



A DGPS/Radiobeacon Receiver for Minimum Shift Keying with Soft Decision Capabilities

Tim Wescott

www.wescottdesign.com



A DGPS/Radiobeacon Receiver for Minimum Shift Keying with Soft Decision Capabilities

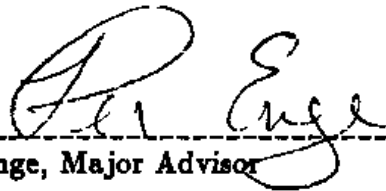
by
Tim Wescott

A Thesis
Submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
degree of Master of Science
in
Electrical Engineering
by

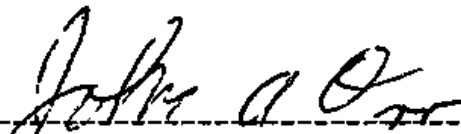


August 17, 1990

APPROVED:



Dr. Per K. Enge, Major Advisor



Dr. John A. Orr, Head of Department

A DGPS/Radiobeacon Receiver for Minimum Shift Keying with Soft Decision Capabilities

The Global Positioning System (GPS) is now in operation, and many improvements to its performance are being sought. One such improvement is Differential GPS (DGPS), where known errors in the GPS broadcast are identified and the corrections broadcast to the end user [12]. One implementation of DGPS being considered is the use of coastal marine radio direction finding (RDF) radiobeacons in the 285-325kHz band as transmitters for the DGPS broadcast. The normal RDF beacon signal consists of a continuous carrier on a one kilohertz boundary plus a Morse-code identification signal 1025Hz above the carrier. In the DGPS/radiobeacon implementation proposed for the US coastal regions, the differential data link signal uses minimum shift keying (MSK) at a data rate of 25, 60, 100, 200, or 400 baud (the exact baud rate has not yet been decided). This MSK signal is centered between the RDF beacon carrier and identification signal.

At the frequencies that these radiobeacons are operated, the prevailing atmospheric noise is both non-Gaussian and very strong. This noise characteristic makes the design of a long-range data link difficult. One solution that has been proposed is the use of forward error correction (FEC) coding of the data. The performance of FEC decoders can be improved by the use of a soft decision receiver, which delivers both bit decisions and information about the validity of the bit decisions [15].

This work describes the design of a radio receiver for DGPS/radiobeacon service which is capable of reception of 400 baud MSK in the DGPS/radiobeacon band. The receiver is designed to be easily augmented to provide soft decisions and easily modified to receive MSK at data rates of 25 to 400 baud. The radio is a microprocessor controlled dual conversion superheterodyne with an audio frequency of 1kHz. The demodulator runs on the same microcomputer that controls the radio. The weak-signal performance of the demodulator is very good: the vs. bit error rate performance of the demodulator is only a couple of dB worse than the theoretical performance of differential phase-shift keying. The radio has a noise floor of -114dBm referenced to its 500Hz wide audio bandwidth and a 3rd order inter modulation intercept of +7dBm for a dynamic range of 83dB,

This work concludes with a thumbnail analysis of the operations needed to implement a soft bit decision estimator, and some suggestions for the implementation of said soft bit decision estimator.

Preface

The Story of the Web Edition

This Master's Thesis is a companion to a radio receiver that I built for Per Enge, then of Worcester Polytechnic Institute in Worcester, Massachusetts, in 1989 and 1990. The motivation for the receiver on Per's part was to get his hands on something that would receive the then-experimental US Coast Guard differential GPS data link broadcasts. Per needed a radio receiver that would not only receive MSK at the correct frequency, but would make “soft” decisions, so that he and his team of graduate students could experiment with advanced forward-error-correction decoding schemes. I needed a thesis topic.

My search for a thesis advisor was a well-thought out, highly detailed study. Basically, I took a list of all the electrical engineering faculty at WPI that had anything remotely to do with communications equipment, and I went knocking on doors. Fortunately for both Per and I, his was the first office on my list that was occupied. I explained that I was looking for a thesis topic that would involve communications equipment, preferably radio. He pulled out a schematic of a radio receiver and said “can you build something like this?”

I had reservations about the radio he wanted me to build. As I found out much later, he had reservations about my ability to build a radio. But the project worked out well for both of us – I got to build a radio receiver for my Master's Thesis and I got some excellent training in writing book-length documents. Per got what turned out to be only the second receiver designed for USCG DGPS radio beacon service. The receiver served him well for several years, and in the end I was fortunate enough to get it back to add to my collection.

When I wrote down this Master's Thesis, I thought I was doing it to check off a box in my graduation requirements, to learn a bit about writing really long (and even slightly scholarly) documents, and to make my Thesis advisor happy. I kept a copy of it on my bookshelf mostly out of pride, although I did have occasion to refer to one or two tidbits contained within. I assumed that I had wrung all of the use out of it that it was ever going to get.

Then, 15 years after it was written, I started coming across queries on USENET asking how to design and construct MSK receivers. People wanted to build MSK receivers, but were unable to find information on how to do the job. After some fruitless web searches, I decided that since I'd already done it once I should write a comprehensive article on the design and construction of an MSK receiver. After re-reading my Master's thesis, I decided that I already had, and all I needed to do was to post it to the web.

I solicited help on USENET, and Joel Kolstad, KE7CDV, kindly stepped forward and scanned my entire thesis into electronic form. Since that time, this thesis has been popular enough reading that we have run the whole thing through OCR software, turning it into text, and have made it into a document that is (hopefully) more readable than the original scanned-in version.

Keep in mind as you read this that the methods presented in this document are just one way to implement an MSK receiver. I designed this receiver and it worked, but I would never, ever suggest that it is the optimal design – in fact, many of the central design decisions of this radio

were made late at night under intense schedule pressure by a neophyte engineer. So read it for the basic theory and as an example of how *one* such radio was built, but don't assume that it can't be vastly improved upon.

Table of Contents

| | | |
|-------|---|----|
| 1 | Introduction..... | 6 |
| 1.1 | The Global Positioning System..... | 6 |
| 1.2 | DGFS and Radiobeacons..... | 7 |
| 1.3 | Minimum Shift Keying..... | 8 |
| 1.4 | Atmospheric Noise and Forward Error Correction..... | 8 |
| 1.5 | Side Information..... | 8 |
| 1.6 | Outline of Thesis..... | 8 |
| 2 | The Radio..... | 10 |
| 2.1 | The Mixing Scheme..... | 10 |
| 2.2 | The Local Oscillator..... | 12 |
| 2.3 | Front end Circuitry..... | 14 |
| 2.3.1 | Antenna Coupler..... | 14 |
| 2.3.2 | RF filter, Mixer..... | 15 |
| 2.3.3 | IF preamp..... | 15 |
| 2.4 | IF Filter..... | 16 |
| 2.5 | IF Amplifier..... | 16 |
| 2.6 | 2nd Mixer and 2nd Local Oscillator..... | 17 |
| 2.7 | Audio Amplifier and AGC Amplifier..... | 17 |
| 2.8 | Microprocessor..... | 17 |
| 2.8.1 | The MC68HC11..... | 17 |
| 3 | The Demodulators..... | 19 |
| 1.7 | MSK..... | 20 |
| 3.1.1 | MSK Detection..... | 23 |
| 3.1.2 | Synchronization..... | 24 |
| 3.2 | Microprocessor based demodulator..... | 25 |
| 3.3 | The Timing Recovery Algorithm..... | 26 |
| 3.3.1 | Phase error estimation..... | 29 |
| 3.3.2 | The gain blocks..... | 29 |
| 3.3.3 | Loop Filter Calculations and Bandwidth..... | 32 |
| 3.3.4 | Limit Cycles..... | 33 |
| 3.3.5 | Lock Acquisition..... | 34 |
| 3.3.6 | Carrier Stability and Carrier I\Loop Bandwidth..... | 34 |
| 3.4 | The Correlation Detector..... | 36 |
| 4 | Tests of Radio Performance..... | 39 |
| 4.1 | Bit error rate in the presence of Gaussian noise..... | 39 |
| 4.2 | BER in the presence of an interfering carrier..... | 43 |
| 4.3 | 3rd Order Inter modulation Performance..... | 45 |
| 4.4 | Receiver Noise Floor..... | 47 |
| 5 | Summary, Some Comments on Soft Bit Decisions, and Further Work..... | 49 |
| 5.1 | Summary..... | 49 |
| 5.2 | Some Comments on Soft Bit Decisions..... | 49 |
| 5.2.1 | The Auxiliary Channel Method..... | 49 |

| | |
|--|-----|
| A DGPS/Radiobeacon Receiver for Minimum Shift Keying with Soft Decision Capabilities | 7 |
| 5.2.2 The In-Channel Method..... | 49 |
| 5.3 Suggestions for Further Work..... | 52 |
| 1 Appendix A..... | 54 |
| 1.1 The Continuous-time MSK Demodulator..... | 54 |
| 2 Appendix B..... | 57 |
| 2.1 Receiver Software..... | 57 |
| 3 Appendix C..... | 87 |
| 3.1 Transmitter Software..... | 87 |
| 4 Appendix D..... | 104 |
| 4.1 Schematics..... | 105 |
| 5 Bibliography..... | 111 |

1 Introduction

1.1 *The Global Positioning System*

The Global Positioning System (GPS) is a worldwide real-time navigation system, currently being developed by the Department of Defense, that will broadcast ranging signals from a constellation of 24 satellites. Various mechanisms limit the accuracy of users' range measurements and ultimately their position determinations. These mechanisms include some errors that vary rapidly in time and space, and others that vary slowly. The slowly varying errors include unmodeled ionospheric delay and "selective availability", which is intentionally introduced for purposes of national defense. Slowly varying errors can be corrected using differential techniques, providing a channel is available to convey the differential correction information to users.

Differential GPS (DGPS) requires a local reference station, a communication link to users, and user GPS receivers [12]. The reference station has its own high quality GPS receiver that continuously measures range to all satellites in view, and determines local range errors using its known surveyed location. Corrections for the local errors are then communicated to users, who remove the errors from the ranging information determined by their own GPS receiver. Civilian users can improve their position fixing accuracy from 100 meters to better than 30 meters using differential corrections, making DGPS potentially suitable for many precision civilian navigation applications.

1.2 DGFS and Radiobeacons

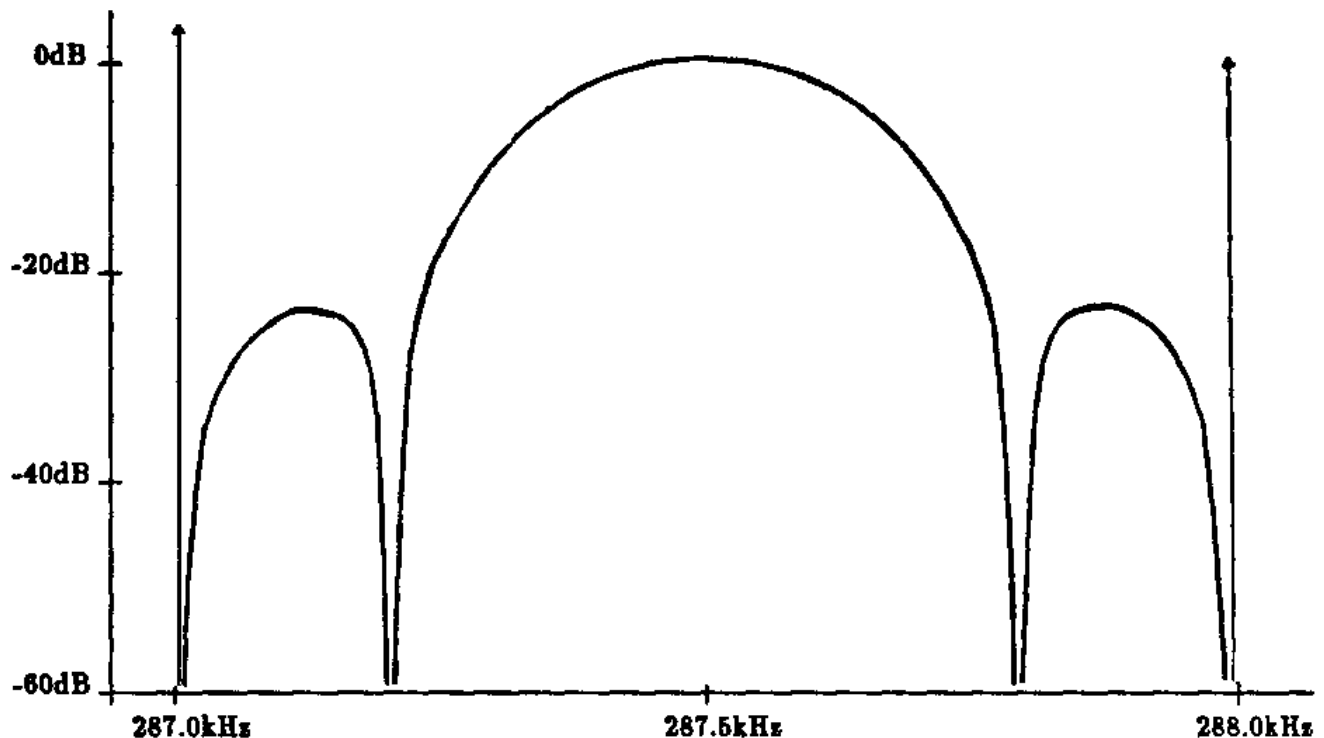


Figure 1: Example spectrum of a DGPS/Radiobeacon operating at 287kHz with a 400 baud minimum shift keying signal between the radiobeacon carrier and ID signals

The US Coast Guard maintains and operates a system of marine radiobeacons, located along the Atlantic, Gulf and Pacific coasts and on the Great Lakes [4]. These radiobeacons provide a 2-line fix capability out to 50 nautical miles or the 100 fathom curve, whichever is greater. They operate in the 285-325 kHz band, with a nominal channel spacing of 1kHz. Similar radiobeacons are operated by various European countries at a channel spacing of 1kHz, which is currently being reduced to 500Hz [6].

These radiobeacons offer an attractive communications link for DGPS because they are widespread, located near major harbors, and, being MF, would require inexpensive user receivers. At this writing, it appears that the messages will be transmitted using minimum shift keying (MSK) using baud rates from 25 to 400, and perhaps using forward error correcting (FEC) with the MSK carrier 500Hz above the radiobeacon carrier with a 3dB subcarrier protection ratio (for the US Coast Guard version) [6]. Figure 1 shows an example of the spectrum of a radiobeacon with a 400 baud DGPS MSK data link. Previous investigators have found that a 100 baud DGPS signal can be transmitted on a radiobeacon link with minimal interference to radio direction finding sets [6].

1.3 Minimum Shift Keying

The most likely candidate for the transmission format for the DGPS data link is minimum shift keying

(MSK). MSK can be thought of either as phase coherent frequency shift keying (FSK) with a deviation ratio of 1/2 (ie. the difference between the mark and space frequencies is 1/2 the baud rate), or as sinusoidally weighted offset quadrature phase shift keying (OQPSK). A thorough analysis of MSK is given in [1] and [2], and a review is given at the beginning of chapter 3. MSK can be generated as phase coherent FSK, and optimally demodulated in Gaussian noise as PSK [1, 2].

MSK has a number of advantages for a DGPS data link. It is spectrally compact [2], which is an advantage in the bandwidth-limited LF range. Because it is a type of FSK it has a constant envelope, which makes it easier to transmit. It lacks sudden phase changes, so it is only minimally distorted when being radiated by a highly reactive antenna (which is important at LF, where the antenna is usually only .05 wavelength long).

1.4 Atmospheric Noise and Forward Error Correction

At 300 kHz, the most important limit to DGPS/Radiobeacon range is atmospheric noise, largely due to lightning. This noise is bursty and highly non-Gaussian. Specifically, the probability density of this noise is much greater in the high-power 'tails' than is the case with Gaussian noise [10].

In the case of Gaussian noise, the bit error rate (BER) can be reduced by increasing the bit interval (ie., decreasing the baud rate) or by increasing the transmitted power. This is also true for typical atmospheric noise, but the improvement in the BER for a given increase in power or bit interval is not nearly as great as for Gaussian noise.

Another way to reduce the ultimate BER is to use forward error correction (FEC) [15]. Using a rate 1/4 FEC code at 400 baud the performance of a DGPS/Radiobeacon link is slightly better than using no FEC and a 50 baud link [11].

1.5 Side Information

FEC algorithms can yield higher performance if they are given additional information about the validity of the data that they are decoding, ie., if the demodulator determines that a bit decision is questionable, that bit is flagged for the FEC decoding algorithm [15]. I have built a radio and demodulator which collect enough information to make decisions about this side information. The design of the radio will be presented in this report.

1.6 Outline of Thesis

This thesis is organized onto three major chapters, two minor chapters, and some appendices.

Minor chapters. Chapter one introduces this thesis. Chapter five gives numerous suggestions for further work on the subject of extracting side information and concludes the thesis

Major chapters. Chapter two is a description of the radio from the antenna to the input of the MSK demodulator. Chapter three is a discussion of MSK, MSK demodulators in general, and a description of the final MSK demodulator design settled on. Chapter four describes, and presents the results of, several performance tests for the radio and demodulator.

Appendices. Appendix A is a description of an analog implementation of MSK demodulation which

does not work well in environments where the noise level is high enough to require FEC, but works very well at lower noise levels. Appendix B is a listing of the receiver software in Motorola 68HC11 assembly code.

2 The Radio

The basic radio design ahead of the demodulator is a fairly standard double conversion superheterodyne receiver. Figure 3 is a block diagram of the receiver. The antenna is connected to a coupler, which acts as a preamp and an impedance converter, allowing the use of a one-meter whip antenna. The signal from the coupler is fed into the RF filter, which is a five pole Chebychev bandpass filter, designed to pass frequencies in the 275-325 kHz range. The filtered signal is then mixed up to 455 kHz and amplified. This IF signal is filtered by a Collins mechanical filter, which is roughly Gaussian with a 500 Hz bandwidth.

I will start the description of this radio by describing the mixing scheme. Then I will describe the local oscillator (LO). Finally, I will describe the all of the modules along the main signal path: the radio frequency (RF) filter and the mixer, the intermediate (IF) preamp, the IF filter, the IF amplifier, the second mixer and second LO, the audio amplifier and AGC amplifier, and the microprocessor.

2.1 The Mixing Scheme

The mixing scheme is based on the following criteria: ready availability of IF components, ease of design of the RF bandpass filter, ease of design of the local oscillator, and relative freedom from harmonic mixing.

455kHz is a very common IF frequency, and it is easy to obtain IF filters which operate at this frequency. As discussed below, an IF frequency of 455kHz also satisfies the other criteria.

The local oscillator (LO) frequency was chosen to be above the IF and RF frequencies. This causes the LO frequency to be the sum of the desired receive frequency and the IF frequency

($f_{lo} = f_{rf} + f_{if}$). The LO range is 730-780kHz.

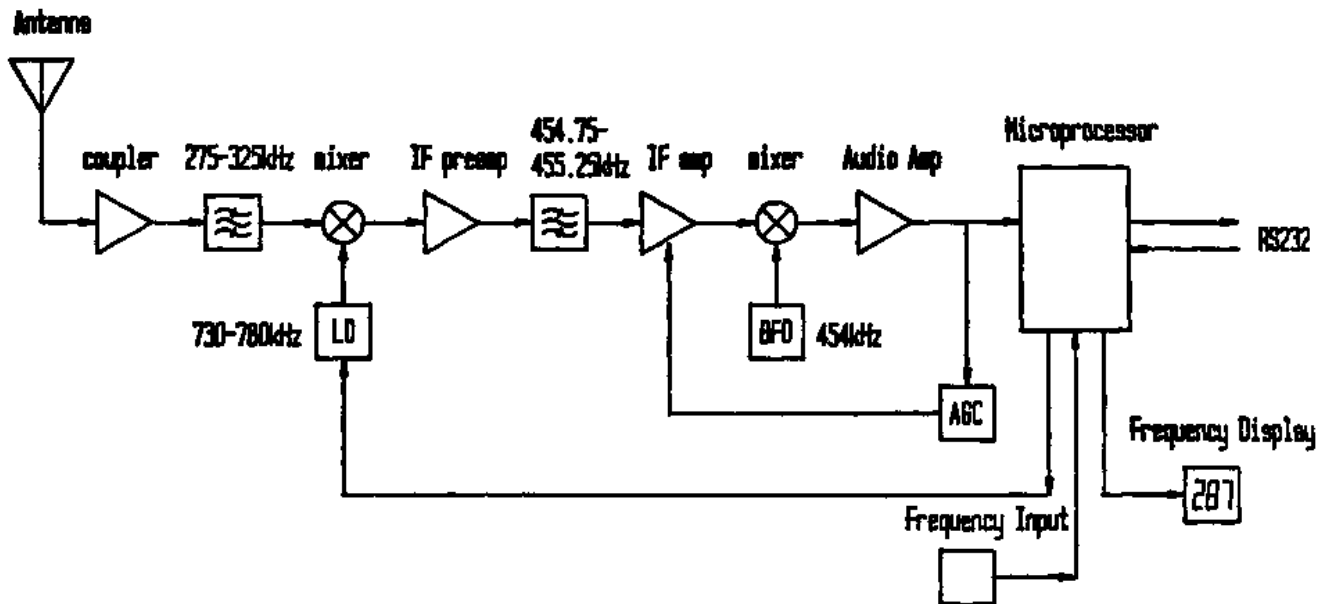


Figure 2: WPI DGPS/Radiobeacon Receiver Block Diagram

Placing the LO above the RF and IF has several advantages. First, the frequency difference between the desired receive band and the band of possible image frequencies is maximized, simplifying the design of the bandpass filter. An image frequency is an undesired RF frequency that can mix to the IF frequency and be received. For example, if the desired receive frequency is 287.5kHz, the LO frequency is

$$f_{lo} = 455\text{kHz} + 287.5\text{kHz} = 742.5\text{kHz}$$

. If there is a signal at the mixer with a frequency of 1197.5kHz, it will be received as well

$1197.5\text{kHz} - 742.5\text{kHz} = 455\text{kHz}$. Second, placing the LO above the IF and RF frequencies minimizes the relative LO range. In this case, the change in LO frequency is only 1.07:1. Finally, the number of possible harmonic mixing combinations is reduced.

Harmonic mixing occurs when harmonics (or the fundamental) of the LO mix with the harmonics or fundamental of some unwanted signal that is being passed by the input filter. For example, if the desired signal were at 302.5kHz, the LO frequency would be 757.5kHz, and the mixer would be converting the 302.5kHz signal to 455kHz ($757.5\text{kHz} - 302.5\text{kHz} = 455\text{kHz}$). However, if the radio is receiving a strong signal at 303.125kHz the fourth harmonic of that signal could be generated in the mixer, mix with the LO fundamental, and be received as a valid signal

$$4 * 303.125\text{kHz} - 757.5\text{kHz} = 455\text{kHz} .$$

| | | LO Harmonics | | | |
|--------------|---|--------------|---------|-----------|-----------|
| | | 0 | 1 | 2 | 3 |
| RF Harmonics | 0 | 0 | 725-785 | 1450-1570 | 2175-2355 |
| | 1 | 250-350 | 375-585 | 1100-1320 | 1825-2105 |
| | 2 | 500-700 | 25-285 | 750-1070 | 1475-1855 |
| | 3 | 750-1050 | 0-325 | 400-820 | 1125-1605 |
| | 4 | 1000-1400 | 215-675 | 50-570 | 775-1355 |
| | | | | | |

Table 1: Table of possible harmonic-mixing combinations. Each entry shows the range of possible mixer outputs for LO and RF ranges. Any range that spans 455kHz (boldface) can be a source of harmonic mixing.

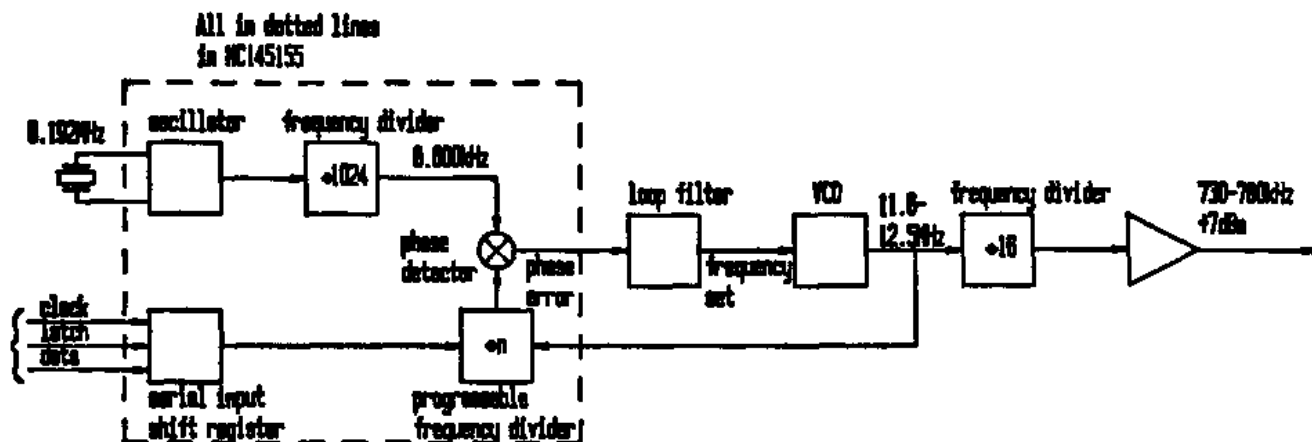
Table 1 shows the harmonic mixing combinations possible with the mixing scheme used. It can be seen from Table 1 that the lowest-order combination possible is the fundamental of the LO mixing with the fourth harmonic of the incoming RF signal. With the mixer I am using (indeed with most mixers) the most severe harmonic mixing problems involve the fundamental of the signal and harmonics of the LO. Because the LO frequency is above both the IF and RF range, it is impossible to get such a combination with this radio.

There is a second mixing operation in this radio. After the IF is filtered and amplified, it is converted down to audio for demodulation. The IF filter passes frequencies of approximately 500Hz from the IF center frequency of 455kHz. The lowest possible audio frequency that will have the same passband as the IF is 1kHz. Therefore the IF is mixed down to 1kHz with a 454kHz beat frequency oscillator.

The individual blocks in Figure 2.1 are now described in greater detail.

2.2 The Local Oscillator

Figure 2 shows a block diagram of the synthesized local oscillator (LO). In the interests of rapid synthesizer lock and low phase noise, I decided to run the VCO at a high frequency and divide it down to the desired LO frequency.

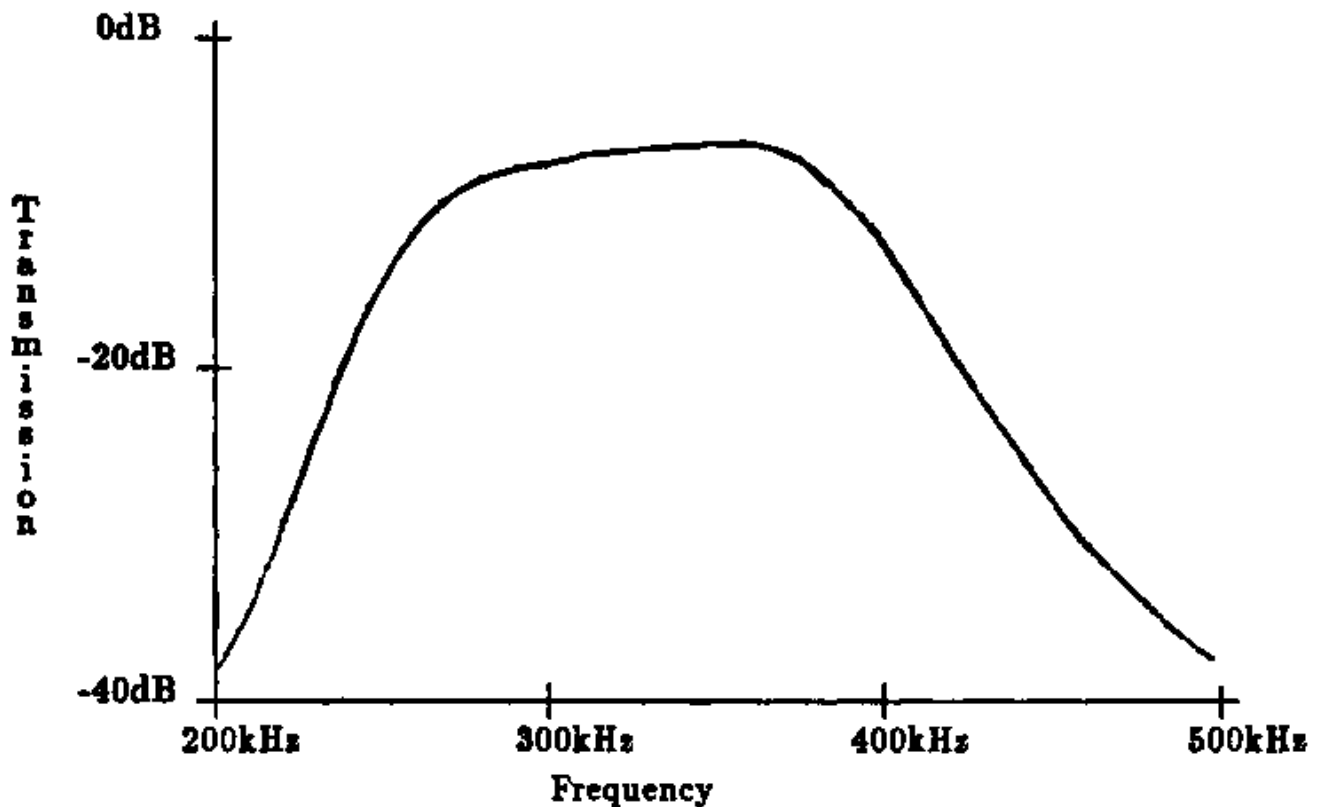


The LO must produce frequencies in the range of 730-780kHz in 500Hz steps, with phase modulation 60dB down from the carrier. If these frequencies were to be synthesized directly, I would need a loop bandwidth of about one radian per second. A loop with such a low bandwidth would have a settling time of about five seconds. Also, because this low bandwidth is achieved by high attenuation in the loop filter, there would be serious stability problems in the oscillator caused by signals bypassing the loop filter and modulating the oscillator directly [8]. If a VCO range of 11.688-12.488MHz in 8kHz steps is used, and the VCO output is divided by 16, the phase modulation criteria can be relaxed by 24dB. This relaxed phase modulation requirement, combined with the higher reference frequency, means that a loop bandwidth of 45 radians per second can be used, for a settling time of about a tenth of a second, and much better oscillator stability.

The core of this synthesizer is a Motorola MC145155 serially programmed PLL frequency synthesis chip [7]. This chip provides a reference oscillator, a pin-programmable frequency divider for the reference oscillator, a serially programmable frequency divider for the output frequency and a phase detector, all on one chip. For this radio, the reference oscillator frequency is 8192kHz, which is divided down by 1024 for a reference frequency of 8.000kHz. The VCO frequency is divided by 1460-1560 then compared to the reference frequency. This results in the VCO output frequency being 11.68-12.48MHz in 8kHz steps.

The output of the VCO is coupled to a 74HC161 divider where it is divided by 16, to a range of 730-780kHz. It is amplified by a 2N3904 transistor stage to the +7dBm needed by the mixer.

The VCO itself is a grounded gate Colpitts oscillator, using an MPF-102 as its active element. The oscillator is tuned by a pair of MV2115 varactors in parallel with a 47pF capacitor and a $1.4 \mu H$ inductor.



2.3 Front end Circuitry

Because the ambient atmospheric noise level is quite high at 300 kHz, I paid much more attention in my design to making a radio resistant to overload, rather than making a radio with an incredibly low noise figure. This goal was achieved by minimizing the pre-mixer gain, and using robust designs for the mixer and post-mixer IF preamp.

2.3.1 Antenna Coupler

A resonant antenna for this frequency band would be on the order of one quarter of a kilometer long. Such an antenna would be inconvenient, even on a large ship. Standard practice for radios of this type is to use a one meter whip antenna and an active antenna coupler and amplifier that provides an impedance transformation between the whip and the 50 input to the RF filter.

2.3.2 RF filter, Mixer

The RF filter is a fifth order Chebychev, with a bandwidth of 50 kHz and center frequency of 300 kHz. It has an overall loss of 7dB. Figure 4 shows the measured shape of the filter passband.

The mixer is a Mini-Circuits GRA-6 diode-ring mixer. It was chosen for its resistance to overload and its ease of use. This mixer requires an LO drive of +7dBm (5mW) and must be terminated with

$50\ \Omega$ loads at both the IF and RF ports. If the mixer is driven and terminated properly, it should have a 3rd order intermodulation input intercept of +11dBm¹.

2.3.3 IF preamp

A preamp is inserted between the mixer and the IF filter. This filter solves a variety of problems that would crop up if a passive impedance matching network were attempted. In order to obtain the best mixer performance, the mixer needs to be terminated by a $50\ \Omega$ load. The IF filter has an input impedance that varies wildly with frequency around the IF center frequency, and it must be terminated at each end with a $500\ \Omega$ resistance in parallel with a 30pF capacitor. Also, the IF filter has about 12dB of attenuation at its center frequency. Furthermore, the match between IF filter and the first IF amplifier was accomplished by a transformer and a ballast resistor, which adds another 2dB of attenuation. Any passive network between the mixer and the IF filter would either have a high amount of loss or would have the same wild impedance variations as the IF filter.

The rated noise figure at the input to the IF amplifier is 6dB [14]. Therefore, the noise figure at the input to the IF filter is about 20dB. In order for the noise figure of the IF preamp to dominate, it had to have a power gain in excess of 20dB. A gain of 24dB was chosen.

The mixer has a 3rd order intermodulation output intercept of +4dBm. Therefore, if the 3rd order intermodulation performance of the radio is to avoid degradation, the IF preamp must have an input intercept that is better than +4dBm. With a gain of 24dB, the output intercept should be +28dBm.

The IF preamp circuit uses heavy feedback both to achieve this intermodulation performance without using too much standing current in the transistor and to provide impedance matching to the mixer and IF filter.

2.4 IF Filter

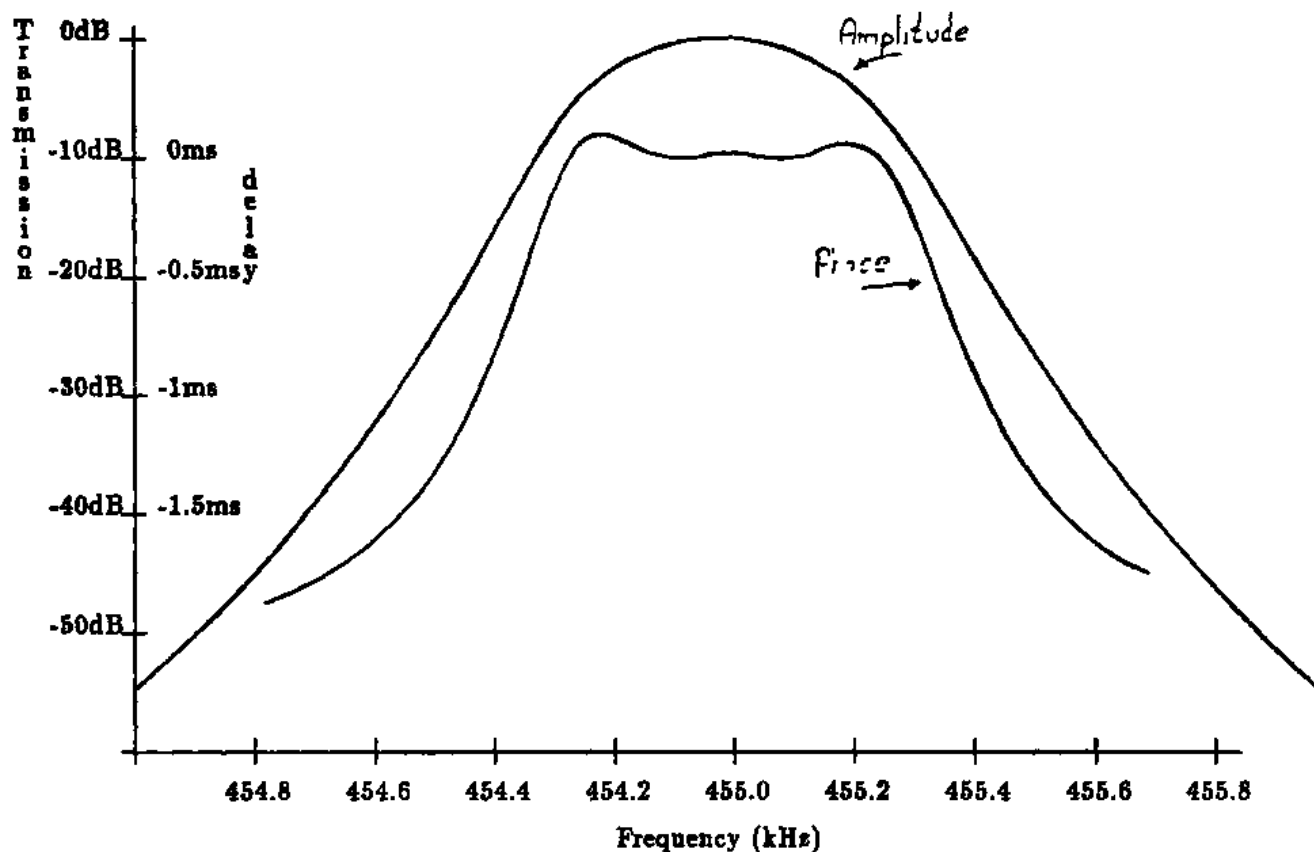


Figure 5: Amplitude and phase characteristics of the Rockwell-Collins 526-8627-010 mechanical IF filter

The IF filter sets the bandpass characteristic of the entire radio. Figure 5 shows the passband characteristics of the IF filter. The filter is a four-pole mechanical filter, Rockwell-Collins part number 526-8627-010. It was made specifically for the conditions encountered in radiobeacon DGPS reception. In particular, it was designed to pass a 400 baud MSK signal with minimal intersymbol interference, while blocking the carrier and ID signal expected to be 500Hz above and below the center frequency of the data stream². It has a 6dB bandwidth of 500Hz, a 100us phase delay bandwidth of 500Hz, and a 60dB bandwidth of 2kHz. At 500Hz away from the center frequency (where one expects to find the radiobeacon carrier or ID signal) it has an attenuation of 25dB. All of the specifications of the filter could be improved with a six or eight pole filter but this filter is adequate for the task.

2.5 IF Amplifier

The radio section was designed for just enough amplification to bring a 5uVrms signal at the antenna up to 1Vrms at the output of the audio amplifier. Because of the other amplifiers in the chain, this works out to an IF amplification of 76dB. I also wanted about 90dB of automatic gain control (AGC) range, to give me AGC action beyond the dynamic range of the front end.

The IF amplifier is a pair of Motorola MC1350 IF amplifier ICs. These provide up to 50dB of gain and 56dB of AGC range each. To get maximum gain, I would have had to use transformer or inductive coupling. Since I did not need this much gain, the amplifiers are resistively coupled. In the end, the IF amplifier chain has about 88dB of gain and over 100dB of AGC action.

2.6 2nd Mixer and 2nd Local Oscillator

A demodulator which works at audio frequencies is easier to build than one which works at a 455kHz center frequency. Therefore, the 455kHz IF signal is mixed down to a 1kHz center frequency audio signal.

The mixer used is a Signetics NE602 monolithic mixer chip. This chip has input and output impedances of $50\ \Omega$ and $1500\ \Omega$, exhibits 18dB of mixer gain, and will provide a 100mV undistorted output. The oscillator in the NE602 is not used. Rather, a CD4060 oscillator/divider and a 7.264MHz crystal are set up to provide a stable 454kHz signal. The output of the CD4060 is attenuated to provide the 200mV p-p signal required by the NE602.

With high IF inputs the NE602 blocks, to the extent that for a large input it can have almost no output at all. This can allow the AGC loop to enter a positive feedback state where increasing IP gain makes the NE602 block and reduce its output, causing the AGC circuit to increase the IF gain, etc. To work around this problem, a pair of diodes were put in parallel across the IF inputs to the mixer, which limits the input amplitude to .6Vp-p. With the input to the mixer limited in this manner, it always has a high enough output to actuate the AGC circuit.

2.7 Audio Amplifier and AGC Amplifier

The audio amplifier is a straightforward differential amplifier that provides a voltage gain of 20dB. 10nF capacitors are used to filter out the high frequency components of the mixer output.

The AGC is designed to keep the signal to the demodulator fairly constant while being insensitive to the large amplitude small duration excursions in the signal envelope that result from lightning noise. The audio peaks are rectified, and the resulting DC is integrated to derive the AGC. The AGC is designed for a time constant of about 200ms.

2.8 Microprocessor

The radio is entirely microprocessor controlled. This makes it easy to interface a panel control, digital display, and frequency synthesized LO. Microprocessor control also allows for easy modification to the operation of the radio.

2.8.1 The MC68HC11

The Motorola MC68HC11 microcontroller was chosen as the radio CPU for a variety of reasons. It has all the features one would expect in a microcontroller, including the ability to use onboard program ROM, onboard RAM, and comprehensive I/O capabilities. More importantly, it has hardware multiply, various timers that can be configured as timed interrupts or as individual vectored hardware interrupts, and an on board sample and hold A/D converter with four or eight inputs.

As we shall see in later chapters, this processor is fine for my original concept, but it is barely fast enough to run an all software demodulator.

3 The Demodulators

Two different demodulators were built for this radio. One uses the microcontroller to sample the MSK at audio, and has the entire demodulator implemented in software. It was designed for very high frequency stability. Its design will be discussed in this chapter. The other is a purely analog design. It runs at a center frequency of 1kHz, and uses CD4046 phase lock loop ICs as its basic frequency reference element. It works well for low to medium noise levels, but cannot maintain lock for high noise levels. Because the analog design does perform well at bit error rates below 10^{-3} it is sufficient for non FEC use, so its design is presented in Appendix A.

This chapter is organized as follows. Section 3.1 is an overview of MSK. Once MSK has been explained, a theoretical model for a MSK demodulator is presented. Thus, section 3.2 covers the theoretical model for a MSK detector, assuming that the proper synchronization signals are available, and Section 3.3 covers the theoretical model for the synchronization signal recovery. After the theoretical model is presented (and the reader has an idea of how a MSK demodulator is supposed to work), a microprocessor based demodulator design is presented. Section 3.4 presents the operations performed by the microprocessor based synchronization recovery algorithm and goes in depth into the design of the phase-locked loops in the synchronization recovery algorithm. Finally, section 3.5 presents the operations performed by the microprocessor based detector algorithm.

1.7 MSK

1 1 0 1 0 0 0 0 1 1 (a)

0 0 1 1 1 0 0 0 1 0 (b)

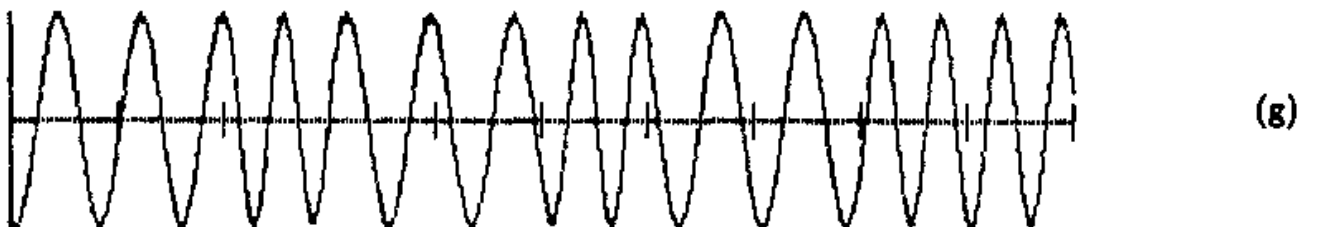
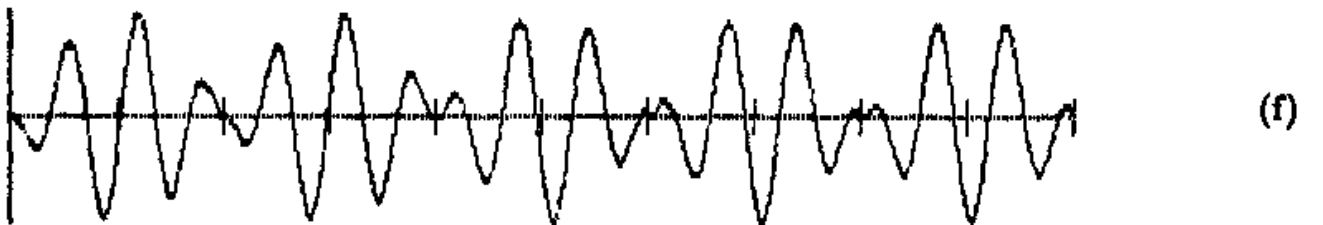
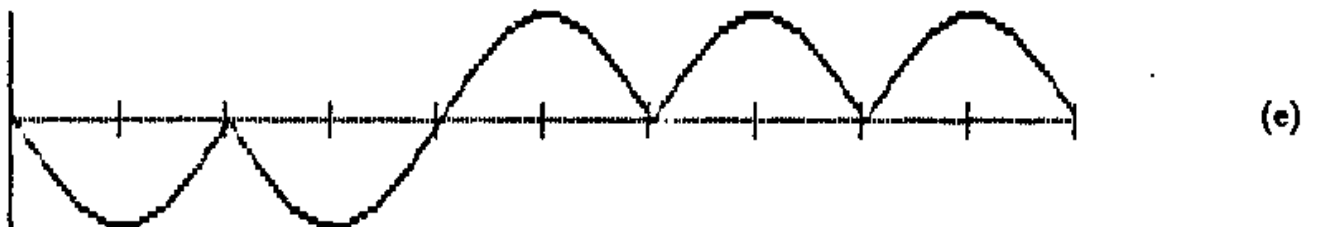
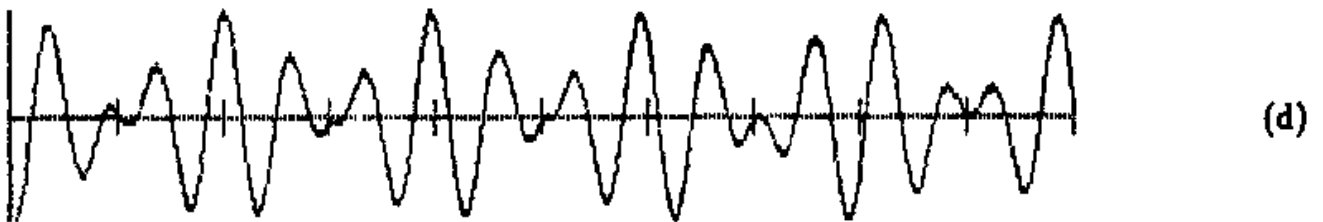


Figure 6: Various waveforms that are part of MSK demodulation

This radio is designed to receive MSK signals. In designing an optimum demodulator for MSK it is best to think of MSK as offset quadrature PSK (OQPSK) with sinusoidal pulse weighting [2]. If

$a_i(t)$ and $a_q(t)$ are the bits transmitted by the inphase and quadrature components of the OQPSK signal, then an MSK signal would be equal to

$$a(t) = a_i(t)\cos(\pi t/2T_b)\cos 2\pi ft + a_q(t)\sin(\pi t/2T_b)\sin 2\pi ft \quad (1)$$

where T_b is one bit interval. Figure 11 shows the $a_i(t)$, $a_q(t)$ and the various components of the MSK signal defined by (3.1). Figure 11(a) shows an example bit stream to be sent.

Like all types of phase modulation, MSK suffers from phase ambiguity at the demodulator. The cure for this is to use differentially encode the bitstream to be sent. Figure 11(b) shows the bitstream of Figure 11(a) after it has been differentially encoded.

This encoded bitstream must be broken up into two parts before being sent over the inphase and quadrature channels. Figure 11(c) shows the inphase multiplier waveform, with the corresponding multiplier values shown as ± 1 inside the waveform. This encoded waveform is shown multiplied by the inphase carrier to yield the inphase signal in Figure 11(d) [Figure 11(d) is the first term in (3.1)]. Similarly, the sinusoidally shaped quadrature multiplier and the quadrature signal are shown in Figure 11(e) and Figure 11(f). The composite signal, as given by (3.1) is shown in Figure 11(g).

Using a trigonometric identity, (3.1) can be rewritten as

$$a(t) = \cos[2\pi ft + b_k(t)\pi t/2T_b + \phi_k] \quad (2)$$

where $b_k(t) = -a_i(t)a_q(t)$ and $\phi_k(t) = (a_i(t)+1)\pi/2$. Figure 11 and equation (3.2) show that MSK is an FSK signal with signaling frequencies $f_m = f + 1/4T_b$ and $f_a = f - 1/4T_b$. In addition, a one in our original bitstream corresponds to a high frequency (mark) tone, and a zero in the original bitstream corresponds to a low frequency (space) tone. Indeed (as shall be described later) if one has an easy method to produce phase-coherent FSK with the proper deviation ratio, it is simple indeed to produce MSK.

3.1.1 MSK Detection

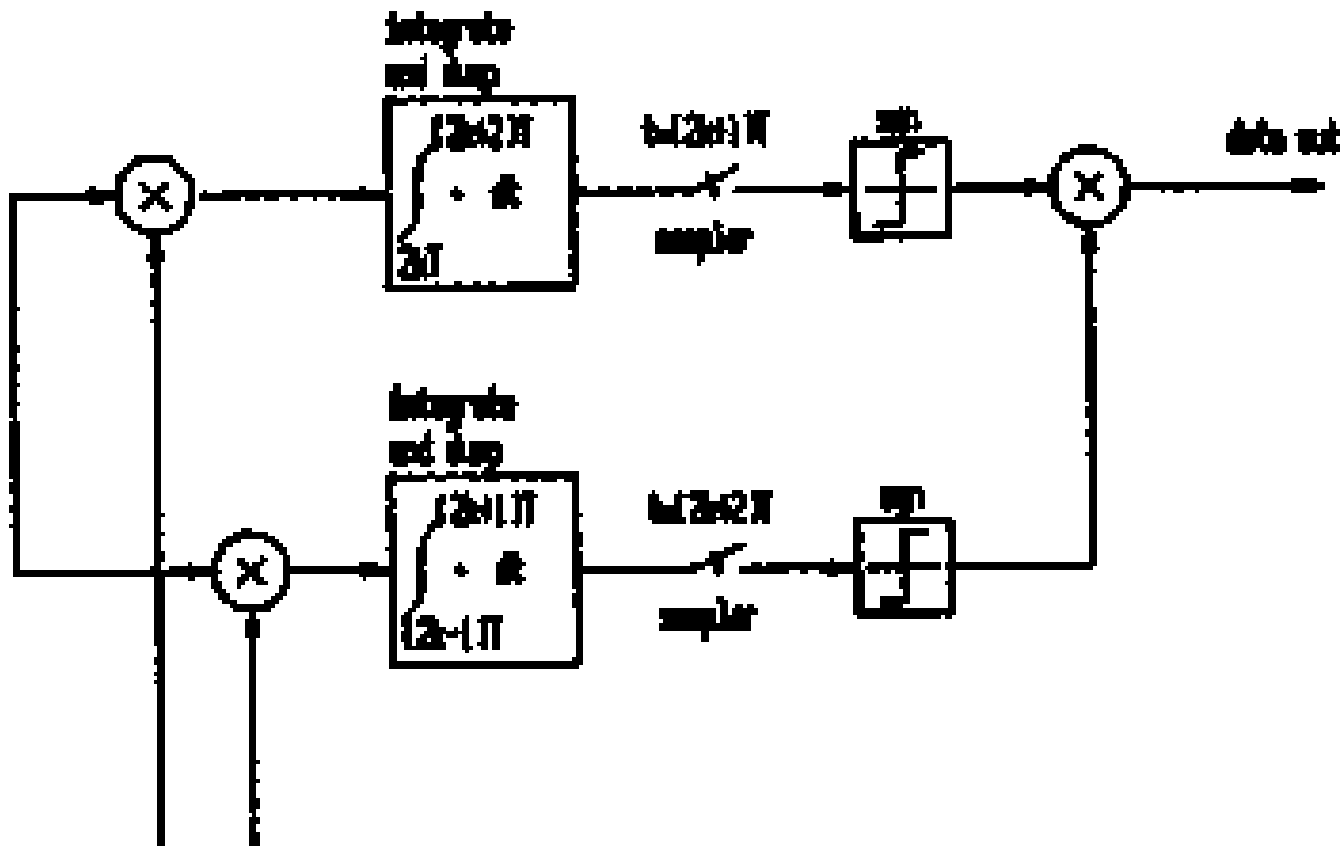


Figure 7: Basic MSK Correlation Detector. This part of the MSK demodulator implements the multiply and integrate-and-dump functions

If one treats MSK as OQPSK it is easy to demodulate. Figure 7 shows a typical MSK demodulator [2]. The incoming signal is multiplied by the inphase and quadrature signals

$$z(t) = 2\cos(\pi t/2T_b)\cos 2\pi ft$$

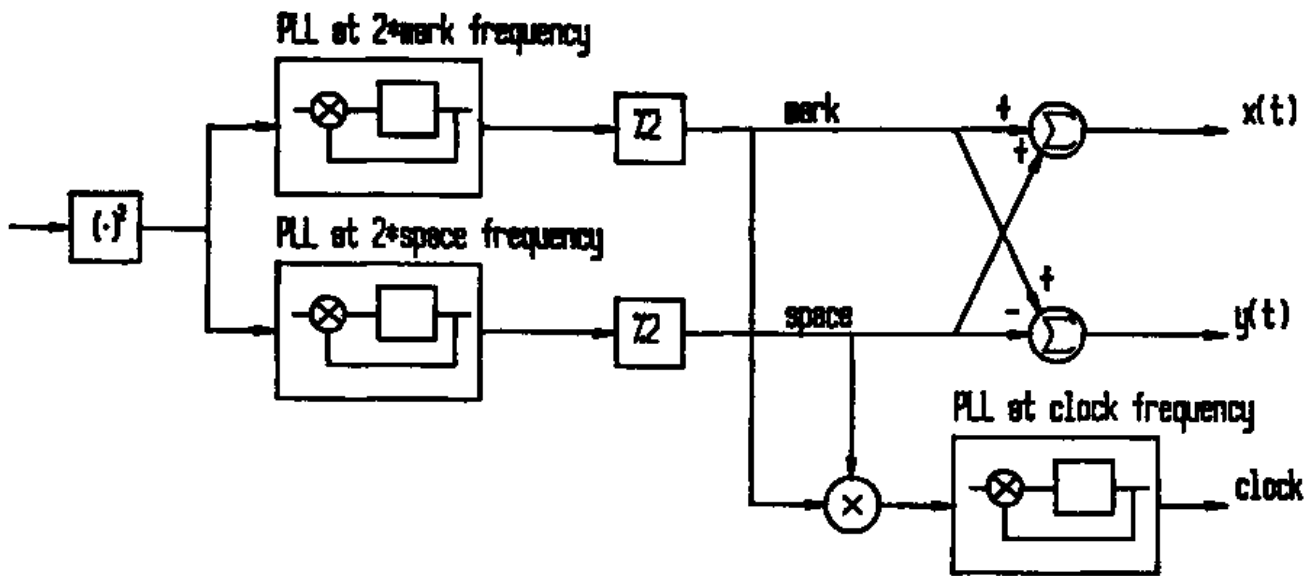
and

$$y(t) = 2\sin(\pi t/2T_b)\sin 2\pi ft$$

(the multiplication by two will become clear later) followed by integrate and dump circuits. This multiply-integrate operation constitutes correlation detection or matched filtering, which is an optimum detection operation for a signal in white Gaussian noise with no intersymbol interference.

Note that this detector is not an optimum detector for non-Gaussian noise. However, interference from the radiobeacon carrier and identification signal (which occurs at approximately ± 500 Hz) makes wideband signal processing of a 400 baud signal virtually impossible. This detector is near optimum for the signal conditions encountered. If a lower baud rate were to be used with this radio, other, more nearly optimum (wide band) detection schemes for non-Gaussian noise should be investigated (10). These schemes would include hard limiters, clippers, blankers, or some other sort of zero memory non-linearity to suppress the impulse noise.

3.1.2 Synchronization



With almost any coherent demodulator, the correlation defector does not require the major effort, rather, the difficulty lies in achieving synchronization with the incoming signal. MSK demodulation is no different. Figure 8 shows an example of a MSK synchronous recovery circuit [2]. If MSK is passed through a squarer, it produces strong discrete spectral components at twice the mark and space frequencies [2]. This becomes apparent if we square (3.2).

$$\begin{aligned}
 s^2(t) &= \cos^2[2\pi ft + b_h(t)\pi t/2T_b + \phi_k] \\
 &= (1 + \cos[4\pi ft + b_k(t)\pi t/T_b + 2\phi_k])
 \end{aligned}
 \tag{3}$$

Notice that the $2\phi_k$ term becomes either 0 or 2π , so it drops out. What we are left with is a pair of tones with a deviation ratio of $1/T_b$ (known as Sunde's FSK), with the phase for each tone a constant. This means that we can lock onto each of these tones with a phase-locked loop, divide each output frequency by two, and produce the pair of reference signals:

$$s_1(t) = \cos(2\pi ft + \pi t/2T_b) \tag{4}$$

$$s_2(t) = \cos(2\pi ft - \pi t/2T_b) \tag{5}$$

The sum and difference $s_1 + s_2$ $s_1 - s_2$ produce the reference carriers $x(t)$ and $y(t)$ shown in Figure 8. This can be seen by applying trigonometric identities to (3.4) and (3.5):

$$\begin{aligned}
 x(t) &= 2\cos(\pi t/2T_b)\cos 2\pi ft \\
 &= \cos(2\pi ft + \pi t/2T_b) + \cos(2\pi ft - \pi t/2T_b) \\
 &= s_1 + s_2
 \end{aligned}
 \tag{6}$$

and

$$\begin{aligned}
 y(t) &= 2\sin(\pi t/2T_b)\sin 2\pi ft \\
 &= \cos(2\pi ft + \pi t/2T_b) - \cos(2\pi ft - \pi t/2T_b) \\
 &= s_1 - s_2
 \end{aligned} \tag{7}$$

For a clock reference, s_1 and s_2 can be multiplied:

$$\begin{aligned}
 s_1(t)s_2(t) &= \cos(2\pi ft + \pi t/2T_b)\cos(2\pi ft - \pi t/2T_b) \\
 &= (\cos 4\pi ft + \cos \pi t/T_b)/2
 \end{aligned} \tag{8}$$

When this signal is locked onto with a phase-locked loop designed to run at a frequency of $1/2T_b$, the output will be proportional to $\sin \pi t/T_b$ which is the desired timing waveform.

3.2 Microprocessor based demodulator

The discrete-time microprocessor based demodulator offers numerous advantages over the continuous-time demodulator. Many of the shortcomings of the continuous-time demodulator could be overcome by careful design and more expensive components, but using a microprocessor based demodulator cures these shortcomings while adding tremendous versatility.

This demodulator is a fairly faithful rendering of the functions delivered by the basic demodulator given in [2] and described at the beginning of this chapter. The block diagrams that describe this demodulator are largely rearrangements of the block diagrams at the start of this chapter (Figure 7 and Figure 8 — basic carrier recovery and basic coherent detector).

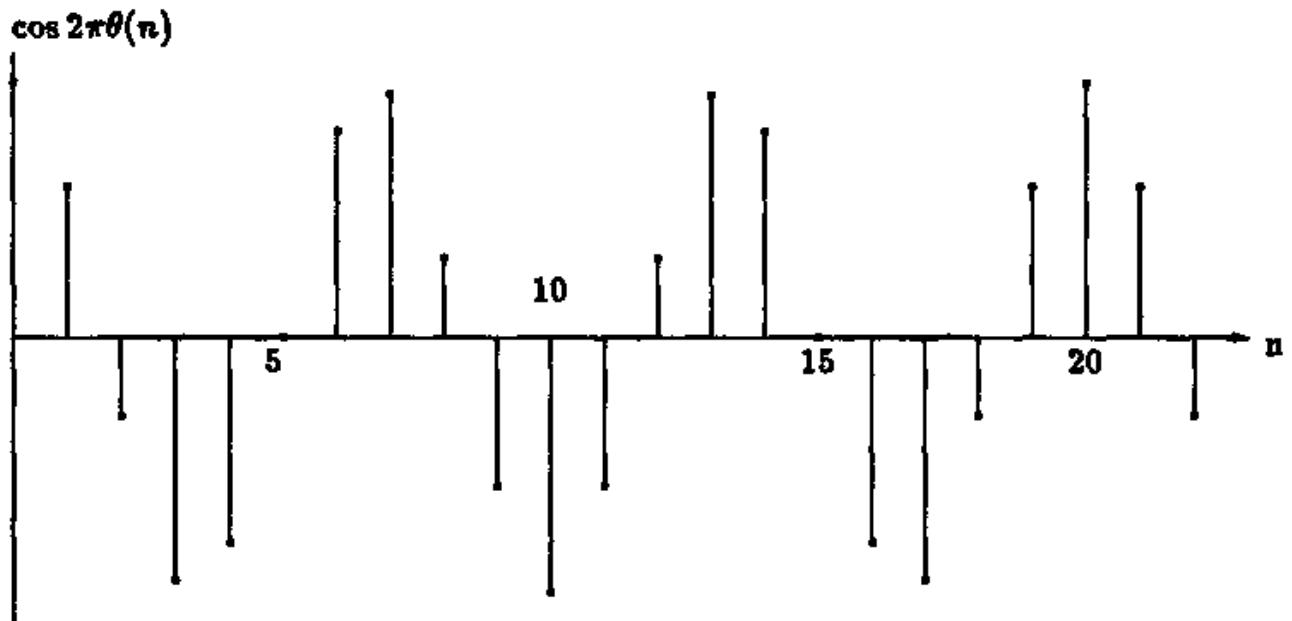
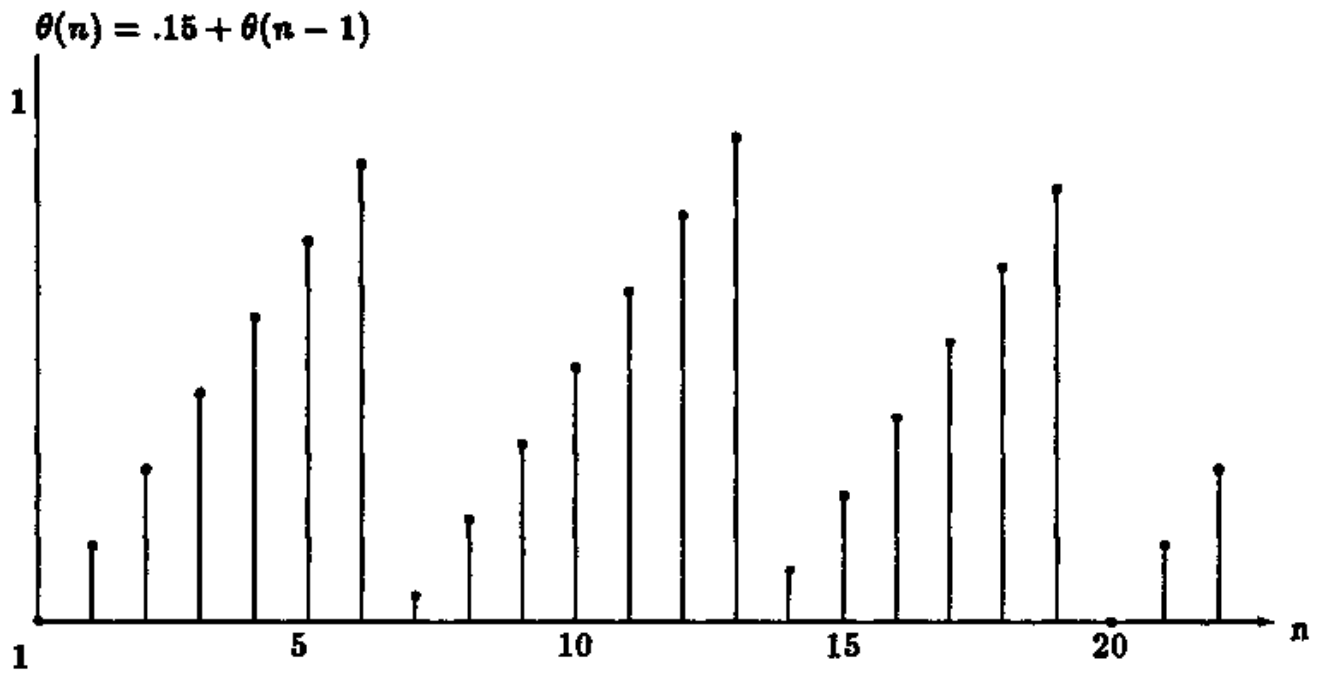
A complete listing of the receiver software, including the demodulator routine, is contained in Appendix B.

Frequency stability was the original motivation for going to a microprocessor based demodulator. The low frequency stability available from the CD4046 phase-locked loop chips required such high loop bandwidths that the loops could not maintain lock in the presence of high levels of noise. A crystal based frequency generation scheme would provide the necessary frequency stability. Since a microprocessor based approach offers versatility as well as frequency stability, that approach was chosen. The microprocessor samples data at intervals which are determined by the on board timer. The timer interval is derived from the crystal controlled microprocessor clock, which gives us the required crystal frequency stability.

The demodulator algorithm is implemented as an interrupt routine in the MC68HC11 microcontroller. The routine is called 15 times per bit interval (once every $166.5 \mu s$, about six times the audio center frequency) under the control of an output timer on board the microcontroller. It reads the last A/D conversion value, then starts a new conversion.

3.3 The Timing Recovery Algorithm

Since the main motivation for this demodulator was phase stability, I will start my description of the demodulator by describing the phase and bit timing recovery phase-locked loops.



Rather than using oscillators to generate my reference signals, a direct digital synthesis – variable phase ramp approach is used. Figure 9 gives an example of direct digital synthesis. In this type of frequency synthesis, one establishes a phase accumulator with a range of 0 - 1. When the accumulator overflows,

the phase value just wraps around. If one wants a sinusoidal waveform from the synthesizer, one takes the phase ramp from the accumulator and calculates the sine of the phase.

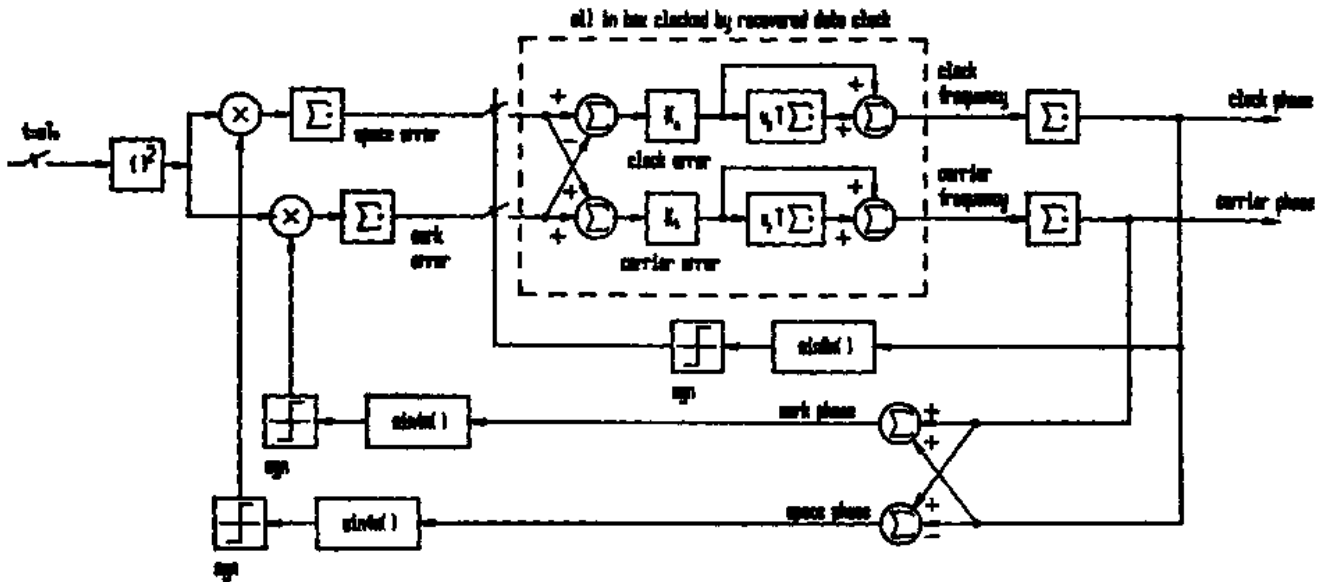


Figure 10: WPI DGPS/Radiobeacon microprocessor based synchronization recovery. This is a block diagram of the operations performed by the synchronization algorithm* Note that the signal is sampled at approximately 6k Hz, but that the phase lock loop calculations are performed at the baud rate. This was done to save processing time

Figure 10 is a block diagram of the microprocessor based synchronization recovery algorithm. Notice that this algorithm does not lock to the mark and space frequencies directly. Rather, it translates the mark and space phase errors into clock and carrier phase errors, then locks the clock and carrier to the incoming signal. This was done for two reasons: One, the algorithm is timed by the transitions of the bit clock, so we want to have the bit clock available directly. Two, this allows the option of having different bandwidths for the clock and carrier, which is nice because the clock can be depended on to be much more stable than the carrier.

Because I am using a relatively slow processor, I had to pay quite a bit of attention to writing time-efficient code. Most of the effort involved reducing the amount of processing done during each sample interval. The overall algorithm is separated into two parts. One part is executed every sample interval, and does the minimum amount of processing necessary for one sample interval. This part checks the clock phase before returning. When the clock phase indicates that the end of a bit interval has been reached, the second part of the algorithm is executed. I will indicate in my descriptions of the algorithms which parts are running at the sample speed and which parts are running at the bit clock speed.

I will describe the phase recovery algorithm starting at the output of the phase-lock loop filters, then trace the signal around and finally describe the phase-lock loop filters. I will be using the notation developed in [8] to describe the various parts of the loop.

3.3.1 Phase error estimation

At each sample interval, the clock summer adds the clock increment (proportional to the clock frequency) to the clock phase. Since the clock frequency is approximately 100Hz (one quarter of the bit rate), and the sample interval is $166.5 \mu s$, this increment is

$$f_{cl} = T_s(100\text{Hz}) = (166.5 \mu s)(100\text{Hz}) = .01665$$

where $T_s = 166.5 \mu s$ is the sample interval. The carrier summer works the same, except the increment

$$f_{ca} = 1000\text{Hz}(166.5 \mu s) = .1165$$

The clock and carrier phase become:

$$\theta_{cl}(m) = [\theta_{cl}(m-1) + f_{cl}(m)] \text{ mod } 1 \quad (9)$$

$$\theta_{ca}(m) = [\theta_{ca}(m-1) + f_{ca}(m)] \text{ mod } 1 \quad (10)$$

The mark and space frequencies are symmetrical about the carrier frequency and differ by one half the baud rate. Since the clock frequency is one quarter the bit rate, the mark and space phase can be derived from the clock and carrier phase:

$$\theta_m(m) = [\theta_{ca}(m) + \theta_{cl}(m)] \text{ mod } 1 \quad (11)$$

$$\theta_s(m) = [\theta_{ca}(m) - \theta_{cl}(m)] \text{ mod } 1 \quad (12)$$

The incoming signal is sampled at approximately 6kHz. For timing recovery, this signal is then squared. The mark error is calculated by multiplying the squared signal by a square wave at twice the mark frequency and the result added to a variable for the mark error. The space error is calculated the same way using a square wave at twice the space frequency. Overall, the summer between the phase detector and the loop filters has an effect on noise and stability similar to that of a low pass filter in a continuous-time phase locked loop. As long as the bandwidth of the phase recovery loops are kept fairly low (which is what we want to do anyway) the effect of this summation step on loop stability can be neglected.

In order to execute this multiply and add operation in the minimum of processor time the processor checks the phase, and if it is in the interval $[1/4, 1/2)$ or the interval $[3/4, 1)$ (ie. if the second most significant bit is one) the signal is subtracted from the error variable, otherwise the signal is added to the error variable. This has the effect of a square wave multiply and add, but is much faster for this processor than using the multiply instruction. Ideally, the processor would look up the sine of that phase and do a full multiply and add, but this processor is not fast enough.

3.3.2 The gain blocks

At the end of a bit interval, when the algorithm has collected phase error information, the algorithm goes into the phase-locked loop filter calculations. The clock phase error is calculated from the difference of the mark and space phase errors (the carrier error is calculated from the sum). Because the clock and carrier loops are identical except for the center frequencies and loop bandwidths, I will only describe the clock loop filters. However, I will note any differences between the clock and carrier loops.

If we look at the squarer-multiplier-summer combination as a phase detector, we can calculate the phase detector gain. The peak-peak voltage of the signal present at the input of the A/D converter is 1Vp-p, and its DC average is 2.5V. Eq. (3.2) can be modified to show the signal received at the A/D converter;

$$\begin{aligned} r(t) &= (.5 V) \cos[2 \pi f t + b_h(t) \pi t / 2 T_b + \phi_h] + 2.5 V, \\ &= (.5 V) \cos \theta_r(t) + 2.5 V \end{aligned} \quad (13)$$

where $\theta_r(t) = 2 \pi f t + b_h(t) \pi t / 2 T_b + \phi_h$. In discrete time, $r_d(m) = (.5 V) \cos \theta_r(m T_s)$ where the variable m represents one sample interval.

This r_d gets sampled and converted by the microprocessor, and turned into a 2's complement fraction;

$$r(m) = (.1) \cos \theta_r(m T_s) \quad (14)$$

This input signal gets squared, to yield the synchronization signal

$$\begin{aligned} r^2(m) &= (.01) \cos^2 \theta_r(m T_s) \\ &= (.01) (1 + \cos[4 \pi f m T_s + b_h(t) \pi m T_s / t_b + 2 \phi_h]) \end{aligned} \quad (15)$$

The mark and space references can be obtained by locking onto this signal with a pair of phase-locked loops at twice the mark and space frequencies, as is done in the theoretical model. Alternately, the loops can be made to lock onto the clock and carrier phases and the mark and space phases can be calculated from them. This algorithm follows the latter approach.

If we have estimates of the mark and space phase errors we can compute the clock and carrier errors. We define the mark and space errors at the start of a bit interval as $\theta_{sm}(n)$ and $\theta_{es}(n)$, respectively (note the use of n as opposed to m to denote that the interval is a bit interval rather than a sample interval). The clock phase is one half the difference between the mark and space phases, while the carrier phase is one half the sum of the mark and space phases. In other words,

$$\theta_{ecl}(n) = [\theta_{em}(n) - \theta_{es}(n)] / 2 \quad (16)$$

(clock phase error)

$$\theta_{eca}(n) = [\theta_{em}(n) + \theta_{es}(n)] / 2 \quad (17)$$

(carrier phase error).

The synchronization signal in (3.15) can be used to get estimates of the mark and space phase errors at every bit interval. In the case of the mark phase, the synchronization signal is multiplied by the sine of twice the mark phase:

$$\begin{aligned} E_m(m) &= r^2(m) \sin 4 \pi \beta_m(m) \\ &= (.005) [1 + \cos 2 \beta_r(m)] \sin 4 \pi \beta_m(m) \\ &= (.005) [\sin 4 \pi \beta_m(m) + \cos 2 \beta_r(m) \sin 4 \pi \beta_m(m)] \end{aligned} \quad (18)$$

This signal has components at either the clock frequency or approximately DC, twice the mark frequency, and at approximately four times the mark frequency. Because the phase locked loops are essentially low pass filters, and because the loop filters are preceded by the summation operation, we can ignore all but the DC component. If a mark is transmitted, we can assume that

$\theta_r(m) - 2\pi\theta_m(m)$ remains constant over the bit interval such that

$$\theta_r(m) - 2\pi\theta_m(m) = \theta_{em}(n)$$

and there will be a DC component to $\hat{e}_m(m)$:

$$e_m(m) = (.0025)\sin 2[\theta_{em}(n)]$$

. If a space is transmitted, there will be no DC component, and we can say that

$$e_m(m) = 0$$

The error signal $\hat{e}_m(m)$ is summed over one bit period (15 samples). After this summation, we can assume that all of the AC components of $\hat{e}_m(m)$ have been filtered out, leaving only

$$e_{dm}(n) = \sum_{m=15n}^{15(n+1)} e_m(m)$$

(because of the low-pass nature of phase-lock loops, any AC that gets passed at this stage will be filtered out later). Note that at the end of a bit period, $e_{dm}(n)$ has one of two different expected values depending on the bit that was transmitted. If we assume that there is equal probability of a mark or a space being sent, the overall expected value is

$$\begin{aligned} E\{e_{dm}(n)\} &= (.5)E\{e_{dm}(n)|\text{mark transmitted}\} + (.5)E\{e_{dm}(n)|\text{space transmitted}\} \\ &= (.5)(15)(.0025)\sin 2[\theta_{em}(n)] + 0 \\ &= (.01875)(\sin 2[\theta_{em}(n)]) \end{aligned}$$

The forgoing analysis would be identical for the space error, so I won't repeat it here, other than to note that I will use $e_{ds}(n)$ for my space error variable.

In this algorithm, these mark and space error variables are used to obtain clock and carrier error variables. These are calculated as

$$e_{cl}(n) = e_{dm}(n) - e_{ds}(n)$$

for the clock error, and

$$e_{ca}(n) = e_{dm}(n) + e_{ds}(n)$$

for the carrier error.

If we make the approximation that the sine of a small angle is equal to that angle in radians, then we can find the expected value of the dock error estimate for small mark and space phase errors.

$$\begin{aligned} E\{e_{el}(n)\} &= E\{e_{dm}(n) - e_{ds}(n)\} \\ &= (.1875)(\theta_{em}(n) - \theta_{es}(n)) \\ &= (.0375)\theta_{ecl}(n) \end{aligned}$$

Therefore, the phase detector gain is

$$K_d = \frac{E\{e_{cl}(n)\}}{\theta_{ecl}(n)} = .0375 \text{ rad}^{-1}$$

The foregoing analysis of the clock loop phase detector gain is exactly the same as the analysis of the carrier loop detector gain. Hence, in my analysis of the carrier loop, I will use eq. (3.23) as the expression for phase detector gain.

The gain of the frequency to phase summer is simply the number of samples per bit times the bit rate;

$$K_o = (15)(400\text{Hz}) = 6000\text{Hz}$$

The other two blocks that contribute phase gain around the loop are the k_{hl} block, which is the block that the designer can adjust to control the loop bandwidth, and the $\sin 4\pi$ block. The $\sin 4\pi$ block contributes a gain of $4\pi \text{ rad / cycle}$.

3.3.3 Loop Filter Calculations and Bandwidth

In a phase locked loop the bandwidth is equal to the gain around the loop, so the bandwidth for this loop is

$$\begin{aligned} K_{cl} &= 4\pi K_d K_{hl} K_o \\ &= (4\pi \text{ rad / cycle})(.0375 \text{ rad}^{-1})(6000\text{Hz})K_{hl} \\ &= (2827 \text{ rad / sec})K_{hl} \end{aligned}$$

. If, for example, $K_{hl} = 2^{-10}$,

$$\begin{aligned} K_{cl} &= (2^{-10})(2827 \text{ rad / sec}) \\ &= 276 \text{ rad / sec} \end{aligned}$$

For stability the loop filter requires a zero at $\omega_2 < K_{cl}/8$. In this case, we need

$$\omega_2 < .35 \text{ rad / sec} \quad [8]. \quad \text{With } \omega_2 = .39 \text{ rad / sec},$$

$$\omega_2 T_b = .00098 = 2^{-10}$$

We can find the RMS phase error for this loop if we know some details about the input noise. For a phase-locked loop with a squarer at the input, the phase error at the output of the loop is related to the loop bandwidth (B_L), input filter noise bandwidth (B_i), and input signal to noise ratio (SNR_i) [8]. If we assume Gaussian noise, this RMS phase error is

$$\theta_c = \sqrt{\left(\frac{B_L}{B_i SNR_i}\right) \left(1 + \frac{1}{2SNR_i}\right)}$$

. If we assume that FEC will allow valid data reception at a BER rate of .01 out of the demodulator, then we must have phase lock loops that can perform at this noise Level. The theoretical signal to noise level that would cause a .01 BER with the IF filter in this radio is 4.6dB, and the IF filter noise bandwidth is 490Hz. Therefore, the RMS phase error at the output of the loop with a bandwidth of

2.76rad/sec would be

$$\theta_a = \sqrt{\frac{.44Hz}{(490Hz)(2.9)} \left(1 + \frac{1}{5.8}\right)}$$

which is more than acceptable.

The carrier phase recovery loop is exactly the same as the clock phase recovery loop, with the exception that the center frequency is 1000Hz rather than 100Hz, and the value of the K_{el} multiplier may be higher, for a higher loop bandwidth. In practice, it seems that values of $K_{hl}=2^{-10}$ to 2^{-6} and $K_{ha}=2^{-6}$ work best (see chapter 4 for test results).

3.3.4 Limit Cycles

The output of the phase detector in this algorithm is a 16 bit fractional number (the value ranges from -.5 to .5, with the smallest possible increment = 2^{-16}). In a digital control system such as this, one must pay attention to the possibility of limit cycles. A limit cycle is a low-level oscillation in a digital control system caused by the finite resolution of the state variables [13, pp345-353]. To avoid limit cycles of magnitude greater than .088 radians (5 degrees), there must be enough bits of precision at the output of the K_{hl} multiplier so that at least its lowest order bit is affected by a .088rad error at the phase detector. Similarly, there must be enough bits of precision at the output of the w_2T multiplier so that its lowest bit is affected by a .088rad error at the phase detector. For a .088rad error the output of the phase detector will be:

$$e_{cl} = (.0375 rad^{-1})(.088 r) = .0033 < 2^{-6}$$

For $K_{ch}=2^{-10}$, the output is less than what can be represented in two bytes;

$$(.0033)(2^{-10}) < 2^{-14}$$

The next larger convenient size for the output of the K_{hl} multiplier is three bytes, which is more than enough precision for our limit cycle criterion. The output of the w_2T summer is around 2^{-10} smaller than the output of the K_{hl} multiplier, so the w_2T summer must have one more byte of precision for a total of four bytes.

Overall, the amount of precision provided is enough for keep the limit cycle amplitude below 4 milliradians, which should be below the expected RMS noise at the output of the loops in all but the quietest of conditions.

The instantaneous frequency is determined by the sum of the outputs of the K_{cl} and w_2T summers in Figure 10. In practice, this instantaneous frequency is calculated then stored. The word size chosen for this is three bytes, to match the K_{cl} summer. Using this size, rather than four bytes, ignores the final byte of the w_2T summer, but I feel it will not introduce any more limit cycle problems.

3.3.5 Lock Acquisition

Phase lock loops have a linear phase detector characteristic is highly non-linear, and the analysis is very involved, so rather than going into it here the reader is referred to [3] and [8]. One result is presented here however. For our carrier loop, if our initial frequency error is w_{ea} and the input noise is negligible, the lock time will be

$$T_{pa} = \frac{(w_{ea}/K_{ca})^2 - 1}{w_2}$$

From this we see that the lock time is proportional to the square of the frequency error over the loop bandwidth. Therefore if we want fast lockup and small loop bandwidth, it is important to make sure that the initial frequencies of the clock and carrier loops will be close to their final values. Also, it is a good idea to insure that if the radio receives only noise inputs for a prolonged period of time the loop frequencies do not wander too far away from the correct values.

Three things have been done to deal with these frequency uncertainty problems. One, when the radio is turned on, the microprocessor initializes the loop frequencies to their nominal values (100Hz for the clock loop, 1000Hz for the carrier loop). Two, after every iteration of the $w_2 T$ summers (one for the carrier, one for the clock) the frequency value is checked against frequency limits ($1000 \pm 10\text{Hz}$ for the carrier, one for the carrier, $100 \pm .5\text{Hz}$ for the clock).and if these limits are exceeded, the frequency is set to just inside the limit. Finally, the bandwidth of the carrier loop is larger than the bandwidth of the clock loop.

3.3.6 Carrier Stability and Carrier Loop Bandwidth

I mentioned in the last section that the carrier frequency is much less certain than the clock frequency. In this section I will give a rough analysis of why this is, and I will demonstrate the effects of changing the loop bandwidth on carrier reference stability and loop acquisition time.

I will assume that the DGPS/Radiobeacon clock and carrier frequencies are derived by frequency division (or synthesis) from a single crystal controlled source. Therefore, both the clock and carrier frequencies of the transmitted signal can be depended on the be accurate to at least $\pm 10\text{ppm}$.

There are some frequency references in the receiver that affect the carrier as well: one, the clock frequency reference at the receiver is synthesized from a single crystal controlled source (the microprocessor crystal), two, the receiver carrier frequency reference is affected by the difference between the LO frequency and the BFO (second LO) frequency, which are crystal controlled by two different crystals. Three, the carrier frequency is also affected by the sampling rate of the microprocessor, which is derived by dividing down the microprocessor clock.

We can assume that the initial frequencies of all of the crystal controlled sources in the receiver are adjusted to $\pm 1\text{ppm}$ to start (this is a calibration which can be carried out by the microprocessor at the time the radio is manufactured). We can also assume that the crystal frequency never varies more than $\pm 10\text{ppm}$ with temperature and that the relative crystal error will approximately track with temperature variations.

Ultimately, the clock frequency only depends on the crystal source at the transmitter and the crystal

source at the receiver. Since both of these sources are accurate to $\pm 10\text{ppm}$, the worst that we can expect of the clock is that it will be $200\text{Hz} \pm 20\text{ppm}$ (remember that the clock loop reference is one half the bit rate). It follows that the maximum clock frequency variation at the demodulator will be:

$$\begin{aligned} e_{fcl} &= (200\text{Hz})(1*10^{-3}) + (200\text{Hz})(1*10^{-3}) \\ &= 42.5\text{rad/sec} \end{aligned}$$

. This is small enough to be ignored.

The carrier frequency can vary much more. If we assume that the transmitter is accurate to $\pm 10\text{ppm}$, and that the LO and BFO frequencies track, the carrier frequency variation at the demodulator will be:

$$\begin{aligned} e_{fca} &= (780\text{kHz} - 454\text{kHz})(1*10^{-5}) + (350\text{kHz})(1*10^{-5}) \\ &= 6.76\text{Hz} \\ &= 42.5\text{rad/sec} \end{aligned}$$

Unlike the possible dock frequency error, this error is quite significant. If we wanted the carrier loop to be able to acquire this frequency instantly, we would need a bandwidth larger than 42.5rad/sec . For a loop bandwidth of exactly 42.5rad/sec and a theoretical BER of .01, the rms phase noise in the loop output is

$$\begin{aligned} \theta_c &= \text{sqrtfrac} 6.76 (490\text{Hz})(2.9)(1 + \text{frac} 15.8) \\ &= .074 \text{ rad} \end{aligned}$$

which is rather high. To reduce this phase noise, we must reduce the bandwidth of the loop, thereby increasing the acquisition time.

If we use a K_{la} of 2^{-8} then the carrier loop bandwidth is $K_{ca} = 11.0\text{rad/sec}$. Using equation (3.27), this gives us an acquisition time of

$$\begin{aligned} T_{pa} &= \frac{\left(\frac{w_{as}}{K_{ca}}\right)^2 - 1}{w_2} \\ &= \frac{\left(\frac{42.5\text{rad/sec}}{11.0\text{rad/sec}}\right)^2 - 1}{.39\text{rad/sec}} \\ &= 36\text{sec} \end{aligned}$$

if we leave the loop zero at $w_2 = .39\text{rad/sec}$. As a worst case, this acquisition time is bearable, although it could be better. In practice, at room temperature the demodulator attains lock immediately with $K_{ca} = 11.0\text{rad/sec}$.

Chapter 4 has some test results for the effect various values of the carrier loop bandwidth have on the error rate performance of the demodulator.

3.4 The Correlation Detector

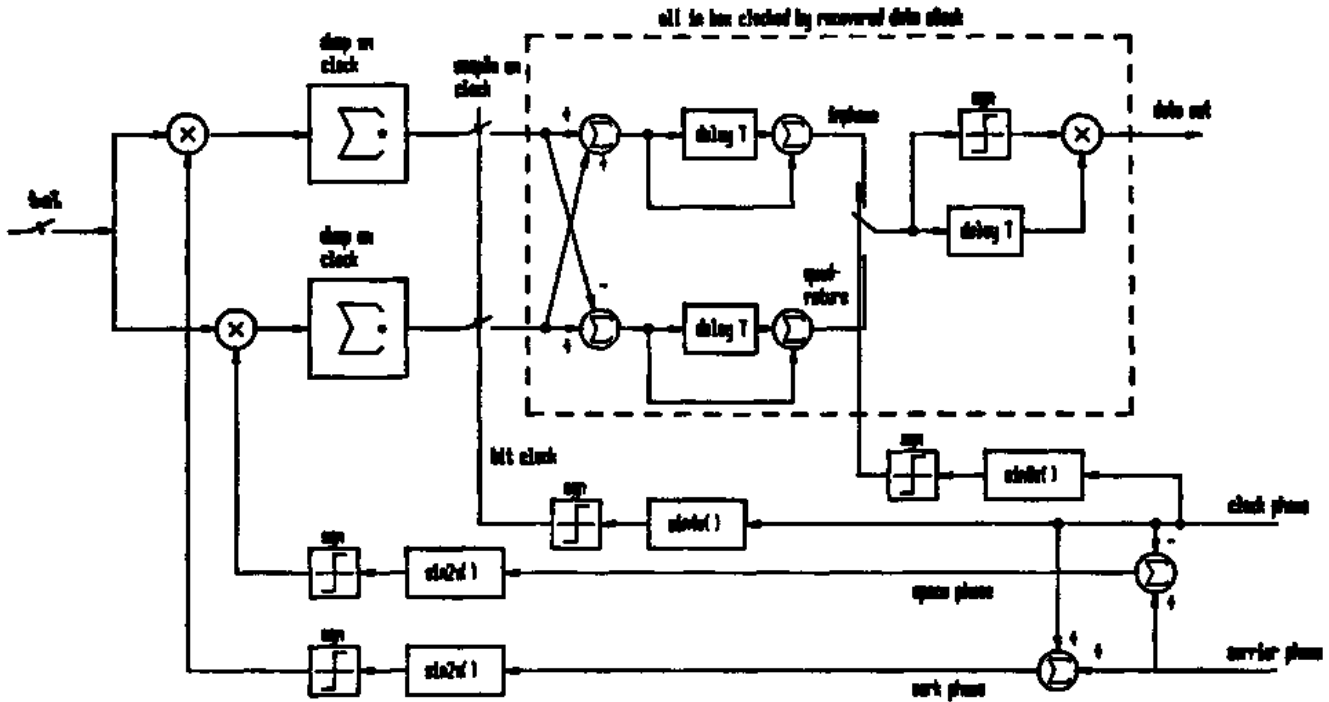


Figure 11: Microprocessor based correlation detector.

Figure 6 shows the block diagram of the correlation detector algorithm used in this radio. With a few exceptions the operations of this detector are exactly equivalent to the operations of the prototypical MSK detector presented at the beginning of this chapter. The two main exceptions are that this demodulator operates in discrete time rather than continuous time, and the multiply operations use square wave inputs rather than sinusoidal inputs in order to speed up the processor.

While the operations performed are the same, the layout is quite different, so I will describe the detector in detail.

The sampled signal, $r(m)$, and the mark and space phases, $\theta_s(m)$ and $\theta_m(m)$, are exactly the same as those used in the synchronization recovery algorithm. Expressions for these variables are given in (3.11), (3.12), and (3.14).

The MSK detector in Figure 7 gets its inphase and quadrature outputs by adding or subtracting the mark and space tones to get the inphase and quadrature signals, then multiplying these signals by the incoming signal. We felt that reducing the inphase and quadrature signals to simple square waves before the multiplication would leave too much information out of the signals. To get around this, the discrete time algorithm multiplies by the constant envelope mark and space tones first, then does the addition/subtraction as a later step.

I will explain the validity of this idea. To make the explanation simpler I will express this with integrations. The concepts are exactly the same for the case using summations. If we let $d_i(2n+1)$ be the output of the inphase integrator, and $d_q(2n+2)$ be the output of the quadrature integrator in

Figure 7, these numbers are equal to:

$$\begin{aligned}
 d_i(2n+1) &= \int_{(2n-1)Tb}^{(2n+1)Tb} r(t)z(t)dt = \int_{(2n-1)Tb}^{(2n+1)Tb} r(t)[\cos\theta_m t + \cos\theta_s t]dt \\
 &= \int_{(2n-1)Tb}^{(2n+1)Tb} r(t)\cos\theta_m tdt + \int_{(2n-1)Tb}^{(2n+1)Tb} r(t)\cos\theta_s tdt \\
 d_q(2n+1) &= \int_{(2n+2)Tb}^{(2n)Tb} r(t)y(t)dt = \int_{(2n+2)Tb}^{(2n)Tb} r(t)[\cos\theta_m t - \cos\theta_s t]dt \\
 &= \int_{(2n)Tb}^{(2n+2)Tb} r(t)\cos\theta_m tdt - \int_{(2n)Tb}^{(2n+2)Tb} r(t)\cos\theta_s tdt
 \end{aligned}$$

. These integrals all integrate over a period of $2T_b$, They can be broken up into integrals over T_b . For example,

$$\int_{(2n)Tb}^{(2n+2)Tb} r(t)\cos\theta_m tdt = \int_{(2n)Tb}^{(2n+1)Tb} r(t)\cos\theta_m tdt + \int_{(2n+1)Tb}^{(2n+2)Tb} r(t)\cos\theta_m tdt$$

I will define a new pair of integrator outputs, $d_m(n)$ and $d_s(n)$ to be

$$\begin{aligned}
 d_m(n) &= \int_{nTb}^{(n+1)Tb} r(t)\cos\theta_m tdt \\
 d_s(n) &= \int_{nTb}^{(n+1)Tb} r(t)\cos\theta_s tdt
 \end{aligned}$$

With these definitions, (3.29) becomes

$$d_i(2n+1) = d_m(2n) + d_m(2n+1) + d_s(2n) + d_s(2n+1)$$

and (3.30) becomes

$$d_q(2n+1) = d_m(2n) + d_m(2n+1) - d_s(2n) - d_s(2n+1)$$

As shown in Figure 6, the demodulator algorithm executes (3.34) and (3.35) in the slower section, while the only operation going on in the faster section is the acquisition of $d_m(n)$ and $d_s(n)$ (as given in 3.32 and 3.34). Ultimately, the operations performed by this algorithm are equivalent to the operations performed in Figure 7 with the blocks rearranged.

This microprocessor-based demodulator ends up working very well indeed. Its performance is less than 1dB worse than the theoretical optimum for this type of demodulator. Test results for this measurement and other pertinent measures of the radio's performance are given in the next chapter.

4 Tests of Radio Performance

I tested the radio for its performance under a variety of criteria. These were: demodulation performance in the presence of Gaussian noise, demodulation performance in the presence of nearby CW interference (to simulate the effects of the radiobeacon's own carrier on the radio performance), single-signal blocking characteristics, two-tone intermodulation input intercept, and receiver equivalent input noise. Each of these tests are discussed in the following sections.

4.1 Bit error rate in the presence of Gaussian noise

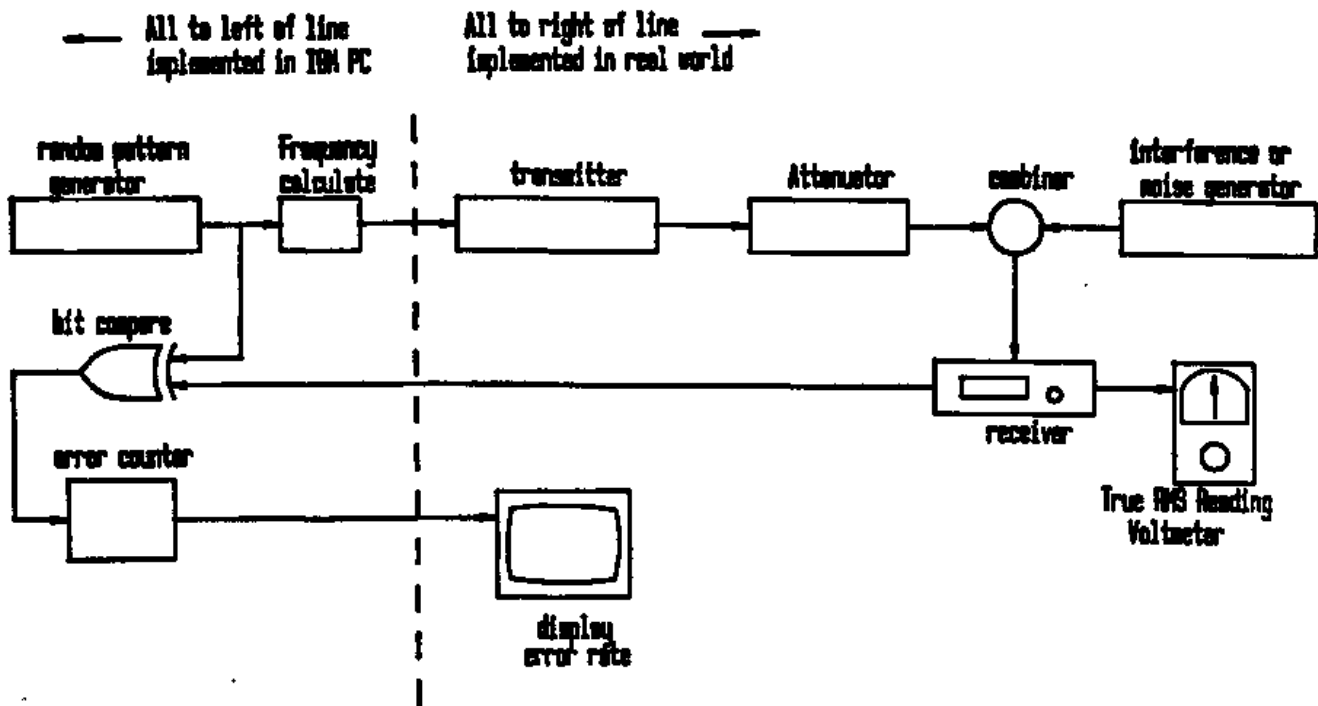


Figure 12 shows the test bed used to obtain measurements of demodulation performance in the presence of Gaussian noise and demodulation performance in the presence of nearby CW interference. Both of these tests are similar in that an interfering signal is added to the intended signal, and the intended signal is adjusted while the bit error rate is measured.

In both cases, the actual bit error rate (BER) is measured with the help of a computer. As shown in Figure 12, the pseudo random sequence is generated by a ten stage tapped shift register, which gives a 1023 bit sequence. The output of this sequence generator is applied to the frequency calculator and the corresponding transmit frequency is sent to the transmitter. The transmitter transmits a mark or a space, then the signal is attenuated and corrupted by either interference or noise. The signal is received by the receiver under test, and the bit decision is compared with an appropriately delayed transmit bit. If the bits do not match, then the error counter is incremented. At intervals, the value in the error counter is updated to the screen.

The following procedure is used to measure the BER vs. signal to (Gaussian) noise ratio (SNR). First, the audio output of the radio is connected to a true RMS reading voltmeter. Then the AGC circuit is defeated, and the radio is connected to a generator of wideband RF Gaussian noise. The IF gain is adjusted until the RMS noise voltage at the the audio output is some convenient value (I used 1V). Next, the noise generator is switched of and the transmitter turned on. The transmitter output is adjusted until the RMS signal voltage is equal to what was established by the noise generator. Finally, the AGC is enabled, and the BER and SNR are measured and recorded for various settings of the transmitter attenuator.

In order to compare the measured performance with the theoretically optimum performance, we must know the relationship that gives E_b and N_o given the audio RMS voltage. The filter transmission characteristics (from Figure 2.4) were converted from graphical form to numerical data at 50Hz intervals. This set of data was used to calculate the audio RMS as a function of N_o and the audio RMS as a function of E_b . Ultimately, it was found that at a SNR of 0dB the E_b/N_o ratio is 0.7dB.

As mentioned in chapter 3, there is a trade off between the carrier loop bandwidth and its acquisition time. To get some data on the effect of carrier loop bandwidths, these tests were conducted for various carrier loop bandwidths. In all cases, the clock loop bandwidth was held constant at 2.76rad/sec.

| SNR | BER | BER | BER |
|------|------------------|---|--|
| (dB) | theoretical | $K_{ca} = 2^{-8}$ $K = 5.5 \text{rad/sec}$ | $K_{ea} = 2^{-6}$ $K = 22 \text{rad/sec}$ |
| 10 | $1.65 * 10^{-6}$ | $2.07 * 10^{-5}$ | $7.3 * 10^{-5}$ |
| 8 | $1.41 * 10^{-4}$ | $5.0 * 10^{-4}$ | $1.00 * 10^{-3}$ |
| 6 | $2.5 * 10^{-3}$ | $4.8 * 10^{-3}$ | $7.4 * 10^{-3}$ |
| 4 | .0162 | .0239 | .0285 |
| 2 | .055 | .072 | .083 |
| 0 | .121 | .133 | .155 |
| -2 | .203 | .222 | .254 |
| -4 | .282 | .30 | .33 |
| -6 | .35 | .37 | .39 |

Table 2: Bit error rate vs. SNR for various carrier loop bandwidths

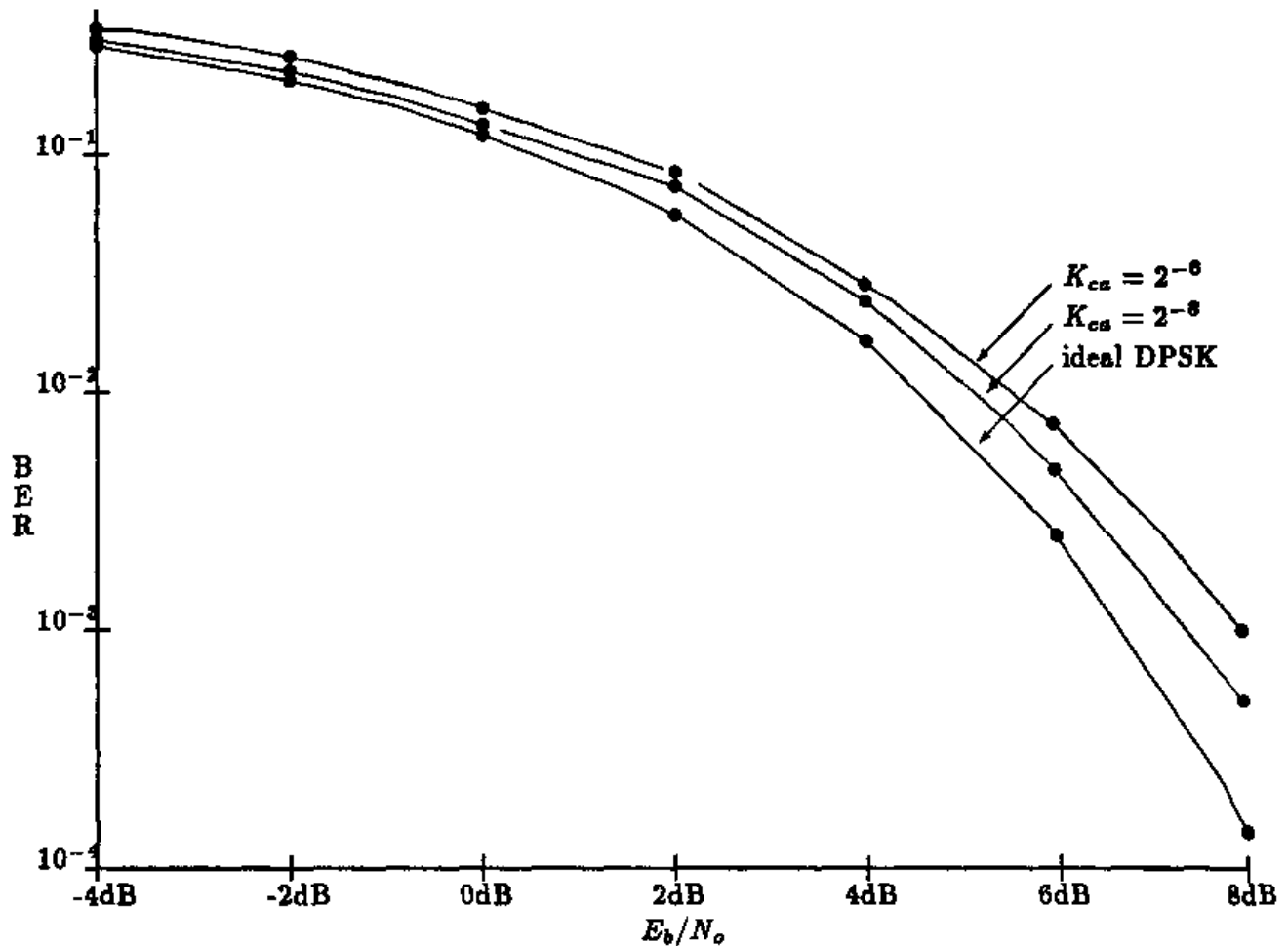


Table 2 lists the performance results for these various loop bandwidths, while Figure 13 presents the results graphically. For the carrier loop bandwidth of 5.5rad/sec, the error rate performance is only .6 to 1dB worse than the theoretical optimum for MSK, which is not too bad.

Notice that the loops managed to maintain lock in the face of noise bad enough to cause a BER of .39. The goal of a demodulator that can perform well in an environment that requires FEC has certainly been demonstrated.

In doing these tests, it was discovered that the loops could not maintain lock for bandwidths greater than $K = 88\text{rad/sec}$ ($K_{ca} = 2^{-4}$) and would act erratically for bandwidths smaller than

$K = 1.4\text{rad/sec}$. One can easily predict the loss of lock for the greater bandwidths — all of the stability calculations were done assuming a continuous time loop, and when the bandwidths start to approach the sample frequency the continuous-time approximation falls apart. I believe that the erratic behavior at the lower loop bandwidths is due to phase noise in other oscillators in the system, because the loops were fairly happy with clock bandwidths as low as $K = .172\text{rad/sec}$ ($K_{cl} = 2^{-13}$).

4.2 BER in the presence of an interfering carrier

Measurements of BER in the presence of interfering carriers at 500Hz from the center frequency were done to simulate the effect of the radiobeacon carrier on the demodulator performance, and also to investigate the hypothetical situation of receiving a weak DGPS/Radiobeacon signal in the presence of a strong radiobeacon at a neighboring frequency. I made this measurement at signal levels low enough to avoid any non-linearities in the receiver front end. This test is conducted in exactly the same manner as the Gaussian noise test, with the exception that a CW generator at 500Hz above or below the MSK center frequency is used instead of the RF noise generator.

| SIR | BER | BER |
|------|-------------------------|-------------------------|
| (dB) | $f_r = 288\text{kHz}$ | $f_r = 288\text{kHz}$ |
| | $f_R - 287.5\text{kHz}$ | $f_R - 288.5\text{kHz}$ |
| -22 | 0 | 0 |
| -24 | 2.0×10^{-5} | 0 |
| -26 | 9.5×10^{-4} | 0 |
| -28 | .023 | 0 |
| -30 | .37* | 2.6×10^{-5} |
| -32 | * | 1.0×10^{-3} |
| -34 | * | 6.5×10^{-3} |
| -36 | * | .051 |
| -38 | * | .171 |
| -40 | * | .21* |
| -42 | * | * |

Table 3: Bit error rate vs. signal to interference ratio. Entries marked with an asterisk (*) represent interference levels where the loops have lost lock

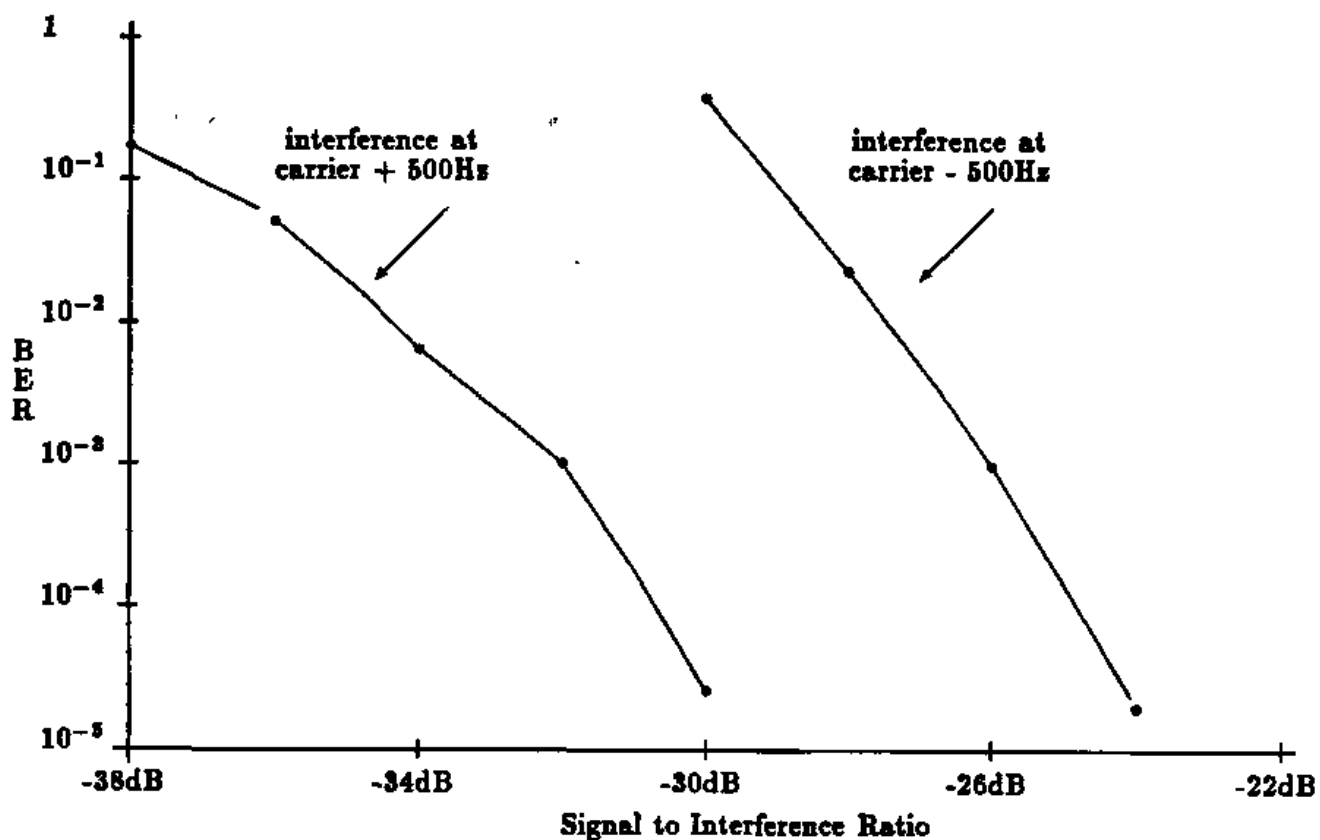


Table 3 lists the results of this test for interfering frequencies that are 500Hz above and 500Hz below the desired receive frequency. Figure 14 presents these results graphically. When the data was taken at high error rates the demodulator took a long time to recover from the interference, which suggests that the input interference affects the phase-lock loops. Notice that the error rate goes from very good to horrible in a very small span of input power. This is most likely because the interfering signal overwhelms the squaring function in the phase locked loops at this point. Also notice that the resistance to this interference is 10dB better when the interference is on the low side of the signal. When the interference is 500Hz below the carrier at RF, it hits a frequency of 1500Hz at audio, which places it at the sampling frequency of the demodulator. After sampling, this interference is very close to DC, and should be modulated out of existence in the phase-lock loops.

4.3 3rd Order Inter modulation Performance

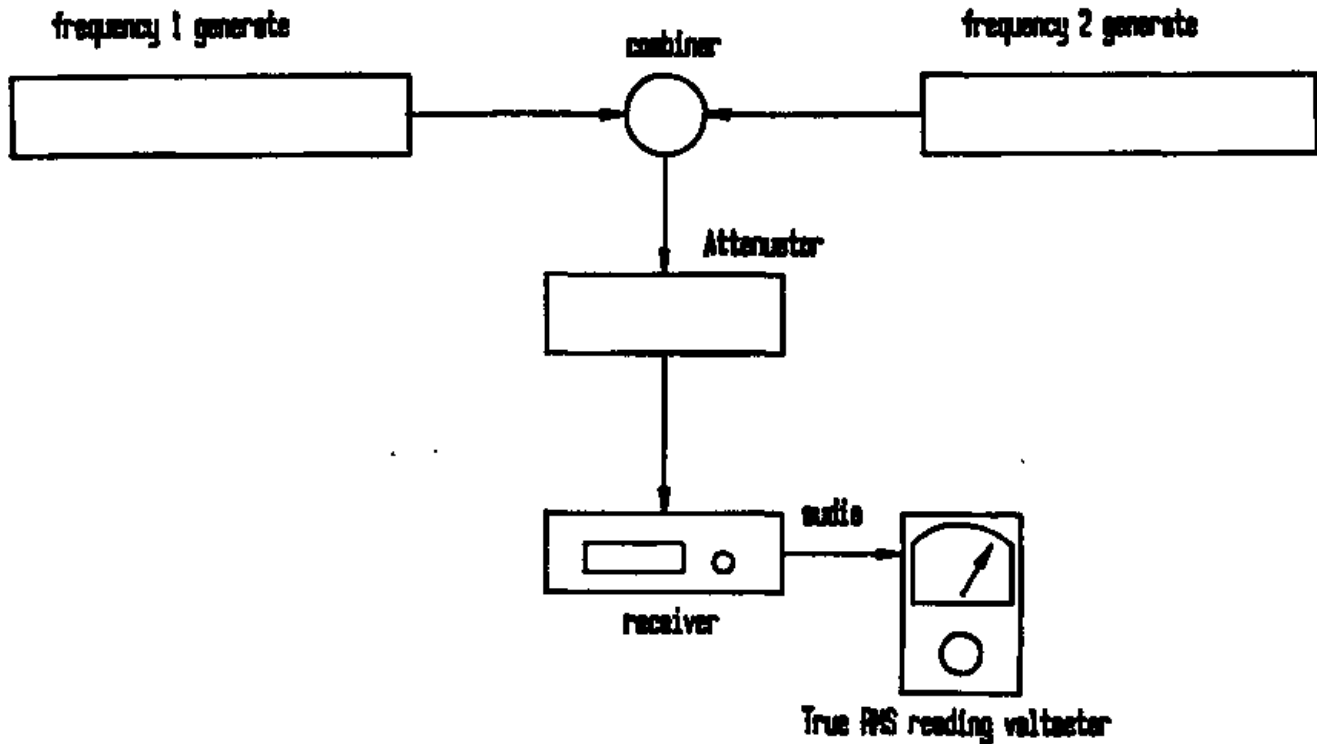
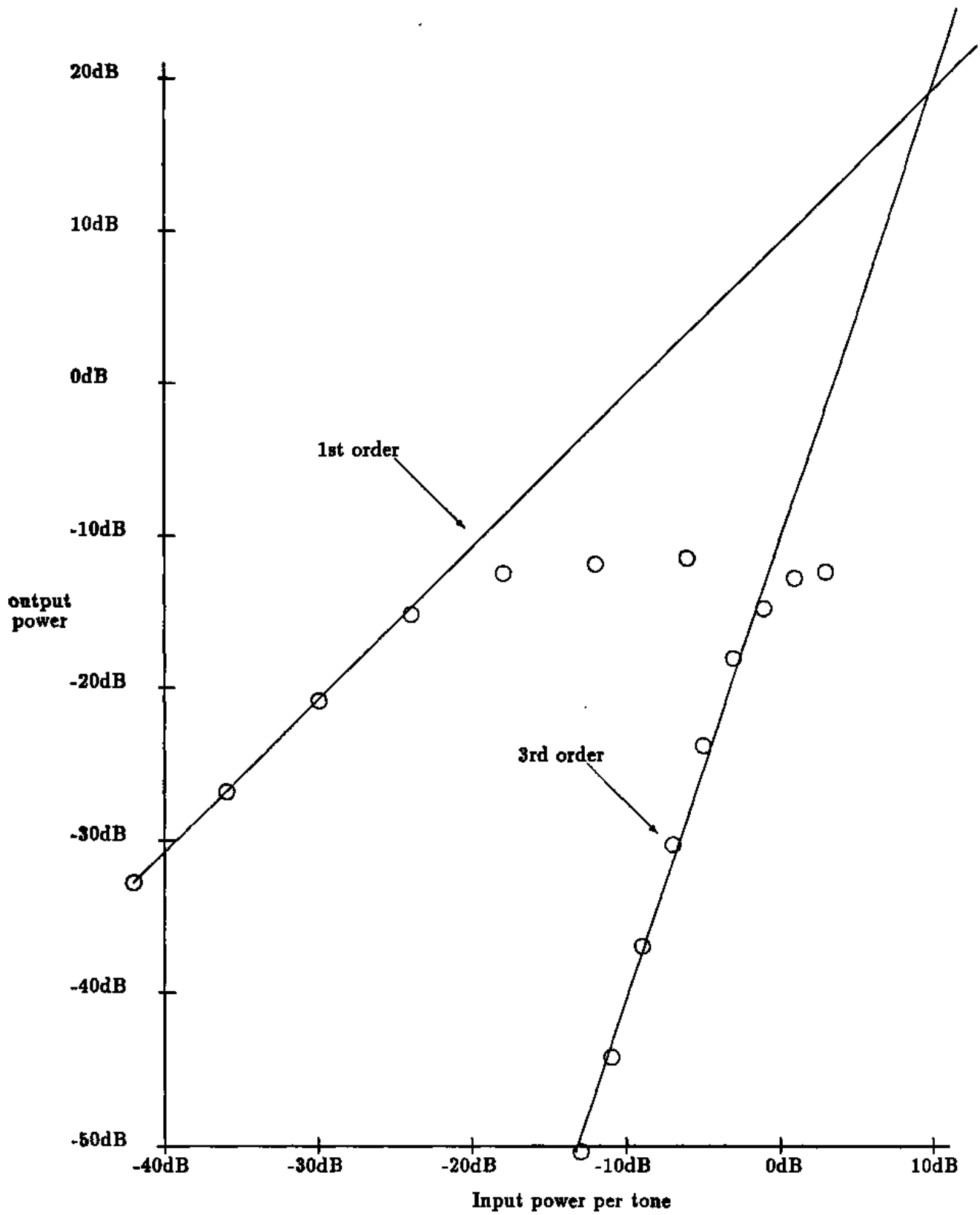
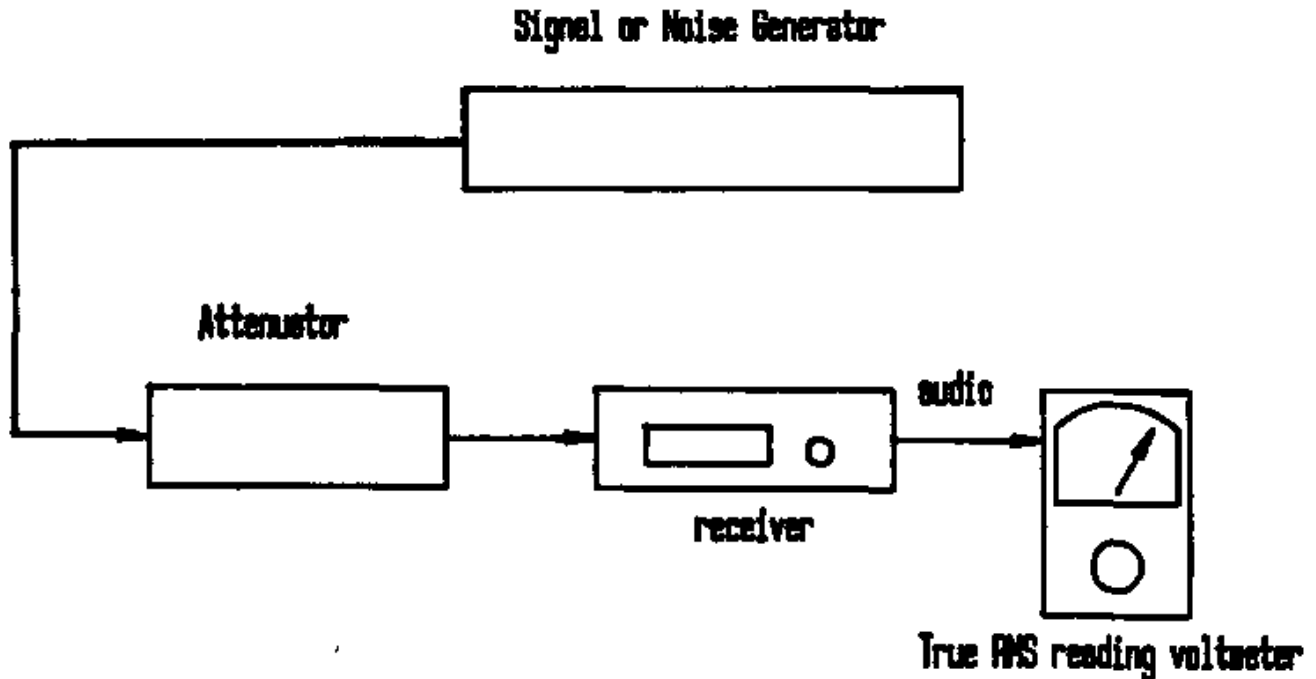


Figure 15: 3rd order intermodulation test bed

Figure 15 shows the test bed used to obtain a measure of the 3rd order intermodulation performance of the receiver. In order to measure this performance, the receiver gain from the RF input to the audio output (demodulator input) was measured at 287.5kHz RF input for a given IF gain setting (the AGC was once again disabled). Then a pair of signals at 302.5kHz and 317.5kHz was applied to the RF input and the resultant audio output (the 3rd order intermodulation product) was measured as the signal power was varied. Figure 16 shows the numerical results of these tests. When the two sets of input power vs. output level data are plotted as in Figure 16, one can calculate the 3rd order intermodulation intercept. For this radio, this intercept comes out to be +9.8dBm. This is neither very good, nor what was expected. The mixer input intercept is around +11dBm, and the RF filter has about 7dB of loss, so the input intercept to the radio should be around +18dBm. I suspect that the problem is either the termination of the mixer or, more likely, that the IF preamp is not robust enough.



4.4 Receiver Noise Floor



The test setup in Figure 17 was used to measure the noise characteristics of the receiver. The receiver AGC was disabled, and a small signal was injected to the receiver at its RF port, and the signal magnitude was adjusted until the output at audio was 10dB greater than the output with no signal input. This occurred at an input carrier amplitude of 104.7dBm. From this we can deduce the effective noise power at the input to the receiver. This is

$$\text{noise power} = -104.7\text{dBm} - 9.5\text{dB} = -114.2\text{dBm}$$

. This is also known as the 'noise floor' of the receiver. Knowing the noise floor of the receiver and the effective noise bandwidth of the IF, we can deduce the receiver noise figure:

$$\begin{aligned} NF &= 168\text{dBm} + \text{noise floor} - 10\log(B_i) \\ &= 168\text{dBm} - 114.2\text{dBm} - 10\log(490) \\ &= 27\text{dB} \end{aligned}$$

Notice that this is not a very good noise figure, but since we are putting the antenna coupler before the RF input to the radio, we can reduce the overall effective input noise by increasing the amplification in the antenna coupler. Overall, the tests of demodulator performance were very encouraging, and the tests of the basic radio performance were rather disappointing. In a way this is not surprising. We ended up expending much more effort on the demodulator design for this radio than we did on the basic radio itself. I think that were one to implement this radio as a practical design, some design effort should be expended to make the radio a little more robust and a little less noisy.

5 Summary, Some Comments on Soft Bit Decisions, and Further Work

5.1 Summary

I have described a radio receiver built to receive MSK in the 285-325kHz bands. This receiver does this job well. The most notable features of this receiver are its ability to be modified for soft bit decisions, its ability to maintain demodulator lock for very high noise levels, and the microprocessor based demodulator, which is by its nature very flexible.

5.2 Some Comments on Soft Bit Decisions

The original aim of the work in this thesis was to develop a radio/demodulator combination that could give soft bit decisions (or generate side information - the two are equivalent). While this goal was not achieved, quite a lot of groundwork has been done in the development of hardware and software to get the job done.

According to [15], the soft decision will be sufficient if it is some one-to-one mapping of the probability that a zero was transmitted given the signal received at the demodulator, I will represent this by $P(0|s)$. The easiest such one-to-one mapping that I can think of is an estimate of $P(0|s)$ itself. Note that in the case of a hard bit decision, this probability estimate takes only two values, and becomes the output of the demodulator.

There are two basic ways that soft decisions can be made. One is to extract information about $P(0|s)$ directly from the input to the demodulator. The other is to monitor some other (hopefully empty) channel to detect noise bursts, and use this auxiliary information along with the demodulator output to calculate $P(0|s)$.

5.2.1 The Auxiliary Channel Method

In the case of ordinary white Gaussian noise, it is useless to use the auxiliary channel method, because the noise processes received in two spectrally disjoint channels will be independent. In the case of atmospheric noise, however, it has shown experimentally that this is not the case [10]. Specifically, the instantaneous noise power of the two channels show positive correlation. This means that a side information generator using an auxiliary channel could be as simple as monitoring the noise power in the auxiliary channel and tagging the output bits as errors when the auxiliary channel noise power exceeds some threshold. The drawback to the auxiliary channel method is that one needs to have an auxiliary channel available. Such a channel is not available on this radio. Furthermore, it would be expensive to build a radio with such an auxiliary channel.

5.2.2 The In-Channel Method

Although it is not immediately apparent, there is quite a bit of information about $P(0|s)$ at the input to the demodulator. Again, in the case of ordinary white Gaussian noise, all of this information will be available at the output of the demodulator (i.e., the only processing needed is that provided by the

demodulator itself). The treatment given in [15] for soft-decision processes gives a good example of the processing necessary for Gaussian noise. In the case of atmospheric noise, there is more information about $P(0|s)$ at the input to the demodulator than at the output.

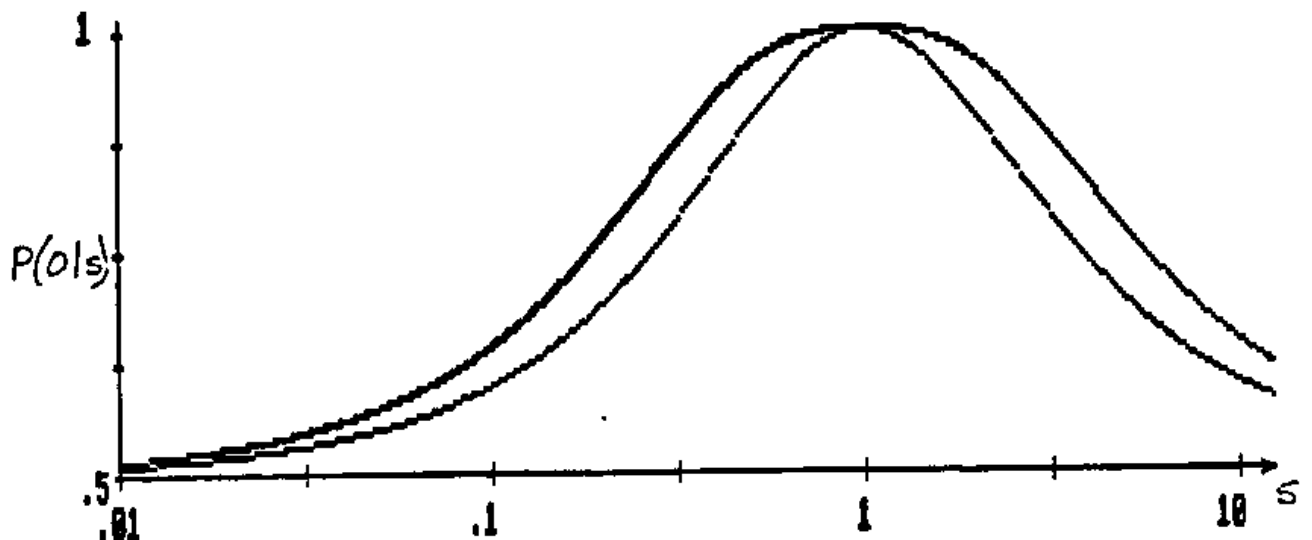
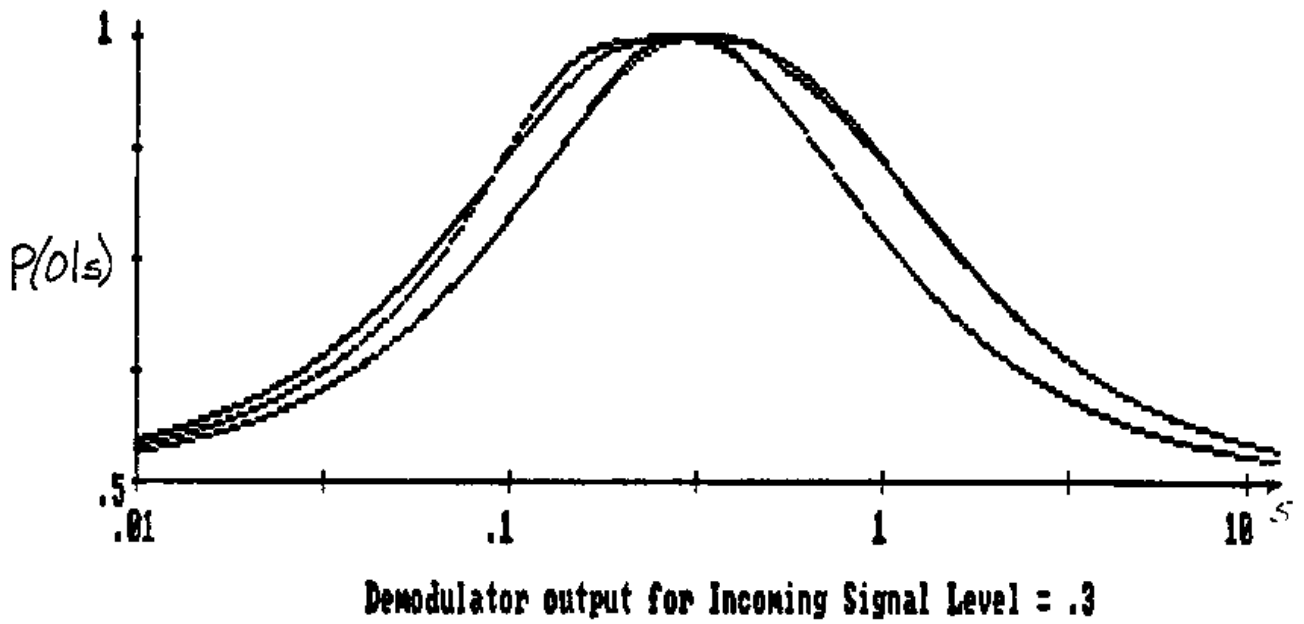
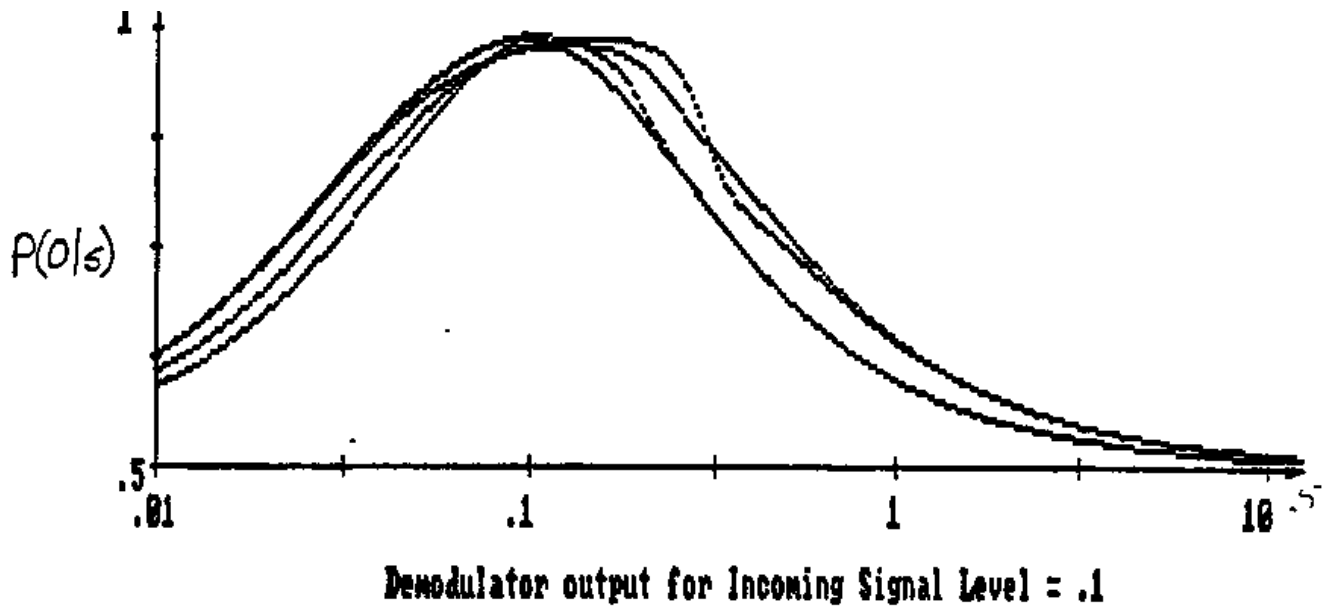


Figure 18: $P(0|s)$ vs. incoming signal level for various incoming signal levels and atmospheric conditions

If we ignore the time structure of the atmospheric noise, then $P(0|s)$ depends only on s , the output of the demodulator, and we can just inspect this output for each bit to decide on our estimate of $P(0|s)$. Figure 18 shows $P(0|s)$ for various input signal strengths and weather conditions, using the probability densities for atmospheric noise developed in [10]. If we examine the plots of Figure 18, several things become apparent: First, on a semi-log plot, the shape of the curves do not change for varying signal strengths. Second, on a semi-log plot, the curves are symmetrical about the incoming signal strength.

The shape of the plots in Figure 18 suggest a method for estimating $P(0|s)$. If we know the signal strength, we can divide the demodulator output by the signal strength and get a normalized signal that can be applied to a look-up table to get a quantized estimate of $P(0|s)$. We notice that with only one look-up table set up for tropical and frontal conditions, this method will be inaccurate for the quiet and quiet-night conditions. However, further thought will reveal that while the estimate will be inaccurate, it will still be a monotonic function of $P(0|s)$, and the raw error rate will be better anyway, so the accuracy is not as necessary.

Unfortunately, the calculations of the the expected value of the absolute value of the signal ($E\{|z|\}$) has a bias that changes with signal strength and noise conditions. This means that if we depend on an estimate of the signal strength based on the average of the absolute value of the signal, our estimate of the signal strength will be similarly biased, and our $P(0|s)$ estimate will be flawed. However, if the absolute value of the signal is truncated at twice the signal strength, the bias is removed, which suggests a mechanism to estimate the signal strength accurately.

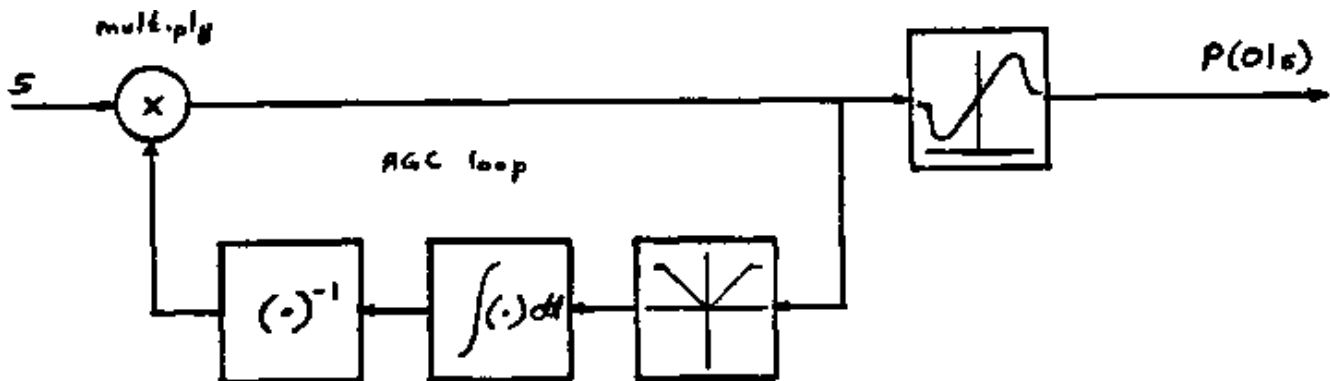


Figure 19 shows a proposed post-demodulator processor that would take the output of the demodulator and process it into an estimate.

5.3 Suggestions for Further Work

This radio lacks the ability to make soft decisions in non-Gaussian noise. The greatest area of work that remains to be done with this radio is to develop a soft decision post-processor for the demodulator.

No effort was made with this radio to design the phase lock loops to be optimum in atmospheric noise. It is suggested in [10] that the correct use of non-linearities in the phase lock loop signal path could improve the loop performance.

No effort was made to investigate schemes for demodulation at baud rates lower than 400 baud. While it is straightforward to modify the demodulator algorithm into an optimum demodulator for Gaussian

noise at any baud rate lower than 400, the problem arises that the demodulator is no longer optimum for atmospheric noise. Also, the effect that optimum demodulation has on the soft-decision properties of the demodulator should be investigated.

1 Appendix A

1.1 The Continuous-time MSK Demodulator

Initially I attempted to realize the demodulator with continuous time analog circuitry. While the resulting demodulator ends up having a number of drawbacks for this application (as we shall see) I feel the basic design is worth presenting here.

This demodulator, like the microprocessor based one, is derived almost directly from the diagrams presented in the Pasupathy paper [2] (Figure 8, Figure 9). Because the bulk of the demodulator is taken up with the functions required to synchronize the demodulator to the incoming signal I will describe them first.

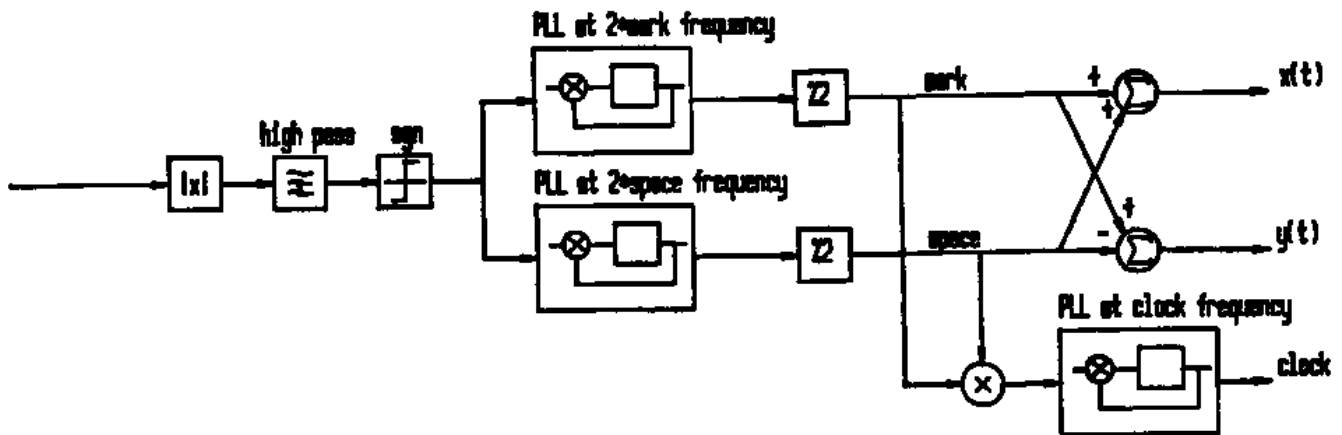


Figure 20 is a block diagram of the continuous time synchronization recovery circuit. The incoming signal is full-wave rectified and run through a comparator to give a squaring effect. The resulting signal is applied to a pair of phase locked loops (based on CD4046 PLL chips) with center frequencies of 2200Hz and 1800Hz. The bandwidth of the PLLs are adaptable, being B_1 when the loops are in the acquisition mode and B_2 in the locked mode. These reference frequencies are divided by two to give the mark and space references. The clock is derived by multiplying the mark and space frequencies and using yet another PLL to give a clean clock signal at one half the bit rate.

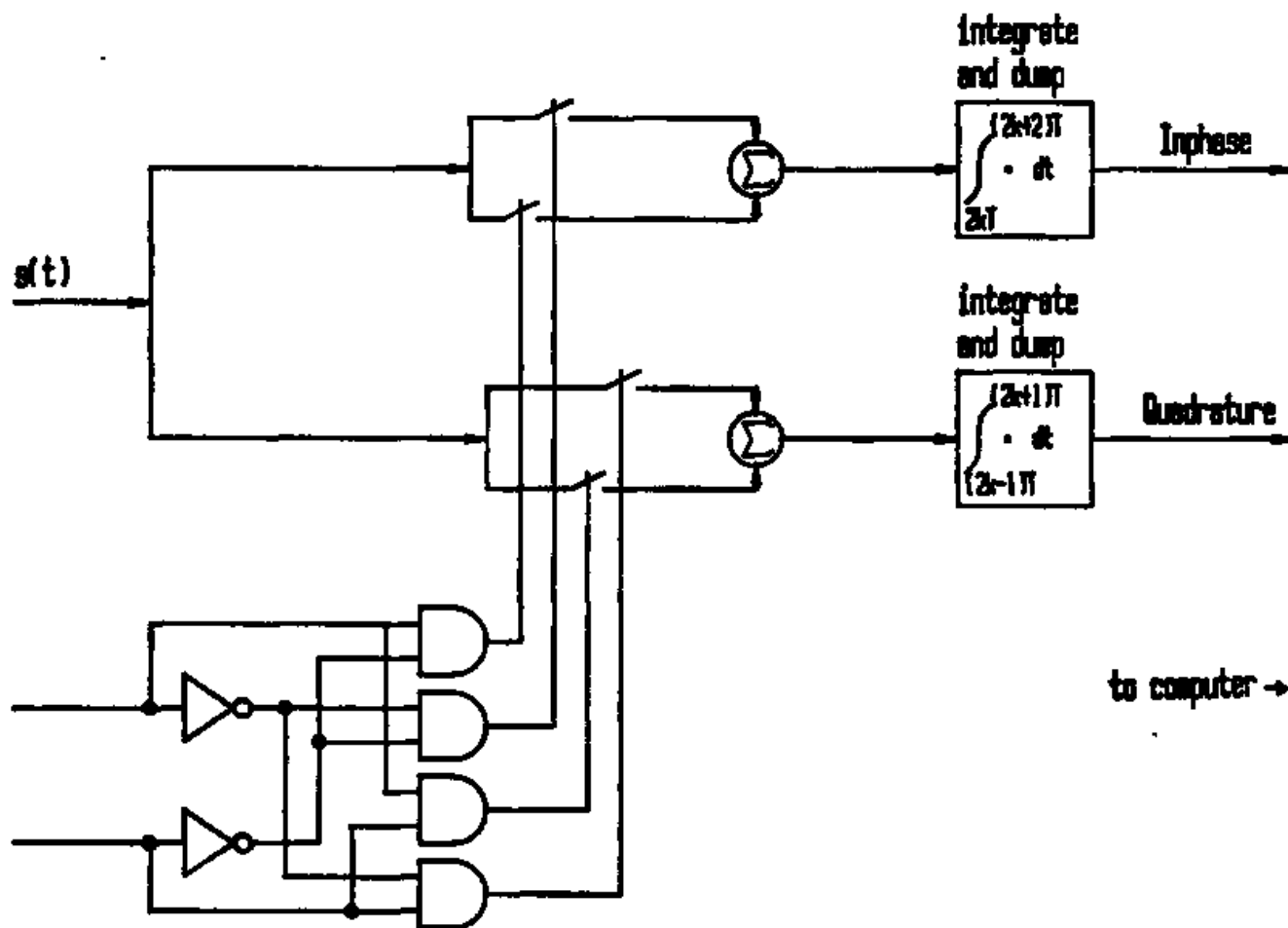


Figure 21 is a block diagram of the detector circuit. This circuit uses an analog multiplexer to implement the multiply operation of figure 3.2. The inphase channel is demodulated by multiplying the input signal by the sum of the mark and space references, then integrating the result. When the clock transitions high to low, the integrator is read and dumped. Similarly, the quadrature channel is demodulated by multiplying the input signal by the difference of the mark and space references, then integrating the result. When the clock transitions low to high, the integrator is read and dumped. Notice that the demodulation operation is not complete; it is lacking a final multiplication. This final operation is performed in the microprocessor.

Most of the drawbacks of this demodulator stem from the lack of stability in the oscillators, which forces the use of fairly high-band width loop filters. The VCO frequency in the CD4046 is affected by much more than just the frequency input. It depends heavily on supply voltage and the (usually inaccurate) resistor and capacitor values. I found experimentally that the best initial oscillator frequency accuracy I could expect was about $\pm 20\text{Hz}$. The in-lock filter bandwidth that I settled on is (experimentally) the smallest I could get without problems with drift.

2 Appendix B

2.1 Receiver Software

This software was written for the Motorola 68HC11 microprocessor. The following is a complete listing of the program that runs the receiver.

```
*****
*
*                               Radio version 3.12
*
*                               Tim Wescott April 6, 1990
*
*   This is the initial, experimental version of the firmware for
*   a differential GPS receiver being built for Per Enge at Worcester
*   Polytechnic Institute.  The firmware is for a real-time controller
*   that is embedded into a fairly standard receiver capable of
*   receiving 400 baud MSK data in the 285kHz - 325kHz band.  Because
*   the tasks are time critical and event driven, the firmware is
*   almost entirely interrupt driven.
*
*   The functions that this computer in providing are:
*
*   Monitor the frequency input knob, and output
*   frequency changes to the display and the synthesizers
*
*   Implement the demodulator in software, with variable
*   loop bandwidth.
*
*   Send the bit decisions out to a host computer via
*   the RS232 transmitter.
*
*****
```

```
* This section contains the declarations for all of the static
* variables and buffers needed by the program
```

```
* debounce constants
dbtime equ    $100                * 128us delay
timeout equ   $2000
synofst equ   500+911             * synth setting for 250 kHz rx
*                               with new filter
dspofst equ   $50                 * low byte of 250kHz offset
timer equ     $0800              * timeout setting
```

```

* lock threshold constants
lockthr equ    $0680          * lock threshold
lockhys equ    $04c0

* sample/demodulator constants
samples equ    15             * number of samples taken at 400 baud
samtime equ    5000/samples   * sample time for sample rate

baudr  equ    400             * the baud rate of the demod
clk1   equ    40*baudr/samples * this wild and wonderful expression
clk2   equ    48*baudr/samples+25/50 * gives the starting clock increment

clkstrt equ    clk1+clk2      * starting bit clock frequency
clkmax  equ    clkstrt+4      * maximum bit clock frequency
clkmin  equ    clkstrt-4      * minimum bit clock frequency

carstrt equ    $50ce/samples*8 * starting carrier frequency
* note that for a 1kHz carrier this would be $5000/samples*$8
carmax  equ    carstrt+$100    * maximum carrier frequency
carmin  equ    carstrt-$100    * minimum carrier frequency

clbw   equ    $09             * clock bandwidth
clzr   equ    $0b             * clock zero
cabwi  equ    $09             * starting carrier bandwidth
* this may have to be changed to 05
cabwl  equ    $09             * in lock carrier bandwidth
cazri  equ    $0b             * starting carrier zero
cazrl  equ    $0b             * in lock carrier zero

*****
org    $00

* These are locations to handle frequency entry, storage, and display
freq   fcb    $00             * frequency offset from 250kHz
synfreq fdb    $00             * calculated number for synth
dapfreq fdb    $00             * bit pattern for the display
spiflag fcb    $00             * flags for the spi interrupts
oldknob fcb    $00             * last setting of frequency knob

* these are locations for lock-detect
inlock fcb    $00             * space for lock memory
lockcrr fcc    '000'          * filter variable
scratch fcc    '000'          * scratch pad

* these are demodulator-related memory locations
mul_sch fcb    $0             * scratch pad for multiply
old_out fcb    $0             * old output bit

recr   fdb    $0             * receive scratch pad
dem_i_1 fdb    $0             * last value of inphase summer
dem_q_1 fdb    $0             * last value of quadrature summer
dem_sp  fdb    $0             * 'space' variable
dem_mk  fdb    $0             * 'mark' variable

```

```

dem_spe fdb      $0          * 'space' phase error variable
dem_mke fdb      $0          * 'mark' phase error variable
dem_scr fdb      $0          * scratch pad

clk_pht fdb      $0          * temporary clock storage
car_phe fcc      'pea'      * carrier phase error
clk_phe fcc      'pel'      * clock phase error
car_ph  fcc      'pha'      * current value of carrier phase
clk_ph  fcc      'phl'      * current value of clock phase
car_inc fcc      'ina'      * current value of carrier increment
clk_inc fcc      'inl'      * current value of clock increment

car_phu fcc      'capu'     * carrier phase update
clk_phu fcc      'clpu'     * clock phase update
car_frq fcc      'cafq'     * current value of carrier frequency
clk_frq fcc      'clfq'     * current value of clock frequency

```

```

*****

```

```

        org      $e000          * start of ROM

*this is the look - up table for the display digits
disptab fcb      $80, $a6, $48, $42, $26, $12, $10, $c6, $00, $06

* First we need to set some values
* we will be doing some loads into the register set

startup ldx      #base          * base address of registers
        ldw      #$00ff        * top of ram

* initialize the serial port

        ldaa     #$31          * 4800 baud
        staa     baud,x

        ldaa     #$08          * 8 bit word
        staa     sccr1,x

        ldaa     #$08          * no interrupts, enable TI
        staa     sccr2,x

        ldaa     #$3a          * set port D directions consistent
        staa     ddrd,x        * with serial ports

* initialize the SPI port to 62.5Kbaud, positive clock, normal phase,
* interrupts off, port D push - pull

        ldaa     #$53
        staa     spcr,x
        bset     portd,x #$20          * reset the latch

* turn on the A/D converter

```



```

        bset    option,x #$80          * set ADPU bit (power up A/D)

* initialize the various demodulator variables

        ldy    #recr                  * zero out the demodulator
        clra                   * memory block
main0   staa   0,y
        iny
        cpy    #clk_frq+4
        bne   main0                  * loop until entire block = 0

        ldd    #carstrt              * set carrier to its
        std    car_inc                * initial frequency
        std    car_frq
        ldd    #clkstrt              * set clock to its
        std    clk_inc                * initial frequency
        std    clk_frq

* initialize clock capture and port A

        ldaa   #$0f                  * capture all edges of frequency
        staa   tct12,x                * knob input capture pins

        ldaa   #$83                  * enable frequency knob interrupts
        staa   tmsk1,x                * and timer compare
        staa   tflg1,x                * reset all input capture flags

        bset   pactl,x #$80           * enable pa7 output
        bset   oclm,x #$80           * set pa7 = 0 on
        bclr   ocld,x #$80           * ocl timeout

* turn on the lock acquisition circuitry

        bclr   porta,x #$40

* initialize the sample timer count

        ldd    tcnt                  * get currant time
        addd   #$080                 * calculate target time
        std    tocl                  * store it away to wait for sample

* enable interrupts

        cli

* Initialize the spi to a known (useful) state

        ldaa   sp^r,x                * this ought to clear, then set,
        staa   spdr,x                * the spif flag

* set the receive frequency to 287kHz

```

```

        ldaa    #$37
        staa    freq
        j^r     syncalc
        j^r     dspcalc

* get the current setting of the frequency set knob

        ldaa    porta,x
        anda    #$03                * we want only the last two bits
        staa    oldknob
        lsra
        sora    oldknob            * get value in straight binary
        staa    oldknob            * save it

* now spend some time reassuring the COP watchdog

mainlp  ldaa    #$55                * arm COP reset
        staa    coprst,x
        ldaa    #$aa                * reset COP timer
        staa    coprst,x

        brset   porta,x #$20 main1
        bset    porta,x #$20
        bra     mainlp

main1   bclr    porta,x #$20

* Put your main - loop code here!

        bra     mainlp

*****
* interrupt routine to start the sample process
* and run the demodulator

sample  ldx     #base

        ldab    adr1,x            * get input
        subb    #$80                * make into 2's comp

        bpl     sam1                * sign extand racr
        ldaa    #$ff
        bra     sam2

sam1    clra

sam2    std     recr

        ldaa    #$02                * start the A/D converter
        staa    adctl,x

```

```

        ldd     tocl,x           * reset the timer for the
        addd   #samtime        * next interval
        std     tocl,x
        bclr   tflfl,x #7f     * clear the ocif flag

A_DcoEp ldd     clk_ph          * update clock phase
        addd   clk_inc         * into temporary space
        std     car_ph

        adda   clk_pht         * get mark phase

        bpl    €_Dc01          * multiply by sign of sin mark
        ldd    dem_mk
        subd   recr
        std    dem_mk
        bra    A_Dc02

A_Dc01  ldd    dem_mk
        addd   recr
        std    dem_mk

A_Dc02  ldaa   car_ph
        suba   clk_pht        * get space phase

        bpl    A_Dc03          * multiply by sign of sin space
        ldd    dem_sp
        subd   recr
        std    dem_sp
        bra    A_Dc04

A_Dc03  ldd    dem_sp
        addd   recr
        std    dem_sp

A_Dc04  ldaa   recr+1         * get received input
        bpl    A_Dc05         * get absolute value
        nega

A_Dc05  tab
        mul

        std    recr           * stash it away

        ldaa   car_ph         * get mark phase
        adda   clk_pht

        bita   #$40           * get sign of twice mark phase
        beq    A_Dc06         * multiply by twice mark phase
        ldd    dem_mke        * and subtract
        addd   recr
        std    dem_mke
        bra    A_Dc07

A_Dc06  ldd    dem_mke

```

```

        subd    recr
        std     dem_mke

A_Dc07  ldaa    car_ph          * get space phase
        suba    clk_pht

        bita    #$40           * get sign of twice space phase
        beq     A_Dc08         * multiply by twice space phase
        ldd     dem_spe        * and subtract
        addd    recr
        std     dem_spe
        bra     A_Dc09

A_Dc08  ldd     dem_spe
        subd    recr
        std     dem_spe

A_Dc09  ldaa    clk_pht        * check for clock rolled over
        eora    clk_ph         * if bit 6 has changed, then cloc
        bita    #$40           * has rolled over
        bne     A_Dc0

A_Dc0
A_Dc00  ldd     clk_pht
        std     clk_ph
        ldaa    clk_inc+2      * make up for 2-byte add in main loop
        ldab    #400*samples/baudr
        mul                    * extra increment to add
        addd    clk_ph+1
        std     clk_ph+1
        ldaa    clk_ph
        adca    #0
        staa   clk_ph         * stash it away permanently

        ldaa    car_inc+2      * make up for 2-byte add in main loop
        ldab    #400*samples/baudr
        mul                    * extra increment to add
        addd    car_ph+1
        std     car_ph+1
        ldaa    car_ph
        adca    #0
        staa   car_ph         * stash it away permanently

        brset   clk_ph #$40 A_Dc1
        beet    porta,x #$08
        ldd     dem_sp
        subd    dem_mk
        addd    dem_q_l
        bra     A_Dc2

A_Dc1   bclr    porta,x #$08
        ldd     dem_mk
        addd    dem_sp
        addd    dem_i_l        * get bit to send

```

```

A_Dc2   tab
        sorb    old_out
        andb    #$80
        staa    old_out          * save old demod output
        pshb

        ldd     dem_mk          * shift all demods
        addd    dem_sp
        std     dem_i_1

        ldd     dem_sp
        subd    dem_mk
        std     dem_q_1

        ldd     dem_mke        * get carrier error
        addd    dem_spe
        std     car_phe+1

        ldd     dem_mke        * get clock error
        subd    dem_spe
        std     clk_phe+1

        ldd     #0             * zero out channel accumulators
        std     dem_mk
        std     dem_sp
        std     dem_mke
        std     dem_spe

        cli                    * it's safe to interrupt

A_Dc3   pulb
        brclr   scsr,x #$80 A_Dc3 * make rs232 available
        stab    scdr,x          * and transmit

A_Dc8   ldaa    clk_phe+1      * sign extend clock error to mab
        bpl     A_dc11
        ldaa    #$ff
        bra     A_Dc12

A_Dc11  clra
A_Dc12  staa    clk_phe
        staa    clk_phu

        ldaa    car_phe+1     * sign extend carrier error
        bpl     A_Dc13
        ldaa    #$ff
        bra     A_Dc14

A_Dc13  clra
A_Dc14  staa    car_phe
        staa    sar_phu

        ldd     car_phe+1     * low pass filter the carrier error
        std     scratch

```

```

        ldd     lockcrr+1          * update the in lock filter
        subd   lockcrr
        std    lockcrr+1
        ldaa  lockcrr
        abca  #0
        staa  lockcrr

        ldd     scratch          * get the absolute value of
        bpl    A_Dc14a          * the phase error

        ldd     #0
        subd   scratch

A_Dc14a addd   lockcrr+1          * low pass filter the absolute value
        std    lockcrr+1          * of the phase error for
        ldaa  #0                  * lock info
        adca  lockcrr
        staa  lockcrr

* Mighty Machineria's spaghetti code city!
* or, I wish I had a faster processor
A_Dc14e ldd     lockcrr          * now get filter value & check
        brset  porta,x #$40 A_Dc14c * branch if loop locked
        subd   #lockthr-lockhys * check if loop should lock
        bhi    A_Dc14f
        bset   porta,x #$40      * if should lock, lock
        bra    A_Dc14g

A_Dc14c subd   #lockthr          * check if loop should inlock
        bla    A_Dc14g
        bclr   porta,x #$40      * if should unlock, unlock

A_Dc14f ldaa  #cabwi            * unlocked loop gain
        bra    A_Dc14d

A_Dc14g ldaa  #cabwl            * locked loop gain

A_Dc14d ldy   #car_phe          * multiply carrier phase
        jar   shift_r          * by high-frequency

        ldy   #clk_phe          * multiply clock phase
        ldaa  #clbw
        jar   shift_r

        ldaa  #$3
        ldy   #0

A_Dc15 ldab   car_phe,y         * set car_phe = car_phe
        stab  car_phe+1,y       * and clk_phe = clk_phe
        ldab  clk_phe,y
        stab  clk_phe+1,y
        iny
        deca

```

```

hne      A_Dc15

ldy      #car_phu+1      * multiply carrier phase
ldd      lockcrr        * by zero frequency
cpd      #lockthr+lockhys * use zero frequency if
bls      A_Dc15a        * close to lock
ldaa     #cazri         * else use high frequency
bra      A_Dc15b
A_Dc15a ldaa     #cazrl
A_Dc15b jar      shift_r

sec
ldd      car_frq+2      * update the carrier frequency
sbc     car_phu+3      * with carry from phase error
sbca    car_phu+2
std     car_frq+2
ldd     car_frq
sbcb    car_phu+1
sbca    car_phu
std     car_frq

cpd      #carmax        * check for carrier overflow
bls      A_Dc16
ldd      #carmax        * if overflow, set to overflow
std     car_frq
bra      A_Dc17

A_Dc16  cpd      #carmin        * check for carrier underflow
bhs      A_Dc17
ldd      #carmin        * if underflow set to minimum
std     car_frq

A_Dc17  ldy      #clk_phu+1      * multiply clock phase
ldaa     #clzr         * by zero frequency
jer      shift_r

sec
ldd      clk_frq+2      * update the clock frequency
sbc     clk_phu+3      * with carry from phase error
sbca    clk_phu+2
std     clk_frq+2
ldd     clk_frq
sbcb    clk_phu+1
sbca    clk_phu
std     clk_frq

cpd      #clkmax        * check for carrier overflow
bls      A_Dc18
ldd      #clkmax        * if overflow, set to maximum
std     clk_frq
bra      A_Dc19

A_Dc18  cpd      #clkmin        * check for carrier underflow
bhs      A_Dc19

```

```

        ldd      #clkmin          * if underflow, set to minimum
        std      clk_frq

A_Dc19 sei
        ldd      car_frq+1       * update the carrier increment
        subd     car_phe+1
        std      car_ing+1
        ldaa     car_frq
        sbca     car_phe
        staa     car_inc

        ldd      clk_frq+1       * update the clock increment
        subd     clk_phe+1
        std      clk_ing+1
        ldaa     clk_frq
        sbca     clk_phe
        staa     clk_inc

        rti

*****
* This routine finds  $(\cos(a+b) + \cos(a-b))/2 = \cos(a)\cos(b)$ 
* the arguments come in in registers a and b, and are returned in a

cosab  aba          * get the sum of the phases
        psha        * save 'em
        sba         * get the difference
        sba
        ldy         #cos_tab    * get base of cosine table
        tab
        aby         * add offset to proper result

        ldaa        ,y          * get  $\cos(a-b)/2$ 

        pulb        * retrieve the sum
        ldy         #cos_tab    * get base
        aby         * add offset
        adda        ,y          * get  $(\cos(a-b) + \cos(a+b))/2$ 

        rts             * this takes 50 instruction cycles

*****
* This routine shifts the 3 byte field pointed to y
* a-8 places to the right (an 8-bit right shift is assumed)
* if a<8 then the field is shifted the appropriate amount to the left

shift_r suba        #$08        * line up a
        beq         sh_end      * if a = 8 then shift not necessary
        bmi         sh_lft      * if a < 8 then shift left

sh_rt  asr          0,y         * shift one place to the right

```



```

        ror    1,y
        ror    2,y
        deca
        bne    sh_rt          * repeat as necessary
        bra    sh_end

sh_lft  asl    2,y          * shift one place to the left
        rol    1,y
        rol    0,y
        inca
        bne    sh_lft       * repeat as necessary

sh_end  rts

*****
SCIint  rti                * hold space for this routine
*****
* interrupt routine to service the spi.  The two things it needs to send
* are the display information and the synthesizer information
* the order of importance is:
*   if display transfer in process, finish it
*   if synthesizer transfer is in progress, finish it
*   if synthesizer transfer is pending, start it
*   if display transfer is pending, start it

SPIint  ldx    #$1000      * make sure you have the correct index

        brclr  spiflag #$10 spi1  * if not synth latch, jump
        bset   portd,x #$20      * else latch onto the synthesizer
        nop
        nop
        bclr   portd,x #$20
        bclr   spiflag #$10      * indicate transfer done
        bra    spi0             * and go to finish

spi1    ldaa   spar,x        * if spi transfer needed, clear spif

        brclr  spiflag #$04 spi2  * if not synth transfer active, jump
        ldaa   synfreq+1         * else get low-order byte of synth
        staa   spdr,x           * and send it
        bclr   spiflag #$04      * indicate transfer done
        bset   spiflag #$10      * set need latch bit
        bra    spi10            * and go to finish

spi2    brclr  spiflag #$01 spi3  * if not display transfer active, jump
        ldaa   dspfreq+1         * else get low-order byte of display
        staa   spdr,x           * and send it
        bclr   spiflag #$01      * indicate transfer done
        bra    spi0             * and go to finish

spi3    brclr  spiflag #$08 spi4  * if not synth transfer pending jump
        ldaa   synfreq          * else get high order byte of
display

```

```

        staa    spdr,x          * and send it
        bclr    spiflag #$08    * indicate transfer pending
        bset    spiflag #$04    * set transfer pending
        bra     spi0           * and go to finish

spi4    brclr    spiflag #$02 spi0 * if not display transfer pending jump
        ldaa    dspfreq        * else get high order byte of synth
        staa    spdr,x          * and send it
        bclr    spiflag #$02    * indicate transfer pending
        bset    spiflag #$01    * set transfer pending
        bra     spi0           * and go to finish

spi0    brclr    spiflag #$1f spi5 * if operation pending, return
        rti

spi5    bclr    spcr,x #$80     * else disable spi interrupts
        rti

*****
* This interrupt service routine is turned on by the frequency
* knob. It must turn on a debounce timer, which causes the computer
* to ignore any switch inputs until things settle down

upf
downf   ldx     #base          * point to base of I/O pile

        ldd     tcnt,x
        addd    #dbtime        * delay
        std     toc2,x         * set debounce for 80ms
        bclr    tflg1,x #$bf    * clear the debounce flag
        bset    tmsk1,x #$40    * disable debounce interrupt
        bclr    tflg1,x #$fc    * clear the switch flags

        rti

*****
* If the switches are open for 20ms, this routine is activated. It turns
* off the repeat interrupt and turns itself off

dbounce ldx     #base
        bclr    tmsk1,x #$40    * disable debounce interrupts

        ldaa    porta,x        * get switch settings
        anda    #$03

        staa    scratch
        lara    eora           * make into binary
        eora    scratch

        tab
        suba    oldknob        * save value
        stab   oldknob        * get increasing/decreasing info
        * save current knob position

```

```

        bita    #$01                * check for valid knob read
        beq     dbnc0              * if not valid, ignore

        bita    #$02                * check for direction
        bne     dbnc1              * if down, decrement frequency
        ldaa   #$01                * else increment frequency
        adda   freq
        daa
        staa  freq
        bra    dbnc2

dbnc1  ldaa   #$99                * decrement frequency
        adda   freq
        daa
        staa  freq

dbnc2  jar    syncalc              * start the synthesizer
        jar    dspcalc              * and the display
        ldaa   #$f4                * turn on demod seek
        staa  lockcrr

dbnc0  rti

*****
* calculate number to load into synthesizer, and start the spi

* calculate the number

syncalc  ldaa   freq                * get bcd offset byte
        tab
        andb   #$f0                * save high digit in B
        anda   #$0f                * low digit in A

        lsrb
        aba
        lsrb
        lsrb
        aba
        lsrb
        aba
        tab
        ldaa   #0

        lslb
        addd   #synofst            * times two for synthesizer
        std    synfreq             * add to offset for IF and base
        * frequency and store

* start the spi

        bclr   spiflag#$14         * clear the transfer in progress flags
        bset   spiflag#$08         * set the transfer pending flag
        bset   spcr.x #$80         * enable spi interrupts

        rts

```

```

*****
* calculate bit pattern for the display and start the transfer
* calculate bit pattern for the display

dspcalc    ldaa  freq                * get decimal offset
           adda  #dspofst            * add to base
           daa
           tab                          * save result
           ldaa  #0
           rola                          * get carry flag into A

           pshb                          * save frequency
           psha                          * save top bit

           lsrb                          * put upper digit into lower
           lsrb
           lsrb
           lsrb
           ldy  #disptab              * get base of look-up
           aby                          * get vector of display byte
           ora  ,y                      * calculate most significant
           staa dspfreq                * bit pattern

           pula                          * recover top bit
           eora #$01                   * and invert it

           pulb
           andb #$0f                   * zero out top digit
           ldy  #disptab              * get base of look-up
           aby                          * get vector of display byte
           ora  ,y                      * calculate least significant
           staa dspfreq+1              * bit pattern

* now start the transfer

           bclr  spiflag    #$01        * clear the transfer in progress flags
           bset  Spiflag    #$02        * set the transfer pending flag

           bset  spcr,x    #$80        * enable spi interrupts

           rts

*****
* This is a table of interrupt vectors for my radio program

           org    $ffd6
           fdb    SCIint              * RS232 service routine
           fdb    SPIint              * serial peripheral service routine

           org    $ffe4

```

```
fdb  A_Dcomp          * A/D complete
fdb  dbounce          * debounce timer done
fdb  sample           * sample timer done
fdb  downf            * frequency decrement
fdb  upf              * frequency increment
fdb  startup          * Demodulator service routine

org  $fff6
fdb  startup          * Various error conditions
fdb  startup          * all of which require a
fdb  startup          * machine reset
fdb  startup
fdb  startup
```

3 Appendix C

3.1 Transmitter Software

This software was written for the IBM PC in small C. This language is a subset of regular C. Specifically, it has only integers and characters for data types, it is only capable of one-dimensional arrays, it doesn't use STRUCT or UNION data types, and it does not implement all of the control structures of ordinary C. None the less, any competent C programmer should be able to read this code.

```

/*****
/*
/*          erate.c          */
/*          Revision 3.1    */
/*          Timmy 4-21-90   */
/*          */
/*    This program runs the interface between an IBM PC and my MSK    */
/* transmitter and receiver.                                          */
/*          */
/*    Transmissions consist of frequency commands to the synthesizer. */
/* These are binary coded, 24 bit long numbers, with the number     */
/* representing the transmit frequency in hertz. Thus, to transmit on */
/* a center frequency of 300kHz, you tell the synthesizer to produce  */
/* 300.1kHz for a mark, and 299.9kHz for a space. The synthesizer     */
/* takes care of loading the frequencies on a 400Hz interval, but the  */
/* transmit routine must keep up with this 400Hz rate to load the new  */
/* frequency for each bit. Transmit information is sent out on the    */
/* parallel printer port.                                             */
/*          */
/*    The transmitted sequence is a 10-bit pseudo-random sequence*/
/* generated by the program. The transmitted information is delayed, */
/* then compared to the received information. The number of bit errors */
/* is counted, and every once in a while the bit error rate is written */
/* to the screen.                                                    */
/*          */
/*****/

#define      null  0
#define      sync  15
#define      up    0
#define      low   1
#define      #asm
printer      equ   378h
#define      #endasm

/* current set of stuff being shifted out */

int  shd;

```

```

/* and we need to know the frequency that we are transmitting*/
int  freq, limit, cumulate;

/* correlation matrix for received data */
int  cnt[8], n, m;
#define  nmin  3
#define  nmax  6

/* and some communications buffer stuff */
char  rr, buff[2], *push, *pull;
int  commint[2];

/* send a byte to the synthesizer */
send()
{
    #asm
    mov  dr,printer ;get port value
    mov  al,bl
    out  dx,al      ;send the byte

    inc  dx

qzsend1:
    in   al,dx      ;get status of transmitter
    and  al,80h     ;check for busy
    jz   qzsend1    ;wait for not busy

    inc  dx
    mov  al,0dh     ;and latch
    out  dx,al
    mov  al,0ch
    out  dx,al
qzsend2:
    dec  al
    jnz  qzsend2
    #endasm
}

/* transmit a mark or a space */
transmit()
{
    #asm

    mov  dr,printer + 1 ;check for transmit ready
qztran1:
    in   al,dx
    and  al,40h       ;if not ready then
    jnz  qztran1      ;wait for transmit ready

    cli                ;block any interrupts

    mov  cx,ss:qzfreq
    mov  ax,ss:qzshd ;get shift register

    and  ah,1

```

```

        jz     qztran2
        add    cx,64h           ;if lsb=1 send mark
        jmp    qztran3

qztran2:
        sub    cx,64h           ;if lsb=0 send space

qztran3:
        mov    bl,4             ;first byte always 4
        call   qzsend

        mov    bl,ch           ;send next
        call   qzsend
        mov    bl,cl           ;two bytes
        call   qzsend

        sti                    ;reenable interrupts

        #andasm
    }

/* make next entry in random sequence */
next()
{
    #asm
    mov    bp,sp               ;this do-hicky implements a
    mov    bx,[bp + 2]         ;ten-bit shift register sequence
    mov    al,bl               ;on the appropriate operand
    mov    cl,3                 ;tap on third bit
    shr    al,cl
    and    al,1h
    xor    al,bl               ;get the new bit to insert
    and    al,1h               ;filter out other bits
    shl    al,1                ;line up to tenth bit
    shr    bx,1                 ;shift operand
    or     bh,al               ;insert bit
    #endasm
}

/* Get the transmit frequency */
getfreq(f)
int *f;
{
    int not_valid;
    not_valid = 1;
    while(not_valid)
    {
        puts("enter the transmit frequency in kilohertz");
        *f = getint(10);
        puta("\n\r\0");
        not_valid = (*f >= 327) | (*f <= 262);
        if (not_valid)
            pl("frequency must be between 262kHz and 327kHz");
    }
}

```



```

        *f = *f*1000 + 500
    }

/* this is a VERY s^^^y error handler */
error();

/* check for a byte from the receiver and return with it */
receive()
{
    #asm

    mov     bp,ss:qxpull
    mov     al,qzbuff[bp]      ;get current receive character
    ror     bp,1
    mov     ss:qzpull,bp
    cbw
    mov     bx,ax

    #endasm
}

/* increment the bit-in-error counter if the bits don't match */
count()
{
    #asm

    mov     bp,sp      ;get a nice little pointer
    mov     bx,[bp + 2] ;get receive bit into bl
    and     bl,80h     ;strip out bit
    mov     ax,ss:qzshd ;get transmit bit into al

    mov     cx,ss:qzn
    mov     bp,cx
    rol     ax,cl
    cmp     al,bl
    js     qzcnt1
    sal     bp,1
    inc     qzcnt[bp]
qzcnt1:

    #endasm
}

/* initialization and interrupt routine for the communications port */
comm()
{
    #asm

    push    ds
    push    es

    mov     ax,350ch

```

```

int    21h          ;get communications port interrupt vector
mov    ss:qzcommint,bx
mov    bx,es
mov    ss:qzcommint[2],bx

mov    dx,cs
mov    ds,dx
mov    dx,offset rx10
mov    ax,250ch
int    21h          ;set interrupt vector to interrupt service

mov    ax,00c3h     ;4800 baud, 1 stop bit, no parity, 8 bit word
mov    dx,0
int    14h          ;initialize the comm port

mov    dx,40h       ;get segment
mov    ds,dx
db     8bh,16h,00,00 ;(mov dx,[0]) get comm port address

cli                    ;make sure there are no interrupts

add    dx,4         ;modem control register
mov    al,0fh       ;DTR, RTS, OUT1, OUT2
out    dx,al

in     al,21h       ;get interrupt masks
and    al,0efh      ;turn off comm port mask
out    21h,al

sti

;***** reset the interface *****

sub    dx,3         ;interrupt enable register
mov    al,0
out    dx,al

add    dx,4         ;line status register
in     al,dx

inc    dx           ;modem status register
in     al,dx

db     8bh,16h,00,00 ;(mov dx,[0]) get comm port address
in     al,dx        ;received character

add    dx,2         ;interrupt ID register
in     al,dx

dec    dx           ;enable interrupts on UART
mov    al,5         ;received character and rx line status
out    dx,al

mov    ss:qzpush,00

```

```

mov    ss:qzpull,00      ;initialize pointers

mov    al,20h           ;end-of-interrupt
out    20h,al

pop    es
pop    ds
ret

```

;This is the interrupt service routine

```

rxio:
sti

push   dx
push   ax
push   es
push   ds
push   bp

mov    al,20h
out    20h,al           ;reenable interrupt controller

mov    ax,stack        ;all data is in the stack segment
mov    es,ax           ;(I am using the extra segment here)

mov    dx,40h          ;get segment
mov    ds,dx
db     8bh,16h,00,00   ;(mov dx,[0]) get comm port address

add    dx,
in     al,dx
test   al,01
jnz    rxil

cmp    al,06           ;test for rx line status
jne    rxil2
add    dx,3           ;if line status, read line status
in     al,dx
jnp    rxil

rxil2: db 8bh,16h,00,00 ;(mov dx,[0]) get comm port address
in     al,dx          ;assume that other thing is rx register
mov    bp,es:qzpush
mov    es:qzbuff[bp],al ;put byte into buffer
xor    bp,01         ;toggle pointer
mov    es:qzpush,bp

rxil:  pop    bp
       pop    ds
       pop    es
       pop    ax
       pop    dx

```

```

    iret

    #endasm
}

/* set the display for its initial value */
initdsp()
{
    int    n;
    char  s[7];
    clrscr();

    n = 23;
    while(--n) putchar('-');    /*make a line of dashes*/

    n = freq/100;                /* convert frequency back to 100Hzs */
    if (n < 0) n = 3276 + n;
    else n = 2622 + n;          /* adjust for signed arithmetic */

    intoatr(s,n);                /* display number */
    n = 0;
    while (++n < 5) a[n] = s[n+1];    /* stick in decimal point */
    s[4] = '.';                  /* for kHz */
    puts("    transmit frequency =");
    puts(s);
    puts("kHz    ");

    n = 23;
    while(--n) putchar('-');    /* finish off with more dashes */
}

/* turn off interrupts and exit */
goodby()
{
    #asm

    cli
    in    al,21h
    or    al,10h                ;mask communications interrupt
    out   21h,al

    push  ds
    mov   ds,ss:qzcommint[0]
    mov   dx,ss:qzcommint[2]
    mov   ax,250ch
    int   21h                    ;reset interrupt vector
    pop   ds

    mov   ax,00c7h                ;4800 baud, 1 stop bit, no parity
    mov   dx,0                    ;(and turn off interrupts)
    int   14h                    ;initialize the comm port

    sti

```

```
#endasm

exit();
}

main()
{
    int    syncflag, running, rx;

    getfreq(&freq);
    initdisp();
    shd = 1;
    n = 8;
    while(--n) cnt[n] = 0;
    comm();

    /* now go into main tranceive loop */
    while (1)
    {
        /* if character, see what operator wants */
        if (pcdoa(11,0))
        {
            /* if character is a space, then reset */
            /* the cumulative error count */
            transmit();
            receive();
            if (getchar() == ' ')
            {
                limit = 0;
                cumulate = 0;
            }
            else
            {
                puts("\n\rdo you wish to stop?");
                if (getchar() == 'y') goodbye();
                puts("\n\r");
                getfreq(&freq);
                initdisp();
            }
        }

        m = 20;
        while (--m) transmit();

        /* collect data to find which channel is correct */
        running = 100;
        while (running-->0)
        {
            n = 3;
            while(++n <= 5)
            {
                transmit();
                count(receive());
            }
        }
    }
}
```

```

        shd = next(shd);
    }
}
transmit();
receive();
/* if the sequence is aligned to n=5 then realign */
if (cnt[4] > cnt[5]) receive();

transmit();
receive();
n = 4;
cnt[4] = 0;

/* count up the errors in 1000 bits */
running = -20;
while (++running <= 1000)
{
    if (running);
    else m = cnt[4];
    transmit();
    count(receive());
    shd = next(shd);
}
cnt[4] = cnt[4] - m;
/* check for believable data and */
/* add errors to cumulative total */
if (cnt[4] < 450)
{
    cumulate = cumulate + cnt[4];
    limit++;
}
transmit();
receive();
/* now display, then clear the correlation register */
#asm
mov  ah,2          ;locate the cursor to the left top
mov  dx, 0200h    ;of screen
mov  bh, 0        ;page zero
int  10h
#endasm
puts("errors per 1000 =");
putint(cnt[4]);
transmit();
receive();

/* and display, then the cumulative register */
#asm
mov  ah,2
mov  dx,0400h
mov  bh, 0        ;page zero
int  10h
#endasm
puts("errors per");
putint(limit);

```

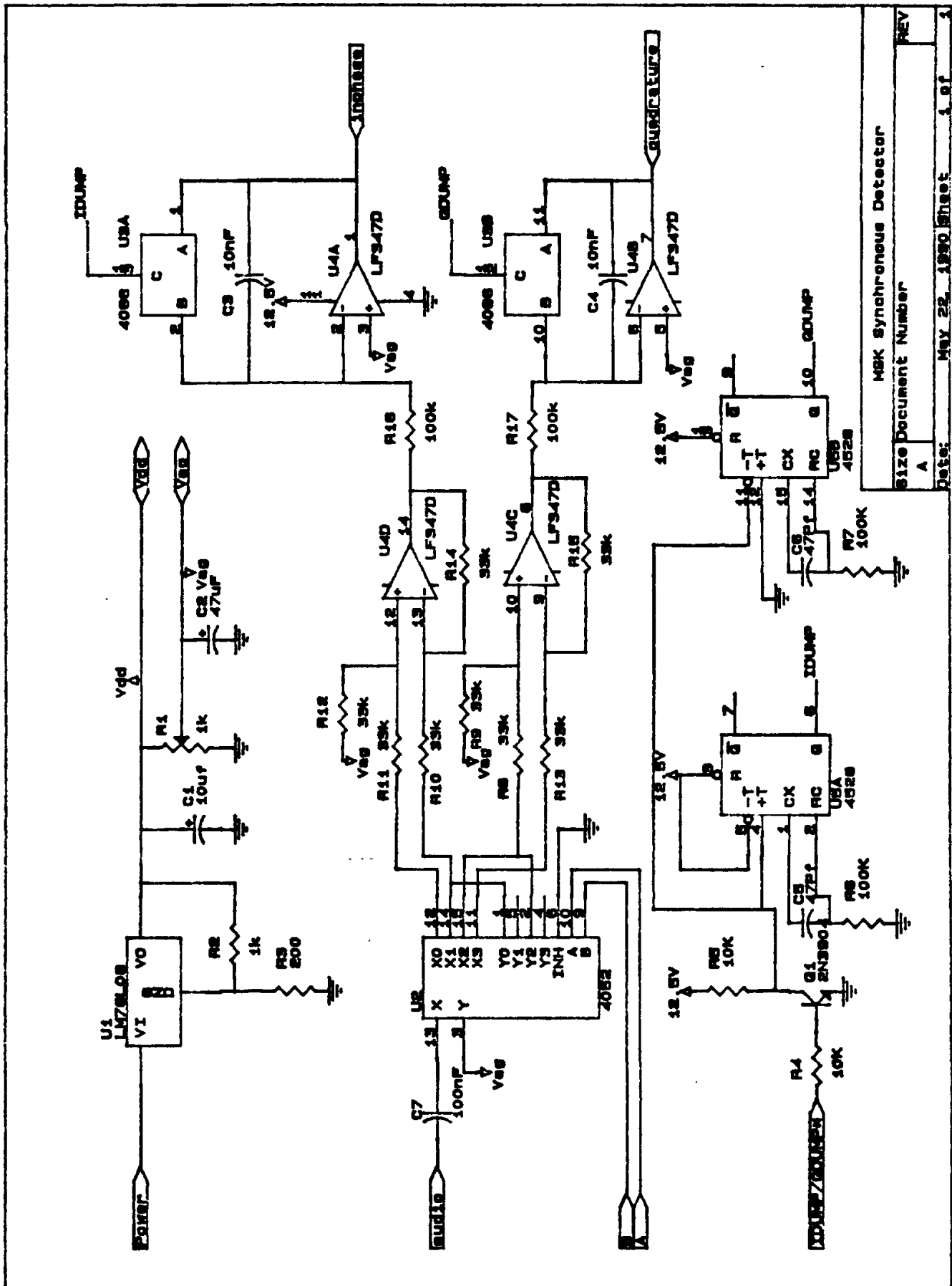
```
puts("000 =");
putint(cumulate);
transmit();
receive();

/* check for overflow */
if (limit > 9999)
{
    limit = 0;
    cumulate = 0;
}
transmit();
receive();

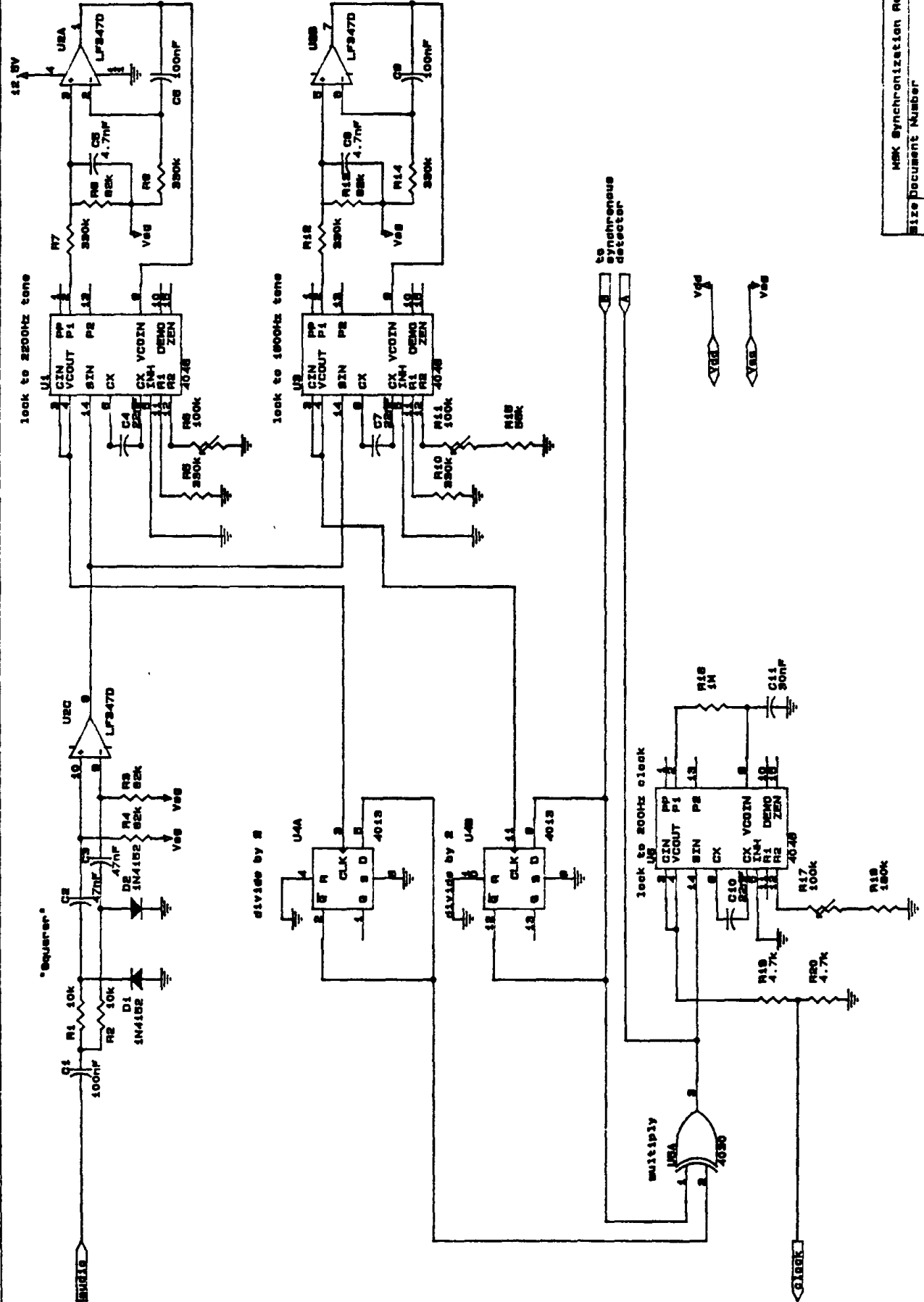
n = 3;
while(++n <= 5) cnt[n] = 0;
}
```

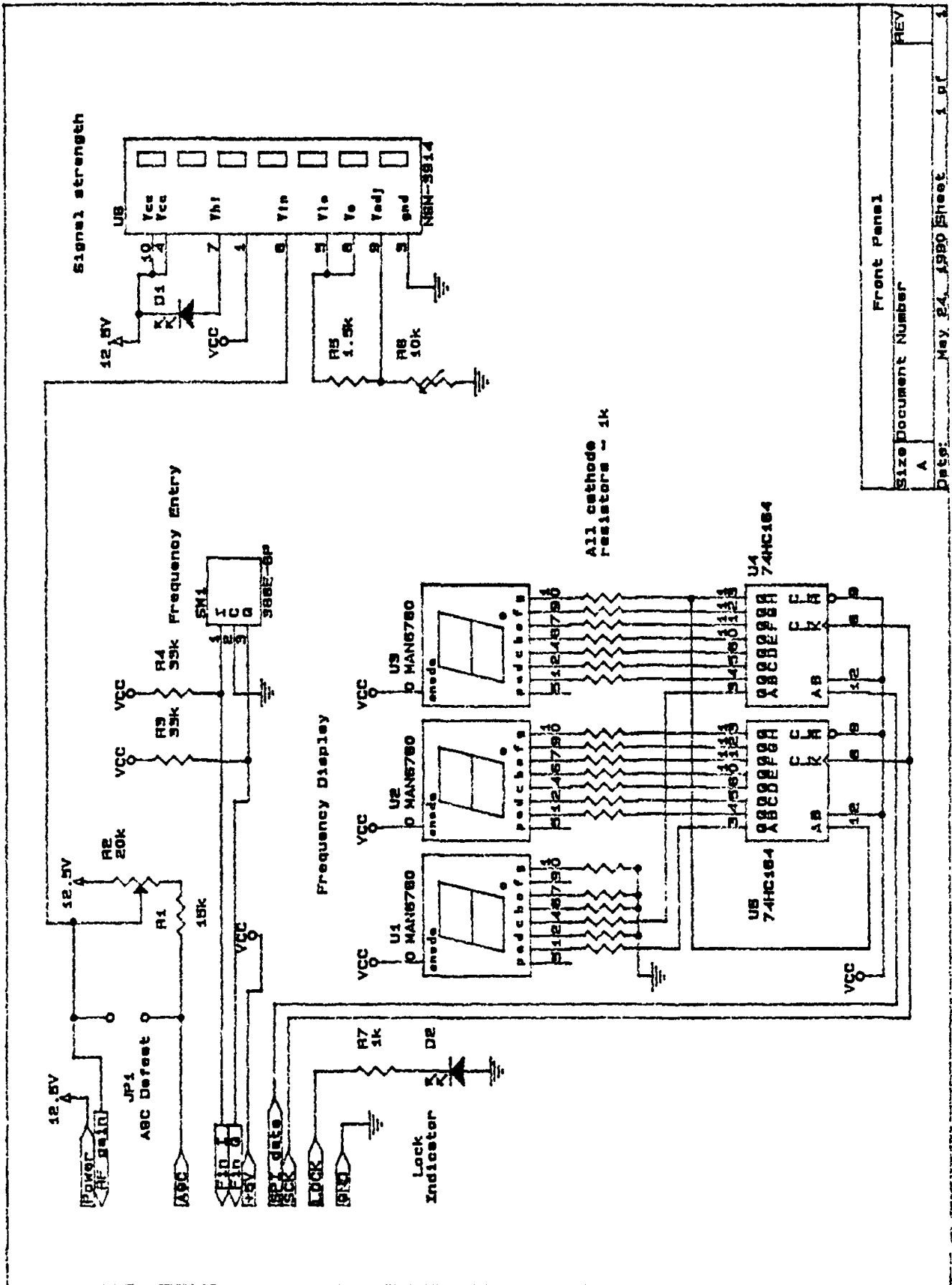
4 Appendix D

4.1 *Schematics*

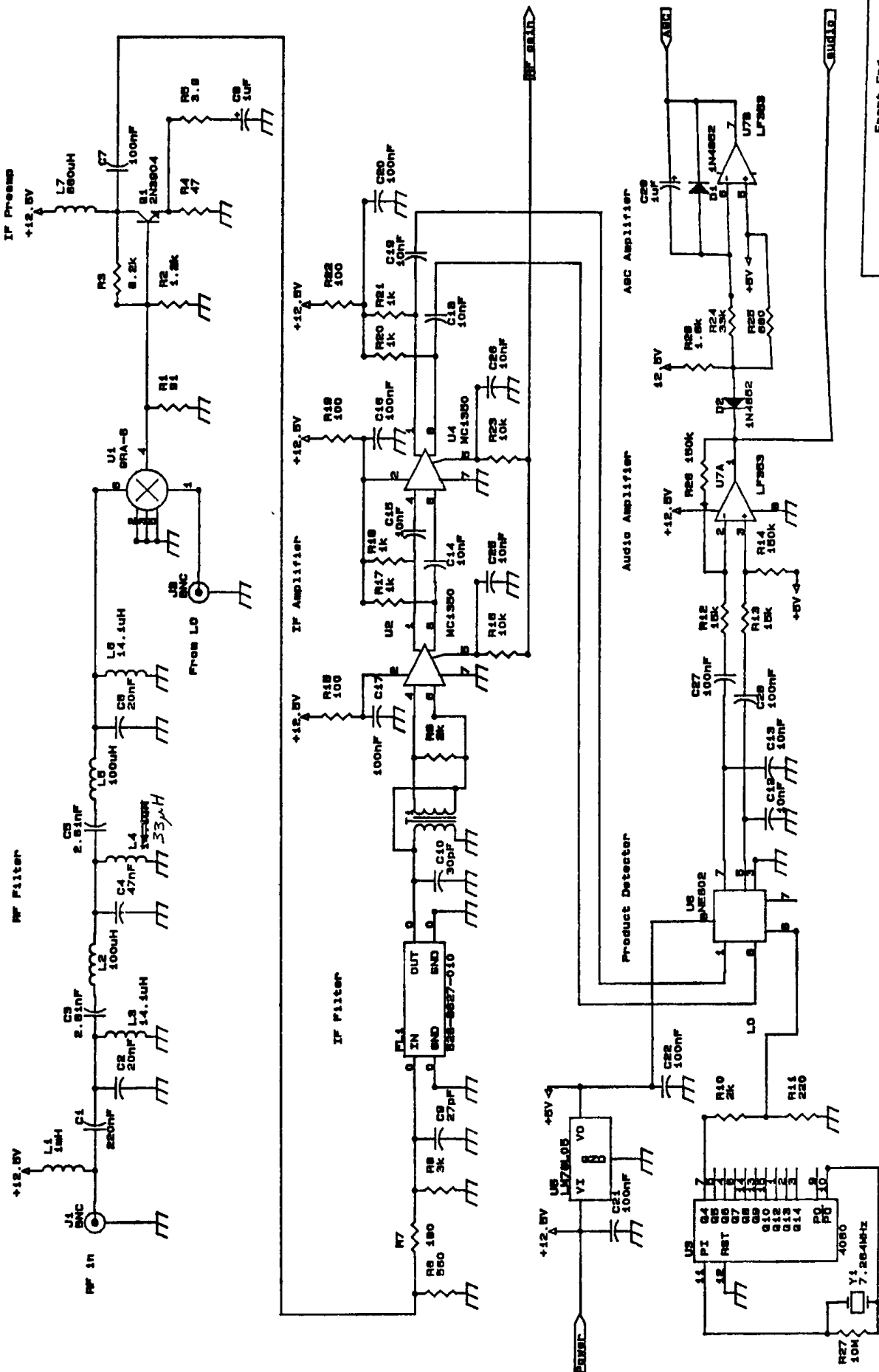


MBK Synchronous Detector
 Size Document Number
 A
 Date: May 22, 1990 Sheet 1 of 1
 REV





| | |
|----------------------|---------------------------|
| Front Panel | REV |
| Size Document Number | A |
| Date: | May 24, 1980 Sheet 1 of 1 |



| | |
|----------------------|------------------------|
| REV | 1 |
| Size Document Number | B |
| Date | MAY 24 1980 FRONT 1.01 |

Front End

5 Bibliography

- [1] R De Buda Coherent demodulation of Frequency-Shift Keying With Low Derivation Ratio
IEEE Transactions, 1972, COM-20, pp. 429-435
- [2] S. Pasupathy, Minimum Shift Keying: A Spectrally Efficient Modulation IEEE
Communications Society Magazine, July 1979, Vol. 17, No. 4, pp. 14-22
- [3] Floyd Gardner, Phaselock Techniques Second Edition, John Wiley & Sons
- [4] Letter from J.T. Noblet, LTJG, US Coast Guard to Dr. Rudolph M. Kalafus dated April 24, 1984
- [5] Regional Arrangement concerning Maritime Radiobeacons in the European Area ITU, Paris –
1951
- [6] M.F. Ruane, P.E. Enge, K Olson Test Experience using Marine Radiobeacons for DGPS
Communications Record of the 1988 IEEE Position, Location, and Navigation Symposium,
Orlando, FL Nov. 1988
- [7] Motorola, Inc CMOS/NMOS Special Functions Data 1988 pp. 6-84 to 6-94
- [8] Dan H. Wolaver, Phase-Locked Loop Circuit Design Prentice-Hall, New York, to be published
in 1991
- [9] Harry L. Van Trees, Detection, Estimation, and Modulation Theory John Wiley & Sons, New
York, 1968
- [10] Donald A. Feldman, An Atmospheric Noise Model with Application to Low Frequency
Navigation Systems Thesis, MIT, Massachusetts Institute of Technology

- [11] M. F. Ruane, P. K. Enge, K.E. Olson and D. Petrasewski, Sea Trials, Message Delay and Network Design for SGPS/Radiobeacons Proceedings of the Second International Technical Meeting of the Satellite Division of ION, Colorado Springs Sept 1989
- [12] P.K. Enge, R. Kalafus, M.F. Ruane, Differential Operation of the Global Positioning System IEEE Communications Magazine, vol. 26, no. 7, July 1988
- [13] C.H. Houppis, G.B. Lamont Digital Control Systems – Theory, Hardware, Software McGraw-Hill, Inc., New York, 1985
- [14] Motorola Inc., Motorola Linear and Interface ICs 1988, pp 9-15 to 9-18
- [15] G.C. Clark, J.B. Cain Error-Correcting Coding for Digital Communications Plenum Press, New York, 1981

About the Author:

Tim Wescott has a Master's degree in Electrical Engineering and over 20 years of experience embodying control and communications theory in software. He has designed and written numerous communications and digital control algorithms, which have been implemented on 8, 16, and 32 bit conventional processors as well as DSP chips. He has extensive experience with turning difficult theoretical problems into practical working solutions that can be understood and maintained by ordinary engineers. He has a working knowledge with C, C++ and assembly language for a number of processors, and can write algorithms that are easily translated into practice, or he can translate these algorithms into practice in the above mentioned languages.

Mr. Wescott currently owns and operates Wescott Design Services, where he provides expertise to clients all over the USA and in Canada and England.