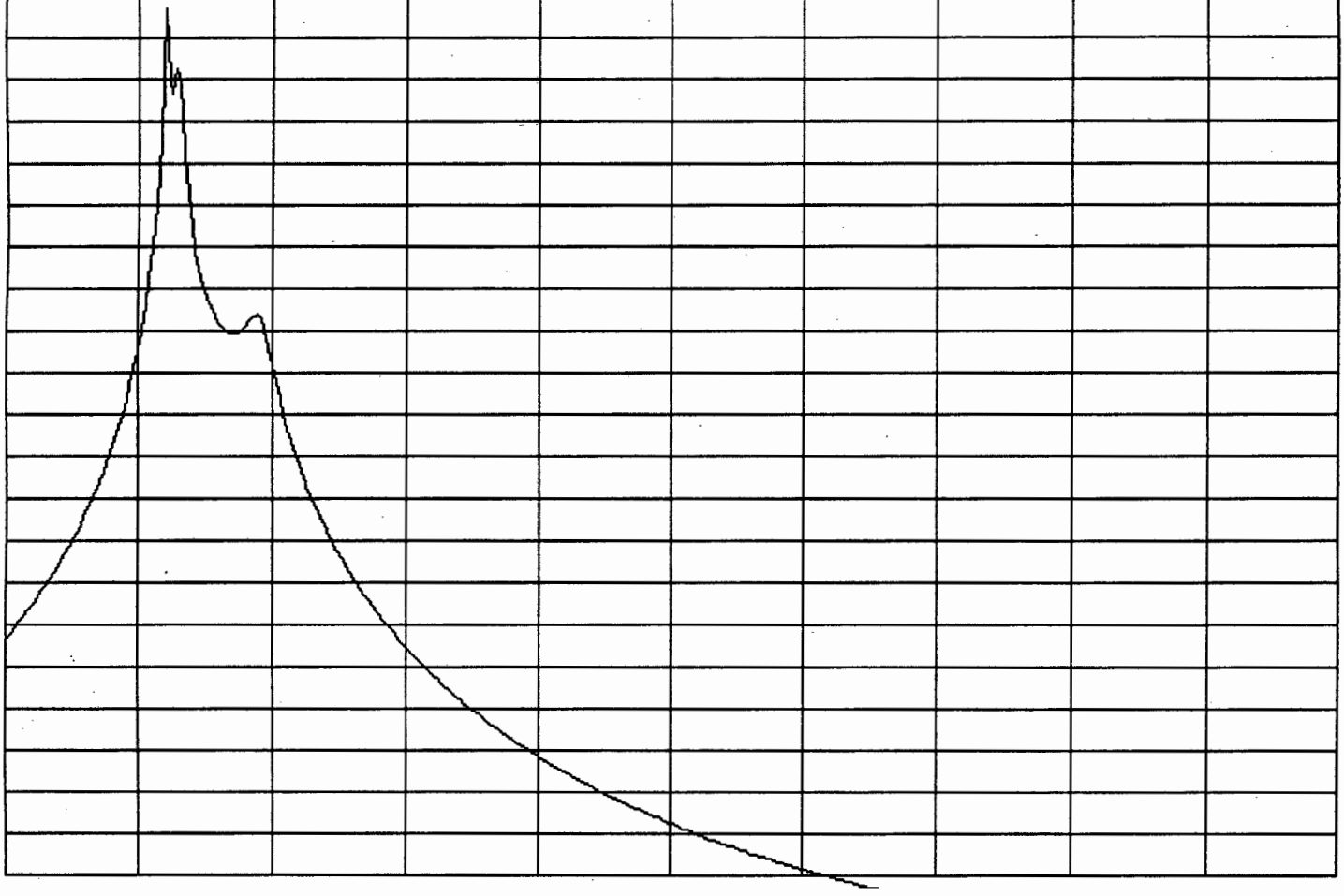

Software Generated Sinusoids using Burg Method.
Resolution Separation: 15 Hz

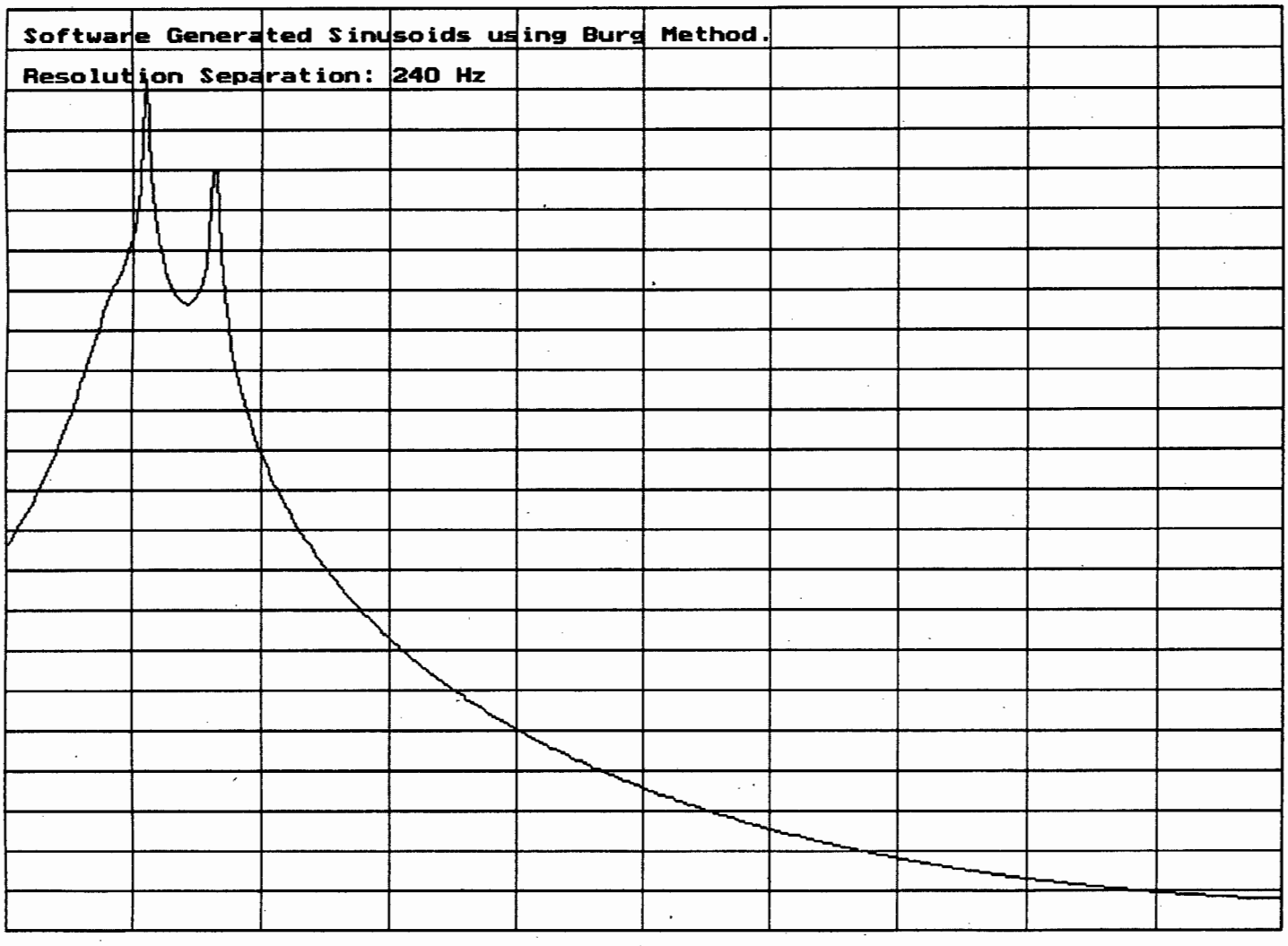

Software Generated Sinusoids using Burg Method.
Resolution Separation: 30 Hz

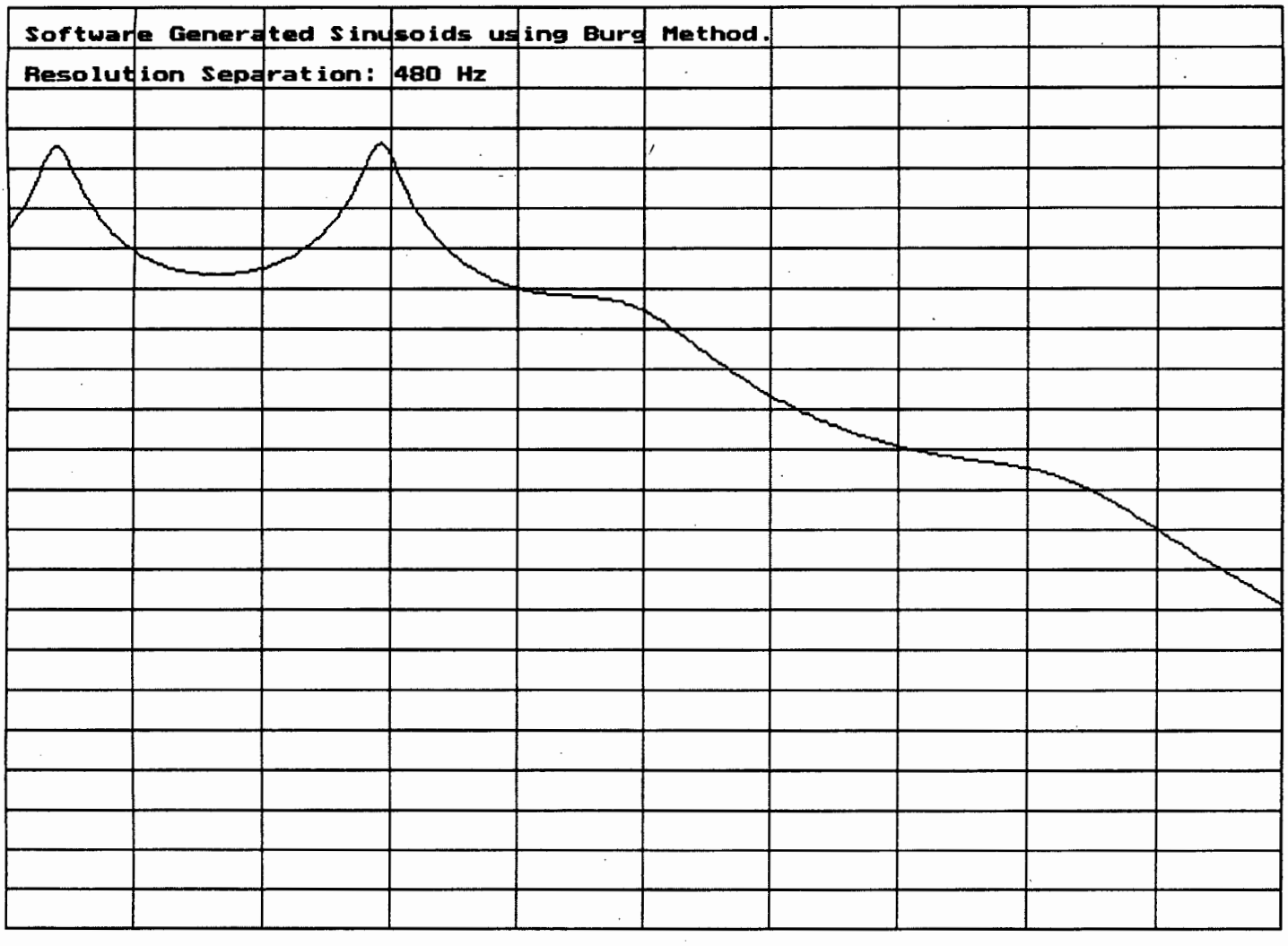

Software Generated Sinusoids using Burg Method.
Resolution Separation: 60 Hz

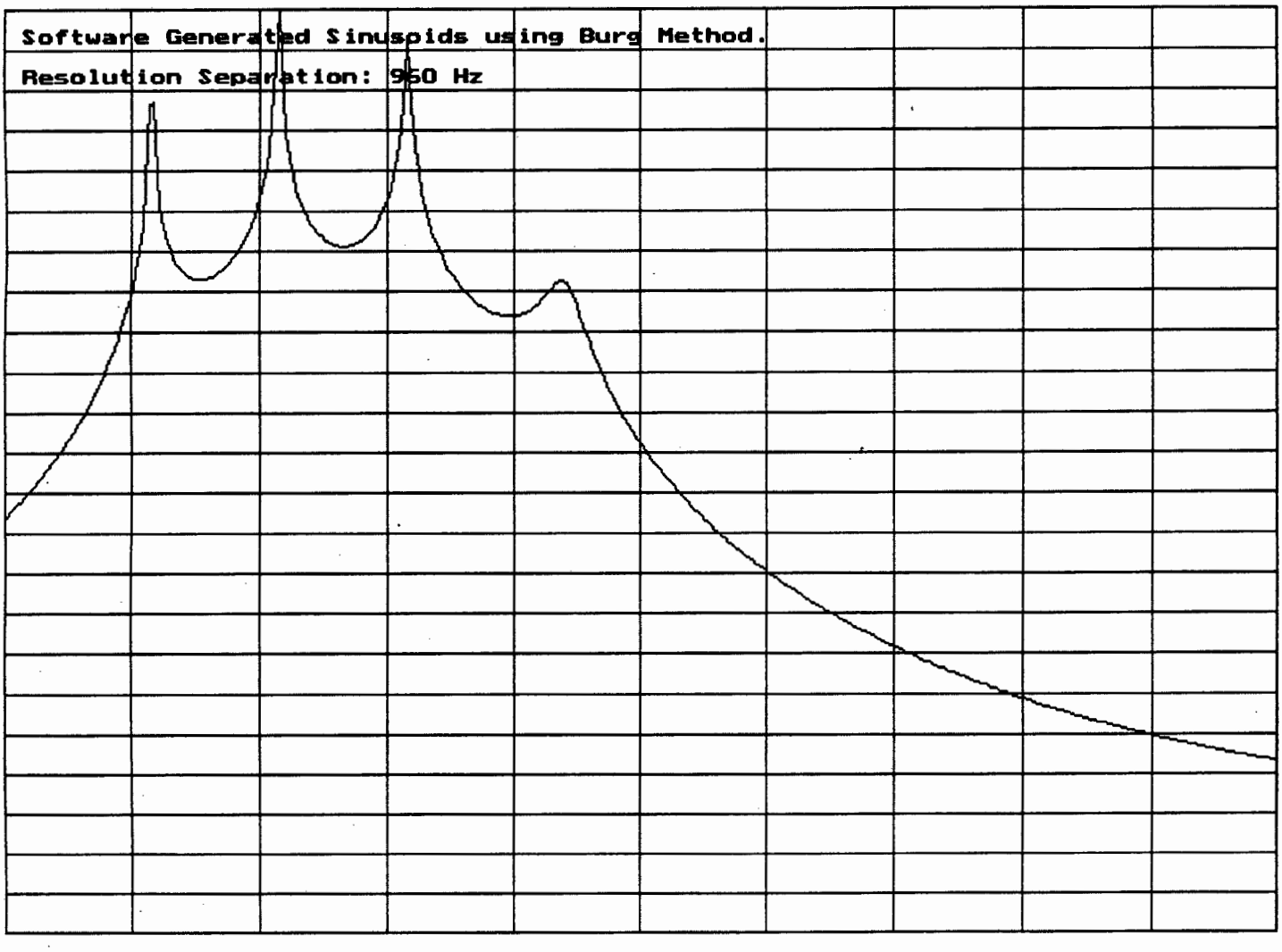
Software Generated Sinusoids using Burg Method.

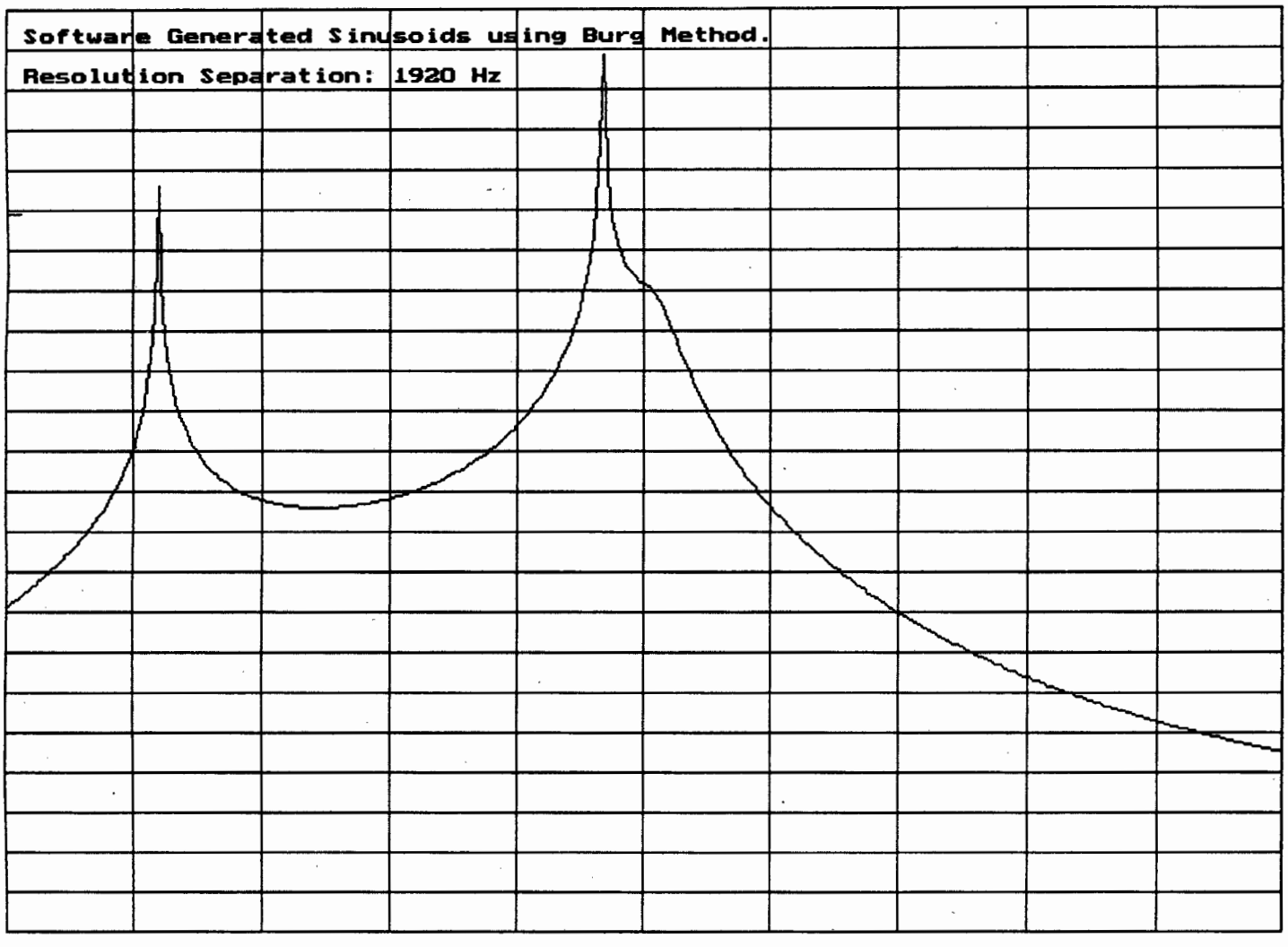
Resolution Separation: 120 Hz

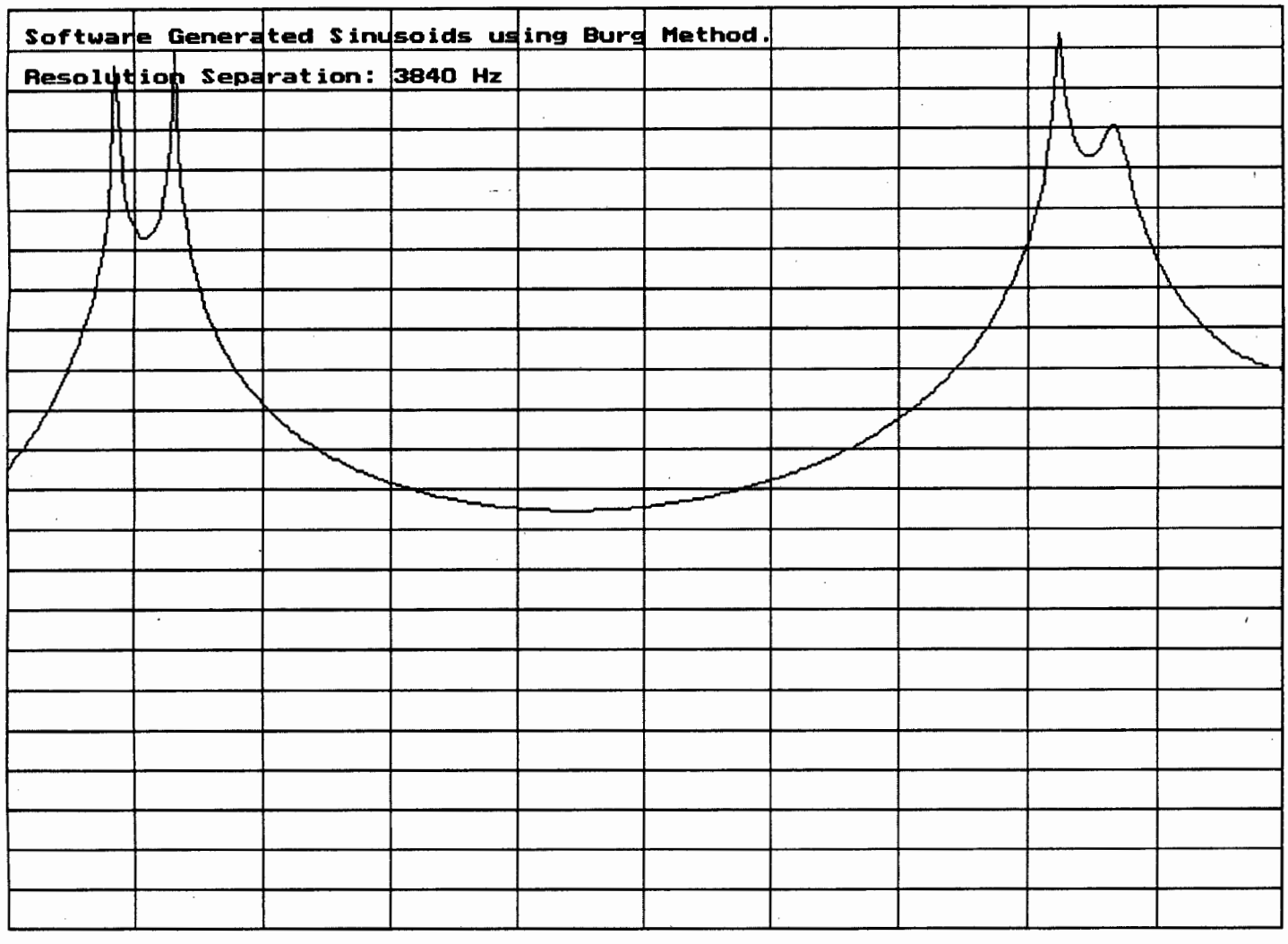



Software Generated Sinusoids using Burg Method.
Resolution Separation: 240 Hz

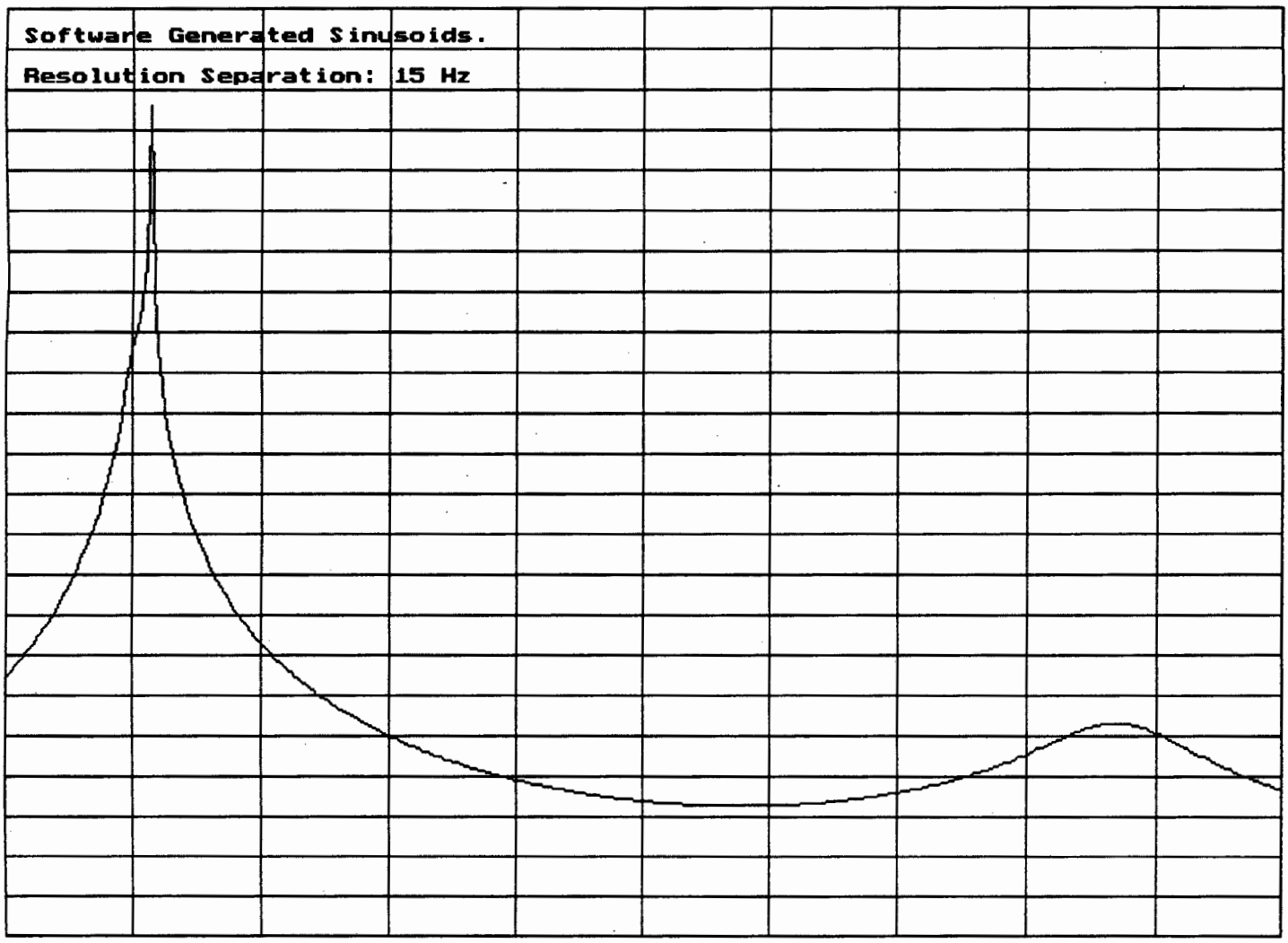

Software Generated Sinusoids using Burg Method.
Resolution Separation: 480 Hz

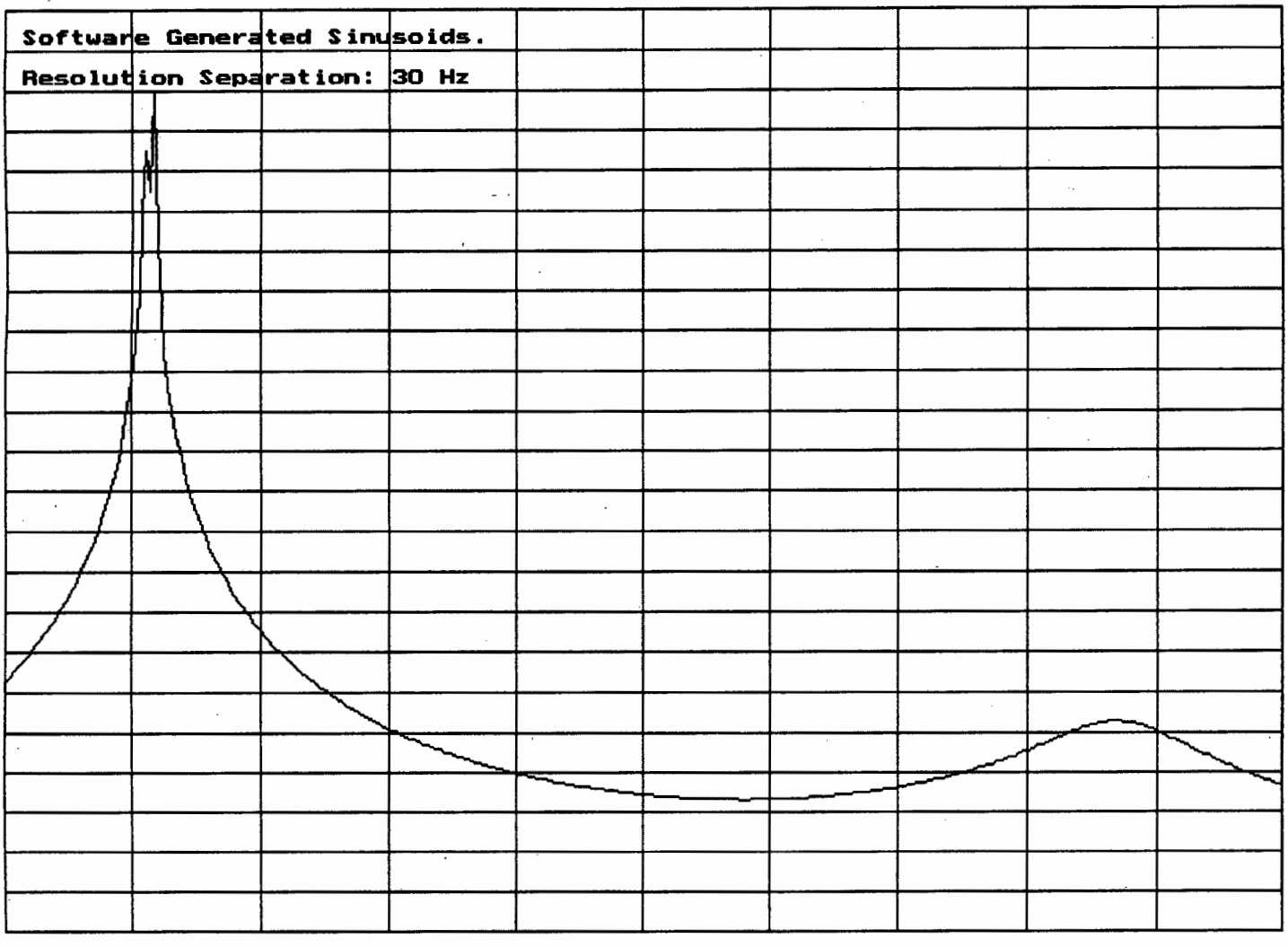

Software Generated Sinuspids using Burg Method.
Resolution Separation: 960 Hz

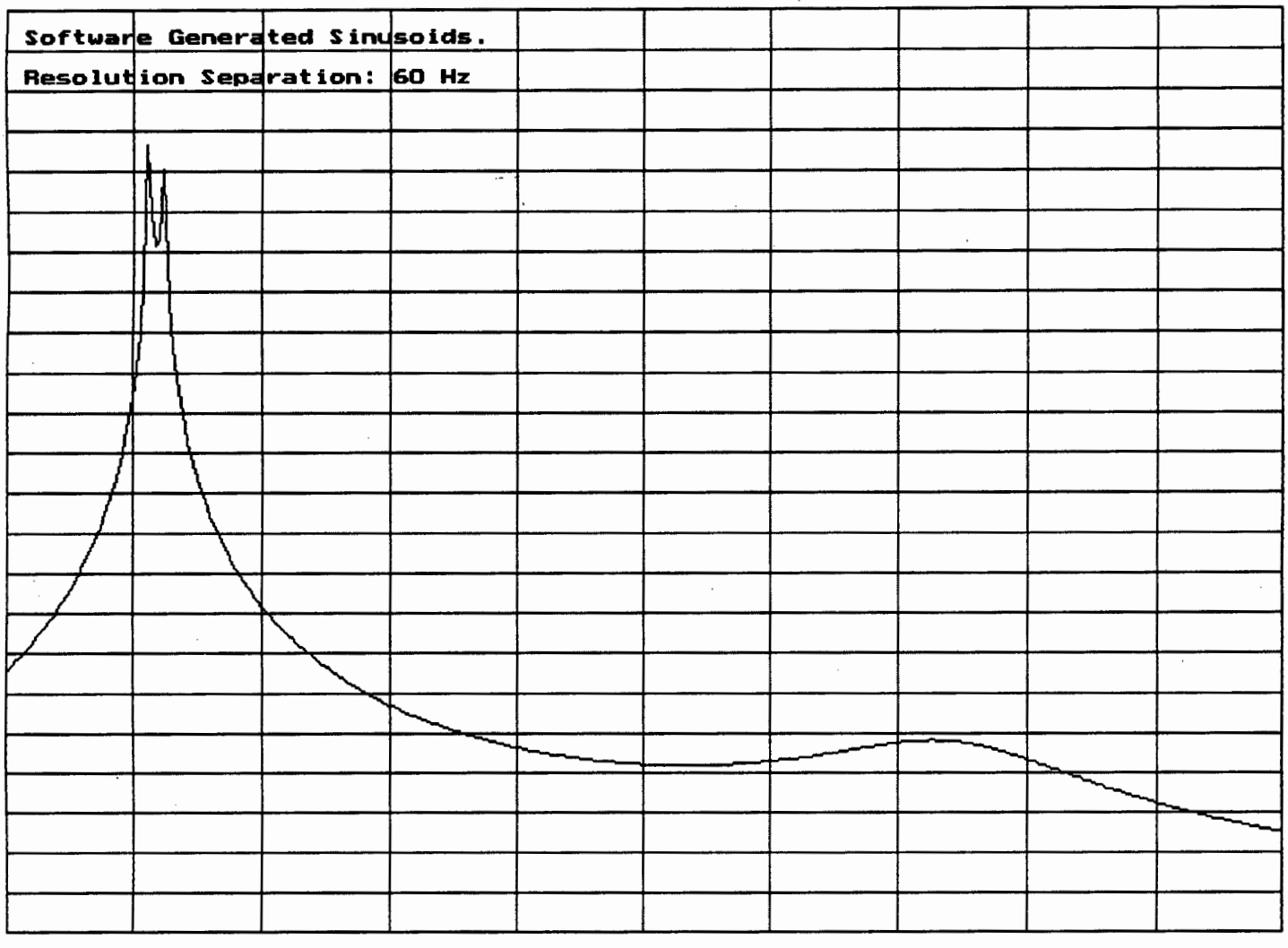

Software Generated Sinusoids using Burg Method.
Resolution Separation: 1920 Hz

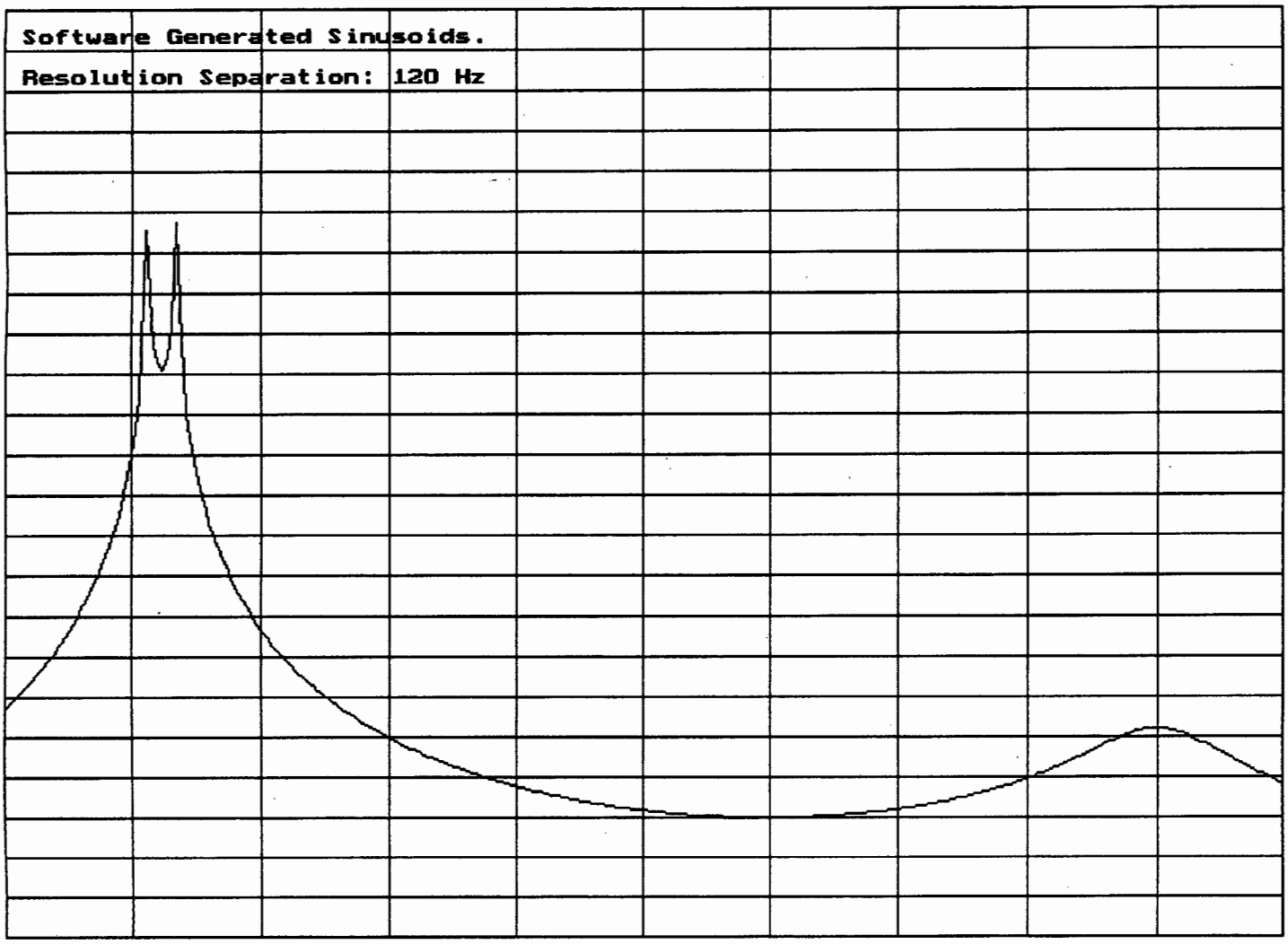

Software Generated Sinusoids using Burg Method.
Resolution Separation: 3840 Hz

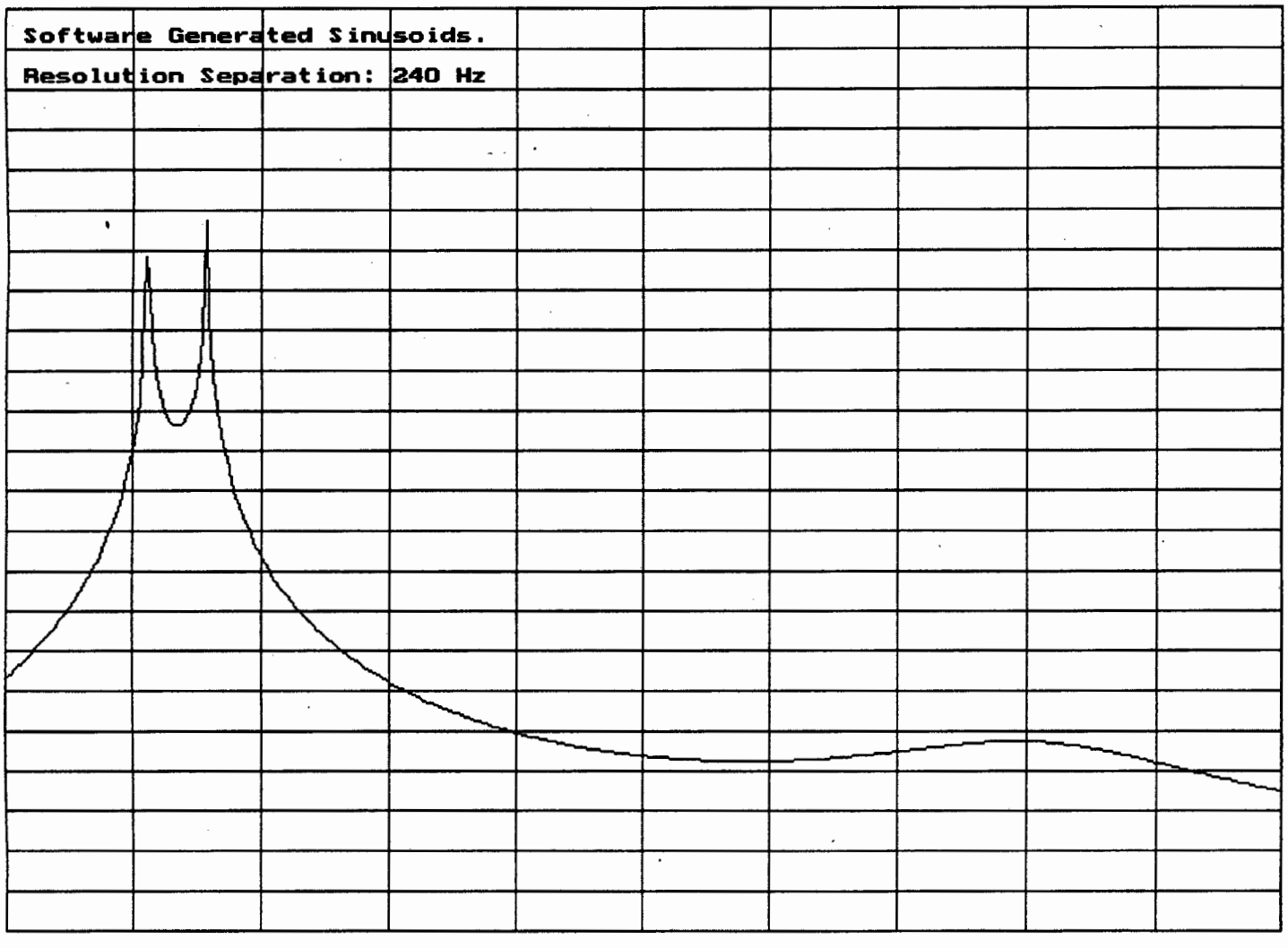
MODIFIED COVARIANCE SPECTRAL PLOTS FOR VARIOUS RESOLUTION TESTS

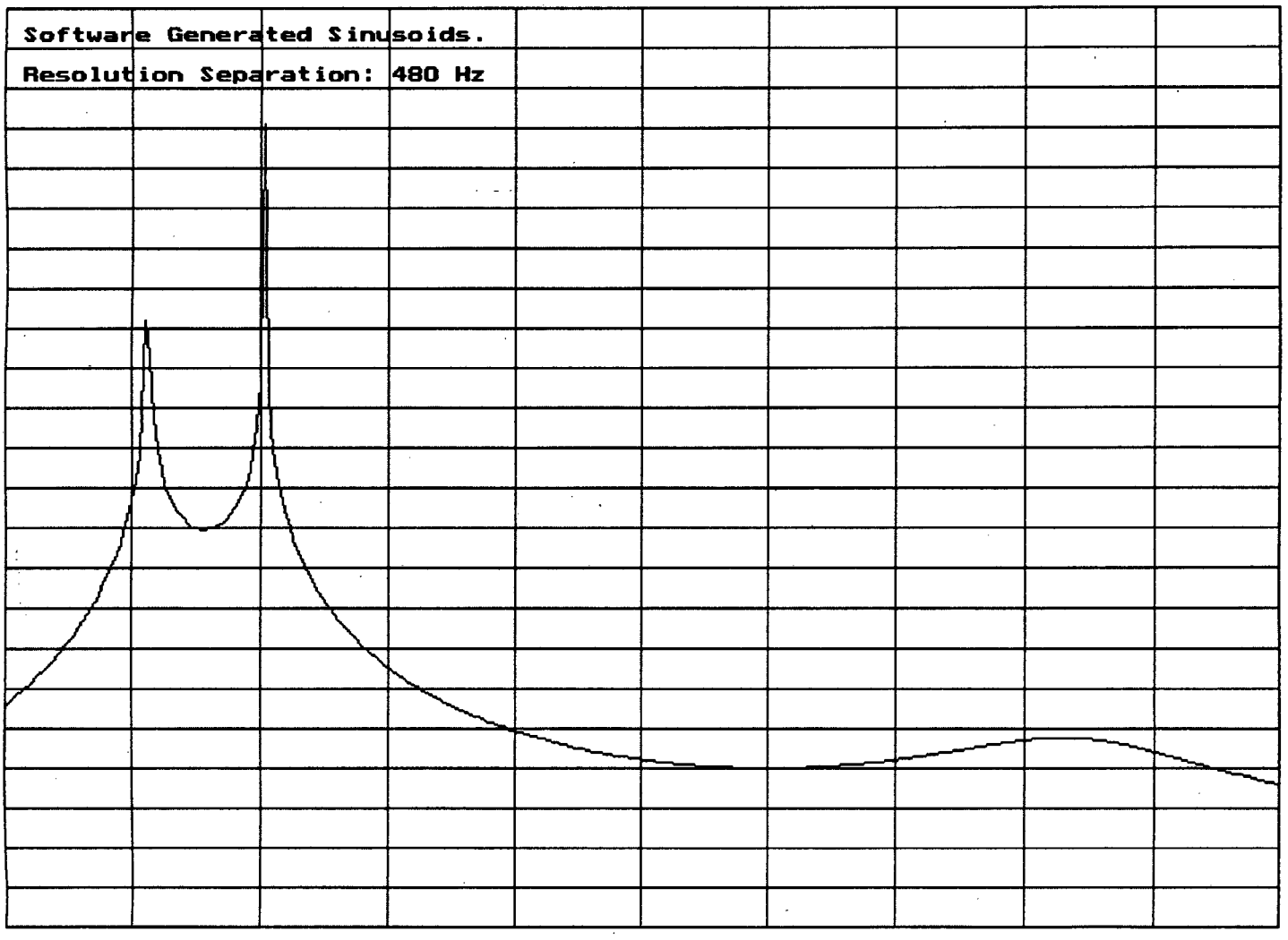

Software Generated Sinusoids.
Resolution Separation: 15 Hz

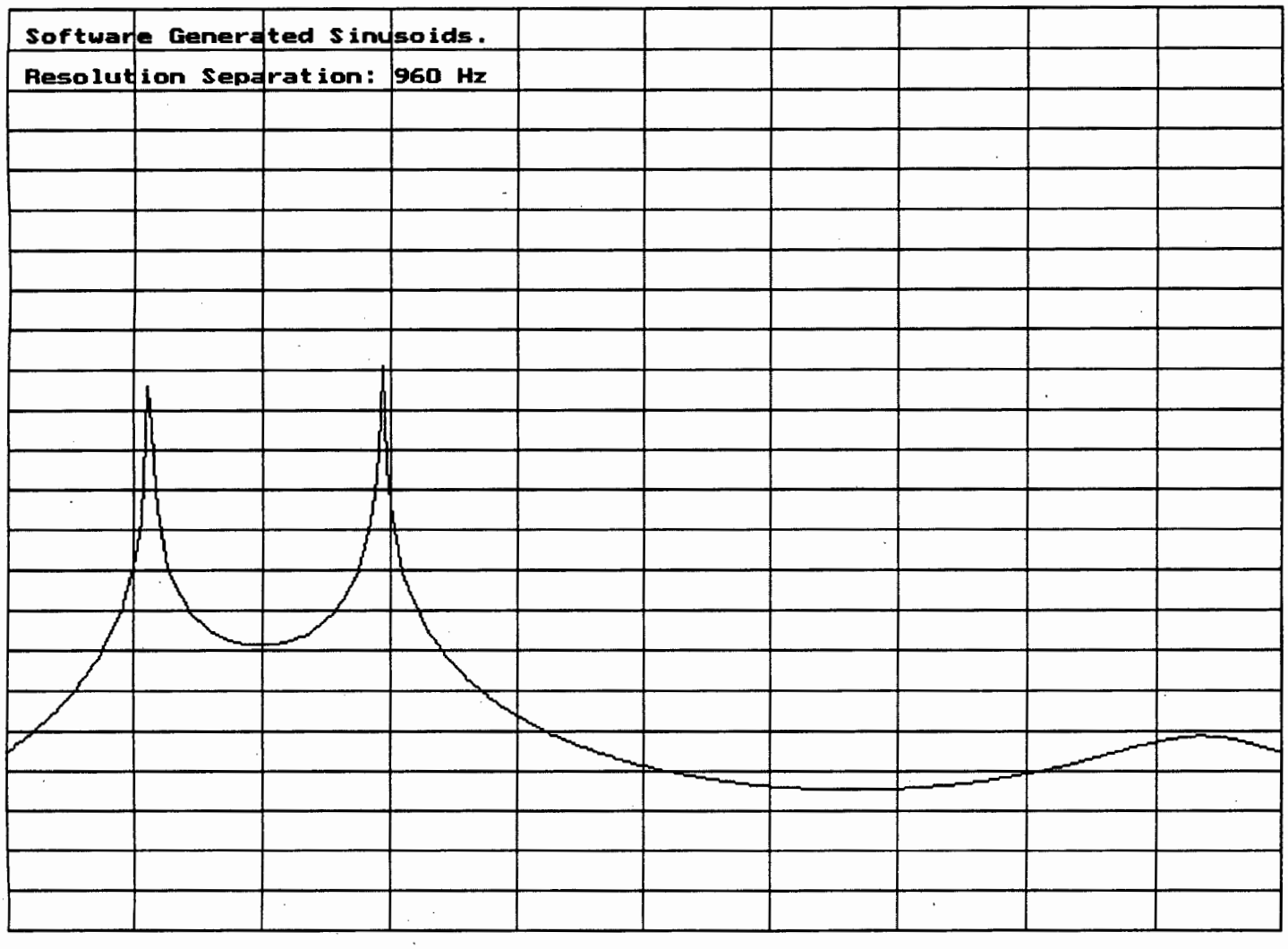

Software Generated Sinusoids.
Resolution Separation: 30 Hz

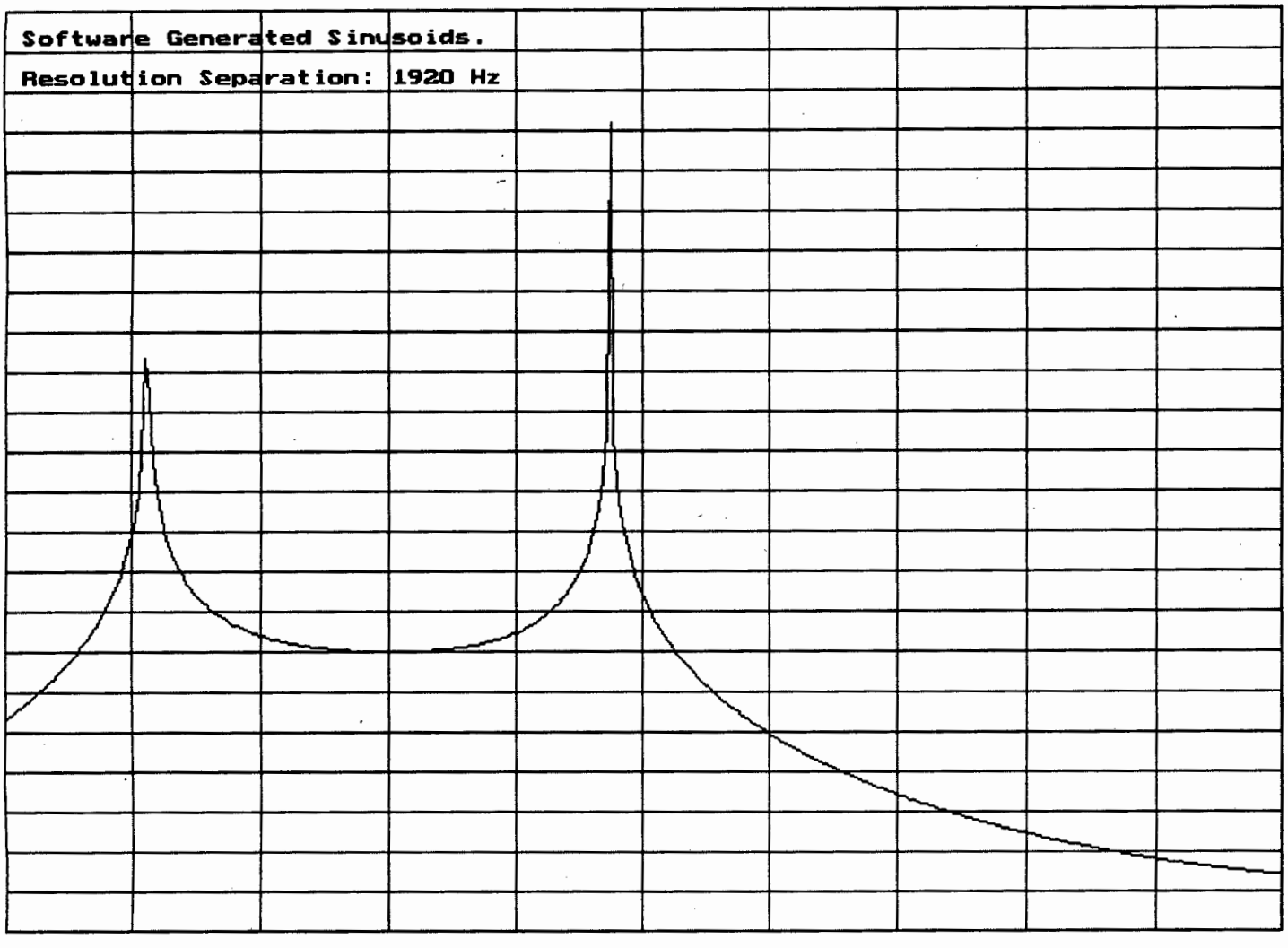

Software Generated Sinusoids.
Resolution Separation: 60 Hz

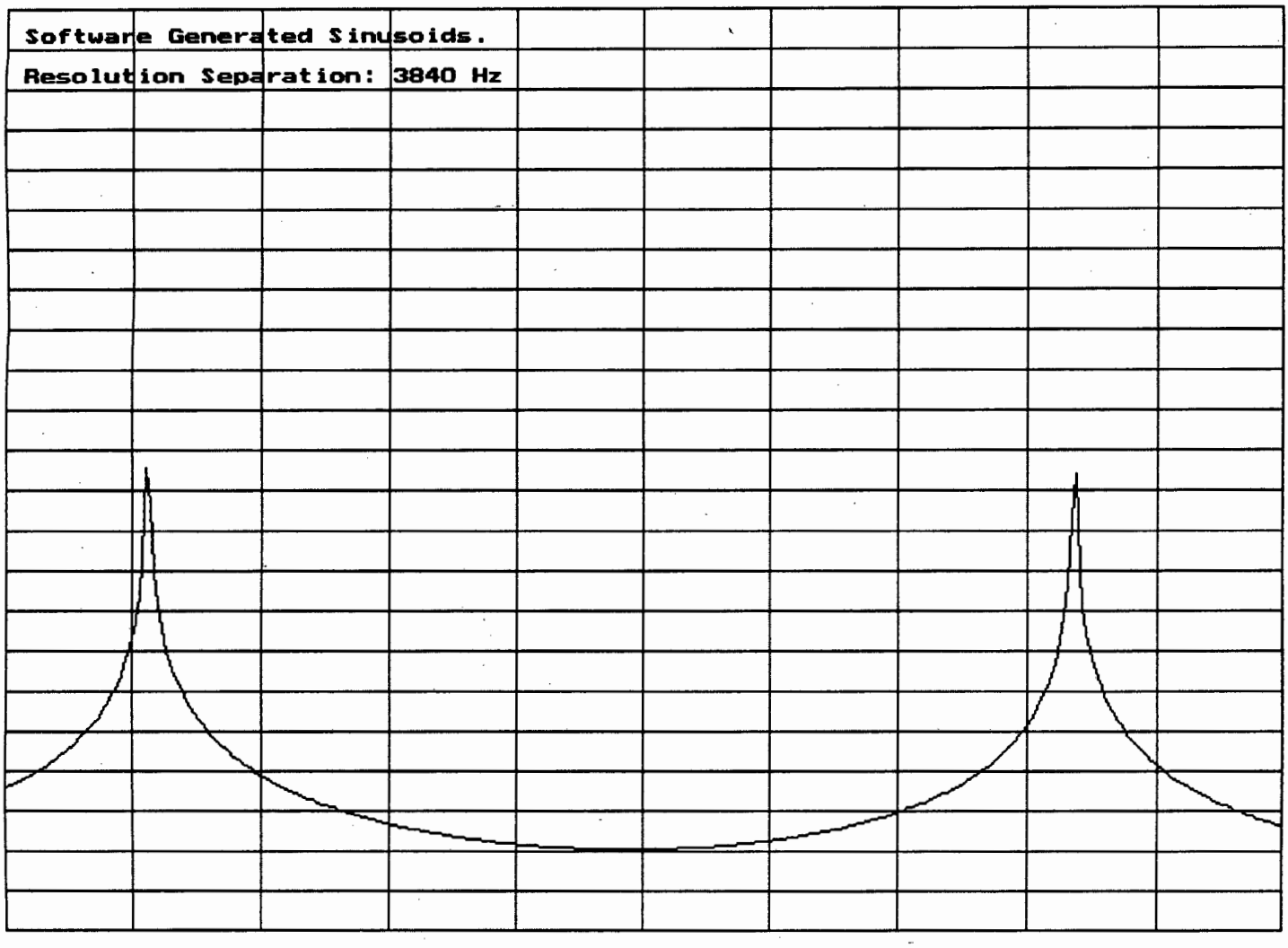

Software Generated Sinusoids.
Resolution Separation: 120 Hz


Software Generated Sinusoids.
Resolution Separation: 240 Hz


Software Generated Sinusoids.
Resolution Separation: 480 Hz


Software Generated Sinusoids.
Resolution Separation: 960 Hz


Software Generated Sinusoids.
Resolution Separation: 1920 Hz


Software Generated Sinusoids.
Resolution Separation: 3840 Hz

APPENDIX P

LISTING OF PROGRAM GET_DAT1

```
program get_dat;

{ Program written in Pascal by A. Bas }

{ This program reads the values into the PC via a 12 bit PC30D A/D card.
  DMA transfer is used for high speed. The data is plotted on the screen,
  and the data is stored to disc. }

{$S+}    {Stack checking on}
{$I+}    {I/O checking on}
{$N+}    {numeric coprocessor}


Uses
    Crt, pc30,graph;

const
     base_add =   $700;
     num_samples = 70;
var

 a_d :  array[0..1000] of integer;
 quit_f : boolean;
 i, test : integer;
 Ch : char;
 out_file : text;
 f_name : string;
 sample : real;

procedure ver_chk;

var

  r:   real;
  i:   integer;

begin

   i := version;

   r := hi(i) + lo(i)/100;
   clrscr;
   GotoXY(25, 2);
   write('PC-30 Driver version ', r:4:2);
   quit_f := false;

   if diag <> 0 then begin
        writeln;
        writeln('PC-30 dignostics report A/D not installed or other fault.');
        quit_f := true;
   end;

end;

procedure set_up;

var
   Test : integer;

begin
     Init;
```

```
      Clean;
      Ad_clock(1);
      Ad_prescaler(5);
      Dma_init;
      Test := Sd_chan(0,num_samples,5,a_d[0]);
      writeln;
      writeln('TEST1 = ',test);

end;


procedure axes;

var xaxis_pointer,yaxis_pointer :integer;

begin
     moveto(639,0);
     lineto(0,0);
     lineto(0,400);
     lineto(639,400);
     lineto(639,0);
     moveto(0,0);

     xaxis_pointer := 0;
     repeat
          moveto(xaxis_pointer,0);
          linerel(0,400);
          xaxis_pointer := xaxis_pointer + 64;
     until xaxis_pointer > 639;

     yaxis_pointer := 0;
     moveto(0,0);
     repeat
          moveto(0,yaxis_pointer);
          linerel(639,0);
          yaxis_pointer := yaxis_pointer + 20;
     until yaxis_pointer > 400;

end;

procedure timeview;

var     i : integer;
        grdriver, grmode : integer;
        adval : real;

begin
      grdriver := detect;
      initgraph(grdriver,grmode,'');
      setgraphmode(grmode);
      setcolor(9);
      axes;
      setcolor(15);
      moveto(639,200);
      lineto(0,200);
      for i := 0 to (num_samples-1) do
          begin
               setcolor(14 + i mod 2);
               adval := ((a_d[i] and 4095) - 2047) / 2048 * 5;
               lineto(round(i * (640 / num_samples)),200 - round(200 * adval / 5))
          end;
          sound(100);
```

```
          delay(100);
          nosound;

          repeat
          until keypressed;
end;

begin
   set_base(base_add);
   ver_chk;
   if quit_f then halt;
   repeat
   clrscr;
   write('Enter file name : ');
   readln(f_name);
   if f_name = '' then f_name := 'c:\alonbas\adc1.dat'
      else
          f_name := 'a:\new2\' + f_name + '.dat';
   writeln(f_name);
   assign(out_file,f_name);
   rewrite(out_file);
   test := 100;
   rtc_off;

   Port[base_add + 11] := 144;
   Port[Base_add + 9] := 1;  { disable a/d }
   set_up;
   repeat
   until (Port[Base_add + 8] and 1) = 0;  {wait for 1 - enable count}
   repeat
   until (Port[Base_add + 8] and 1) = 1;  {wait for 0 - no count}
   Port[Base_add + 9] := 0;  { enable a/d }
   while (test <> 0) do
         begin
              test := Dma_chk;
              writeln(test,'    ',ok_30,' ',par_30,'  ',n_comp_30);
         end;
   for i := 0 to (num_samples - 1) do
       begin
            sample := ((a_d[i] and 4095) - 2047) / 2048 * 5;
            writeln(out_file,sample);
            writeln(i,')  ',sample);
       end;
   Test := Dma_close;
   writeln('TEST2 = ',test);
   close(out_file);
   rtc_on;
   timeview;
   closegraph;
   until false;
end.
```

APPENDIX Q

LISTING OF THE NEW VERSION OF THE HARDWARE SAMPLING PROGRAM (GET_DAT2)

```
program get_dat2;

{ Program written by A. Bas. }

{ This program reads in the analogue values via a DMA transfer from the
  PC30D 12 bit ADC. The program then plots the values and saves the data
  to a file. White noise is added in software ie. pre-whitening.}

{$S+}    {Stack checking on}
{$I+}    {I/O checking on}
{$N+}    {numeric coprocessor}

Uses
    Crt, pc30, graph;

const
     base_add =   $700;
     num_samples = 70;
     noise_pow = 5e-4;         { noise power }

var

 a_d :  array[0..1000] of integer;
 quit_f : boolean;
 i, test : integer;
 Ch : char;
 out_file : text;
 f_name : string;
 sample : real;

procedure ver_chk;

var   r : real;
      i : integer;

begin
   i := version;
   r := hi(i) + lo(i)/100;
   clrscr;
   GotoXY(25, 2);
   write('PC-30 Driver version ',r:4:2);
   quit_f := false;

   if diag <> 0 then begin
        writeln;
        writeln('PC-30 diagnostics - A/D not installed or other fault.');
        quit_f := true;
   end;

end;

procedure set_up;

var  Test : integer;

begin
     Init;
     Clean;
     Ad_clock(1);
     Ad_prescaler(5);
     Dma_init;
```

```
      Test := Sd_chan(0,num_samples,5,a_d[0]);
      writeln;
      writeln('TEST1 = ',test);
end;

procedure axes;

var xaxis_pointer,yaxis_pointer :integer;

begin
      moveto(639,0);
      lineto(0,0);
      lineto(0,400);
      lineto(639,400);
      lineto(639,0);
      moveto(0,0);

      xaxis_pointer := 0;
      repeat
            moveto(xaxis_pointer,0);
            linerel(0,400);
            xaxis_pointer := xaxis_pointer + 64;
      until xaxis_pointer > 639;

      yaxis_pointer := 0;
      moveto(0,0);
      repeat
            moveto(0,yaxis_pointer);
            linerel(639,0);
            yaxis_pointer := yaxis_pointer + 20;
      until yaxis_pointer > 400;

end;

procedure timeview;

var      i : integer;
         grdriver, grmode : integer;
         adval : real;

begin
       grdriver := detect;
       initgraph(grdriver,grmode,'');
       setgraphmode(grmode);
       setcolor(9);
       axes;
       setcolor(15);
       moveto(639,200);
       lineto(0,200);
       for i := 0 to (num_samples-1) do
           begin
                setcolor(14 + i mod 2);
                adval := ((a_d[i] and 4095) - 2047) / 2048 * 5;
                lineto(round(i * (640 / num_samples)),200 - round(200 * adval / 5)
           end;
       sound(100);
       delay(100);
       nosound;

       repeat
       until keypressed;
end;
```

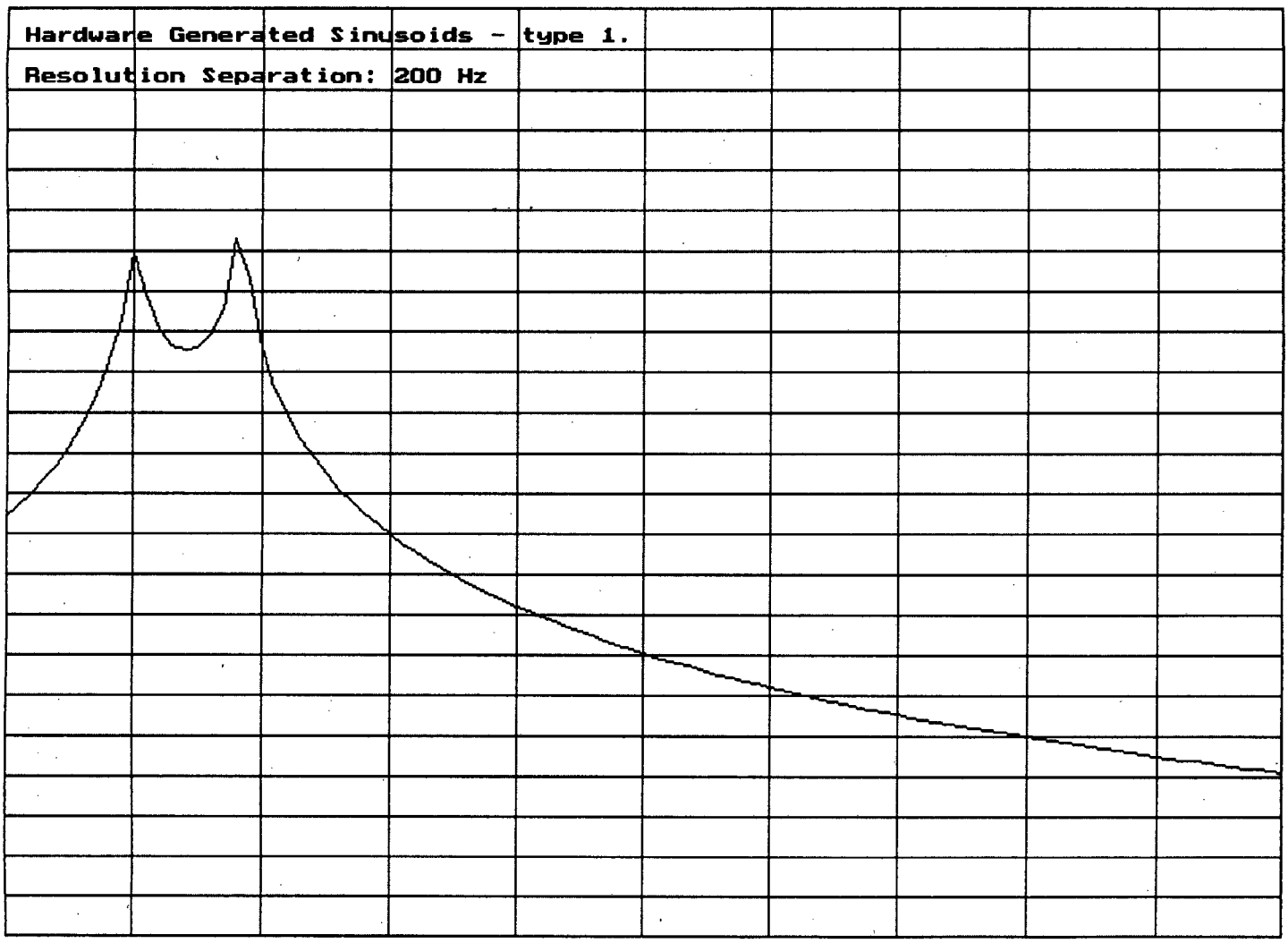
```
function noise : double;

var sum : double;
    a : integer;

begin
     sum := 0;
     for a := 1 to 12 do
              sum := sum + noise_pow * 2 * (((random(32767) / 32767) - 0.5 ));
     {writeln(sum);}
     noise := sum;
end;

begin
   set_base(base_add);
   ver_chk;
   if quit_f then halt;
   repeat
   clrscr;
   write('Enter file name : ');
   readln(f_name);
   if f_name = '' then
      f_name := 'c:\alonbas\adc1.dat'
   else
       f_name := 'a:\new2\' + f_name + '.dat';
   assign(out_file,f_name);
   rewrite(out_file);
   test := 100;
   rtc_off;

   Port[base_add + 11] := 144;
   Port[Base_add + 9] := 1;  { disable a/d }
   set_up;
   repeat
   until (Port[Base_add + 8] and 1) = 0;  {wait for 1 - enable count}
   repeat
   until (Port[Base_add + 8] and 1) = 1;  {wait for 0 - no count}
   Port[Base_add + 9] := 0;  { enable a/d }
   while (test <> 0) do
         begin
              test := Dma_chk;
              writeln(test,'    ',ok_30,' ',par_30,'  ',n_comp_30);
         end;
   for i := 0 to (num_samples - 1) do
       begin
            sample := ((a_d[i] and 4095) - 2047) / 2048 * 5;
            sample := sample + noise;
            writeln(out_file,sample);
            writeln(i,')  ',sample);
       end;
   Test := Dma_close;
   writeln('TEST2 = ',test);
   close(out_file);
   rtc_on;
   timeview;
   closegraph;
   until false;
end.
```

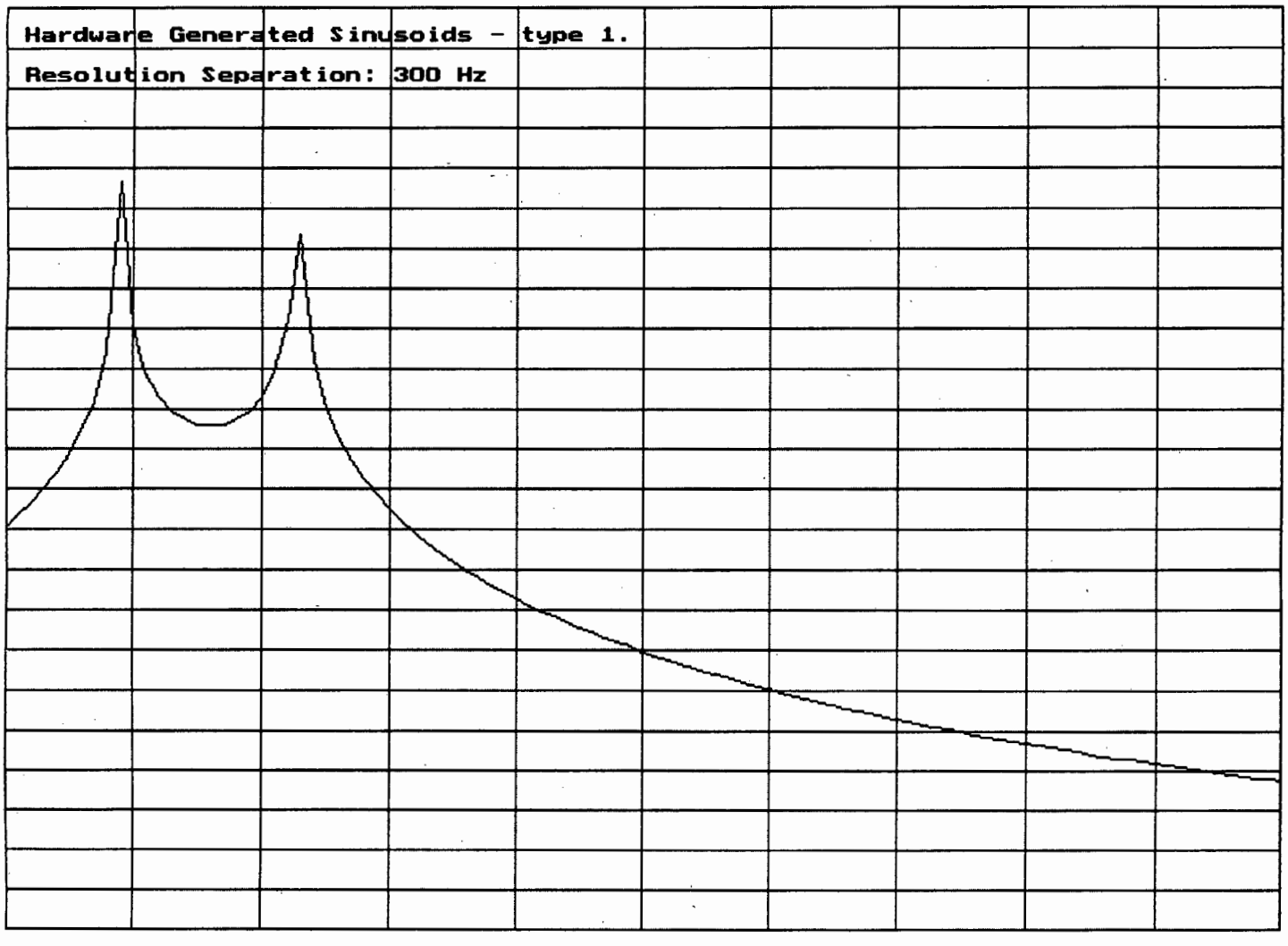
APPENDIX R

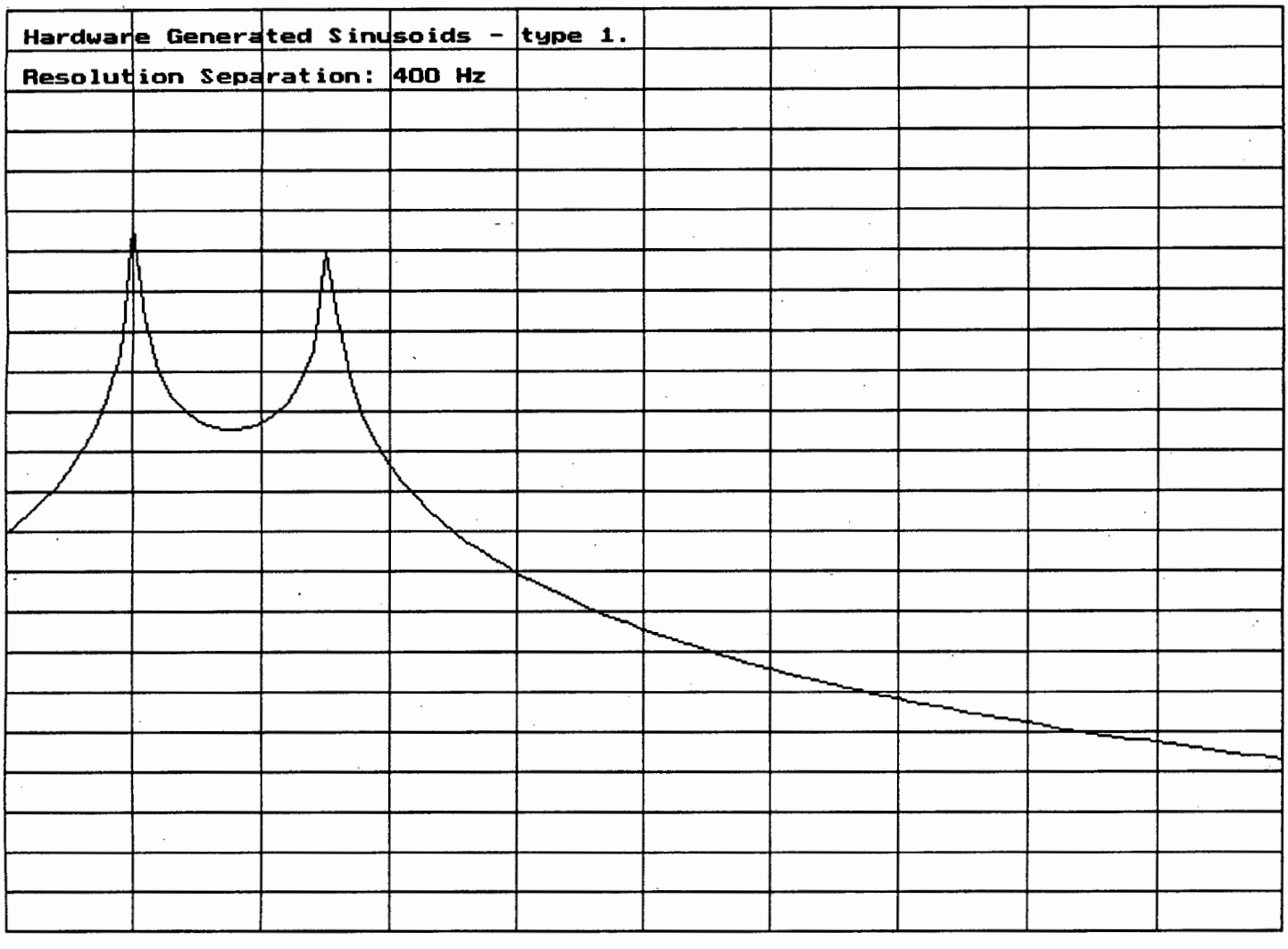
VERSION ONE: HARDWARE SAMPLING SPECTRAL PLOTS FOR VARIOUS RESOLUTION TESTS

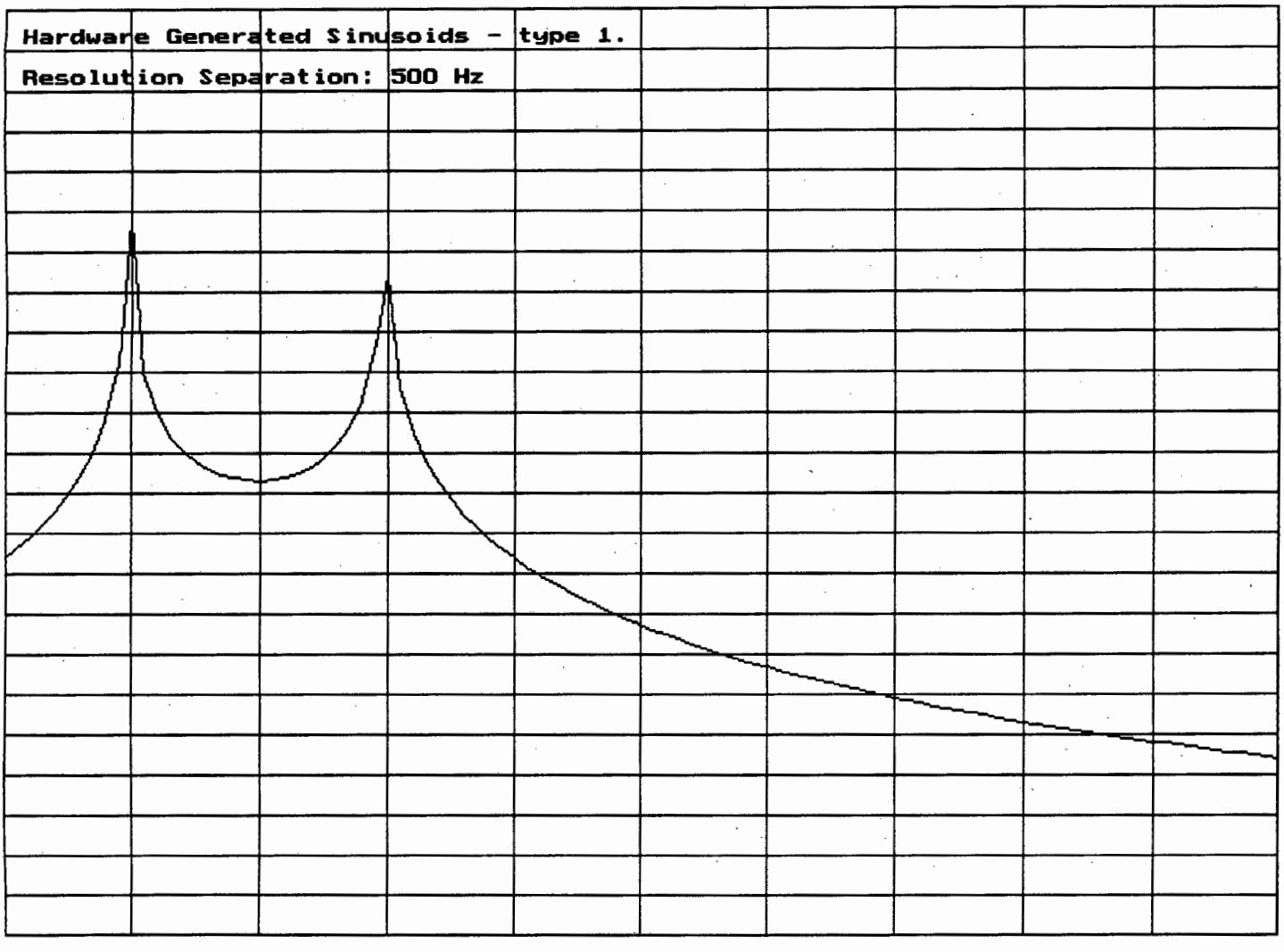
Hardware Generated Sinusoids - type 1.

Resolution Separation: 100 Hz

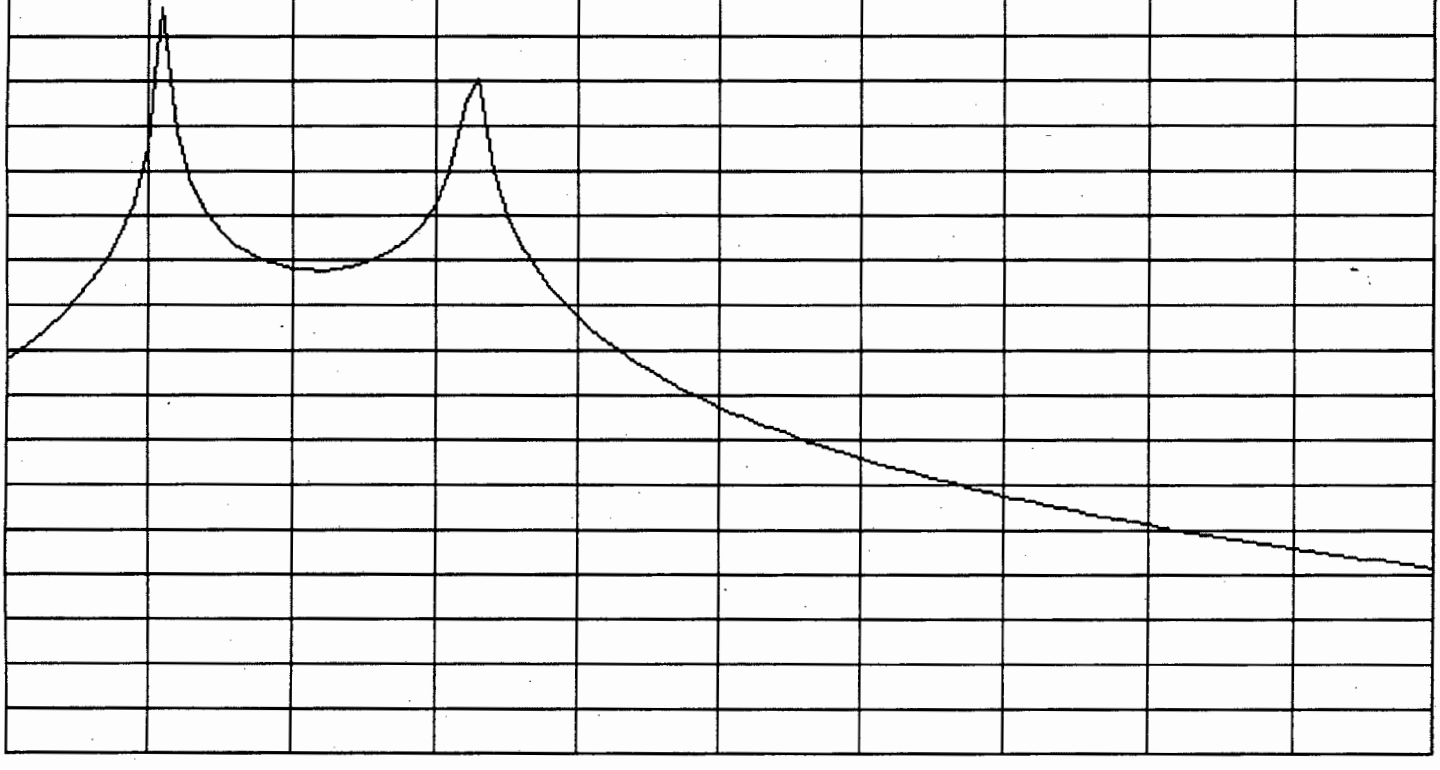

Hardware Generated Sinusoids - type 1.
Resolution Separation: 200 Hz

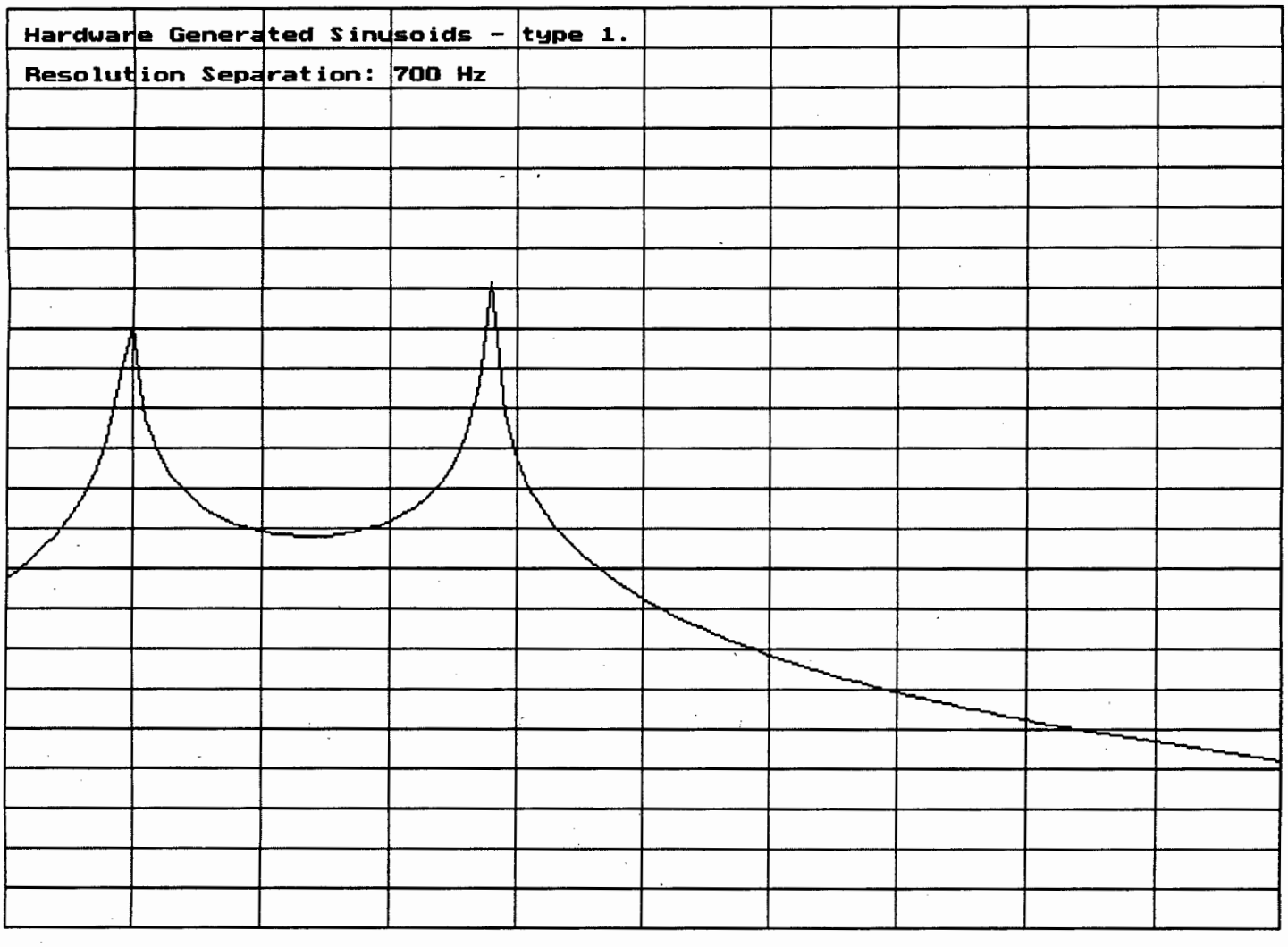

Hardware Generated Sinusoids - type 1.
Resolution Separation: 300 Hz

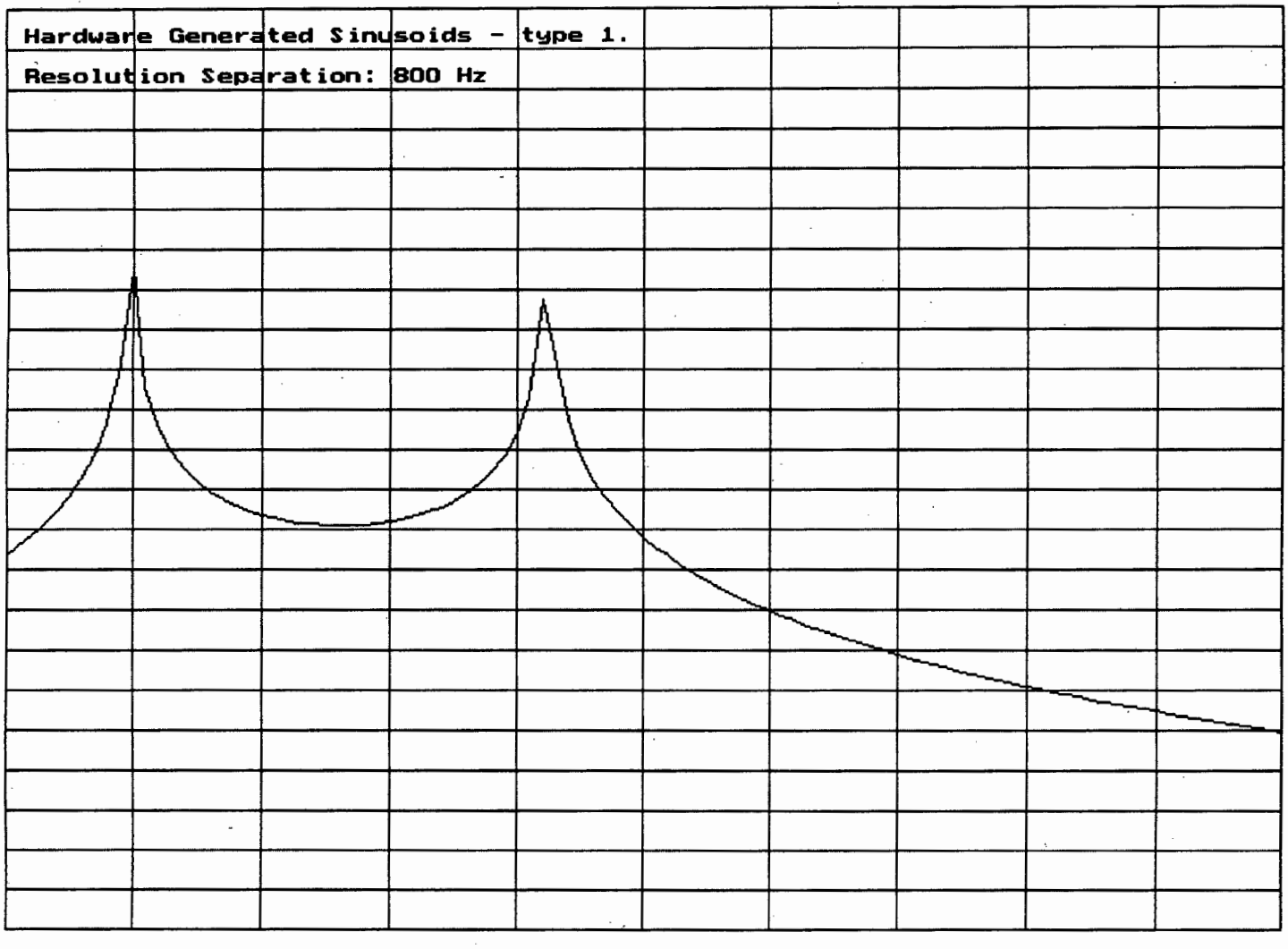

Hardware Generated Sinusoids - type 1.
Resolution Separation: 400 Hz

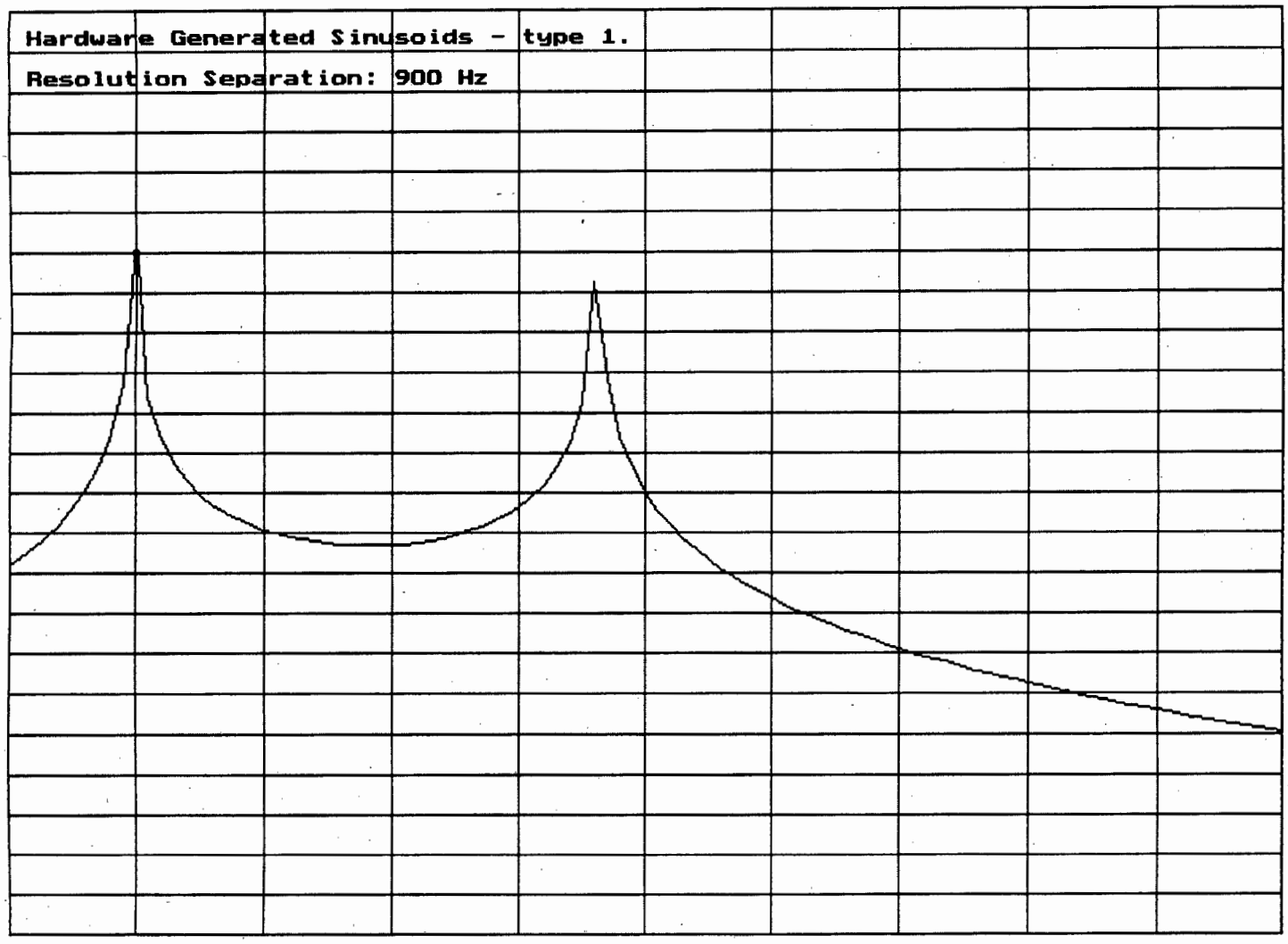

Hardware Generated Sinusoids - type 1.
Resolution Separation: 500 Hz

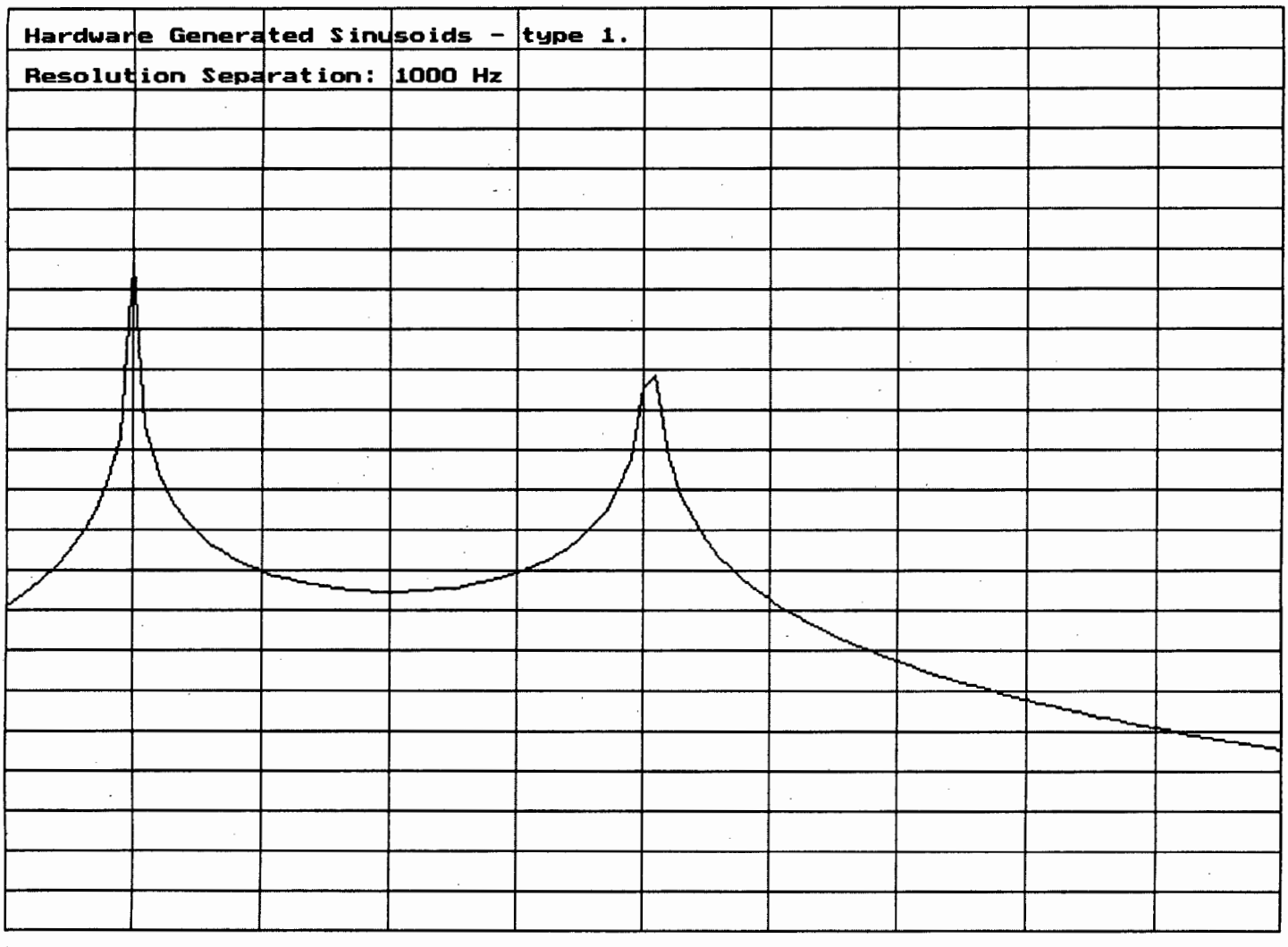
Hardware Generated Sinusoids - type 1.
Resolution Separation: 600 Hz

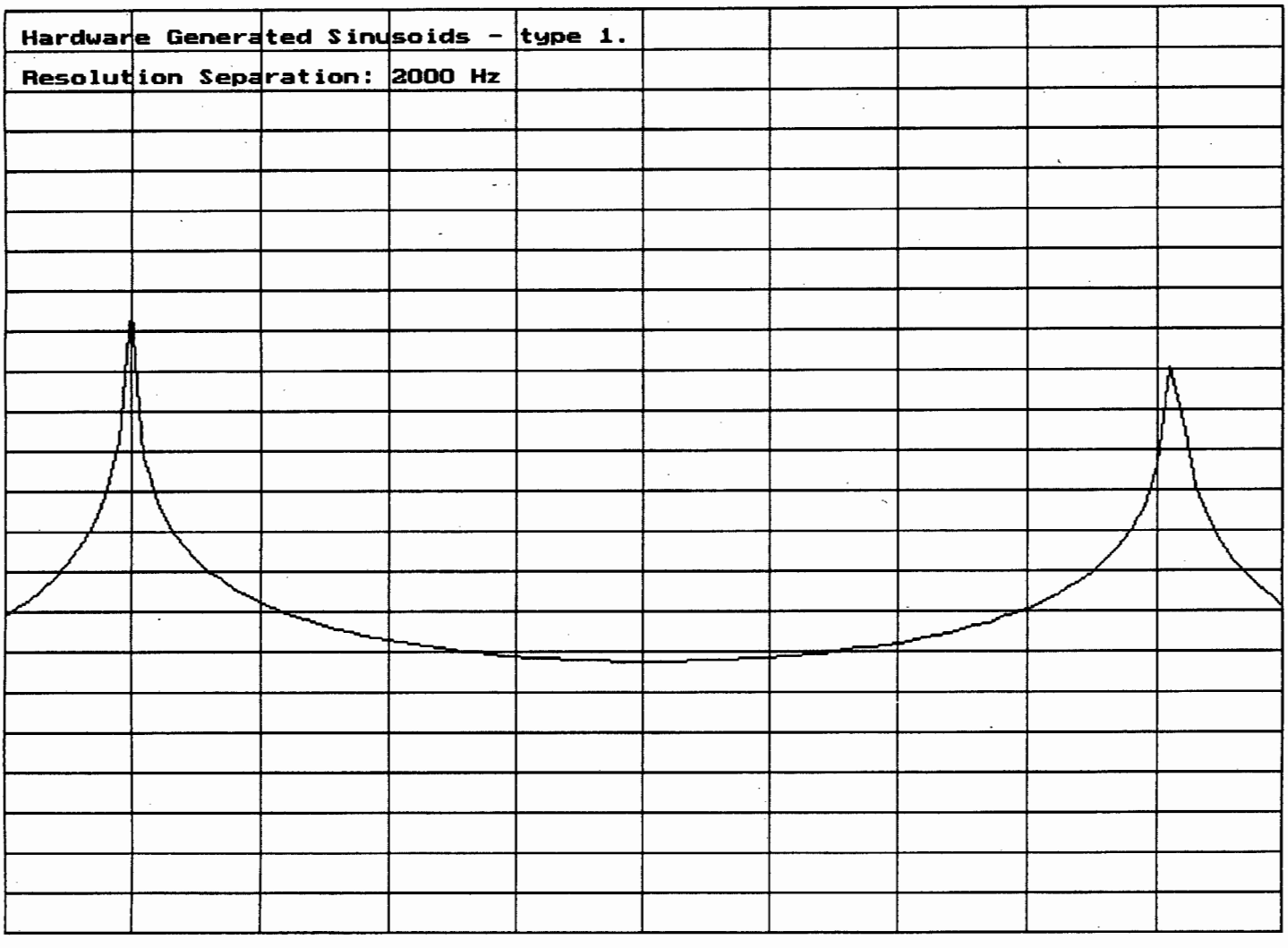



Hardware Generated Sinusoids - type 1.
Resolution Separation: 700 Hz


Hardware Generated Sinusoids - type 1.
Resolution Separation: 800 Hz


Hardware Generated Sinusoids - type 1.
Resolution Separation: 900 Hz


Hardware Generated Sinusoids - type 1.
Resolution Separation: 1000 Hz


Hardware Generated Sinusoids - type 1.
Resolution Separation: 2000 Hz

APPENDIX S

LISTING OF PROGRAM LOOP_SPEC

```
program loop_spec;

{ This program was written by A. Bas. }

{ It reads in the data from the PC30D ADC and calculates the AR parameters
  using the Modified Covariance Method. The final spectrum is plotted on the
  screen. The data can be pre-whitened if desired. }

{$N+}

uses crt,graph,pc30;

const
     base_add = $700;
     num_samples = 70;          {no. of samples}
     maxorder = 100;
     maxo1 = 101;
     maxsamples = 512;
     IP = 35;                   {no. of poles}
     noise_pow = 0;             {noise power can be set to some non-zero
                                 value if pre-whitening is required}
type
  float = double;

  complex = record
                  Re,Im : float;
            end;

  data_pnts = array[1..maxsamples] of complex;

  filt_pnts = array [1..maxorder] of complex;

  filt_pnts1 = array[1..maxo1] of complex;

{$I c:\alonbas\BURGMATH.INC}

var
   th1, th2, tm1, tm2, ts1, ts2, ths1, ths2 : word;
   N : integer;             {number of data samples}
   X : data_pnts;
   ISTAT : integer;
   A : filt_pnts;
   P : float;
   I1 : integer;
   a_d : array[0..1000] of integer;
   quit_f : boolean;
   i, test : integer;
   Ch : char;
   sample : real;
   pi : real;
   l, R1, R2 : integer;
   ARcoef : array[0..50] of complex;
   scale, sumR, sumI, mag, thet : real;
   grdriver, grmode : integer;

procedure axes;

var xaxis_pointer, yaxis_pointer : integer;

begin
```

```
      setcolor(9);
      moveto(639,0);
      lineto(0,0);
      lineto(0,400);
      lineto(639,400);
      lineto(639,0);
      moveto(0,0);

      xaxis_pointer := 0;
      repeat
            moveto(xaxis_pointer,0);
            linerel(0,400);
            xaxis_pointer := xaxis_pointer + 64;
      until xaxis_pointer > 639;

      yaxis_pointer := 0;
      moveto(0,0);
      repeat
            moveto(0,yaxis_pointer);
            linerel(639,0);
            yaxis_pointer := yaxis_pointer + 40;
      until yaxis_pointer > 400;
      setcolor(15);
      moveto(0,200);
      lineto(639,200);

end;

procedure ver_chk;

var
  r : real;
  i : integer;

begin
    i := version;
    r := hi(i) + lo(i)/100;
    quit_f := false;
    if diag <> 0 then begin
         quit_f := true;
    end;
end;

procedure set_up;

var
    Test : integer;

begin
      Init;
      Clean;
      Ad_clock(1);
      Ad_prescaler(5);
      Dma_init;
      Test := Sd_chan(0,num_samples,5,a_d[0]);
end;

procedure MCOV(N, IP : integer; X : data_pnts; var P : float;
               var A : filt_pnts; var ISTAT : integer);

var
    C, D : filt_pntsl;
```

```
   R : filt_pnts;
   LAMBDA, THETA, XI, PSI, val, val2 : complex;
   EF, EB, C1, C2, C3, C4, SAVE1, SAVE2, SAVE3, SAVE4 : complex;
   M, K, MK : integer;            {loop counters}
   DELTA, GAMMA, R1, R2, R3, R4, R5 : float;

const
     zero : complex=(Re:0.0;Im:0.0);

begin
     R1 := 0.0;
     for K := 2 to N-1 do
         begin
              squaremag(X[K],val);
              R1 := R1 + 2.0 * val.Re;
         end;

     squaremag(X[1],val);
     R2 := val.Re;

     squaremag(X[N],val);
     R3 := val.Re;

     R4 := 1.0 / (R1 + 2.0*(R2 + R3));
     P := R1 + R2 + R3;
     DELTA := 1.0 - R2 * R4;
     GAMMA := 1.0 - R3 * R4;

     multiply(X[1],X[N],val);
     conjugate(val,val);
     multiplyCR(val,R4,LAMBDA);

     multiplyCR(X[N],R4,C[1]);

     conjugate(X[1],val);
     multiplyCR(val,R4,D[1]);

     ISTAT := 0;
     M := 0;
     if IP = 0 then
         begin
              P := (0.5 * R1 + R2 + R3) / N;
              exit;
         end;

   repeat
     M := M + 1;
     SAVE1 := zero;
     for K := (M+1) to N do
         begin
              conjugate(X[K-M],val);
              multiply(X[K],val,val);
              add(SAVE1,val,SAVE1);
         end;

     multiplyCR(SAVE1,2.0,SAVE1);

     conjugate(SAVE1,R[M]);

     multiply(X[N],D[1],THETA);

     multiply(X[N],C[1],PSI);
```

```
conjugate(X[1],val);
multiply(val,D[1],XI);

if (M <> 1) then
   for K := 1 to (M-1) do
       begin
            multiply(X[N-K],D[K+1],val);
            add(THETA,val,THETA);

            multiply(X[N-K],C[K+1],val);
            add(PSI,val,PSI);

            conjugate(X[K+1],val);
            multiply(val,D[K+1],val);
            add(XI,val,XI);

            conjugate(X[N+1-M+K],val);
            multiply(X[N+1-M],val,val);
            subtract(R[K],val,R[K]);
            conjugate(X[M],val);
            multiply(val,X[M-K],val);
            subtract(R[K],val,R[K]);

            conjugate(R[K],val);
            multiply(val,A[M-K],val);
            add(SAVE1,val,SAVE1);
       end;
divideCR(SAVE1,-P,C1);
A[M] := C1;
squaremag(C1,val);
P := P * (1.0 - val.Re);
if (M<>1) then
   for K := 1 to (M div 2) do
       begin
            MK := M - K;
            SAVE1 := A[K];

            conjugate(A[MK],val);
            multiply(C1,val,val);
            add(SAVE1,val,A[K]);

            if (K <> MK) then
               begin
                    conjugate(SAVE1,val);
                    multiply(C1,val,val);
                    add(A[MK],val,A[MK]);
               end;
       end;

if M = IP then
   begin
        P := 0.5 * P / (N - M);
        exit;
   end;

squaremag(LAMBDA,val);
R1 := 1.0 /(DELTA * GAMMA - val.Re);

conjugate(LAMBDA,val);
multiply(THETA,val,val);
multiplyCR(PSI,DELTA,val2);
```

```
add(val,val2,val);
multiplyCR(val,R1,C1);

multiply(PSI,LAMBDA,val);
multiplyCR(THETA,GAMMA,val2);
add(val,val2,val);
multiplyCR(val,R1,C2);

conjugate(LAMBDA,val);
multiply(XI,val,val);
multiplyCR(THETA,DELTA,val2);
add(val,val2,val);
multiplyCR(val,R1,C3);

multiply(THETA,LAMBDA,val);
multiplyCR(XI,GAMMA,val2);
add(val,val2,val);
multiplyCR(val,R1,C4);

for K := 1 to (((M-1) div 2) + 1) do
    begin
         MK := M + 1 - K;
         conjugate(C[K],SAVE1);
         conjugate(D[K],SAVE2);
         conjugate(C[MK],SAVE3);
         conjugate(D[MK],SAVE4);

         multiply(C1,SAVE3,val);
         multiply(C2,SAVE4,val2);
         add(val,val2,val);
         add(C[K],val,C[K]);

         multiply(C3,SAVE3,val);
         multiply(C4,SAVE4,val2);
         add(val,val2,val);
         add(D[K],val,D[K]);

         if not(K = MK) then
            begin
                 multiply(C1,SAVE1,val);
                 multiply(C2,SAVE2,val2);
                 add(val,val2,val);
                 add(C[MK],val,C[MK]);

                 multiply(C3,SAVE1,val);
                 multiply(C4,SAVE2,val2);
                 add(val,val2,val);
                 add(D[MK],val,D[MK]);
            end;
    end;

squaremag(PSI,val);
R2 := val.Re;

squaremag(THETA,val);
R3 := val.Re;

squaremag(XI,val);
R4 := val.Re;

conjugate(THETA,val);
multiply(LAMBDA,val,val);
```

```
multiply(PSI,val,val);
R5 := GAMMA - (R2*DELTA + R3*GAMMA + 2.0*val.Re) * R1;

conjugate(XI,val);
multiply(LAMBDA,val,val);
multiply(THETA,val,val);
R2 := DELTA - (R3*DELTA + R4*GAMMA + 2.0*val.Re) * R1;

GAMMA := R5;
DELTA := R2;

conjugate(PSI,val);
multiply(C3,val,val);
conjugate(THETA,val2);
multiply(C4,val2,val2);
add(val,val2,val);
add(LAMBDA,val,LAMBDA);

if not(P > 0.0) then
    begin
         ISTAT := 1;
         exit;
    end;

if not((DELTA > 0.0) and (DELTA <= 1.0) and (GAMMA > 0.0) and
        (GAMMA <= 1.0)) then
        begin
             ISTAT := 2;
             exit;
        end;

  R1 := 1.0/P;
  squaremag(LAMBDA,val);
  R2 := 1.0 / (DELTA * GAMMA - val.Re);
  EF := X[M+1];
  EB := X[N-M];
  for K := 1 to M do
      begin
           multiply(A[K],X[M+1-K],val);
           add(EF,val,EF);

           conjugate(A[K],val);
           multiply(val,X[N-M+K],val);
           add(EB,val,EB);
      end;

  multiplyCR(EB,R1,C1);

  conjugate(EF,val);
  multiplyCR(val,R1,C2);

  conjugate(EB,val);
  multiplyCR(val,DELTA,val);
  multiply(EF,LAMBDA,val2);
  add(val,val2,val);
  multiplyCR(val,R2,C3);

  multiply(EB,LAMBDA,val);
  conjugate(val,val);
  multiplyCR(EF,GAMMA,val2);
  add(val,val2,val);
  multiplyCR(val,R2,C4);
```

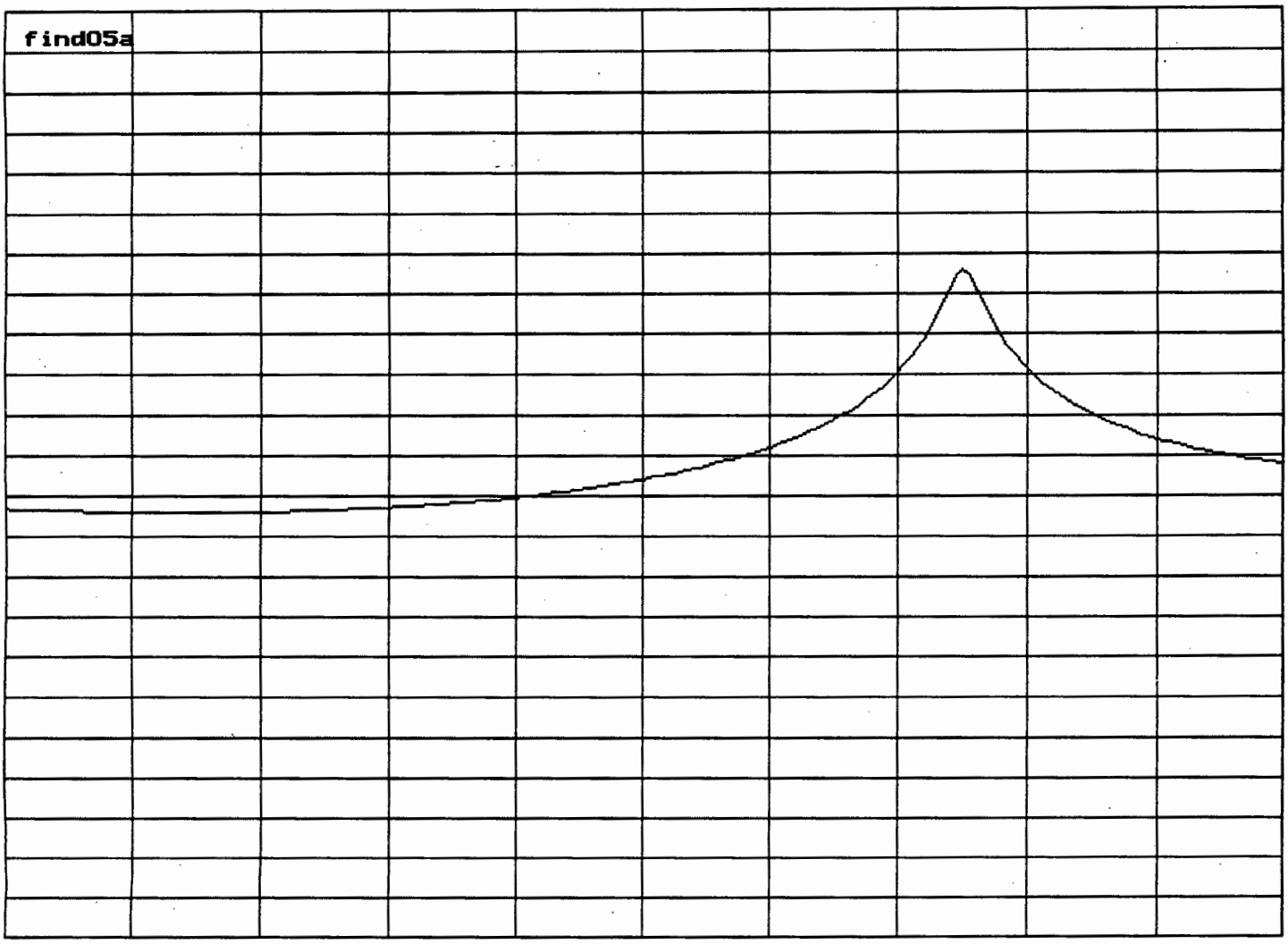
```
      for K := M downto 1 do
          begin
               SAVE1 := A[K];
               multiply(C3,C[K],val);
               multiply(C4,D[K],val2);
               add(val,val2,val);
               add(SAVE1,val,A[K]);

               multiply(C1,SAVE1,val);
               add(C[K],val,C[K+1]);

               multiply(C2,SAVE1,val);
               add(D[K],val,D[K+1]);
          end;

      C[1] := C1;
      D[1] := C2;
      squaremag(EB,val);
      R3 := val.Re;

      squaremag(EF,val);
      R4 := val.Re;

      multiply(EB,LAMBDA,val);
      multiply(EF,val,val);
      P := P - (R3*DELTA + R4*GAMMA + 2.0*val.Re) * R2;
      DELTA := DELTA - R4 * R1;
      GAMMA := GAMMA - R3 * R1;
      multiply(EF,EB,val);
      conjugate(val,val);
      multiplyCR(val,R1,val);
      add(LAMBDA,val,LAMBDA);
      if not(P > 0.0) then
         begin
              ISTAT := 3;
              exit;
         end;
      if not((DELTA > 0.0) and (DELTA <= 1.0) and (GAMMA > 0.0) and
             (GAMMA <= 1.0)) then
         begin
              ISTAT := 4;
              exit;
         end;
    until false;


end;

function noise : double;

var sum : double;
    a : integer;

begin
     sum := 0;
     for a := 1 to 12 do
               sum := sum + noise_pow * 2 * (((random(32767) / 32767) - 0.5));
     {writeln(sum);}
     noise := sum;
end;
```

```
begin
      repeat
            test := 100;
            rtc_off;
            set_base(base_add);
            ver_chk;
            if quit_f then halt;
            Port[base_add + 11] := 144;
            Port[Base_add + 9] := 1;  { disable a/d }
            set_up;
            repeat
            until (Port[Base_add + 8] and 1) = 0;  {wait for 1 - enable count}
            repeat
            until (Port[Base_add + 8] and 1) = 1;  {wait for 0 - no count}
            Port[Base_add + 9] := 0;  { enable a/d }
            while (test <> 0) do
                  begin
                        test := Dma_chk;
                  end;
            Test := Dma_close;
            rtc_on;

   for i := 1 to num_samples do
       begin
            X[i].Re := ((a_d[i-1] and 4095) - 2047) / 2048 * 5;
            X[i].Re := X[i].Re + noise;
            if (X[i].Re >= 4.99) then
               begin
                    sound(500);
                    delay(100);
                    nosound;
               end;
            X[i].Im := 0;
       end;

  N := num_samples;
  MCOV(N,IP,X,P,A,ISTAT);
  writeln;

  case ISTAT of
    0:begin
       writeln(ISTAT,' : Success');
       writeln('sig2 = ',P);
      end;
    1:writeln(ISTAT,': Ill conditioned data');
    2:writeln(ISTAT,': Ill conditioned data');
    3:writeln(ISTAT,': Ill conditioned data');
    4:writeln(ISTAT,': Ill conditioned data');

  end;
      writeln;
      writeln(-10.0 / 2.3 * ln(P):5:2,' dB');
      writeln;
      pi := 3.141592654;
      N :=   8192; {4096;} {32767;} {16384;} {4096;} {16384;}
      R1 :=  320;  {160;} {2000;} {1000;} {160;}  {840;}
      R2 :=  960;  {480;} {3000;} {1500;} {480;}  {1100;}
      scale := 1e5;
      writeln(N,'  ',R1,'  ',R2,'  ',scale);
      delay(500);
      grdriver := detect;
```

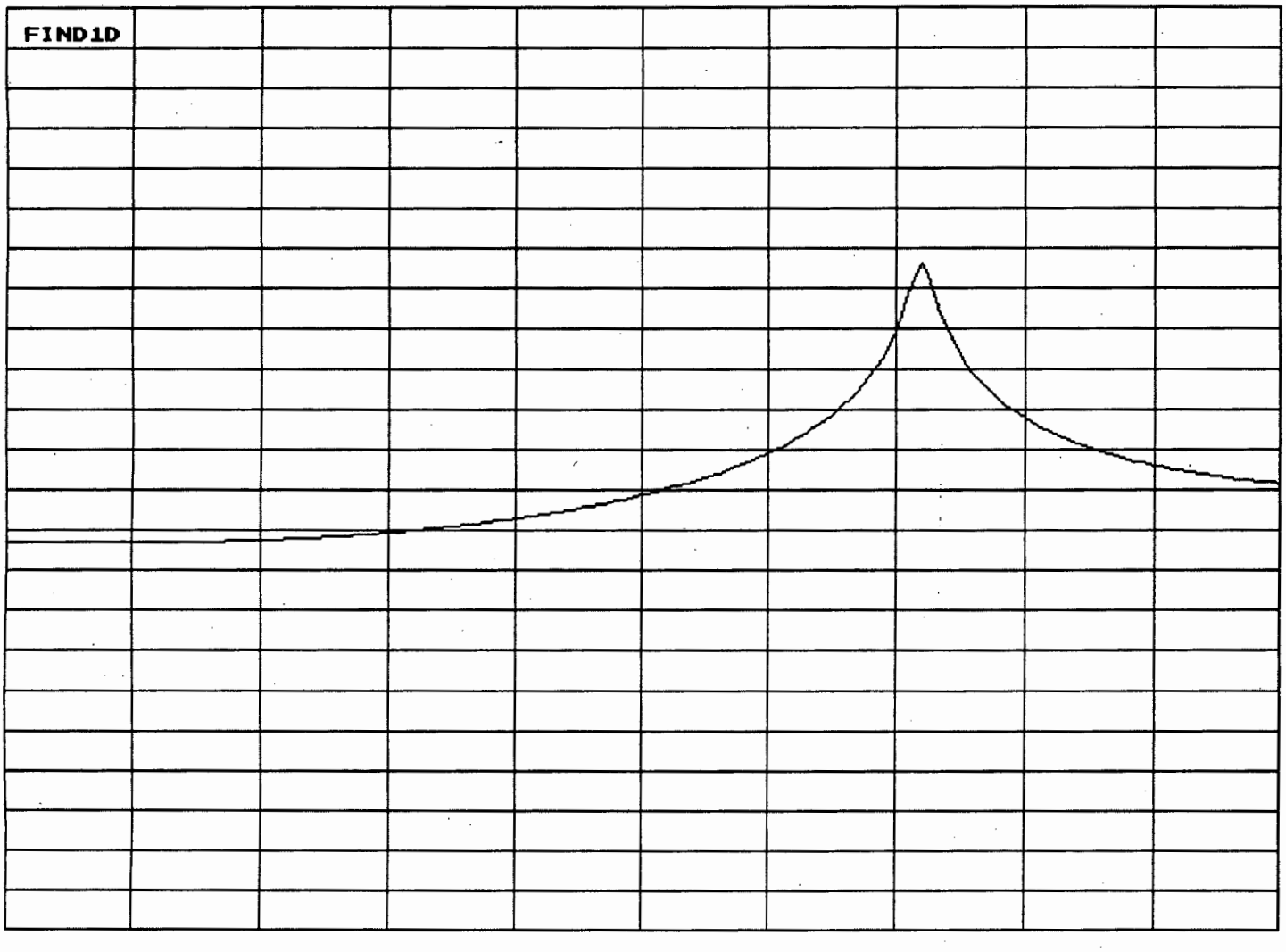
```
        initgraph(grdriver,grmode,'');
        setgraphmode(grmode);
        axes;
        moveto(0,400);
        setcolor(15);

        for i := R1 to R2 do
            begin
                 sumR := 1;
                 sumI := 0;
                 thet := i / N * 2 * pi;
                 for l := 1 to IP do
                     begin
                          sumR := sumR + (A[l].Re * cos(thet*l) + A[l].Im *
                                          sin(thet*l));
                          sumI := sumI + (A[l].Im * cos(thet*l) - A[l].Re *
                                          sin(thet*l));
                     end;
                 mag := SQR(sumR) + SQR(sumI);
                 lineto(round((i - R1) / (R2 - R1) * 640),200 - round(40.0 *
                        ln((1.0 / mag) / scale) / ln(10.0)));
            end;

            sound(100);
            delay(100);
            nosound;

            delay(1000);
            closegraph;
        until false;

end.
```

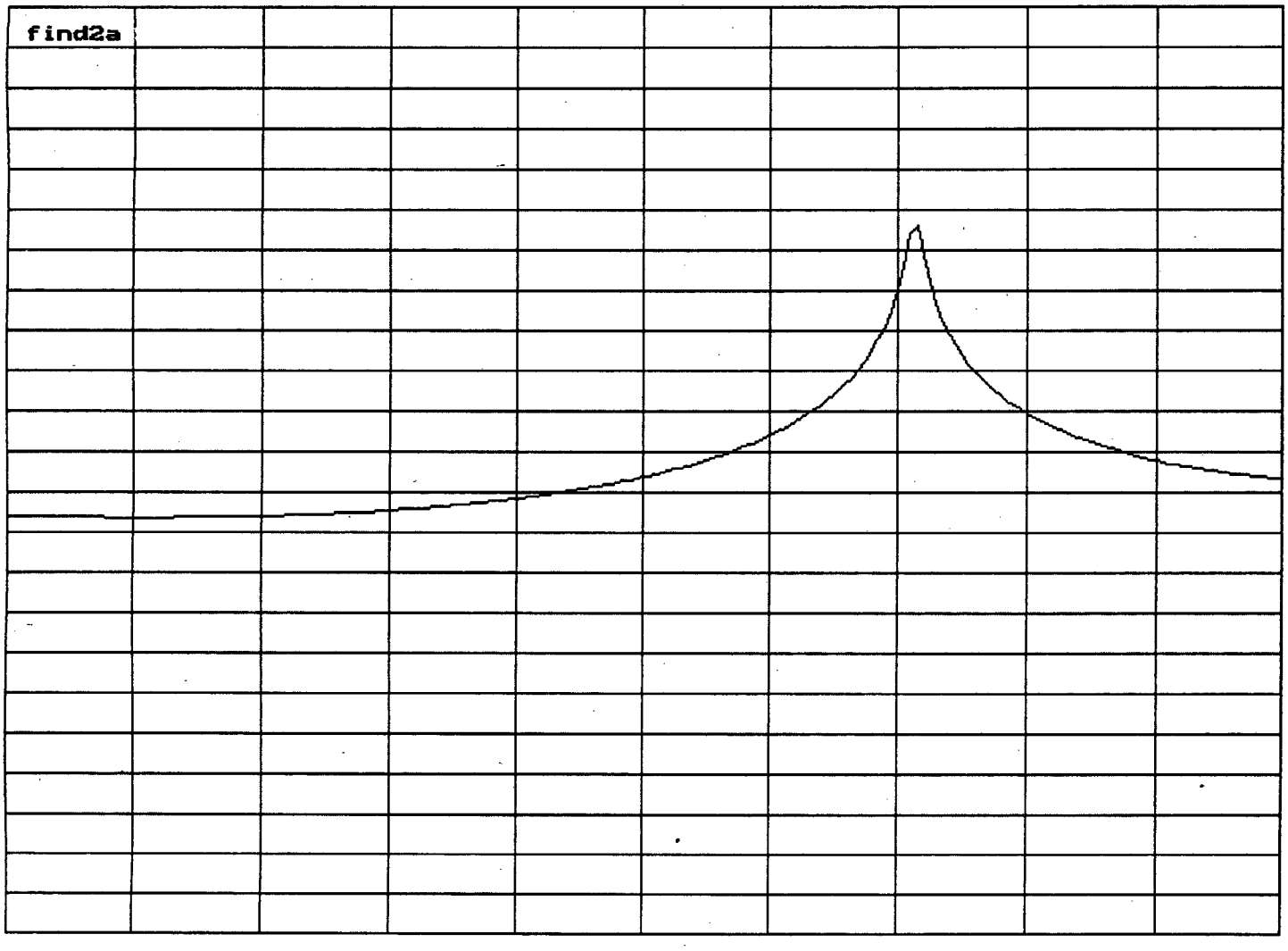
APPENDIX T

SCALED RADAR PLOTS OF SINGLE TARGET RANGE RETURNS

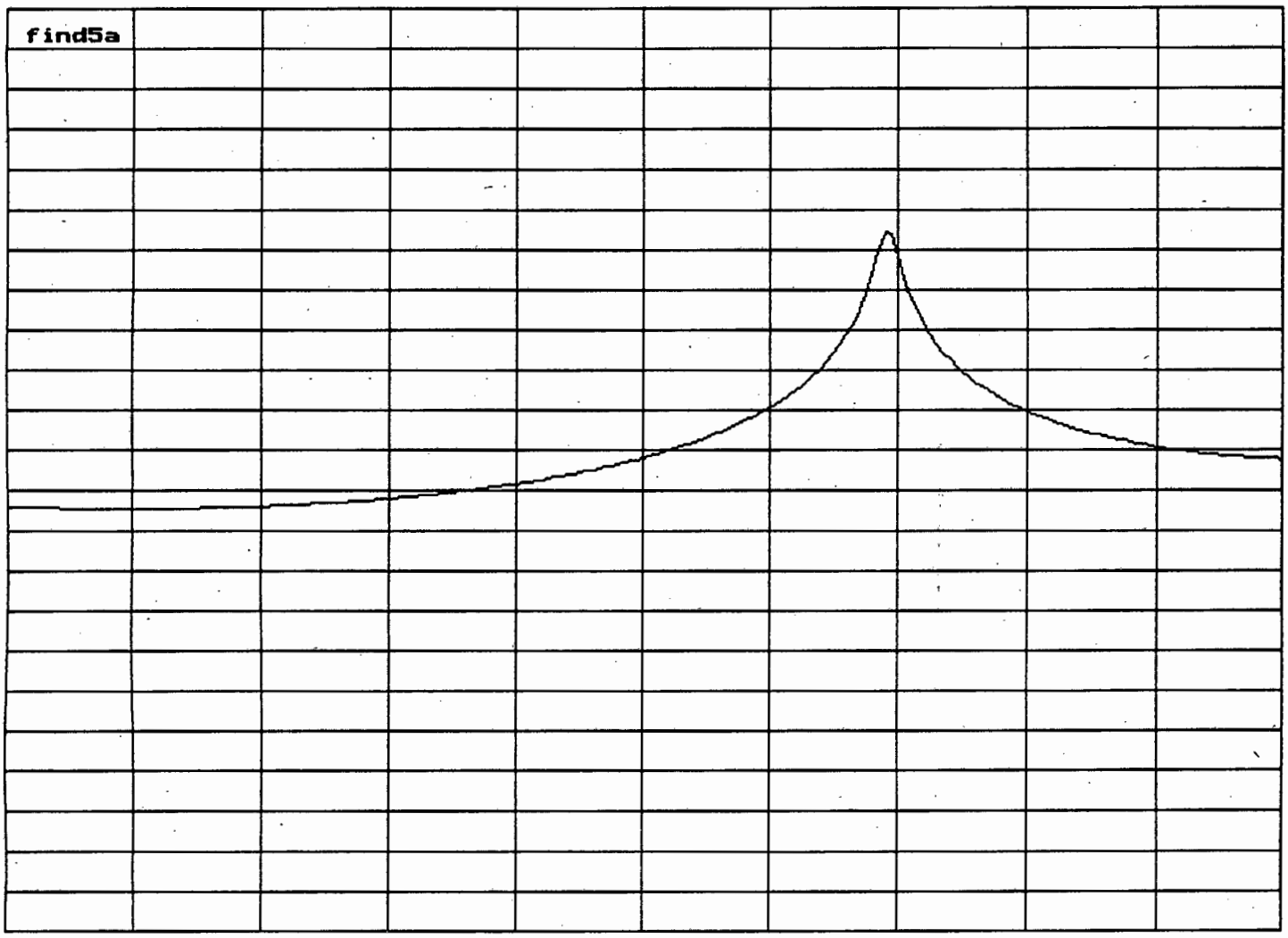


0.5 cm

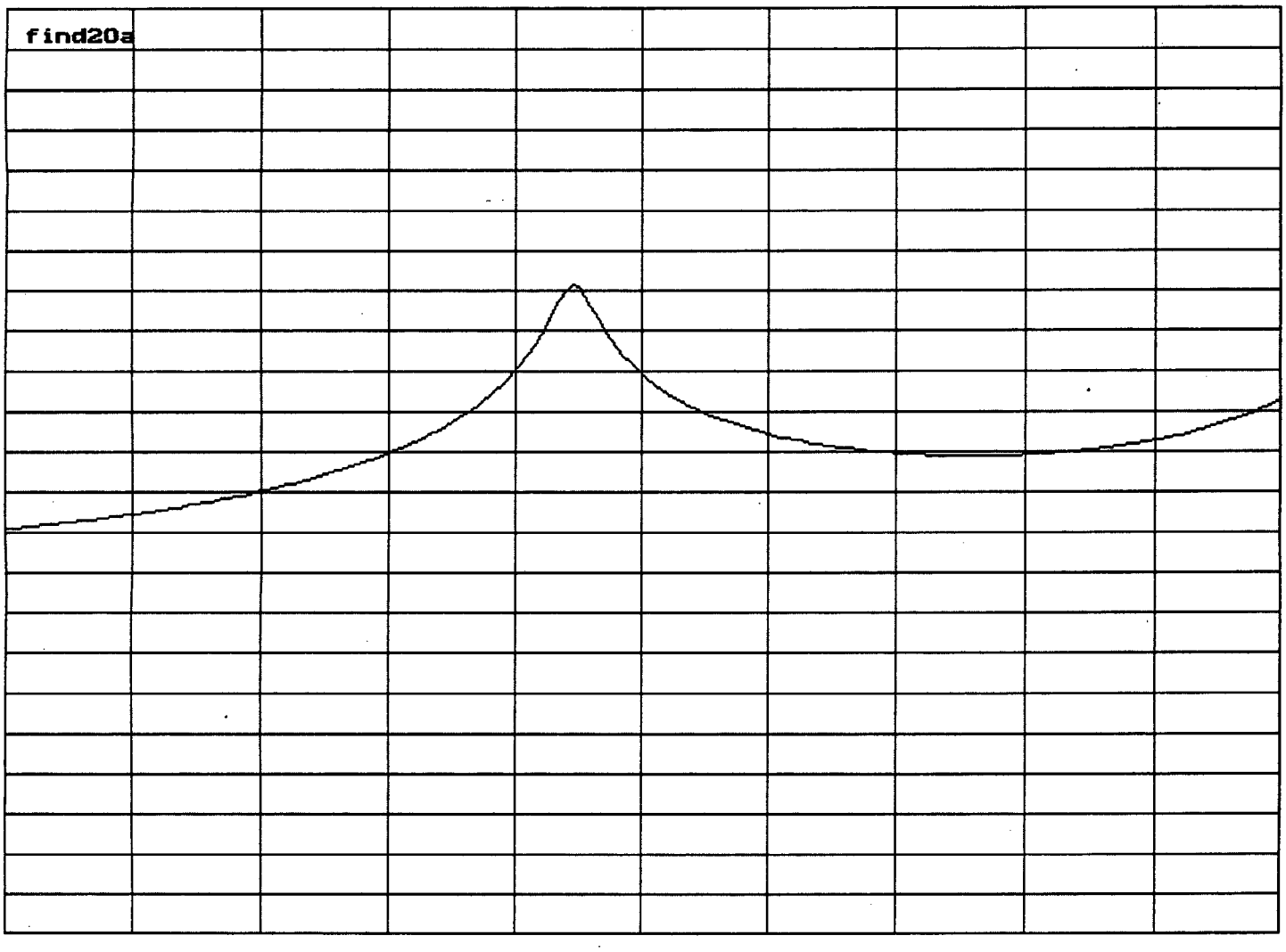

FIND1D
1cm



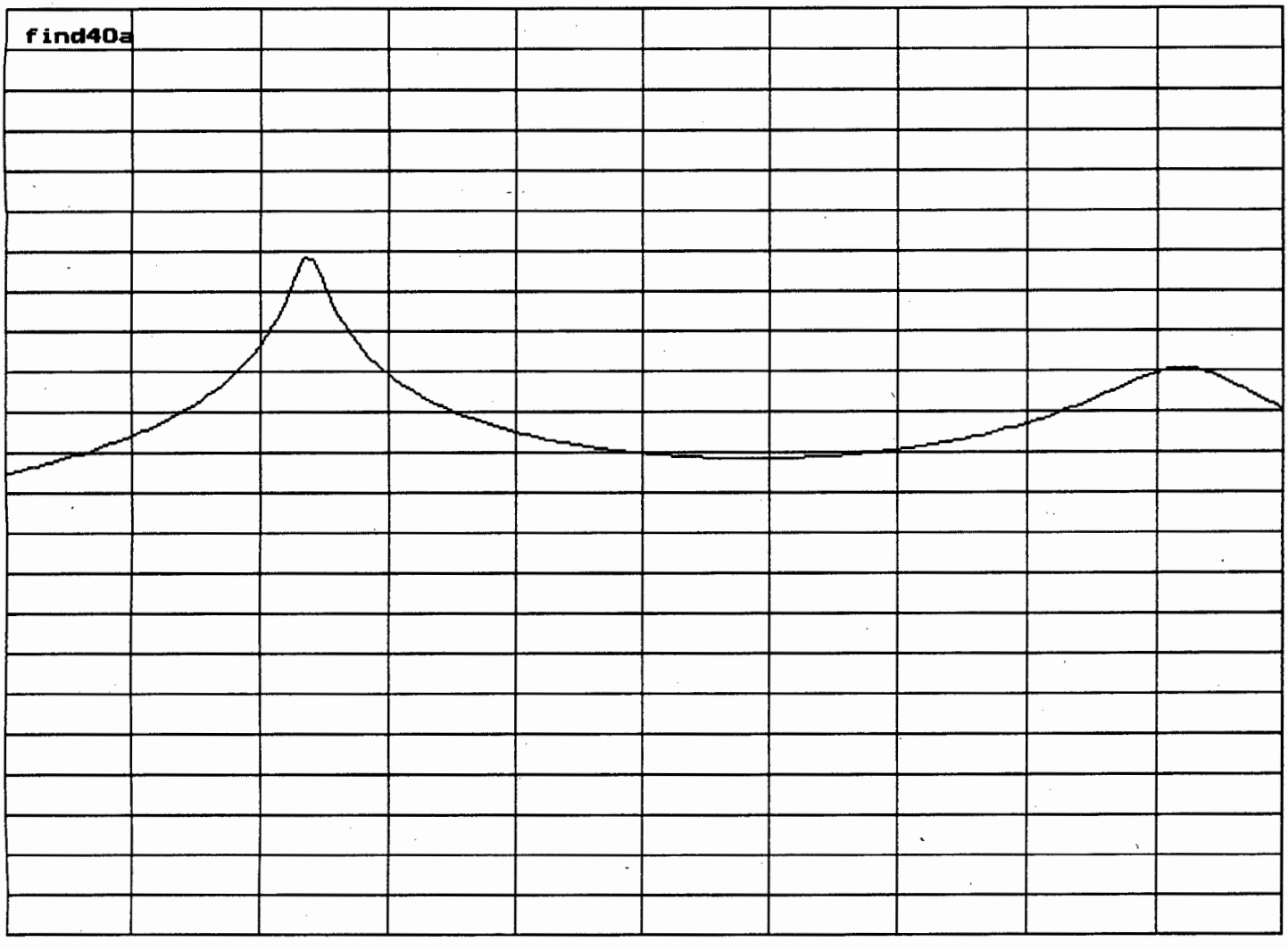
2 cm



5cm



20 cm



40cm

APPENDIX U

MODULATOR & PSU CHANGES FOR THE YIG TUNED OSCILLATOR

The main tuning coil of the YIG tuned oscillator needs to be driven by a current source, since the frequency is proportional to the magnetic field. Therefore, the modulator output has to be modified by a current driver. The circuit diagram in shown below.

U1 is a differential op-amp configuration. VR1 is required to apply a DC offset to the main tuning coil. This is required since the oscillator has a 6.5GHz bandwidth from 4.0GHz. VR2 is used to scale the 20V modulator output. U2 is a current source with buffering darlington transistor Q1. U2 has an extremely high slew rate. The current is programmed by R5. The relation is:

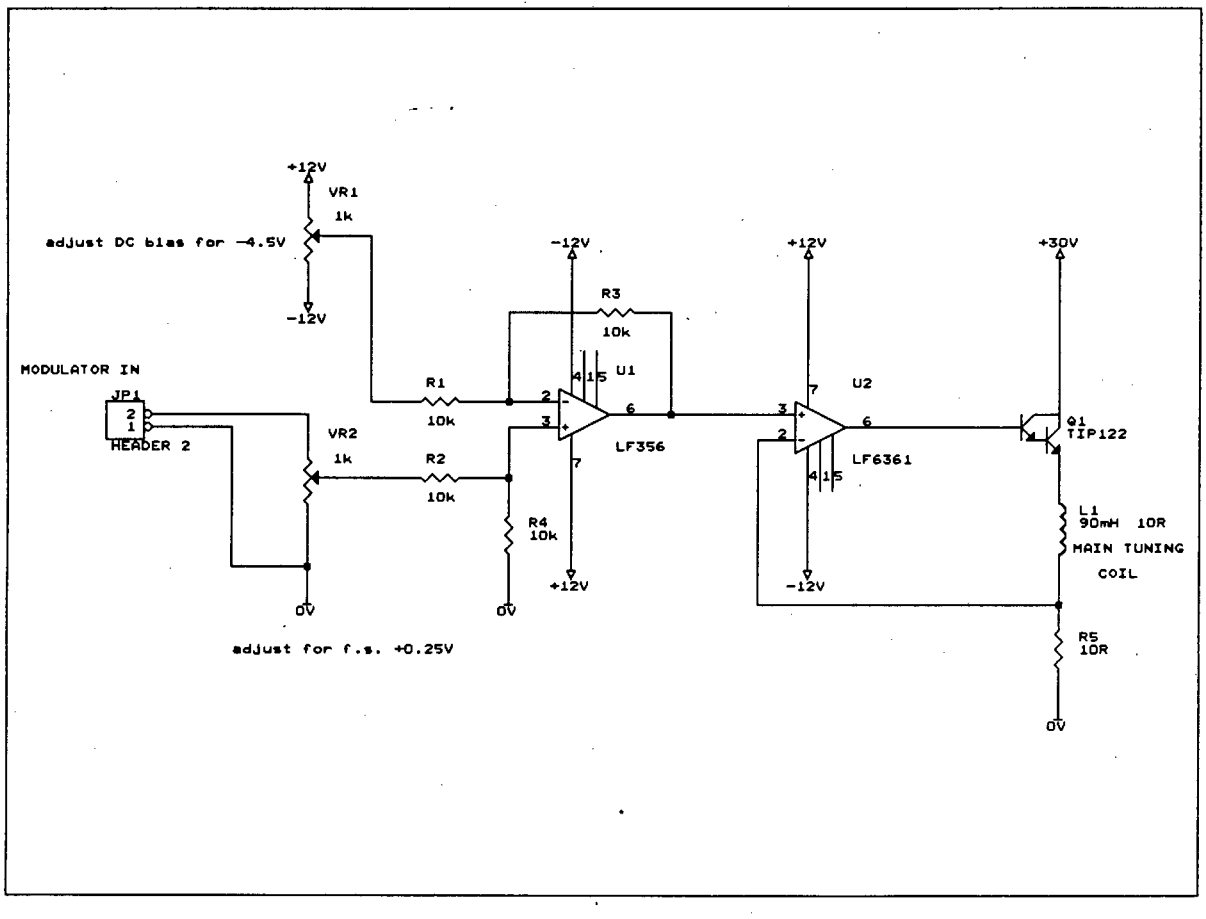
$$I_{out} = \frac{V_{in}}{R5}$$

$$= 0.1 * V_{in}$$

The DC current is set to 450mA; therefore VR1 needs to be set to -4.5V. The main tuning coil has a 20MHz / mA frequency pushing capability, therefore VR2 needs to be set so that the 20V modulator output is scaled to +0.25V for a 25mA modulation signal.

Laboratory power supplies were used to provide the extra supplies required for the YIG tuned oscillator. These were:

1) +15V from system +10V power supply (adjust voltage regulator).
2) -5V
3) +24V heater supply
4) +30V driver supply.



YIG Coil Driver.