Theory, Design and Implementation of a Digital Receiver for the Advanced Communications Technology Satellite (ACTS) Beacons

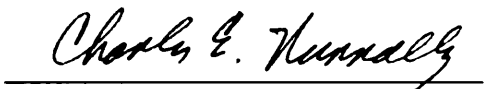
by

William R. Sylvester Jr.

Thesis submitted to the Faculty of
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

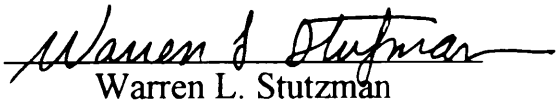
Master of Science
in
Electrical Engineering

APPROVED:



Charles E. Nunnally, Chairman

A. A. (Louis) Beex



Warren L. Stutzman

Timothy Pratt

August, 1992
Blacksburg, VA

c.2

Abstract

This document describes the theory, design and implementation of a digital receiver designed for the ACTS propagation experiments. The Virginia Tech Satellite Communications Group is designing, constructing and distributing eight ACTS propagation terminals (APTs) under a contract with NASA. The terminals will measure the received signal power from the ACTS satellite beacons (20 GHz, 27.5 GHz) in various climates at different elevation angles. The resulting signal power measurement, radiometer and weather measurement data will be used to characterize atmospheric effects on signal propagation at K_a band.

Each APT system will contain two identical, independent digital receivers; one 20 GHz channel and one 27.5 GHz channel. The algorithm implemented on each receiver utilizes the results of multiple fast Fourier transforms to reliably identify the carrier tone amidst neighboring modulation tones. The acquisition procedure can reliably identify the carrier signal at signal-to-noise ratios down to 10 dB in a 20 Hz bandwidth (23 dBHz in a 1 Hz bandwidth, -26 dB relative to clear air conditions on the APT system) in 2 seconds. The receiver then uses a comb filter, two FIR filters and additional FFTs to produce power measurements accurate to 0.1 dB at signal-to-noise ratios down to 5 dB in a 20 Hz bandwidth (18 dBHz in a 1 Hz bandwidth, -31 dB relative to clear air conditions on the APT system) at 1 Hz and 20 Hz sampling rates. The algorithm also provides several supplemental functions including a software selectable detection bandwidth from 2 Hz to 50 Hz (1 Hz increments), spectrum analyzer type output for a 303.333 kHz bandwidth centered on the current carrier frequency and carrier frequency estimates accurate to ±0.5 Hz.

A custom digital signal processing platform has been designed and constructed to implement the receiver algorithm. The system is comprised of an Analog Devices ADSP21020 microprocessor running at 20 MHz, a high

speed, 12-bit analog-to-digital converter with user selectable clock rates from 1 Hz to 1 MHz (0.00463 Hz increments), an analog waveform output channel which has low phase noise and a 1 Hz to 5 MHz output range (0.00463 Hz increments), a serial port, 5 Mbits of RAM (expandable to 20 Mbits) and a bootstrap loader which downloads application-specific executable code through the serial port. The entire system can be constructed for approximately $1600.

The Virginia Tech Satellite Communications Group will deliver eight ACTS propagation experiment terminals in 1993 before ACTS is launched in July, 1993. Consequently, it was not possible to test the receiver using the ACTS beacons. The results of accuracy and linearity tests as well as the results of tests utilizing the OLYMPUS 20 GHz beacon are, however, presented. These tests confirm the accuracy and reliability of the receiver algorithm.

The Virginia Tech Satellite Communications Group is currently constructing 16 receivers for use in 8 ACTS propagation terminals which will be distributed to experimenters across North America early next year.

Abstract

Acknowledgments

I would like to express my appreciation to a number of people without whose efforts this project would not have been successful.

I began my graduate studies under the guidance of Dr. John C. McKeeman. Although Dr. McKeeman left Virginia Tech before I was able to complete my graduate career, I learned a great deal from him. Dr. McKeeman not only provided technical guidance while I worked for him, he was also a friend.

Dr. Charles Nunnally, Dr. Warren Stutzman and Dr. Timothy Pratt furnished invaluable encouragement and assistance after Dr. McKeeman left Virginia Tech. Each of these professors contributed insights which enabled the APT receiver algorithm to evolve into a functional subsystem which could be replicated and integrated with the rest of the APT terminal hardware.

Dr. A. A. (Louis) Beex provided technical assistance with the digital signal processing theory on which the APT receiver algorithm is based. I have taken digital signal processing classes through the Master's level under Dr. Beex. It is to him that I owe what I consider to be a thorough understanding of digital signal processing fundamentals. Dr. Beex's assistance, both in an out of class, was indispensable.

The technical assistance supplied by Analog Devices Inc. was exceptional. The APT receiver project was a β-test site for the ADSP21020 JTAG in-circuit emulator and associated software. Some of those involved with the APT receiver project were apprehensive about using such a new and relatively untested microprocessor system. The timely, thorough help provided by Chris Russell, an applications engineer for Analog Devices, quickly dispelled these concerns.

Randall Nealy (VPI&SU), Will Remaklus (VPI&SU) and Dave Westenhaver (Westenhaver Wizard Works for Auburn University) made their practical design experience available to me during all stages of the APT receiver development and testing. Without their help, the transformation from a

theory-based algorithm to a functioning physical system would have been very difficult.

The assistance provided by Andrew Predoehl, was instrumental in completing and testing the production version of the APT receiver in a timely fashion. Andrew generated the printed circuit board artwork for the production receivers, performed a thermal analysis to ensure the production receivers would operate over the required temperature range and provided indispensable help in all phases of the receiver tests.

Finally, I would like to thank NASA, not only for funding the APT digital receiver development (NASA grant #NAGW-2335), but for continuing to do so after Dr. McKeeman left Virginia Tech. Dr. McKeeman possessed the majority of the digital design experience in the Virginia Tech Satellite Communications Group. When Dr. McKeeman left Virginia Tech, a more conventional analog receiver was considered for use on the APT system. I would like to thank Dr. John Kiebler and Dr. Jack Chakraborty for having the confidence in me to permit me to continue my work on the digital receiver.

Contents

List of Figures

List of Tables

Chapter 1

Satellite Communications

The communication of large amounts of information at fast rates with fewer faults is becoming an integral part of society. High definition television, for example, requires data transmission rates of 80-100 Mbits/sec[1]. Algorithms for the Strategic Defense Initiative require thousands of small satellites (termed Brilliant Pebbles) to communicate threat information over a world wide space network. These new applications of technology are viable only if communication links are capable of reliably transferring the required volume of information at the required rates. Existing technology is often the limiting factor in the development of new ideas because saturated communication channels cannot handle the new volume, speed, and fault tolerance requirements. To meet new communications requirements engineers are exploring new communication media as well as attempting to expand the capabilities of existing media. New technologies such as high bandwidth multi-mode fiber optic links have evolved tremendously in recent years while existing radio link technology has been refined to produce more reliable, higher bandwidth links over a broader range of frequencies.

When designing a communication link, the engineer must consider many factors such as data volume and data rate requirements, the distance between the transmitter and the receiver, tolerable down time, link installation costs, link maintenance costs, link security, etc. Communication via geosynchronous satellites provides an attractive solution to many transcontinental and long distance intercontinental link problems. A satellite communication link is established when the satellite receives a signal from an earth terminal at an uplink frequency, mixes the signal to a downlink frequency and retransmits an amplified version of the signal toward Earth. The benefits of geosynchronous satellite communication links include:

* The need for costly right-of-way privileges is eliminated.
* Achievable link distance far exceeds that of transmitter/receiver line-of-site links[2].
* Long distance links can be established without the cost of cables and costly cable maintenance (i.e. undersea cable repairs).
* Relatively small, and therefore inexpensive, antennas can transmit or receive a directional signal. Hence, multiple users can operate in one geographic area without interference.
* Mobile/portable operations are possible.

However, communication links utilizing geosynchronous satellites also have the following limitations:

* The number of satellites utilizing a given set of frequencies is limited by antenna design.
* Atmospheric phenomena affect signal propagation.
* Satellites are expensive and, once launched, are not generally accessible for maintenance or repairs.

Today most geosynchronous communications satellites in a direct line-of-site from the United States operate at C band (6/4 GHz) or K_u band (14/11 GHz).

Frequency bands below K_u band are very crowded. Hence, new links must be established at higher frequencies such as K_a band (30/20 GHz).

The magnitude of atmospheric effects is strongly frequency dependent. Atmospheric propagation models must, therefore, be developed for potential new frequency bands in order to predict link performance. The Virginia Tech Satellite Communications Group is currently conducting propagation experiments using the European Space Agency's OLYMPUS satellite and will begin conducting similar experiments using NASA's Advanced Communication Technology Satellite (ACTS) in mid-1993. Data obtained from these experiments will aid in the characterization of atmospheric effects at K_a band.

1.1 Atmospheric Effects on Signal Propagation

Signals transmitted from a geosynchronous satellite remain essentially unaltered until entering the earth's atmosphere[3]. The earth's atmosphere disturbs radio waves by phenomena such as gaseous absorption, stratified layering and rain.

Rapid fluctuations in the level of a signal, known as scintillations, are caused by stratified layering of the atmosphere. As the sun shines on the earth's atmosphere each day, the outer layers of the atmosphere warm more quickly than the inner layers. This temperature gradient causes inhomogeneities in the refractive index of the atmosphere thereby changing the effective length of the ray paths over the coupling region between the transmit and receive antennas. The resulting scintillations can enhance or attenuate the signal as shown in Figure 1-1.

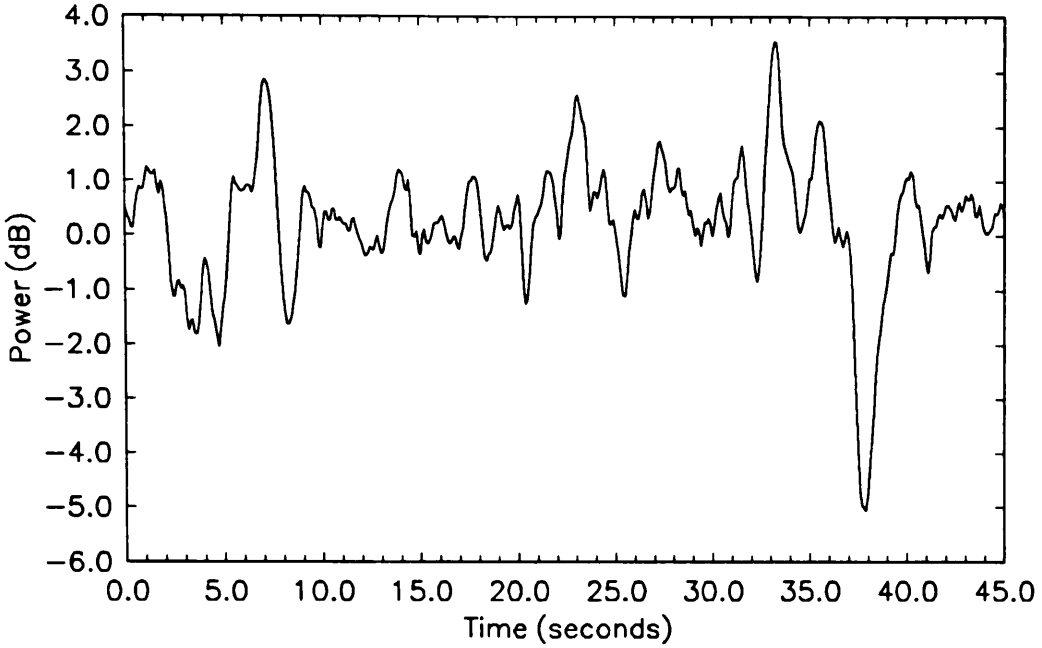


Figure 1-1. Scintillations at 30 GHz.

Microwave signals passing through the earth's atmosphere are attenuated through their interaction with atmospheric gasses. Gaseous absorption attenuates signals in the 10-30 GHz range 1-2 dB depending primarily on elevation angle[4].[1] The predominant clear air attenuating agent over the frequency range of interest is water vapor. High humidity levels or dense clouds along the signal's path may attenuate the signal an additional 1-2 dB.

Rain can attenuate microwave signals so severely that signal reception is no longer possible. For example, Figure 1-2 presents received 12.5, 20 and 30 GHz signal power measurements recorded by Virginia Tech OLYMPUS experiment hardware during an afternoon thunderstorm. As shown, the 20 GHz and 30 GHz signals are attenuated more than 30 dB for more than 10 minutes.

---

[1] ***Elevation Angle:*** the angle between a locally horizontal plane at the earth station and the signal path.

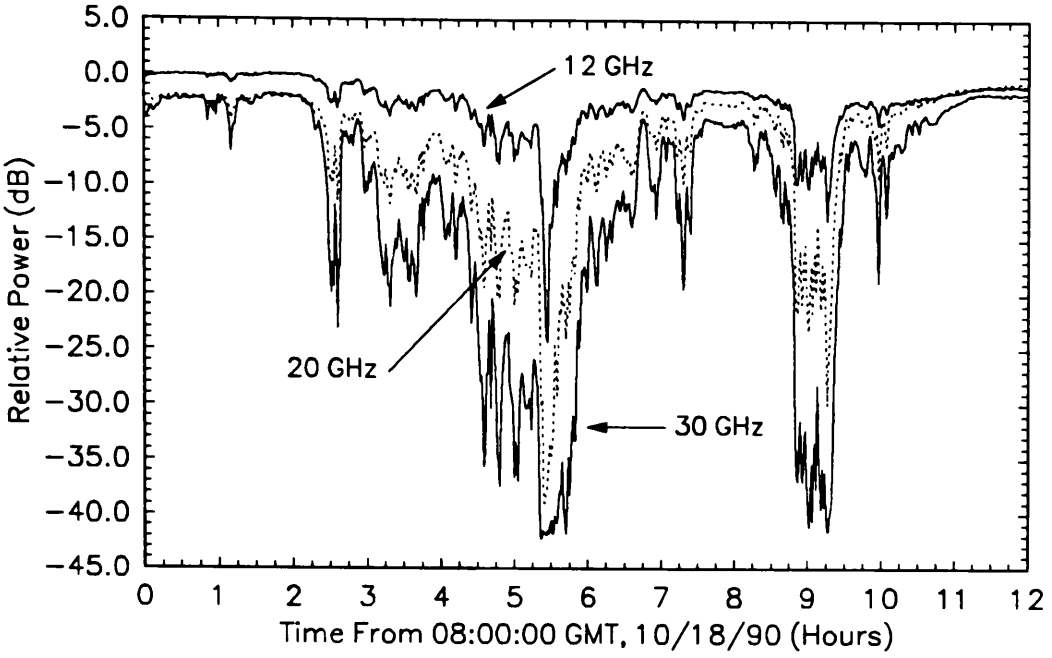


Figure 1-2. Signal attenuation due to rain.

1.2 Propagation Experiments

The design of reliable satellite communication links requires accurate quantitative descriptions of the atmospheric phenomena described above. The Virginia Tech Satellite Communications Group has identified a set of statistics which will aid scientists in obtaining this information. This extensive list of statistics includes scintillation indices, fade slope measurements, fade duration measurements, fade interval measurements, and frequency scaling of attenuation[5]. Such data can be determined from received beacon power measurements recorded at a sampling rate of at least 1 Hz. Unfortunately, much of the existing experimental data at K_u and K_a bands are inappropriate for developing these statistics[6]. The time resolution of data from previous experiments is insufficient for generating accurate fade slope and fade duration measurements. Additionally, many of the previous experiments were performed at sites which viewed a satellite at a high

elevation angle. At high elevation angles a signal's path length through the earth's atmosphere is shorter. Hence the impact of atmospheric phenomena is reduced. A satellite with beacons at K_u and K_a bands viewed from a low elevation angle would provide the signals needed to generate the statistics mentioned above. The OLYMPUS and ACTS satellites satisfy these requirements and therefore provide an opportunity to perform further propagation experiments at K_a band.

OLYMPUS is in geosynchronous orbit at 19° W over the Atlantic Ocean and can be viewed at a 14° elevation angle from Blacksburg, VA. Among other payloads, the OLYMPUS satellite carries three beacon transmitters operating at 12.5, 20 and 30 GHz[7]. The polarization of the 20 GHz signal is switched at a rate of 933 Hz. The 12 and 30 GHz signals are CW carriers. The Virginia Tech Satellite Communications Group has been conducting an experiment utilizing the OLYMPUS satellite since August, 1990. An overview of Virginia Tech's OLYMPUS experiment hardware is shown in Figure 1-3. As shown, the experiment utilizes four analog beacon receivers, three radiometers and two tipping bucket rain gauges. Information from these devices along with other environmental[2] and system status data are assimilated by the data acquisition system (DAS) and logged by a personal computer.

Measurement data from Virginia Tech's propagation experiments utilizing the OLYMPUS spacecraft have established a reliable base of information with which low elevation angle atmospheric propagation models may be developed. Continuous, synchronous collection of signal and weather information permits expected correlations between signal amplitude and weather to be examined and verified through the statistics listed above. This experiment, however, only provides data from a single system at a single elevation angle. Data from a broad range of similar experiments must be

---

[2] wind speed, wind direction, outside temperature, rain temperature, barometric pressure, and relative humidity

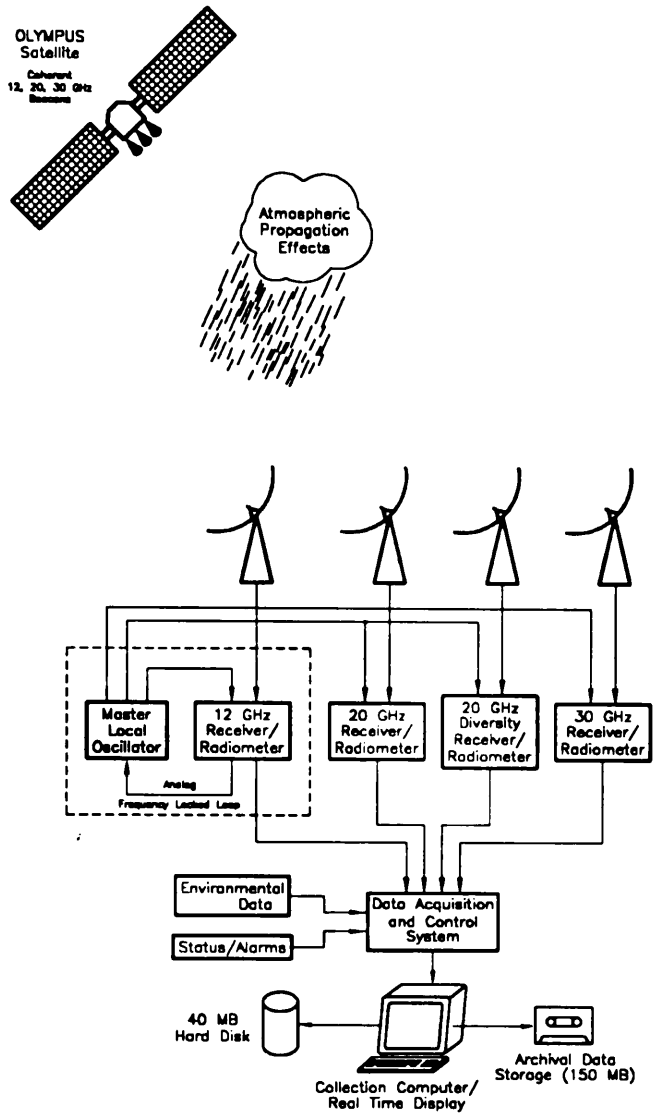


Figure 1-3. Overview of the Virginia Tech OLYMPUS experiment hardware.

assimilated to reinforce the concepts developed upon data from the Virginia Tech OLYMPUS propagation experiment.

Additional propagation data will be produced through experiments using NASA's Advanced Communications Technology Satellite (ACTS). ACTS is

scheduled for launch in July, 1993. Among many other communication experiment payloads, ACTS carries a beacon transmitter payload for propagation experiments; one transmitter operates at approximately 20 GHz, another at approximately 27.5 GHz[8]. NASA has sponsored the Virginia Tech Satellite Communications Group to develop and test one prototype and eight ACTS Propagation Terminals (APTs). Production terminals will be distributed to experimenters in Fairbanks, AK, Fort Collins, CO, Las Cruces, NM, Clarksburg, MD, Tampa , FL, Norman, OK, Blacksburg, VA and Vancouver, British Columbia, Canada. These terminals will enable multiple experimenters to generate a large quantity of propagation data at a diverse range of elevation angles with a consistent set of measurement equipment. An overview of the proposed experiment hardware is shown in Figure 1-4.

Similar to Virginia Tech's OLYMPUS experiment, each APT is comprised of two beacon receivers, two total power radiometers, weather measurement equipment, a data acquisition and control system (DACS) and a personal computer. The ACTS experiments will provide the propagation science community with additional propagation data to reinforce concepts developed upon Virginia Tech's OLYMPUS experiment data.

1.3 Thesis Overview

This document details the development and performance of a low cost, accurate, very versatile digital receiver for use in the APTs. The receiver is capable of acquiring a signal in the bandwidth of interest very quickly even at poor signal-to-noise ratios. The receiver then produces accurate signal power measurements, frequency domain spectra of the bandwidth of interest and carrier frequency estimates at software selectable rates. The receiver has been designed to be very flexible and may be reconfigured to suit a variety of applications. The remainder of this document details the receiver in a

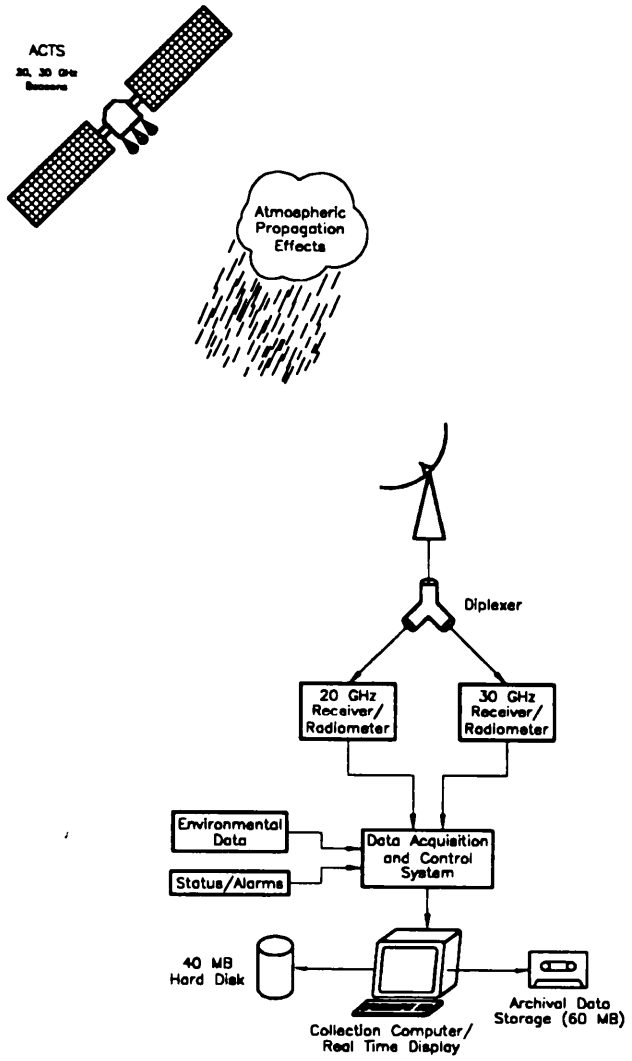


Figure 1-4. Overview of the ACTS experiment hardware.

configuration suitable for measuring the received signal power in the ACTS beacons.

The proposed ACTS experiments must produce accurate, repeatable beacon signal power measurements. Chapter 2 begins with a high level description of functions the APT receiver must perform to produce these measurements.

Quantitative beacon specifications are then outlined and the ACTS Users' Community receiver performance criteria are presented as a minimum set of specifications the APT receiver must meet. Supplemental functions which would facilitate use of the receiver are also presented. Although these functions are not specifically required by the ACTS experiment, they would provide the user with information that is not otherwise available. Chapter 2 concludes with a discussion of issues pertinent to the production of sixteen receivers for use in eight APTs.

The remaining chapters in this document describe the development and testing of the APT receiver. This process is summarized in Figure 1-5.

Chapter 3 compares and contrasts digital versus analog signal power measurement receivers. The ability of each type of receiver to satisfy criteria relevant to the ACTS experiment is identified. The comparison indicates that a digital receiver is better suited to the APT requirements.

Chapter 4 examines existing digital receiver technology. Algorithms implemented on receivers produced by Signal Processors Limited of Cambridge, England, Elektronik Centralen of Denmark, and Virginia Tech are evaluated. The benefits and limitations of each are summarized.

Chapter 5 presents the signal detection and measurement algorithms considered for implementation on the APT digital receiver. The chapter begins by describing the ACTS beacon signal simulation generated to aid in the evaluation of candidate algorithms. The first task of the APT digital receiver is to digitize the analog carrier signal. Various beacon signal digitization schemes are therefore presented and evaluated. Numerous signal acquisition and power measurement algorithms are then proposed; the advantages and disadvantages of each are summarized and simulated performances are discussed. Finally, Chapter 5 concludes by describing the algorithm that was selected for implementation on the APT receiver.

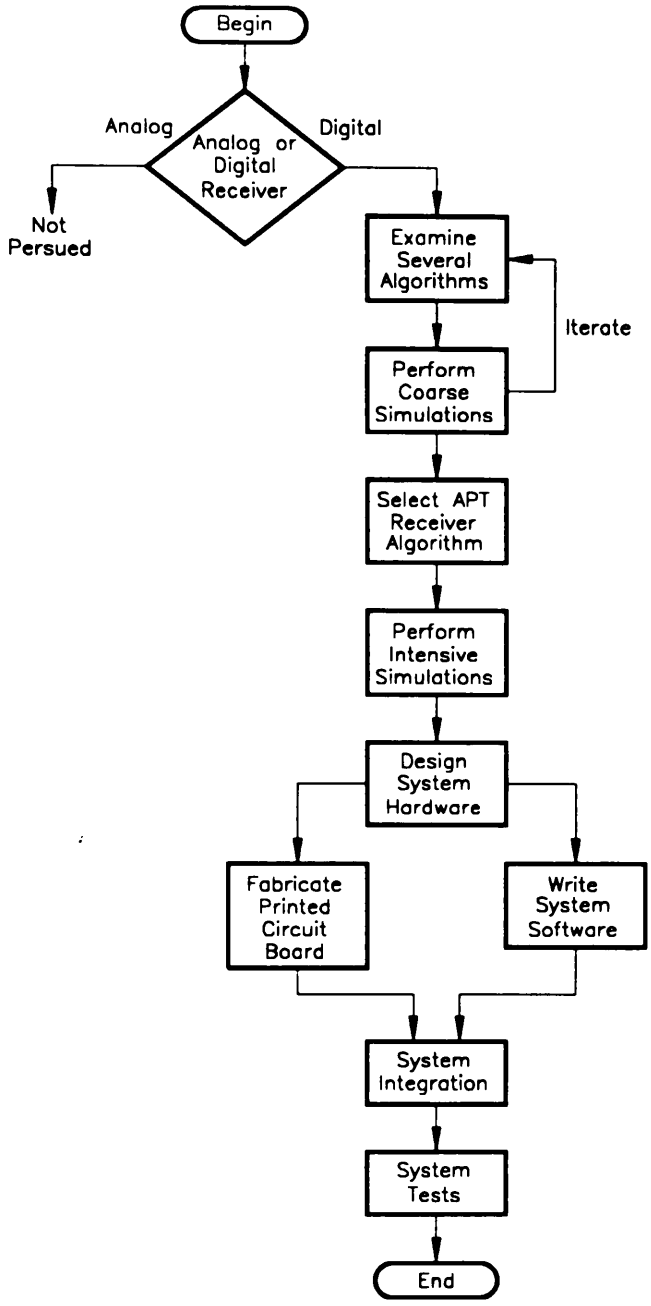


Figure 1-5. APT receiver development process overview.

Chapter 6 provides an in-depth description of the construction of the digital receiver hardware and software. The criteria by which major hardware

components were selected is summarized. All hardware component interfaces are then quantitatively described through the use of timing diagrams. Chapter 6 concludes by outlining the decision to develop the receiver software in assembly language.

Chapter 7 presents test results and final performance specifications of the digital receiver as it has been configured for the ACTS program. Unfortunately, the APTs will be installed before ACTS is launched in July, 1993. Consequently, it was not possible to test the digital receiver with the actual ACTS beacons. To confirm the performance of the APT receiver algorithm, the receiver was tested using the OLYMPUS 20 GHz beacon and signal generator output. The data in this chapter demonstrates that the receiver meets or exceeds all ACTS Users' Community specifications.

Chapter 8 concludes this document by detailing the status of the production APT digital receivers and describing possible future work.

Chapter 2

ACTS Propagation Terminal Receiver Requirements

In order to produce the required carrier signal power measurements, the APT receiver must acquire the carrier signal rapidly and remain locked to that signal during fading. This chapter begins with a qualitative description of the signal acquisition and power measurement functions. Characteristics of the ACTS beacon signals pertinent to the design of the APT receiver are then identified. Quantitative receiver requirements specified by the ACTS Users' Community are outlined as a minimum set of performance criteria each receiver must meet. Supplemental functions suggested by members of the Virginia Tech Satellite Communications Group, NASA representatives and members of the propagation science community are also discussed. Although these features are not specifically required by the experiment, they will provide the user with information that is not otherwise available. The chapter concludes with a discussion of issues pertinent to the production of sixteen receivers for use in eight APTs.

2.1 Overview

The beacon receiver performs three primary functions: detection of the carrier signal frequency, continuous tracking of the carrier signal and detection of the carrier signal power. Each of these functions is described in this section.

2.1.1 Carrier Signal Acquisition

Although the ACTS beacon transmitters are designed to operate at a specific frequency, practical constraints limit the accuracy with which the precise carrier frequency can be predicted. Furthermore, changes in the temperature of the payload, bus voltage variations and crystal aging can significantly alter the carrier frequency. As a result, a range of frequencies over which the carrier frequency can exist has been predicted. Determining the precise location of the carrier signal within the specified bandwidth is referred to as *acquiring the signal*. Since the APT has been designed to function autonomously, the APT receiver must detect when it is out of lock and attempt to reacquire the carrier signal without operator intervention.

Signal acquisition is further complicated by the existence of modulation tones on the 20 GHz beacon. The 20 GHz carrier signal can be modulated with combinations of 14.5, 19, 27.8, 32 and 64 kHz tones[8]. Figure 2-1 displays a simulated frequency domain representation of the 20 GHz carrier with 19 kHz, 32 kHz and 64 kHz modulation. Note, the amplitude of some of the resulting modulation tones is theoretically only 6 dB below the amplitude of the carrier. It is crucial that the receiver be able to identify the carrier signal among neighboring modulation tones. The carrier is the strongest signal in the bandwidth of interest. Should the receiver incorrectly lock to a side tone containing less power, the ensuing signal power measurements would reflect the loss of signal power. Additionally, the ultimate dynamic range of the

receiver would decrease due to the reduced signal-to-noise ratio at the receiver's input.

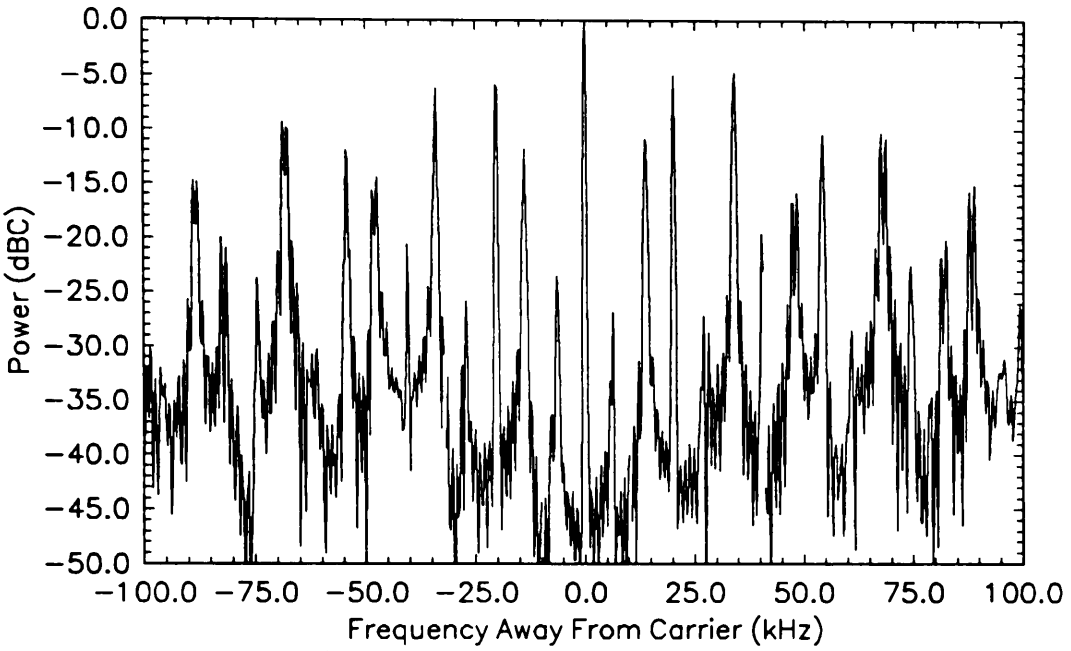


Figure 2-1. Digitally simulated ACTS 20 GHz beacon spectrum. (Created by evaluating time domain equations to simulate 300 kHz complex samples and performing a 1024-point fast Fourier transform)[9].

2.1.2 Carrier Signal Power Measurement

After the receiver has acquired the carrier signal, the receiver must produce signal power measurements. Traditionally, this is accomplished by passing the carrier signal through a very narrow bandpass or lowpass filter and detecting the power at the filter's output.

The detection filter should be designed to optimize the signal-to-noise ratio at the filter's output since signal power measurements can be made more

accurately at higher signal-to-noise ratios. This optimization can be achieved in varying degrees depending on the width of the carrier signal in the frequency domain and the accuracy of the frequency drift tracking algorithm employed. Theoretically, a filter design which shapes the filter's passband to match the shape of the carrier signal would provide the maximum signal-to-noise ratio at the filter's output. Using this matched filter, however, accurate signal power measurements can only be achieved if the signal is precisely centered in the filter's passband (otherwise, significant signal power is lost through the filter). Hence, the use of a matched detection filter places very stringent requirements on the frequency drift tracking algorithm which must lock the carrier to the center of the filter's passband. If an ideal rectangular filter with a wider passband is employed, frequency drift tracking errors would impact signal power measurements less severely but the signal-to-noise ratio at the filter's output would be reduced. For example, suppose the passband of an ideal rectangular filter is 10 Hz wider than the carrier signal. The carrier signal may drift within the passband of the filter without impacting the accuracy of the signal power measurements. Decreasing the width of the filter's passband would improve the signal-to-noise ratio at the filter's output by reducing the noise power in the passband of the filter but frequency drift tracking requirements would become tighter. If the passband of the filter is too narrow to pass the entire carrier signal or if the signal is permitted to drift out of the filter's passband, signal power would be lost through the filter and signal power measurements would be very inaccurate.

Translating the carrier frequency to the center of the passband of a very narrowband filter requires knowledge of the current carrier frequency. Hence, the receiver must detect and compensate for relatively small variations in the carrier frequency due to the net drift effects of the satellite oscillators and the oscillators in the APT itself. More accurate frequency drift tracking permits the use of narrower detection filters. As described above, this ultimately results in more accurate signal power measurements. Specific carrier signal drift rate specifications as well as signal power measurement accuracy and rate requirements are presented in the following sections.

2.2 ACTS Beacon Characteristics

The ACTS beacon characteristics pertinent to the design of the APT receiver are outlined in Table 2-1.

Table 2-1. ACTS beacon characteristics[10].

Carrier Frequency	20 GHz	27.5 GHz
Modulation Frequencies	14.5, 19, 32, 64, 27.8 kHz	none
Lifetime Carrier Bandwidth	±141 kHz	±187 kHz
Maximum Frequency Drift Rate	±555 Hz/15 minutes	±757 Hz/15 minutes
Phase Noise (dBC/Hz @ 1 Hz increments from the mean carrier frequency)	no specification available	no specification available

2.3 APT Receiver Specifications

2.3.1 ACTS Users' Community

The ACTS Users' Community is comprised of those individuals and organizations who will be operating the APTs or conducting experiments utilizing the data obtained from the APTs. The collective experience of the ACTS Users' Community has indicated that propagation measurements produced by a receiver which meets or exceeds the specifications listed in Table 2-2 would be appropriate for the studies described in Chapter 1.

Table 2-2. ACTS Users' Community APT receiver specifications[11].

Dynamic Range	+10 dB to -25 dB (relative to clear air conditions)
Resolution	0.1 dB
Accuracy (in clear air)	0.5 dB
Detection Bandwidth	no specification
Sample Rate	1 Hz

2.3.2 Suggested Enhancements

During the receiver's development, members of the Virginia Tech Satellite Communications Group, NASA representatives and members of the propagation science community identified additional desired APT capabilities. These suggested enhancements are based on the characteristics of the ACTS beacons and the circumstances surrounding the installation and use of the APTs.

2.3.2.1 Spectrum Analyzer Output

The ability to provide the user with frequency domain information about the entire bandwidth of interest would benefit the ACTS experiment in a number of ways. First, it would be possible to verify that the receiver has locked onto the carrier signal and not a side tone. The importance of locking to the correct tone was outlined in Section 2.1.1. Second, the modulation scheme employed on the ACTS 20 GHz beacon is very flexible. The beacon signal can be modulated with multiple combinations of 14.5, 19, 27.8, 32 and 64 kHz tones. The carrier signal amplitude depends on the current modulation format. Using spectrum analyzer type output, an experimenter

could determine which tones are present in the bandwidth. Sudden beacon signal level shifts could be explained by comparing spectra from before and after the shift. Finally it would be possible to verify the integrity of the spectrum. The existence of stray signals that might be present at one experiment site but not another could be verified.

2.3.2.2 Selectable Detection Bandwidth

An ideal oscillator would consistently output a precisely constant frequency. Practical oscillators, of course, cannot achieve this. As described above, the frequency output of practical oscillators varies with temperature and oscillator age. The frequency output of practical oscillators also suffers from phase noise. Phase noise causes very short term, Gaussian distributed perturbations from the oscillator's mean frequency. Phase noise causes signals to have finite width in the frequency domain unlike ideal signals which are represented by a spike without width. Information about the phase noise characteristics of the ACTS beacon signals is very limited.

As mentioned above, signal power measurements are traditionally made by passing a signal through a narrow band filter and measuring the power at the filter's output. To optimize receiver performance, the width of the detection filter should be trimmed to the width of the signal in the frequency domain.

The Virginia Tech Satellite Communications Group will install eight production APTs before ACTS is launched in July, 1993. Hence, the phase noise characteristics of the ACTS beacons will likely remain unknown until after the APTs are delivered. Receiver performance would be enhanced if the bandwidth of the detection filter could be easily adjusted.

2.3.2.3 Sample Rate

The collective experience of the ACTS Users' Community indicates that a signal power measurement sample rate of 1 Hz is sufficient to characterize the atmospheric phenomena of interest. The 1 Hz sample rate is also supported by a study conducted by the Virginia Tech Satellite Communications Group[12]. Some members of the propagation science community have, however, requested that the APT receiver produce signal power measurements at a higher rate to facilitate uplink power control experiments. The capability of producing signal power measurements at a rate of 20 Hz would meet the needs of these experimenters.

2.4 Production Issues

The APT receiver must be easy to replicate because the Virginia Tech Satellite Communications Group must produce sixteen units for use in eight APTs. The receiver design should incorporate minimal adjustments in order to make each receiver easy to assembly and test.

ACTS experiment results will include statistics which incorporate data from multiple sites. Hence APT receiver performance must be consistent. Variation in the spectral integrity or accuracy of the receiver's output is intolerable.

Chapter 3

Analog vs. Digital Receivers

The characteristic traits of analog and digital receivers often identify one type of receiver as more suitable for a given application. The design of the APT receiver began with a comparison that determined which type of receiver would best satisfy the APT requirements. Design time restricted the comparison to consider only high-level generic characteristics of each type of receiver. That is, the comparison did not specifically consider different types of analog receivers (FLL vs. PLL for example) or specific digital receiver algorithms. Rather, the comparison weighed the merits and problems typically associated with each type of receiver against the APT receiver requirements presented in Chapter 2. As shown in the following sections, the comparison concluded that a digital receiver is more suitable for the APT application.

For the reader who is unfamiliar with the distinction between analog and digital receivers, typical implementations of each are summarized below.

Analog Signal Power Measurement Receivers

Frequency lock and signal power measurement functions are very difficult to implement at typical carrier frequencies. Hence, conventional analog receiver technology utilizes successive analog amplifiers, analog filters and frequency mixers to amplify, bandlimit and translate the RF signal containing the carrier to a lower frequency. The first step in the downconversion process is usually amplification because the received signal is typically very weak. The signal is then bandlimited to prevent ensuing amplification stages from being saturated with noise. As shown in Figure 3-1, frequency mixers accept two signals at independent frequencies as inputs and produce a signal containing the sum and difference frequencies.

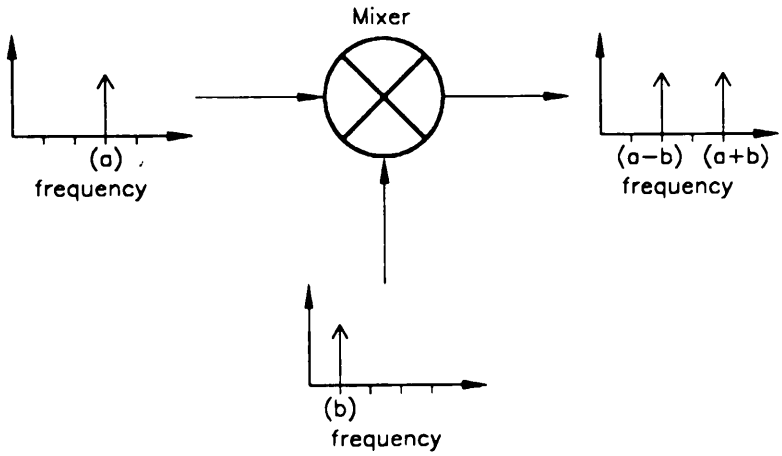


Figure 3-1. Frequency mixer operation.

Typically only one of the sum or difference frequencies is desired. Hence, the undesired component must be attenuated through the use of an analog filter. Once the carrier bandwidth has been translated to a suitable IF, conventional analog receivers employ a phase or frequency locked analog feedback loop to drive a voltage controlled oscillator such that:

$$f_{carrier} - f_{vco} = f_{known} \tag{3-1}$$

As shown in Figure 3-2, the output of the voltage controlled oscillator is used to drive a final analog mixer which locks the carrier at a known frequency. The resulting signal can then be passed through a very narrowband filter which bandlimits the signal to the detection bandwidth. Signal power is detected at the output of this filter. If the detection process is analog, the last frequency mixer often translates the carrier to baseband (dc) where the amplitude of the carrier is simply a voltage. Signal power is then recorded as a function of this voltage.

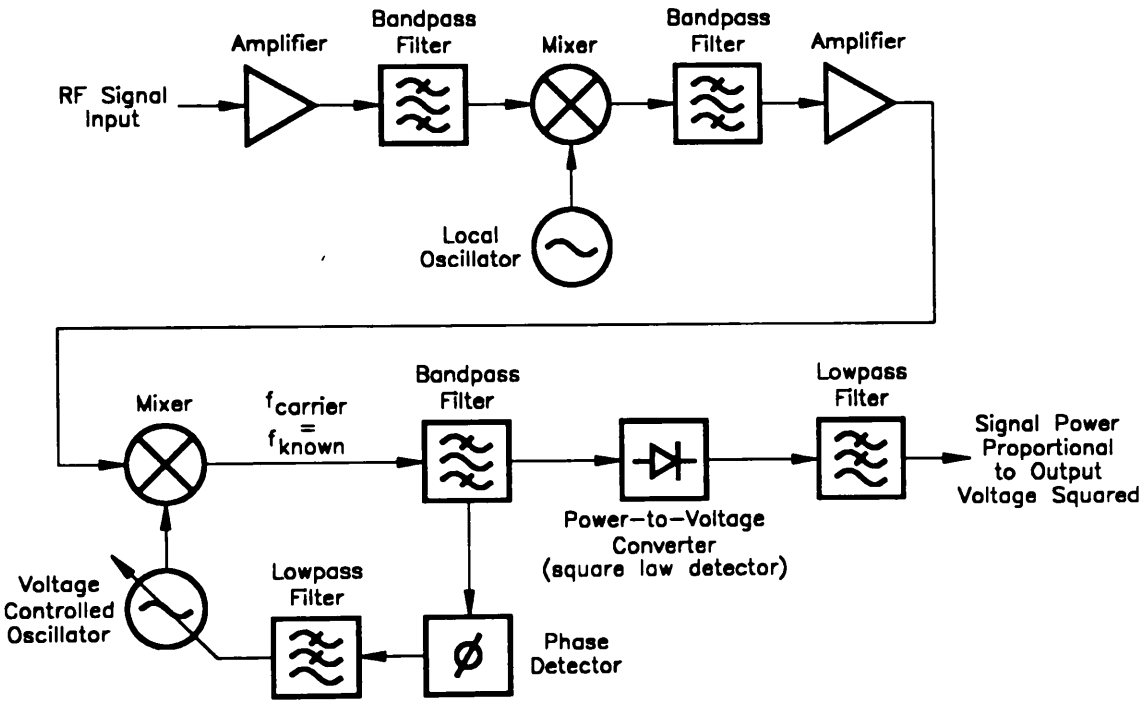


Figure 3-2. Typical analog receiver block diagram; this receiver has a phase locked loop for signal tracking.

Digital Signal Power Measurement Receivers

Digital receivers implement the required carrier signal acquisition and power measurement functions by digitizing the analog input signal and executing various mathematical algorithms on the digital data. The digitization process does not usually occur at the original carrier frequency because the signal must be significantly amplified and bandlimited before it can be accurately digitized. The analog amplification stages and bandlimiting filters required are typically difficult to build at the original carrier frequency. Hence, digital receivers also utilize the analog frequency translation technique described above to translate the carrier signal to a lower frequency at which the signal can be bandlimited, amplified and accurately sampled. See Figure 3-3.

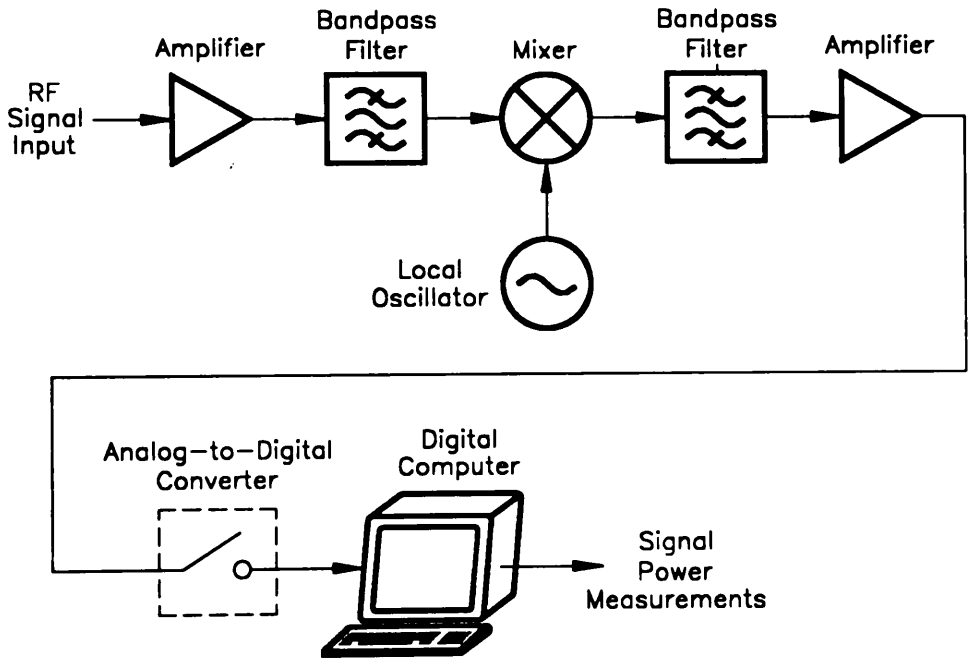


Figure 3-3. Typical digital receiver block diagram.

After the signal has been digitized, digital receivers utilize a digital computer to execute digital signal processing (mathematical) algorithms. These

algorithms determine the carrier frequency, further bandlimit the carrier signal and measure the power in the carrier signal. Although the comparison did not investigate particular digital receiver algorithms, examples of techniques which could be used to perform the these tasks include, Fourier transforms, chirp-Z transforms, finite impulse response filtering, infinite impulse response filtering, and autoregressive modeling. The performance of a specific digital receiver algorithm depends on the format of the carrier signal, the anticipated signal levels and the available processing power.

3.1 Repeatable System Performance

The scope of the proposed ACTS propagation experiments requires that the APT receiver be easy to replicate. Furthermore, the receiver must require minimal adjustments in order to make the receivers easy to assemble and test.

Examining the components utilized in analog and digital receivers, it is clear that digital systems can be replicated far more easily than analog systems. For example, replicating a digital filter can be achieved by simply executing the same software on multiple computers. The gain value at each filter tap will be identical on both systems. The nature of digital systems therefore yields filters which have identical characteristics[3]. Replicating an analog filter is much more difficult. Analog filters are, of course, constructed from physical components. Although designers can obtain analog components with very tight tolerances, no analog component specifications are exact. Two analog filters constructed from components with the same tight tolerances can have measurably different characteristics.

---

[3] Multiple versions of the same digital filter will have the same characteristics to floating point precision if the same numerical rounding and quantization procedures are followed in each system on which the filter implemented.

3.2 Carrier Signal Acquisition

In most receiver designs, it is necessary to determine the frequency of the carrier signal. This task is referred to as *carrier signal acquisition*. Acquiring the ACTS 20 GHz beacon is complicated by the existence of many modulation tones surrounding the carrier. Furthermore, the format of these tones will not be constant but will be altered depending on the state of the satellite. Differentiating the carrier signal from the modulation tones is a straightforward task in the frequency domain because the carrier is the strongest signal in the bandwidth of interest. A number of realizable digital signal processing algorithms exist for translating a time domain signal to the frequency domain.[4] High speed microprocessors can typically implement these algorithms in a few seconds. Hence a digital receiver would be able to reliably discern the carrier signal amidst neighboring modulation tones very quickly. Conversely, this task would be quite difficult for an analog receiver to perform. The analog receiver would have to sequentially translate small sections of the bandwidth of interest through a narrowband analog filter to determine the frequency of the strongest signal in the bandwidth. When the entire bandwidth had been scanned, the approximate location of the carrier could be determined. Although analog systems similar to the one described above have been constructed, they are difficult to design, typically take a relatively long time to scan the bandwidth of interest[5] and require good signal-to-noise ratio to acquire the carrier. Therefore, a digital system would be better suited to acquiring the ACTS 20 GHz beacon signal.

---

[4] Fast Fourier transform, chirp-Z transform, autoregressive modeling, etc.

[5] The maximum scan rate for full filter response is limited by the square of the filter bandwidth[16]. The analog circuit designed by Virginia Tech for the SIRIO experiment took approximately 21 minutes to scan a 125 kHz bandwidth with a 10 Hz filter.

3.3 Detection Bandwidth Flexibility

As discussed in Chapter 2, the output of practical oscillators is imperfect; Phase noise gives the output signal bandwidth in the frequency domain. The bandwidth of the filter used to detect the power in a signal should be determined from the phase noise specifications of the signal. The available measured ACTS beacon phase noise characteristics are not specific enough to determine the optimum detection bandwidth for the ACTS beacons. As such, the performance of the APT receiver would be enhanced if the width of the detection filter could be easily adjusted.

The characteristics of an analog filter are not easily adjusted because analog filters are physically constructed from discrete components. Hence, the task requires physical modification of the filter circuit. Conversely, digital filters may be reconfigured in software. Altering the bandwidth of a digital filter would not require any hardware modifications. Clearly, a digital system is more flexible than its analog counterpart in this respect.

3.4 Long Term Stability

The performance of traditional analog receivers can be significantly degraded from thermal and electrical instabilities.[6] Conversely, digital circuitry maintains constant performance over a wide range of ambient conditions.

[6] amplifier gain variations, phase-locked loop nonlinearities, etc.

3.5 Summary and Conclusions

Table 3-1 summarizes the results of the study conducted to determine which type of receiver (analog or digital) would best satisfy the APT receiver requirements. A digital receiver was selected for implementation on the APT system based on the information contained therein.

Table 3-1 Analog vs. digital receiver performance summary.

Feature	Implementation	
	Analog	Digital
Repeatable System Performance	often a problem	easily replicated
Carrier Signal Acquisition	difficult to design, slow to perform	trivial in the frequency domain
Detection Bandwidth Flexibility	bandwidth changes require hardware modifications	bandwidth changes can be accommodated in software
Long Term Stability	poor	good

Chapter 4

Existing Digital Receiver Technology

The advantages of digital technology outlined in Chapter 3 have been exploited in literally hundreds of receiver applications. Digital pagers, digital cellular telephones and HDTV are but a few examples. Comparatively few of these receivers are designed to make precise signal power measurements, however, due to the limited number of applications for such data. The following sections describe four signal power measurement receivers which utilize digital technology in at least part of their design. The hybrid receivers designed by Virginia Tech and Elektronik Centralen of Denmark rely on analog circuitry to lock the signal of interest at a known frequency but utilize digital techniques to obtain the desired signal power measurements. The digital receivers designed by Signal Processors Limited of Cambridge, England and Virginia Tech (first generation) digitize an unlocked waveform and perform all of the receiver functions digitally.

4.1 Elektronik Centralen Beacon Receiver

Elektronik Centralen of Denmark has developed a receiving system designed for simultaneous measurement of the OLYMPUS satellite 20 GHz and 30 GHz beacons. Both signals are derived from a single antenna but are amplified and downconverted by separate RF front ends. Analog detection circuitry decommutates the 20 GHz beacon signal and detects the co-polar and cross-polar signals. The receiver design takes advantage of the fact that the OLYMPUS beacons are phase coherent to phase lock both detection systems to the less fade prone 20 GHz beacon, thereby extending the dynamic range of the 30 GHz system. The analog detection circuitry generates in-phase and quadrature waveforms for each of the received signals. Analog filters are used to bandlimit these waveforms to 50 Hz before the signals are digitized to create a complex sample stream. The resulting samples are recorded by a personal computer. Note, the desired signal power measurements are obtained from this data using Equation (4-1).

$$Power(dB) = 10 \cdot log_{10}(I^2 + Q^2) \qquad (4\text{-}1)$$

$$where: I = in-phase\ component$$
$$Q = quadrature\ component$$

The dynamic range of this system is entirely dependent on the quality of the analog lock circuitry. The signal is only digitized to derive the desired signal power measurements. When the system is locked, these measurements are accurate to 0.5 dB.

4.2 Virginia Tech Hybrid Receiver

The Virginia Tech Satellite Communications Group has developed a receiving system to make signal power measurements of the OLYMPUS 12.5 GHz, 20 GHz and 30 GHz beacons. The 20 GHz and 30 GHz receivers are very similar to the 12.5 GHz receiver described below.

The 12.5 GHz beacon signal is amplified, filtered and downconverted to a 10 kHz IF by dedicated analog circuitry. Additional analog detection, filtering and mixing circuitry provides a 10 kHz IF beacon signal bandlimited to 200 Hz and locked to a 40 kHz sample clock. A custom *I-Q Detector Card* then digitizes the 10 kHz waveform on the 19th and 40th positive-going transitions of the 40 kHz sample clock. As shown in Equation (4-2), this sampling technique generates samples separated by 90° in phase on the 10 kHz waveform.

- $$\begin{pmatrix} time\ between\ 19^{th} \\ and\ 40^{th}\ sample \\ clock\ transitions \end{pmatrix} = \frac{40-19}{40000} = 5.25 \times 10^{-4}\ second \qquad (4\text{-}2)$$

- $$period\ of\ 10kHz\ waveform = \frac{1}{10000} = 1 \times 10^{-4}\ second$$

- $$\begin{pmatrix} \text{phase difference between} \\ 19^{th} \text{ and } 40^{th} \text{ sample} \\ \text{clock transitions} \end{pmatrix} = \left\{ \frac{\begin{pmatrix} \text{time between } 19^{th} \\ \text{and } 40^{th} \text{ sample} \\ \text{clock transitions} \end{pmatrix}}{\begin{pmatrix} \text{period of} \\ 10\text{kHz} \\ \text{waveform} \end{pmatrix}} \text{ modulo } 1.0 \right\} \cdot 360°$$

$$= \left\{ \frac{5.25 \times 10^{-4}}{1 \times 10^{-4}} \text{ modulo } 1.0 \right\} \cdot 360°$$

$$= 90°$$

Time adjacent real samples are then paired to form complex samples which are input to a 1116-tap Kaiser window FIR filter. The filter is executed 10 times per second to bandlimit the complex sample stream to 3 Hz and reduce the sampling frequency to 10 Hz. The resulting 10 Hz complex data stream is stored by a personal computer. Signal power measurements can then be obtained from each 10 Hz complex sample using Equation (4-1).

Similar to the Elektronik Centralen receiver system, the Virginia Tech hybrid receiver relies on analog circuitry to provide the carrier signal locked at a known, stable frequency. Hence, the dynamic range of this system is entirely dependent on the quality of this analog circuitry. System tests indicate that power measurements produced by the receiver are accurate to 0.1 dB.

4.3 Signal Processors Limited Receiver

Signal Processors Limited (SPL) of Cambridge, England has developed a digital receiver which uses digital frequency locking techniques to track the

signal of interest at signal-to-noise ratios as much as 15 dB below that required by conventional PLL designs. Furthermore, the receiver can reacquire the carrier signal after deep fades at lower signal-to-noise ratios than conventional analog designs by using digital detection techniques.

The beacon signal of interest is input to the SPL receiver at 70 MHz. The receiver system includes a digitally controlled oscillator to downconvert the 70 MHz signal to a 15 kHz IF. To acquire the carrier signal, the bandwidth of interest is examined in sections. The digitally controlled oscillator is updated to translate a section of the bandwidth through a narrow analog filter. The input signal is then digitized at a rate consistent with the width of the analog filter and a 64-point fast Fourier transform is executed. The results of the transform are evaluated to determine the spectral location of the largest spectral peak. The digitally controlled oscillator is then updated and the process repeated to characterize a new section of the bandwidth. When the entire bandwidth has been examined, the spectral location of the strongest signal identifies the approximate carrier frequency. The digitally controlled oscillator is then readjusted to translate the approximate carrier frequency bandwidth through a narrower analog filter. Another 64-point FFT is performed on data taken at a lower sampling rate in order to identify the carrier frequency accurately enough to allow the tracking algorithm to lock to the signal. The acquisition process can scan an 800 kHz bandwidth in approximately 2 seconds and will acquire the carrier signal at signal-to-noise ratios better than 32 dBHz in a 1 Hz bandwidth (-17 dB relative to clear air conditions on the APT system).

Once the receiver has acquired the carrier signal, the digitally controlled oscillator is updated to translate the carrier signal to approximately 15 kHz. The 15 kHz IF signal is bandlimited and then digitized at a rate of 60 kHz. A complex data stream can then be assembled by pairing time adjacent real samples because consecutive real samples are separated by 90° in phase on the 15 kHz IF waveform. Fast Fourier transforms are performed on the complex data stream in order to generate carrier frequency estimates. Carrier

frequency estimates accurate to 1 Hz are readily derived from spectral peak location in the Fourier domain. The digitally controlled oscillator is adjusted, based on this carrier frequency information, to lock the carrier signal at the 15 kHz IF. The signal is then bandlimited to 50 Hz using a digital filter before the complex sample stream is decimated to a sampling rate of 100 Hz and made available to the user. The desired power measurements are then obtained from the complex data stream using Equation (4-1).

The SPL digital tracking scheme fails at 9 dBHz in a 1 Hz bandwidth. Unfortunately, no measurement accuracy information is available. Note, a more complicated version of the SPL receiver has the ability to measure polarization-switched beacon signals. The techniques used to derive the co-polar and cross-polar signals from the copolar signal and perform phase correction are not discussed here because neither of the ACTS beacons are polarization switched.

4.4 Virginia Tech Digital Receiver (First Generation)

The Virginia Tech Satellite Communications Group has developed a digital beacon receiver which utilizes successive fast Fourier transforms as filter banks to acquire, track and measure received signal power in a signal of interest. The receiver digitizes a 10 MHz input signal at 8 MHz as shown in Figure 5-12. This IF/sampling frequency relationship provides digitized samples which are separated by 90° in phase on the sampled waveform. Hence, complex samples may be constructed by pairing time-adjacent real samples. The complex sample stream is immediately decimated to the receiver's input bandwidth of 0.5 MHz. 1024-point FFTs are then performed on the 0.5 MHz complex sample stream. The contents of the resulting 488 Hz-wide bins are examined as the output of 1024 adjacent filters. The

contents of the bin containing the most power are placed into a queue. When the queue contains 1024 complex samples, a second stage FFT is executed. Again, the contents of the resulting 0.5 Hz-wide bins are examined as the output of 1024 adjacent filters. The spectral position of the bin with most power is assumed to represent the mean carrier frequency. Signal power measurements are, therefore, obtained by summing the contents of bins surrounding the bin with the most power to form the output of a detection filter with the desired bandwidth.

This receiver has only been tested on synthesized signals but does appear to produce linear, accurate signal power measurements over a 40 dB dynamic range[7] as long as the input signal remains near the center of a first stage FFT bin. If the input signal power is split between adjacent first stage FFT bins, the accuracy of the ensuing signal power measurements degrades significantly. Runyon notes that this deficiency could be corrected through the addition of a voltage controlled oscillator[13]. Carrier frequency estimates derived from the spectral location of the second stage DFT point with the greatest magnitude could be used to adjust the voltage controlled oscillator to lock the carrier signal in the center of a first stage FFT bin.

As discussed in Section 5.3.1, additional simulations indicate the receiver would not be able to reliably identify the carrier signal at signal-to-noise-ratios below 31 dBHz in a 1 Hz bandwidth. This deficiency would limit the receiver's useful dynamic range to approximately 18 dB on the APT system.

[7] using a 10 bit analog-to-digital converter

Chapter 5

Algorithm Development

Several digital signal processing based signal detection and power measurement techniques were considered for implementation on the APT receiver. These techniques were combined in various ways to form numerous plausible receiver algorithms. To determine the potential performance of each technique, the algorithms were implemented in the C computer language and executed on simulated data. The APT digital receiver algorithm is a hybrid algorithm which incorporates the most effective digital signal processing and frequency translation techniques from all algorithms considered. Performance data generated from a comprehensive simulation of the APT receiver algorithm indicates the receiver will meet the goals established in Chapter 2.

The following sections describe the sampling process utilized by all of the candidate algorithms and present a representative sample of the digital signal processing techniques considered for use on the APT receiver. The simulation of these algorithms was computationally intensive and, hence, very time consuming. As a result, less promising algorithms were simulated less

thoroughly. The level of detail to which each algorithm is described is indicative of the extent to which the algorithm was simulated. This chapter concludes with a detailed description of the APT receiver algorithm.

Note, the following discussions assume the reader has a basic understanding of digital signal processing theory. The interested reader may find additional information on the techniques described below in *Digital Filters and Signal Processing, Second Edition* by L.B. Jackson, Kluwer Academic Publishers, Boston, MA, 1990 and *Digital Signal Processing* by A.V. Oppenheim and R.W. Schafer, Prentice Hall Inc., Englewood Cliffs, NJ, 1975.

5.1 Beacon Signal Simulation

The ACTS 20 GHz beacon was selected for use in the algorithm simulation process because it is the most complex of the ACTS beacons. As outlined in Chapter 2, the ACTS 20 GHz beacon can be modulated in a number of different formats. The ACTS 27.5 GHz beacon is a CW carrier. Hence, the ACTS 27.5 GHz beacon is considered to be a trivial case of the 20 GHz beacon. Analog-to-digital converter samples of the ACTS 20 GHz beacon were simulated by evaluating time domain equations describing the carrier, associated modulation and noise. The following sections describe the construction of these equations.

Modulation

The ACTS 20 GHz beacon can be modulated by a Phase Shift Keying (PSK) telemetry channel and a ranging tone simultaneously. The 20 GHz beacon is derived from a phase modulator where the residual power acts as the beacon source, and the side band power carries telemetry and ranging information[9]. As such, the beacon will be transmitted at a constant power but will be

surrounded by various modulation tones. Equation (5-1) describes the carrier and associated 19 kHz (ranging), 32 kHz (place holder) and 64 kHz (PSK telemetry) tones.

$$S = A \cdot cos(2\pi f_c nT) + \xi_1 \delta \cdot sin(2\pi f_{m1} nT) + \xi_2 \delta \cdot sin(2\pi f_{m2} nT) + \xi_3 \delta \cdot sin(2\pi f_{m3} nT)) \quad (5\text{-}1)$$

where: S = sample value

A = carrier amplitude

f_c = carrier frequency (Hz)

n = sample index

T = sample period

$\xi_{1,2,3}$ = modulation indices = 0.72

f_{m1} = 64 kHz

f_{m2} = 32 kHz

f_{m3} = 19 kHz

$$\delta = \begin{cases} 1, & (nT) \text{ modulo } (2T_{PSK}) < T_{PSK} \\ -1, & (nT) \text{ modulo } (2T_{PSK}) \geq T_{PSK} \end{cases}$$

$$T_{PSK} = \frac{1}{f_{PSK}} = \frac{1}{1024} = 0.000976$$

Although this combination of frequencies does not comprise one of the actual modulation formats on the ACTS 20 GHz beacon, it does form a representative sample of the types of modulation expected on the beacon[9]. Simulated analog-to-digital converter samples were generated by evaluating Equation (5-1) at time instants corresponding to analog-to-digital converter

clock transitions. Figure 5-1 displays a frequency domain representation of the simulated 20 GHz beacon signal.

The spectrum in Figure 5-1 was generated by performing a 1024-point complex FFT on data generated by evaluating Equation (5-1). Note, the time domain signal was not bandlimited in any way. Hence the spectrum in Figure 5-1 does contain digital aliases.

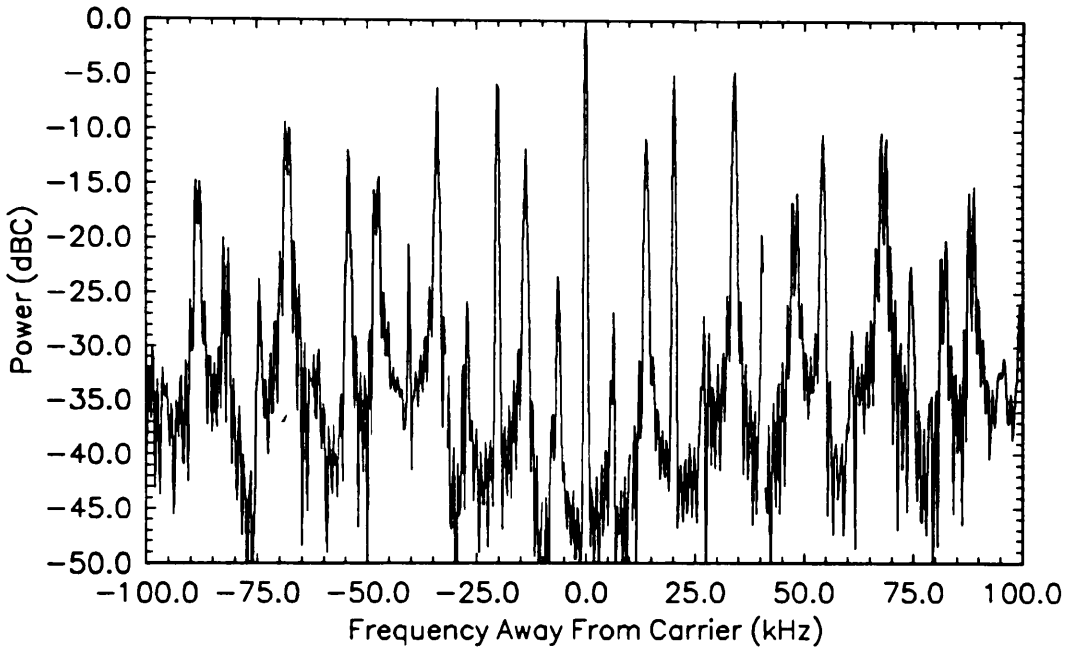


Figure 5-1. Digitally simulated ACTS 20 GHz beacon spectrum[9].

Phase Noise

Phase noise was modeled as Gaussian distributed, very short-term variations in the frequency of the carrier signal. The relative magnitudes of these variations should be determined from the carrier's phase noise specifications. Unfortunately, there are no phase noise specifications for the ACTS beacons at frequencies less than 50 Hz away from the mean carrier frequency.

Furthermore, measured data are available only in graphical form. These factors combine to limit the accuracy of available numerical data. Due to this lack of information, two different plausible phase noise models were incorporated into the beacon simulation. Both models generated carrier frequency deviations by passing scaled, normally distributed random numbers through a low pass filter. The spectral content of the carrier frequency deviations was manipulated by adjusting the scaling factor and the shape of the low pass filter.

To simulate the phase noise in a *typical* oscillator circuit, the first model was tuned to produce frequency deviations whose spectral content decayed as $1/f^3$ (30 dB/decade)[18]. Figure 5-2 shows the frequency domain representation of a carrier signal with phase noise generated by the first phase noise model.

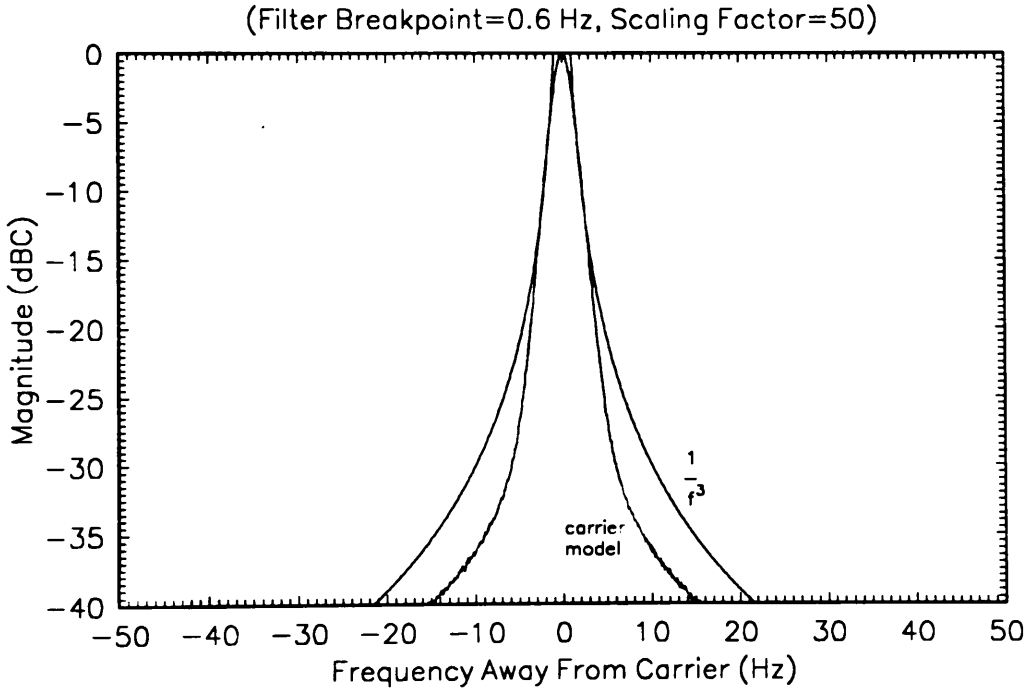


Figure 5-2. Frequency domain representation of a carrier signal with *typical* phase noise characteristics.

The spectral characteristics of data produced by the second phase noise model were tuned to match data extrapolated from available ACTS 20 GHz beacon measured data. Figure 5-3 compares the frequency domain representation of a carrier signal with phase noise generated by the second model to the extrapolated data.

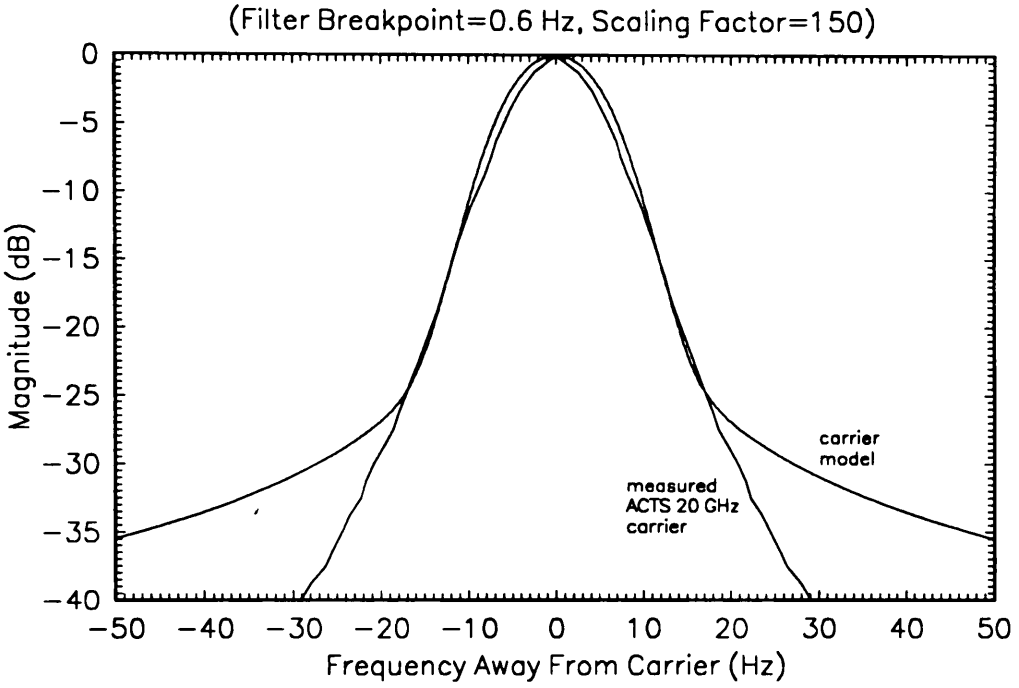


Figure 5-3. Frequency domain representation of a carrier signal with phase noise generated by the model tuned to match available measured ACTS beacon data.

White Noise

The noise level at the receiver's input is dominated by thermal noise from the amplifiers, active filters, and frequency mixers preceding the receiver in the IF chain. Sky noise contributions to the system noise level are relatively

minor. Consequently, during fade conditions signal power is reduced but the system noise level remains essentially constant.

To simulate noise in the bandwidth of interest, and consequently generate the signal-to-noise ratios anticipated at the receiver's input, scaled, normally distributed numbers were added to each simulated beacon sample. A white noise floor 10 dB below the saturation point[8,9] was simulated by scaling normally distributed numbers by 0.1 and adding the result to simulated carrier signal samples. The magnitude of the carrier signal samples was then adjusted to simulate the desired signal-to-noise ratios:

$$A = 10^{\frac{(\alpha+\delta-10.0)}{20.0}} \tag{5-2}$$

where: $A = carrier\ amplitude$

$\alpha = attenuation\,(dB)$

$\delta = signal - to - noise\ ratio\ in$

$a\ 200kHz\ bandwidth\ (dB)$

Frequency Drift

Changes in the temperature of the spacecraft as well as oscillator aging cause the mean carrier frequency to drift slowly. Long term drifts in the mean carrier frequency were simulated by slewing the value of f_c in Equation (5-1).

---

[8] The RMS noise power is set 10 dB below the saturation point of the analog-to-digital converter to ensure scintillation signal level enhancements will be accurately measured.

[9] The full-scale voltage range of the simulated analog-to-digital converter was defined to be ±1 volt.

f_c was incremented according to Equation (5-3) each time Equation (5-1) was evaluated to produce another simulated beacon sample.[10]

$$\Delta f_c = \frac{\delta}{f_s} \tag{5-3}$$

$$\begin{aligned} where: \Delta f_c &= delta\ carrier\ frequency\ (Hz) \\ \delta &= rate\ of\ drift\ (Hz) \\ f_s &= sampling\ frequency\ (Hz) \end{aligned}$$

Quantization

The final step in generating a simulated beacon sample was quantizing the results of the aforementioned operations to the precision of the analog-to-digital converter. Both components of each simulated complex beacon sample were quantized using Equation (5-4).

$$S_Q = Integer(S \cdot 2^{(n-1)}) \tag{5-4}$$

$$\begin{aligned} where: S_Q &= quantized\ sample \\ S &= unquantized\ sample \\ n &= analog-to-digital\ converter\ precision\ (bits) \end{aligned}$$

[10] An additional phase term was added to each sinusoidal component in Equation 5-1 to avoid phase discontinuities when f_c was changed. For example, $\cos(2\pi f_c nT)$ would become $\cos(2\pi f_c nT + \phi)$ where $\phi = \left| f_{c(new)} nT - f_{c(old)} nT \right|$ modulo 2π.

5.2 Digitization of the Beacon Signal

Complex signal digitization often provides several advantages over real digitization. As such, each of the candidate signal detection algorithms was designed to be executed on complex data. For the reader who is not familiar with real versus complex sampling theory, a summary in provided below.

Real Sampling

Real sampling of a waveform is accomplished by measuring the amplitude of the waveform at evenly spaced sample instants as shown in Figure 5-4.

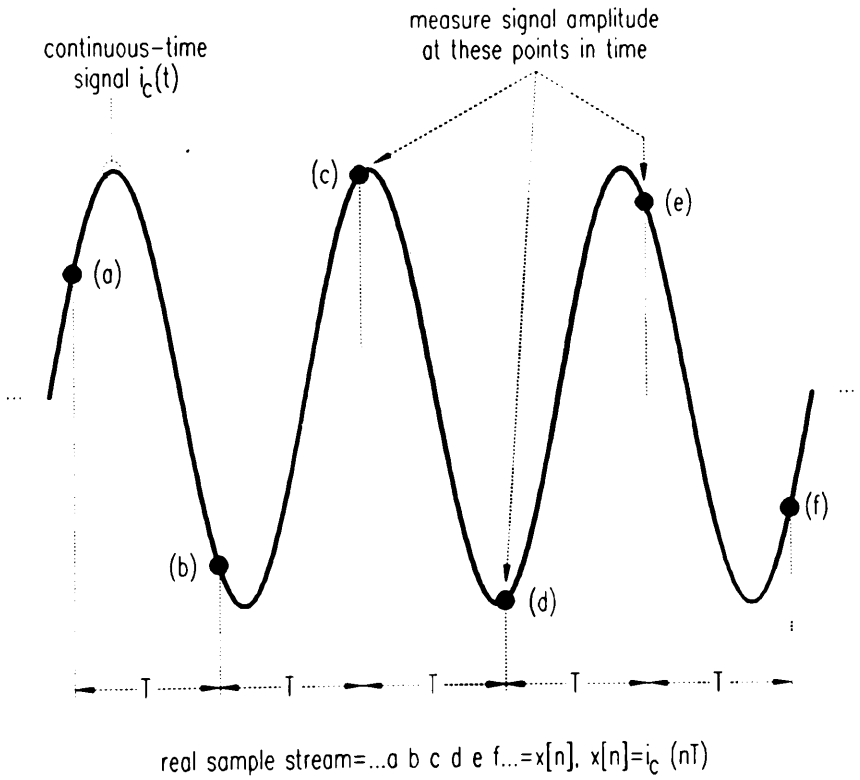


Figure 5-4. Real sampling of a waveform.

To obtain enough information to reconstruct the signal (avoid aliasing), the Nyquist criteria demands that the sampling frequency be at least twice the frequency of the highest frequency component in the sampled waveform[14].

Note, the discrete Fourier transform (DFT)[11] of a real-valued signal is symmetric about the half sampling point as shown in Figure 5-5.

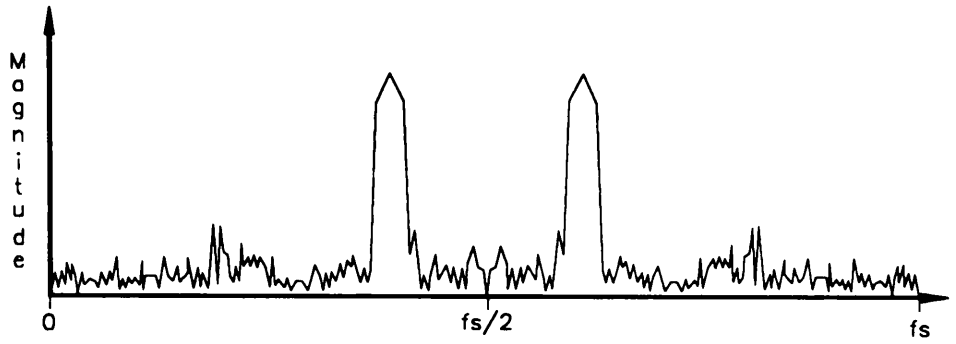


Figure 5-5. Discrete Fourier transform of a real-valued signal.

Complex Sampling

Complex sampling of a sinusoid is performed by simultaneously measuring the amplitude of the signal at two points separated by one quarter wavelength (90° in phase) on the waveform. The resulting in-phase and quadrature measurements are then combined as shown in Figure 5-6 to form one complex sample. Similar to real sampling, the measurement process must occur at evenly spaced sample instants.

To accurately characterize a waveform, complex sampling must be performed at a rate greater than the bandwidth of the sampled signal[17]. For example, if a signal is bandlimited to 10 kHz, complex samples must be generated at a

[11] The fast Fourier transform (FFT) exploits the symmetry of the discrete Fourier transform to compute the transform results in fewer mathematical operations.

minimum rate of 10 kHz. Although aliasing will occur, the contributions of the aliases will be negligible if the signal is significantly bandlimited as shown in Figure 5-7.

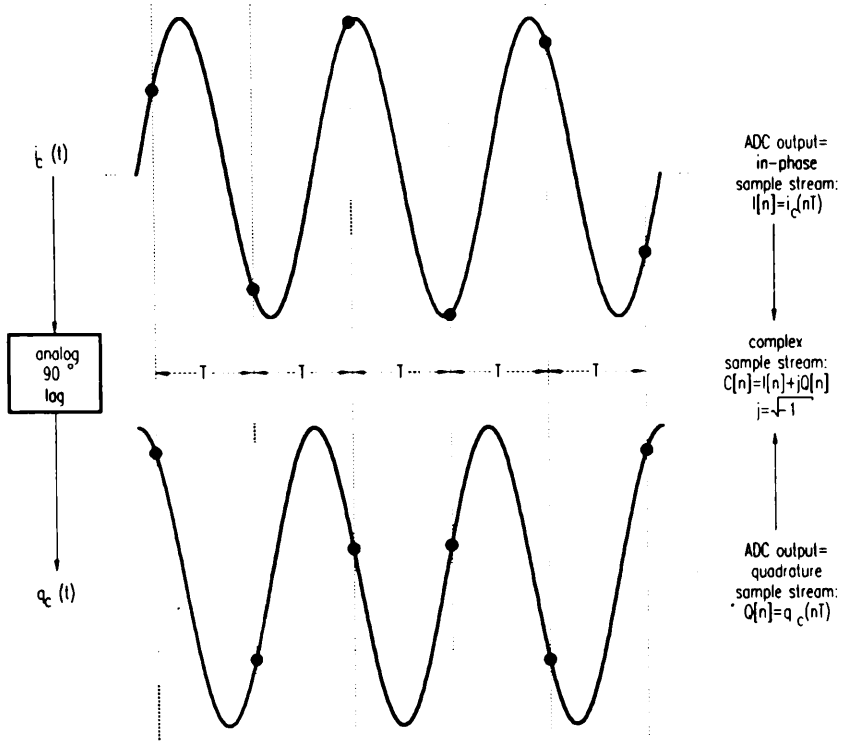


Figure 5-6. Complex sampling of a waveform.

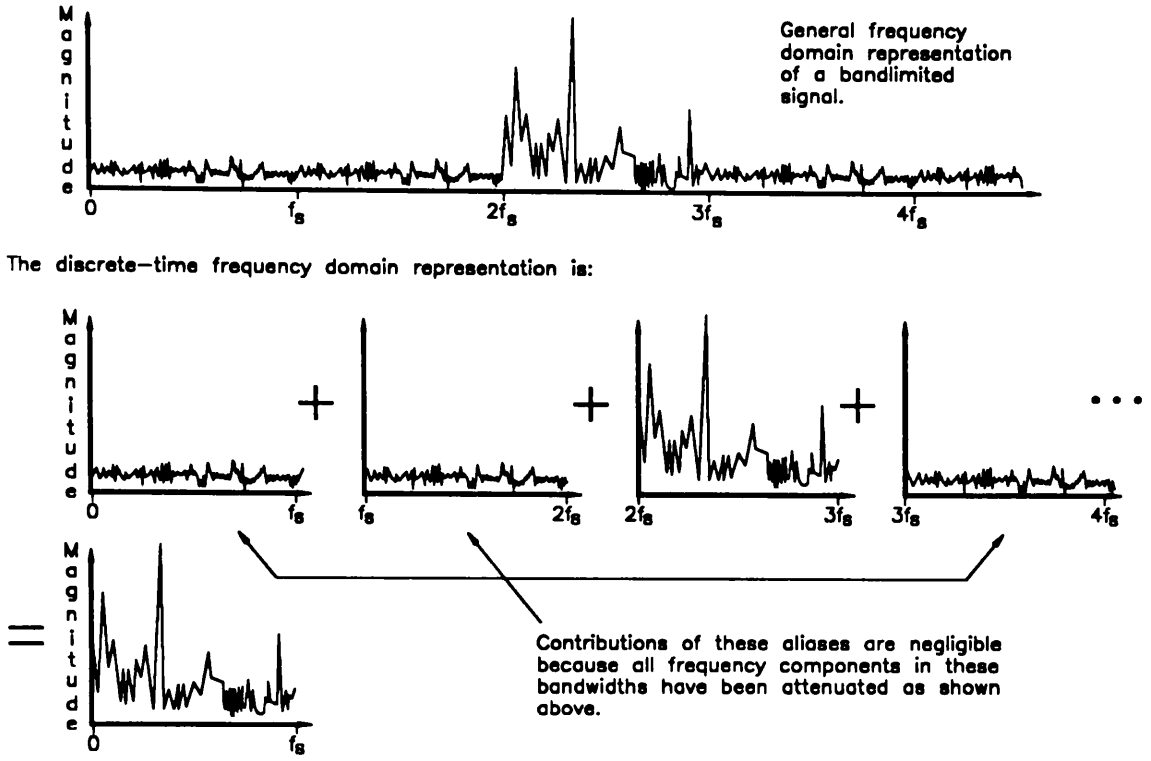


Figure 5-7. Aliasing of a bandlimited, complex-valued signal.

Unlike the DFT of a real-valued signal, the DFT of a complex-valued signal is unique from $n \cdot f_s$ to $(n+1) \cdot f_s$. As shown in Figure 5-8, the DFT of a complex-valued signal is not symmetric about the half sampling point.

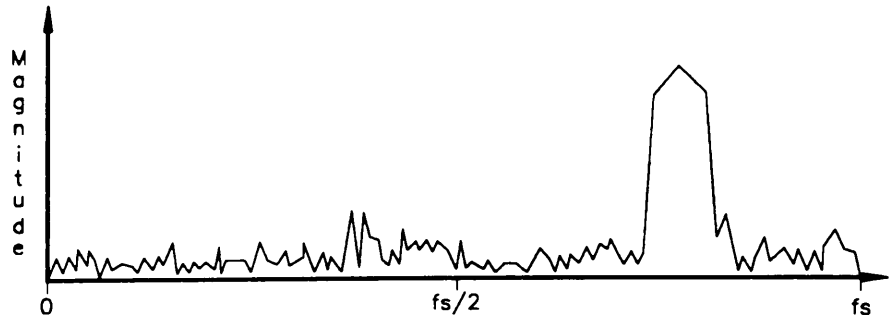


Figure 5-8. Discrete Fourier transform of a complex-valued signal.

5.2.1 Advantages of Complex Sampling

Complex sampling offers several advantages over real sampling with respect to the APT receiver requirements. These advantages are outlined in the following sections.

5.2.1.1 Density of Discrete Fourier Transform Points

A complex DFT produces twice as many DFT points distributed across the bandwidth of interest as its real counterpart. Consequently, better spectral resolution as well as better signal-to-noise ratios are obtained in the complex case.

Recall, the DFT of a complex-valued signal is unique from $n \cdot f_s$ to $(n+1) \cdot f_s$. Hence, an *N*-point complex transform generates *N* DFT points spanning the bandwidth of interest. The real DFT generates only *N/2* points spanning the bandwidth of interest because the real transform is symmetric about its midpoint. The density of DFT points in the complex case is, therefore, twice that of the real case. As a result, the bandwidth represented by each DFT point[12] in the complex case is half that of its real counterpart. Hence, more accurate spectral estimates are obtained. Furthermore, narrower DFT bins contain less noise power and therefore provide better signal-to-noise ratios if the signal of interest can be completely contained in a single bin.

---

[12] This bandwidth is often referred to as a DFT "bin."

5.2.1.2 Frequency Estimation

Frequency estimation based on spectral peak location can produce inaccurate results when real-valued signals are near the half sampling frequency. As outlined above, the DFT of a real-valued signal is symmetric about the midpoint. When the frequency of a real-valued signal approaches the half sampling frequency, leakage from the image peak can interfere with the main peak through superposition. As shown in Figure 5-9, this interference can alter the location of the spectral peaks thus corrupting the corresponding frequency estimates.

The DFT of a complex-valued signal is unique from $n \cdot f_s$ to $(n+1) \cdot f_s$ and thus does not contain image peaks. Hence, image peak interference error is eliminated when complex sampling is performed.

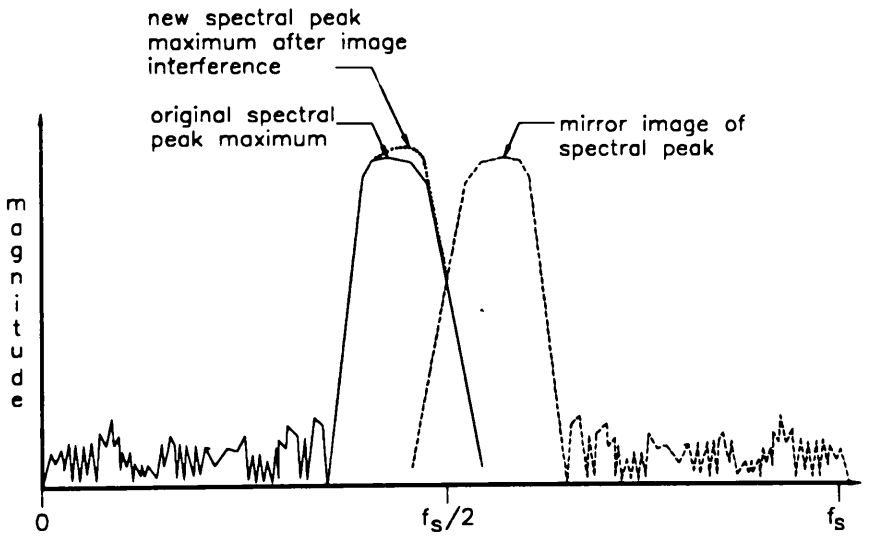


Figure 5-9. Corruption of spectral peak location in the discrete Fourier transform of a real-valued signal.

5.2.1.3 Sampling Rate Requirements

Complex sampling requirements for bandlimited signals are often lower than the corresponding real sampling requirements. As delineated above, complex sampling must be performed at a rate greater than or equal to the bandwidth of the sampled signal. Real sampling must be performed at twice the frequency of the highest frequency component in the sampled signal. For example to uniquely characterize a 100 kHz signal bandlimited to 100 kHz±10 kHz, the signal must be complex sampled at rate of 20 kHz. Conversely, if real sampling is performed, the signal would have to be sampled at 220 kHz. Clearly, the complex sampling rate required is far less demanding than the corresponding real sampling rate.

5.2.2 Complex Sampling Implementations

Recall, complex samples are formed by pairing real samples which are separated by 90° in phase on the sampled waveform. These real samples can be generated in many different ways. The implementation of complex sampling presented in Figure 5-6 is problematic. As shown, this implementation requires two analog-to-digital converters. Analog-to-digital converters meeting the anticipated precision and speed requirements are relatively expensive. Hence this alternative is costly. Furthermore, although analog 90° phase shifters are realizable for specific frequencies, they are very difficult to construct for a range of frequencies. Considerable error would be incurred if the bandwidth under consideration is large.

A digital alternative to the analog 90° phase shifter is the Hilbert transformer. As shown in Figure 5-10, a Hilbert transformer could be used to implement the required 90° phase shift in the digital domain thus eliminating the need for an analog phase shifter and a second analog-to-digital converter.

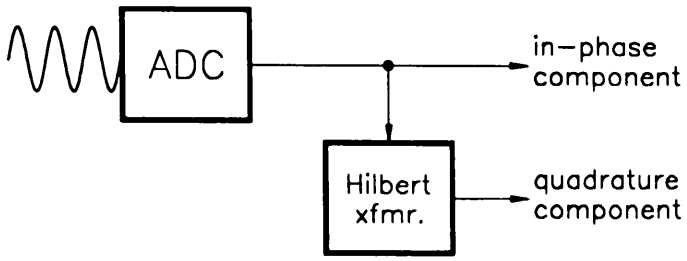


Figure 5-10. Complex sampling utilizing a Hilbert transformer.

Unfortunately, this solution is also problematic. First, similar to the analog phase shifter, an ideal Hilbert transformer is not realizable. FIR approximations to the ideal Hilbert transformer can realize an exact 90° phase shift but will also have an additional linear phase component required for a causal FIR system[17]. Second, the implementation of a Hilbert transformer is computationally intensive, and would be difficult to implement in real time at typical sampling rates.

A third implementation of complex sampling is outlined in Figure 5-11. By sampling at a rate satisfying Equation (5-5) and pairing time adjacent samples, complex sampling can be implemented with no additional mathematical computations using a single analog-to-digital converter.

$$f_{s(real)} = \frac{f_c}{N \pm 0.25} \tag{5-5}$$

where: f_c = *frequency of sampled signal* (Hz)

N = *any integer. N should be selected such that the complex sampling rate is greater than the bandwidth of the sampled signal.*

$f_{s(real)}$ = *analog – to – digital conversion rate* (Hz)

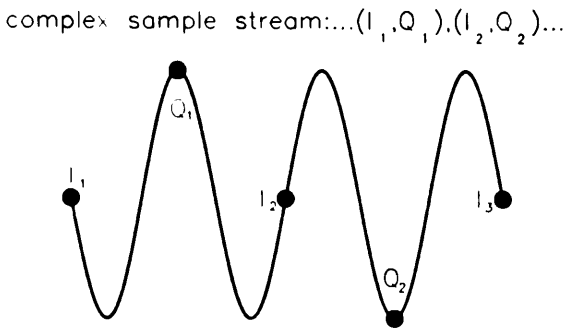


Figure 5-11. Implementation of complex sampling used on the APT digital receiver

This implementation also has limitations however. First, similar to the alternative implementations described above, this implementation only produces perfect in-phase and quadrature samples when the frequency of the sampled signal is f_c. Error is introduced into the sampling process for signals at all other frequencies. That is, the in-phase and quadrature components of each complex sample will not be separated by exactly 90° in phase if the frequency of the sampled signal is not precisely f_c or if there are multiple signals in the bandwidth of interest.

5.3 Alternative Receiver Algorithms

Many different digital signal processing techniques were investigated during the development of the digital receiver algorithm. These techniques were applied in different combinations to form a variety of plausible signal detection and power measurement algorithms. The following sections are intended to provide an overview of the digital signal processing techniques considered rather than comprehensive discussion of each algorithm.

5.3.1 Sequential Fast Fourier Transform Algorithm

The Sequential Fast Fourier Transform Algorithm, illustrated in Figure 5-12, was implemented on the first generation digital receiver designed by Ginger Runyon at Virginia Tech[13]. This algorithm utilizes two stages of fast Fourier transforms as filter banks to identify the carrier signal and make signal power measurements.

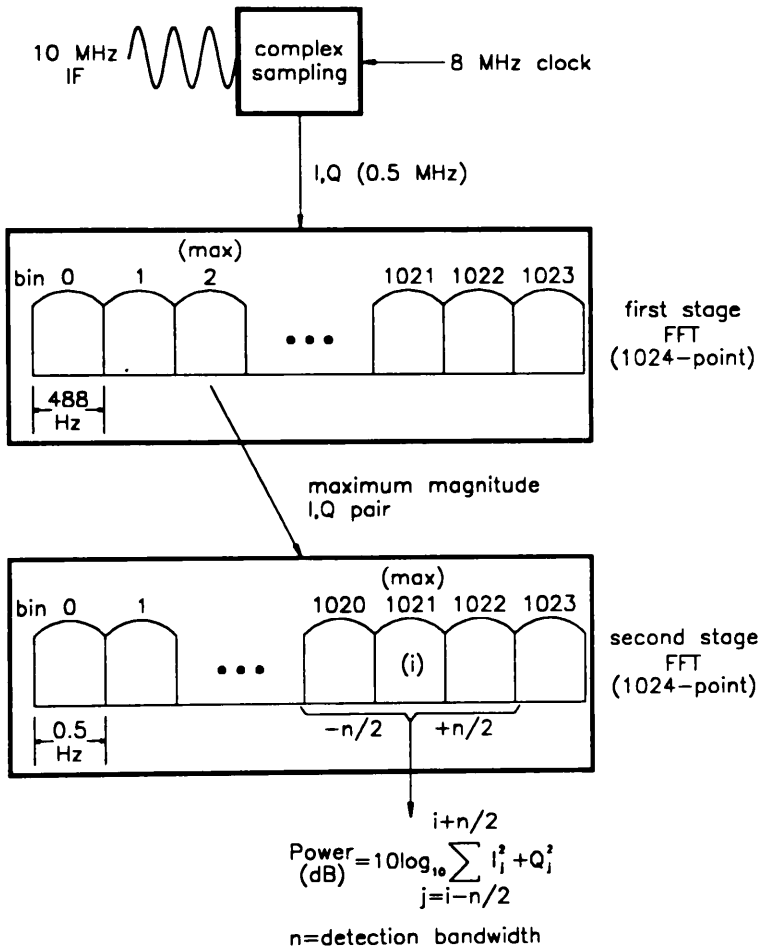


Figure 5-12. Sequential Fast Fourier Transform Algorithm block diagram.

The unlocked IF waveform is complex sampled at a 0.5 MHz rate. When 1024 complex samples have accrued, the first stage FFT is performed. The contents of the first stage FFT bin with the greatest magnitude are then input to the second stage FFT. When 1024 first stage FFTs have been performed, the second stage FFT is executed. Signal power measurements are then computed by summing the power contents of bins surrounding the spectral peak representing the carrier as:

$$Power(dB) = 10 \cdot log \sum_{j=i-\frac{n}{2}}^{i+\frac{n}{2}} I_j^2 + Q_j^2 \qquad (5\text{-}6)$$

where: i = *second stage FFT bin with the greatest magnitude*

I = *real component of* i^{th} *DFT point*

Q = *complex component of* i^{th} *DFT point*

n = *number of second stage FFT bins that must be summed to produce the desired detection bandwidth*

- $first\ stage\ bin\ width = \dfrac{0.5\ MHz}{1024\ bins} = 488.28\ Hz$

- $second\ stage\ FFT\ bin\ width = \dfrac{488.28\ Hz}{1024\ bins} = 0.476\ Hz$

- $n = \dfrac{detection\ bandwidth\ (Hz)}{0.476\ (Hz)}$

To make accurate signal power measurements, the receiver must lock the carrier signal near the center of a single first stage FFT bin. Should any of the carrier power fall into an adjacent bin, the accuracy of the ensuing signal power measurements will be significantly degraded. This coarse frequency

lock was not maintained on the first generation Virginia Tech digital receiver and the effects are noted by Runyon[13]. If, however, the receiver does maintain the required frequency lock, Runyon's simulation results indicate the receiver could generate signal power measurements accurate enough to satisfy Virginia Tech's measurement accuracy goals.

Runyon's simulations executed the Sequential Fast Fourier Transform Algorithm on simulated analog-to-digital converter samples of a stable, CW carrier without noise. The results of these simulations are reproducible. When actual signal-to-noise ratios anticipated at the receiver's input were simulated, however, the algorithm was not able to reliably identify the carrier signal below approximately 18 dB fade conditions. Recall, the first stage FFT bins are approximately 488 Hz wide. This bandwidth is too wide to produce enough signal-to-noise ratio to discern the carrier in deep fade conditions. This simulation result is supported by calculating the anticipated signal-to-noise ratio in a 488 bandwidth as:

$$SNR\ in\ 15\ Hz = 37.42\ dB \quad \textit{(See Appendix A)} \tag{5-7}$$

$$SNR\ in\ 488\ Hz = 37.42 - 10.0 \cdot \log_{10}\left(\frac{488\ Hz}{15\ Hz}\right) = 22.29\ dB$$

In summary, the Sequential Fast Fourier Transform Algorithm cannot meet the APT requirements without modifications to improve the signal-to-noise ratio at the output of the first stage FFT. This algorithm is, however, attractive because it provides accurate signal power measurements, the detection bandwidth is easily adjustable and the algorithm could provide spectral information about the entire acquisition bandwidth by making the results of the first stage FFT available to the user.

5.3.2 Adaptive FIR Filtering Algorithm

The Adaptive FIR Filtering Algorithm exploits the fact that digital filters can be reconfigured in software to compensate for changes in the carrier frequency. Signal power is detected at the output of three FIR filters which are redesigned in real time such that the carrier signal is locked into the passband of the center filter. The investigation of this algorithm was limited to the algorithm's ability to produce accurate signal power measurements. Hence, this algorithm was simulated as if the carrier signal had already been acquired. The signal power measurement configuration investigated is shown in Figure 5-13.

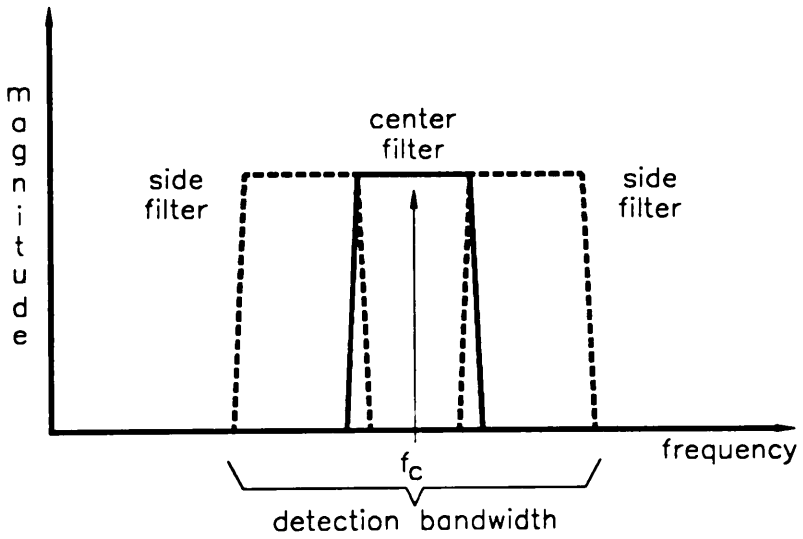


Figure 5-13. Adaptive filter configuration.

At the beginning of the simulation, the filter trio was centered on the carrier frequency. As such, signal power measurements were obtained by summing the power at the output of the three filters spanning the detection bandwidth. When the carrier frequency drifted, a greater proportion of the signal power would appear at the output of one of the side filters. The coefficients of all

three filters were then recalculated to center the passband of the middle filter on the new carrier frequency.

Although this algorithm did produce accurate signal power measurements, the FIR filters required by this algorithm were very long. Consequently, the investigation of this algorithm concluded that the required FIR filters and the occasional recalculation of filter coefficients could not be implemented in real time on the microprocessors available within the digital receiver budget.

5.3.3 Autoregressive Modeling Algorithm

Autoregressive modeling is a method of system identification. As shown in Figure 5-14, the Autoregressive Modeling Algorithm assumes the carrier signal samples are the output of a *black box* system which the modeling process attempts to characterize. This characterization includes the magnitude and frequency of each component of the input signal.

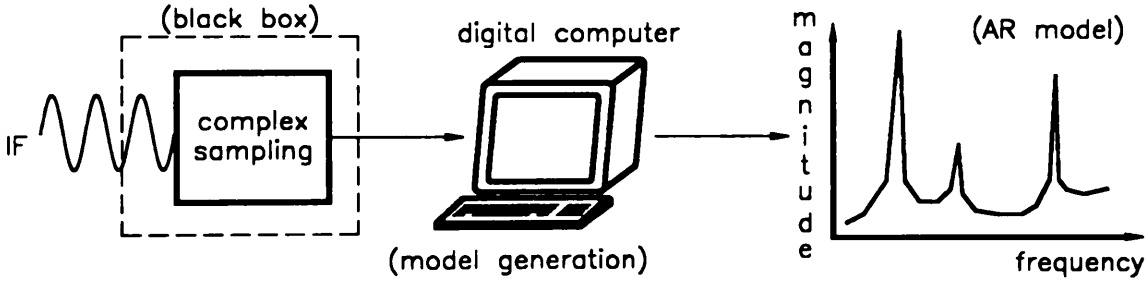


Figure 5-14. Block diagram of the autoregressive modeling algorithm.

There are many variants of the autoregressive modeling process. Unfortunately, the lack of available design time and the designer's inexperience with autoregressive modeling prohibited a comprehensive study of this technique. The simulations performed, did, however, identify several limitations. First, the order of the autoregressive model should be chosen

such that there exists one pole for each frequency component in the input signal. The ACTS 20 GHz beacon can be modulated in a number of different formats. Hence, the number of sinusoids in the bandwidth of interest is variable. Furthermore, in fade conditions, the modulation tones will fall below the noise floor of the receiver before the carrier. As a result, the order of the model should be reduced in fade conditions. Second, the simulation indicated that the modeling process is very sensitive to complex sampling error. For example, assume the frequency of a sinusoid, f_C , is estimated to be $f_C+\Delta f$. Furthermore, assume complex sampling is performed based on the erroneous frequency estimate and an autoregressive model of the system is developed. The simulations performed indicated that the frequency estimates derived from an autoregressive model would contain errors on the magnitude of Δf. Finally, noise in the bandwidth of interest would obviously diminish the performance of the modeling process. Unfortunately, lack of available design time prohibited a thorough investigation of the effects of noise on the modeling process.

5.3.4 Variable Sample Rate Algorithm

The Variable Sample Rate Algorithm attempts to lock the passband of a digital filter to the carrier signal by altering the rate at which the input waveform is digitized. As shown in Figure 5-15, the spectral position of a digital filter is completely dependent on the sampling rate of the data it operates on.

To acquire the carrier signal, the Variable Sample Rate Algorithm examines the bandwidth of interest in sections. The complex sampling rate is adjusted to position the passband of a digital filter in the area of the bandwidth to be characterized. After the unlocked carrier signal samples are bandlimited by the filter, the sample stream is decimated to a rate consistent with the width of the filter. A 1024-point fast Fourier transform is then performed. The

location and magnitude of the largest spectral peak in the output of the transform is recorded. The complex sampling rate is then altered and the process repeated to characterize a new section of the bandwidth. After all sections of the bandwidth are processed, the location of the spectral peak with the greatest magnitude identifies the carrier.[13] The time required to scan the entire bandwidth is inversely proportional to the width of the filter used. Narrower filters will, however, produce better signal-to-noise ratios and

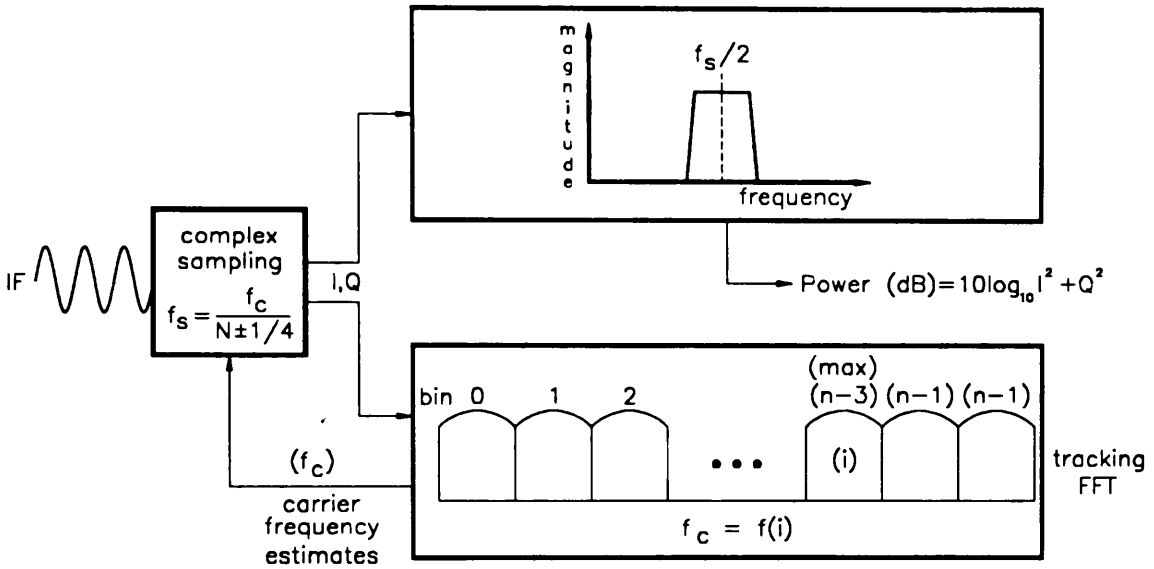


Figure 5-15. Block diagram of the variable sample rate algorithm.

therefore better performance during fades. As a benchmark, the simulations performed indicated that a receiver executing this algorithm could reliably identify the carrier in a 200 kHz bandwidth in 25 dB fade conditions in approximately 7 seconds.

To make signal power measurements, the sampling frequency is updated to translate the carrier signal through a narrowband digital detection filter. Changes in the carrier frequency are detected by performing a 2048-point

[13] Recall, the carrier is always the strongest signal in the badnwidth of interest.

FFT on a sliding window of 1 kHz complex samples. These changes are then compensated for by slewing the sampling frequency such that the detection filter is centered on the new carrier frequency.

Signal power measurements obtained via this algorithm were, unfortunately, inaccurate during sampling frequency updates. Although these measurement errors decayed when the sampling frequency remained constant, relatively small, slow changes in the sampling frequency caused signal power measurement errors up to 0.4 dB in magnitude.

5.3.5 Digital Mix Algorithm

The Digital Mix Algorithm digitally reproduces the frequency translation function provided by the analog frequency mixer. As shown in Figure 5-16, frequency mixing may be accomplished in the digital domain by multiplying complex samples of the first input signal with corresponding complex samples of the second input signal.[14]

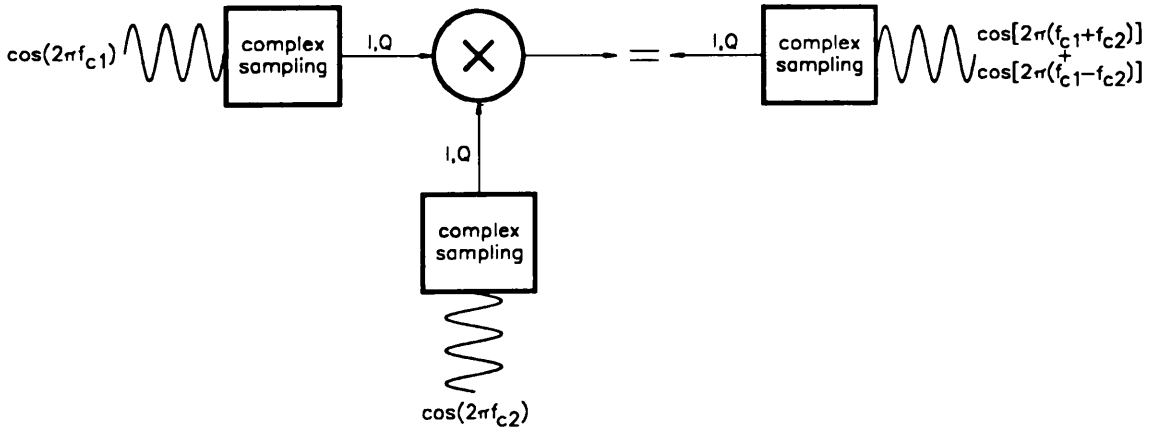


Figure 5-16. Digital mixing process.

[14] Both signals must be complex sampled at the same rate.

Similar to the analog mixing operation described in Chapter 3, the digital mixing operation produces an output signal which contains frequency components at both the sum and difference frequencies of the input signals. Since only one frequency component is typically desired, the unwanted component must be eliminated using a digital filter.

Although the simulation of this technique verified its integrity, our investigation concluded that digital mixing does not offer any advantages over analog mixing. Furthermore, digital frequency mixing is very computationally intensive at typical sampling rates. The digital hardware required to implement complex multiplication in real time is comparatively expensive. Hence, analog mixing could be implemented at lower cost.

5.4 APT Digital Receiver Algorithm

The algorithm selected for implementation on the APT digital receiver is a hybrid algorithm comprised of the digital signal processing and frequency translation techniques judged most effective by the simulations described above. A block diagram of the algorithm is shown in Figure 5-17. The carrier signal acquisition procedure, power measurement procedure, out-of-lock detection strategy and available supplemental functions are discussed in the following sections.

5.4.1 Carrier Signal Acquisition

In carrier signal acquisition mode, the APT receiver determines the current carrier frequency by identifying the strongest signal in the acquisition bandwidth using the results of multiple FFTs. Although the entire acquisition

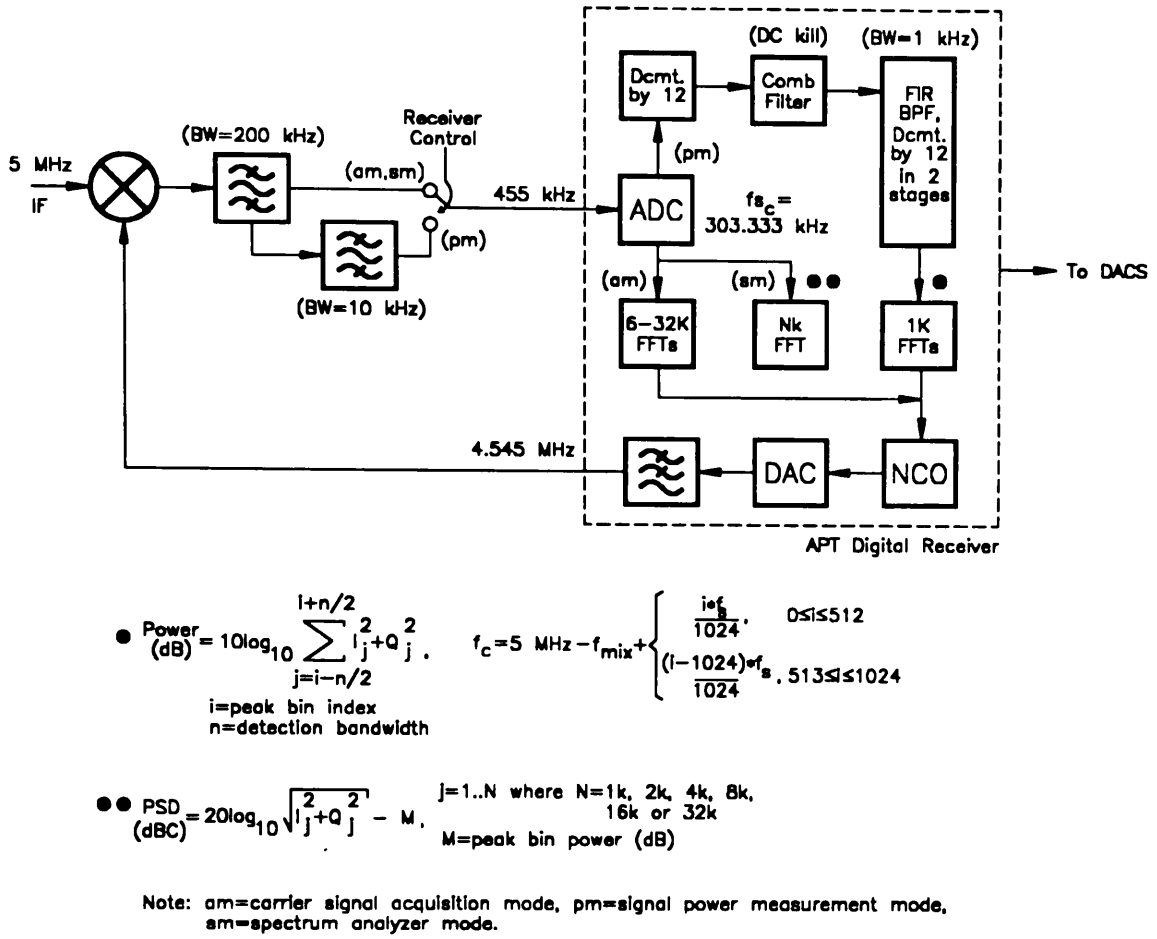


Figure 5-17. APT Digital Receiver Algorithm block diagram.

bandwidth is characterized by a single FFT, the receiver performs multiple FFTs to generate more accurate spectral estimates and reduce the possibility of false lock. The multiple FFT algorithm characterizes the 455 kHz± 151.5 kHz acquisition bandwidth in approximately 2 seconds.

When the APT receiver is in carrier signal acquisition mode, the filter selection switch on the IF board is set so the output of the 200 kHz filter is input to the receiver. Complex sampling is performed on the input signal according to:

$$f_{s(real)} = \frac{f_c}{N \pm 0.25} = \frac{455\,kHz}{1 - 0.25} = 606.667\,kHz \tag{5-8}$$

- *Pair time adjacent real samples*

$$f_{s(complex)} = \frac{f_{s(real)}}{2} = 303.333\,kHz$$

The carrier frequency was assumed to be 455 kHz to minimize complex sampling error in the acquisition bandwidth. Complex sampling error is directly proportional to the error in the carrier frequency estimate. Assuming the carrier is at the center of the acquisition bandwidth minimizes the possible error in the estimate, thereby minimizing sampling error. Simulations were conducted to determine the impact of sampling errors on spectral peak magnitude; the results of these simulations indicate that less than 3 dB of error is incurred over the entire acquisition bandwidth. The resulting 303.333 kHz complex sampling rate is sufficient to characterize the 200 kHz bandwidth provided by the IF filter and provides additional bandwidth to ensure the filter stopbands have rolled off significantly before aliasing occurs.

To identify the carrier signal, the APT receiver locates the strongest signal in the bandwidth of interest by performing a 32768-point FFT on the 303.333 kHz complex data and identifying the spectral peak with largest magnitude.[15] The carrier frequency is then calculated based on the spectral location of this peak using :

$$f_c = \left(1 + \frac{i}{32767}\right) \cdot f_{s(complex)} \tag{5-9}$$

[15] Recall, the carrier is always the strongest signal in the bandwidth of interest.

$$\text{where: } f_{s(complex)} = \text{complex sampling rate}$$
$$= \text{index of FFT bin with the greatest magnitude}$$
$$f_c = \text{carrier frequency estimate}$$

A single carrier frequency estimate generated in this fashion cannot, however, reliably identify the carrier signal. In the discrete Fourier domain it is unlikely that the spectral location of a DFT point will precisely correspond to the carrier frequency. As a result, the magnitude of spectral peaks in the discrete Fourier domain only approximate the magnitudes of the time domain signals they represent. For example, if, in the worst case, the spectral location of the carrier frequency falls directly between two DFT points, each DFT will reflect a 3 dB loss of power as shown in Figure 5-18(a).

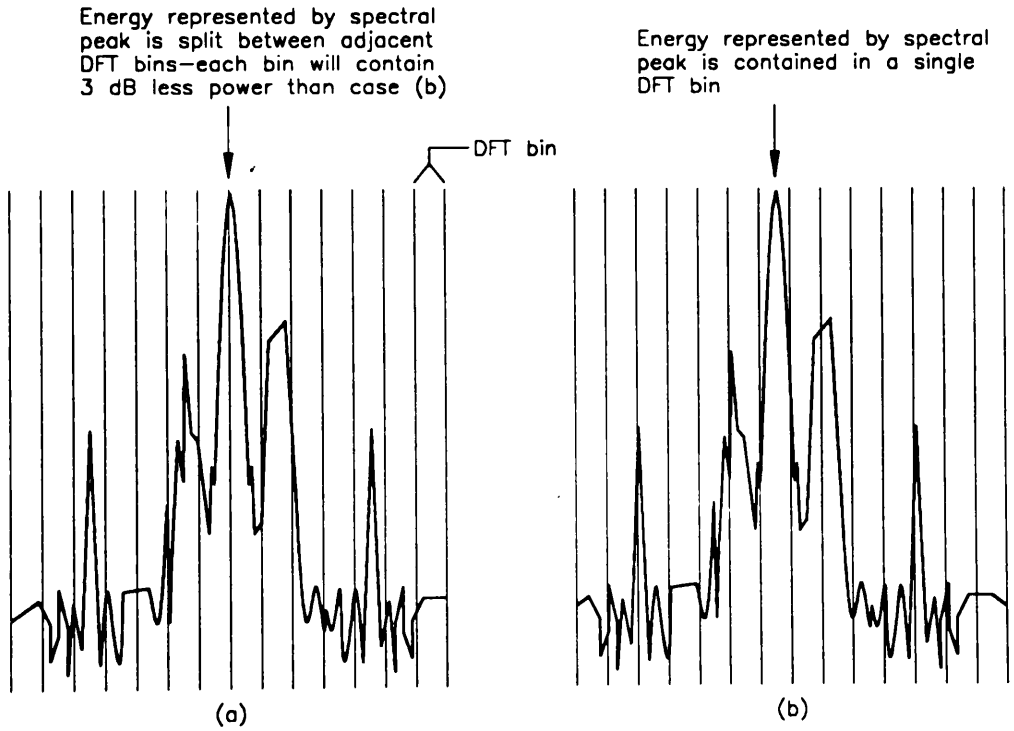


Figure 5-18. Estimating spectral peak magnitude. (a) Location of DFT points causes 3 dB loss in magnitude of spectral peak. (b) Shifted spectrum improves estimation of the spectral peak magnitude.

Errors due to the discrete nature of the FFT combined with errors due to imperfect complex sampling could theoretically reduce the amplitude of the spectral peak representing the carrier signal by 6 dB. Unfortunately, the strongest modulation tones on the ACTS 20 GHz beacon are only 6 dB below the carrier. Hence, a single FFT could mistakenly identify a modulation tone as the carrier signal. To increase the accuracy of the spectral estimates derived from a single 32768-point FFT, the APT receiver shifts the acquisition bandwidth by half of the spectral distance between adjacent DFT points (4.63 Hz)[16] and performs a second 32768-point FFT. The results of both FFTs are then compared to determine location of the spectral peak with the greatest magnitude. This procedure effectively halves the spectral distance between adjacent DFT points to provide an improved estimate of the amplitude of the signals in the bandwidth. See Figure 5-18(b).

Despite this precaution, it is hypothetically possible for random noise in the bandwidth to constructively interfere with a modulation tone such that the amplitude of the modulation tone is temporarily greater than that of the carrier. Although, statistically, the probability of this occurring is very small, the APT receiver further diminishes the possibility of false lock by repeating the entire dual FFT procedure three times to generate three independent carrier frequency estimates. All three carrier frequency estimates must match within anticipated carrier frequency drift and carrier frequency estimate quantization levels before the digital receiver will establish lock and enter signal power measurement mode. These calculations are shown in Equation (5-18):

---

[16] This shift in frequency is performed in the analog domain by altering the frequency of the signal driving the 5 MHz to 455 kHz downconversion.

$$\bullet \begin{matrix} \text{anticipated carrier} \\ \text{drift during the} \\ \text{acquisition process} \end{matrix} = \begin{matrix} \text{maximum} \\ \text{carrier drift} \\ \text{rate} \end{matrix} \cdot \begin{matrix} \text{acquistion} \\ \text{time} \end{matrix} \tag{5-10}$$

$$= 1.69 \frac{Hz}{sec.} \cdot 2 sec.$$

$$= 3.38\ Hz$$

$$\bullet \left(\begin{matrix} \text{carrier frequency} \\ \text{estimate quantization} \\ \text{level} \end{matrix} \right) = \frac{303.333\ kHz}{32768 - point\ \ FFT}$$

$$= 9.25\ Hz$$
$$= \pm 4.68\ Hz$$

$$\bullet\ Acquisition\ threshold = 3.38\ Hz + 4.68\ Hz$$
$$= 8.01\ Hz$$

If the initial carrier frequency estimates do not match, the receiver continues to generate additional estimates, using the dual FFT technique described above, until the lock condition is satisfied.

Extensive simulations were performed on this acquisition algorithm. The results of the simulations, which are shown in Figure 5-19, indicate that the APT digital receiver will reliably identify the carrier at signal-to-noise ratios down to 10 dB in a 20 Hz bandwidth (23 dBHz in a 1 Hz bandwidth, -26 dB relative to clear air conditions on the APT system).

Benchmarks for available digital signal processing microprocessors (DSPμPs) indicated that the APT receiver would be able to perform a 32768-point

complex FFT in approximately 0.2 seconds. Using this benchmark, the time required to execute the entire acquisition algorithm was estimated to be approximately 2 seconds.[17]

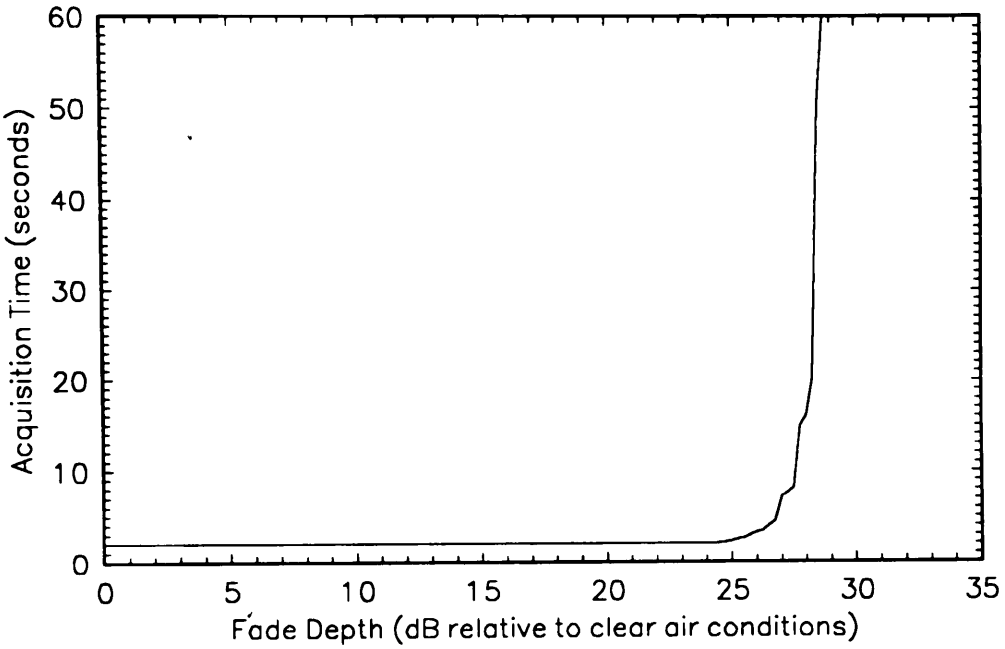


Figure 5-19. Simulated APT receiver acquisition performance.

5.4.2 Measuring Signal Power

In signal power measurement mode, the APT receiver locks the carrier signal to 455 kHz and measures the amount of power in a user selectable detection bandwidth. The APT receiver uses FIR filters implemented in multiple stages to bandlimit the input signal and then performs complex FFTs on a decimated sample stream to produce signal power measurements and detect changes in the carrier frequency.

---

[17] This estimate was later confirmed. See Chapter 7.

When the receiver enters signal power measurement mode, the carrier frequency information obtained in signal acquisition mode is used to translate the carrier frequency to 455 kHz. This translation is accomplished by updating the frequency of the output signal driving the 5 MHz to 455 kHz downconversion. After the carrier signal has been translated to 455 kHz, the receiver sets the position of the filter selection switch on the IF board such that the output of the 10 kHz filter is input to the receiver. See Figure 5-17. The complex sampling rate is then reduced to 25.278 kHz by decimating the 303.333 kHz complex sample stream by 12. This decimation can be performed without any loss of information because the input signal is now bandlimited to 10 kHz. Although, theoretically, a 10 kHz complex sampling rate is sufficient to characterize the bandlimited input signal, the 25.278 kHz complex sampling rate is maintained to ensure that the 10 kHz filter stopbands have rolled off significantly before aliasing occurs. As shown in Figure 5-20, the APT receiver then further bandlimits and decimates the complex sample stream to 1011.111 Hz using two FIR filter/decimate stages. The filtering/decimation process is performed in two stages in order to reduce the number of mathematical operations required.

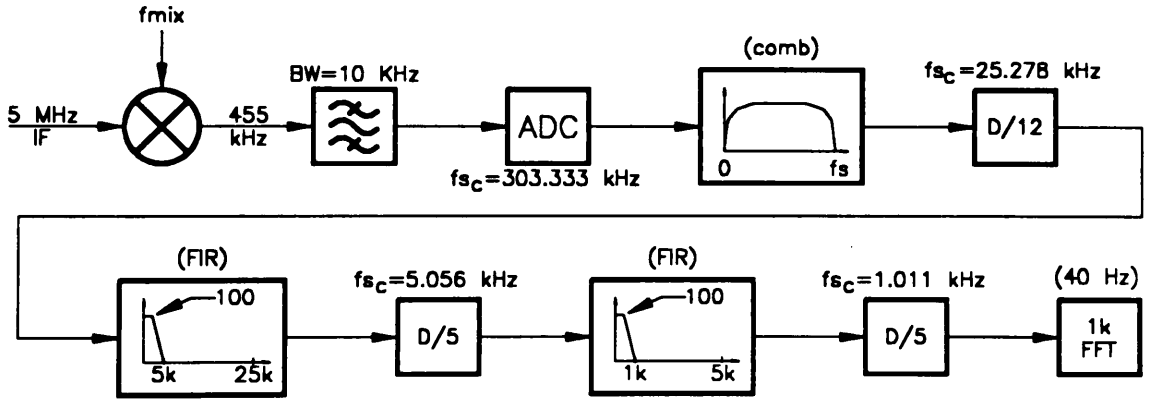


Figure 5-20. APT receiver signal power measurement mode filtering scheme.

Note, all filters in the bandlimit/decimation process have real coefficients. The need for complex filters was eliminated by incorporating the images present in real filters into the filter design. For example, as shown in Figure 5-21, an N Hz-wide bandpass filter for complex data can be created by designing an N/2 Hz-wide highpass filter with real coefficients. The passband of the N/2 Hz-wide highpass filter combines with its N/2 Hz-wide lowpass image to form the desired N Hz-wide bandpass filter for complex data.

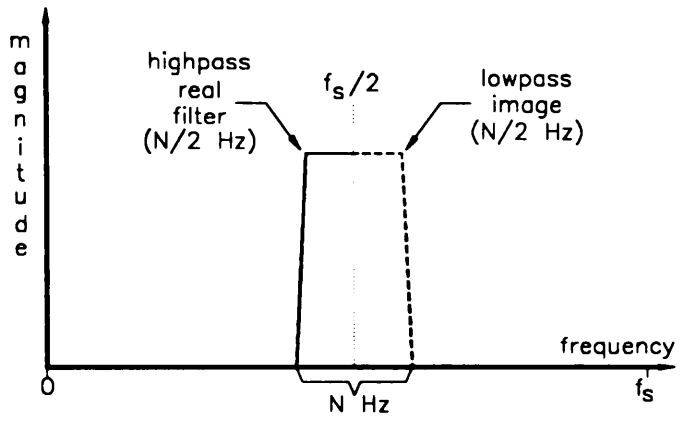


Figure 5-21. Design of a complex bandpass filter with real filter coefficients.

To make signal power measurements and detect changes in the carrier frequency, the APT receiver performs 1024-point FFTs on a sliding window of the 1011.111 Hz complex data. An FFT is performed each time 25 new complex samples enter the window. Hence, approximately 40 1024-point FFTs are performed per second. A Hanning window is applied to the time domain data before each FFT is performed to reduce windowing effects. The results of each transform are used to generate a signal power measurement via Equation (5-11).

$$Power\ (dB) = 10 * log_{10} \sum_{j=i-\frac{n}{2}}^{i+\frac{n}{2}} (I_j^2 + Q_j^2) \qquad (5\text{-}11)$$

where: i = *index of FFT bin with the greatest magnitude*

n = *number of bins that must be summed to produce the desired detection bandwidth*

I_i = *real component of* i^{th} *DFT point*

Q_i = *complex component of the* i^{th} *DFT point*

1 Hz and 20 Hz signal power measurements are then derived from this 40 Hz signal power measurement stream. As shown in Figure 5-22, the 40 Hz stream is bandlimited to 10 Hz using a 30 dB/octave windowing function and then decimated by 2 to obtain 20 Hz measurements. The 40 Hz stream is also bandlimited to 0.5 Hz using a second 30 dB/octave window and decimated by 20 to obtain 1 Hz measurements.[18]

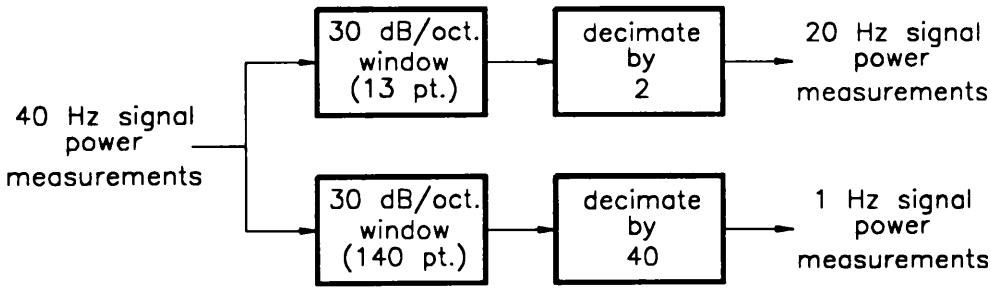


Figure 5-22. Derivation of 20 Hz and 1 Hz signal power measurements from a 40 Hz measurement stream using 30 dB/octave window filtering.

The 30 dB/octave window was selected because it has the widest main lobe and the steepest side lobe attenuation of the commonly used windowing

[18] The windows were not cascaded in order to minimize delay through the system.

functions. Although more rectangular filters can, of course, be designed the delay associated in these filters was intolerable in the APT system.

The passbands of the FIR filters used to bandlimit the complex sample stream to 1011.111 Hz are 100 Hz wide as shown in Figure 5-20. To prevent the carrier signal from drifting out of these passbands, the APT receiver maintains a coarse frequency locked loop using carrier frequency estimates derived from the aforementioned signal power measurement mode FFT results. Equation (5-12) describes the relationship between spectral peak location and carrier frequency.

$$f_c = f_{mix} + \begin{cases} f_{s(complex)} \cdot \left(1 + \dfrac{i}{1024}\right), & 0 \le i \le 512 \\ f_{s(complex)} \cdot \left(1 + \dfrac{(i-1024)}{1024}\right), & 512 \le i \le 1023 \end{cases} \tag{5-12}$$

$$\begin{aligned} \textit{where: } f_{s(complex)} &= \textit{complex sampling rate} = 1011.111\ \textit{kHz} \\ i &= \textit{index of DFT bin with the greatest magnitude} \\ f_{mix} &= \textit{current frequency output to the analog mixer} \\ f_c &= \textit{carrier frequency estimate} \end{aligned}$$

When the carrier frequency drifts more than 25 Hz away from 455 kHz, the receiver slews the frequency of the signal driving the 5 MHz to 455 kHz downconversion to force the carrier signal back toward 455 kHz. Figure 5-23 summarizes the coarse frequency locked loop used by the APT receiver.

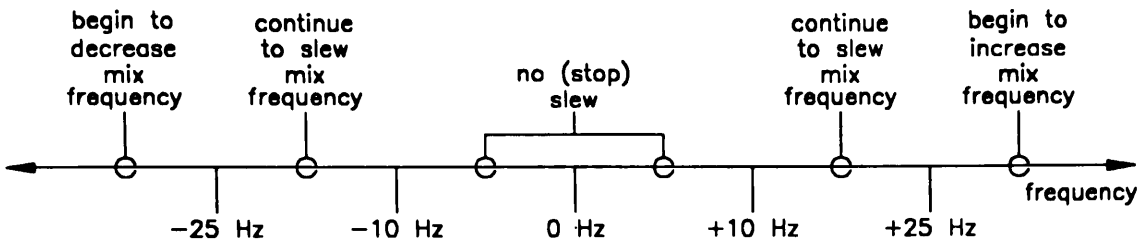


Figure 5-23. APT digital receiver frequency lock scheme.

The signal power measurement and frequency detection techniques described above were simulated at a variety of signal-to-noise ratios. The results of the simulations indicated that the receiver would be able to detect 1 Hz variations in the carrier frequency and would measure the power in the detection bandwidth with 0.1 dB accuracy at frequency drift rates up to 3 Hz/second.

5.4.3 Detecting the Out-of-Lock Condition

To make accurate signal power measurements, the receiver must be able to reliably identify the spectral peak representing the carrier signal in the results of the signal power measurement mode FFTs described in Section 5.4.2. The receiver examines the variance of the difference between consecutive carrier frequency estimates to determine its ability to identify the carrier signal. If the receiver is locked to the carrier signal, the difference between consecutive carrier frequency estimates is small because the carrier frequency drifts relatively slowly. Conversely, when the carrier signal falls below the system noise floor, the receiver attempts to lock to the random noise in the bandwidth, which is approximately white. The corresponding carrier frequency estimates are uncorrelated. Hence, the differences between these estimates have a large variance. A variance threshold of 2.0 has been empirically determined to indicate the out-of-lock condition.

As described above, the receiver generates an estimate of the carrier frequency based on the results of each FFT (40 Hz). Each time the receiver

generates a new carrier frequency estimate, the variance of the differences between the past 40 estimates is computed using Equation (5-13).

$$\sigma^2 = \frac{\sum_{i=1}^{n}(\delta(i) - \mu)^2}{n} \tag{5-13}$$

$$\begin{aligned} \textit{where: } \delta(i) &= f_c(i) - f_c(i+1) \\ f_c(i) &= i^{th}\ \textit{carrier frequency estimate} \\ \mu &= \textit{geometric mean of } \delta(i),\ i = 1..40 \\ n &= 40 \\ \delta^2 &= \textit{variance} \end{aligned}$$

When the receiver determines that it is no longer locked to the carrier signal, the last valid carrier frequency estimate is used to translate the carrier signal to 455 kHz. The receiver then continues to generate carrier frequency estimates based on the 40 Hz signal power measurement mode FFT data for a period of 29.6 seconds in an attempt to reacquire the carrier signal. Recall, the DFT bin width resulting from the 40 Hz signal power measurement mode FFTs is approximately 1 Hz. The DFT bin width resulting from the 32768-point carrier signal acquisition mode FFTs is approximately 9.5 Hz. Hence, better signal-to-noise ratios are obtained when the receiver is in signal power measurement mode. As a result, the receiver will reacquire the carrier signal at a lower signal-to-noise ratio in signal power measurement mode. Even at the maximum carrier frequency drift rate specified in Chapter 2, the carrier will remain in the passband of the FIR filters for 29.6 seconds.[19] If the receiver does not reacquire the carrier signal in this time[20], the receiver will reenter signal acquisition mode.

---

[19] The filter transition bands begin at 455 kHz±50 Hz, $\frac{50\ Hz}{1.69\ Hz/\text{sec.}} = 29.6$ sec.

[20] The receiver assumes that it has reacquired the carrier signal if the variance of 40 consecutive carrier frequency estimates is less than 2.

5.4.4 Available Supplemental Functions

The APT receiver provides several functions not explicitly required by the ACTS propagation experiment. These supplemental functions enhance the receiver's output and demonstrate the versatility of the system.

5.4.4.1 Adjustable Detection Bandwidth

The APT receiver detection bandwidth can be adjusted by the user without interrupting the operation of the receiver. As described in Section 5.4.2, the APT digital receiver computes signal power measurements by summing the contents of a series of DFT bins. Hence the detection bandwidth can be altered by changing the number of bins summed to produce each signal power measurement. The APT digital receiver can provide detection bandwidths ranging from 2 Hz to 50 Hz in approximately 1 Hz increments.

5.4.4.2 Optional High Sample Rate

Although the collective experience of the ACTS Users' Community suggests that a signal power measurement rate of 1 Hz is sufficient to characterize the atmospheric phenomena of interest, several users have requested signal power measurements at a higher sampling rate. To satisfy the needs of these users, the APT receiver provides signal power measurements at a rate of 20 Hz in addition to the required 1 Hz measurements. See Section 5.4.2 for more information.

5.4.4.3 Spectrum Analyzer Output

The APT receiver can provide approximate spectral information about a 303.333 kHz bandwidth centered on the carrier frequency upon request. To do so, the receiver translates the carrier signal to 455 kHz by updating the frequency of the output signal driving the 5 MHz to 455 kHz downconversion. The position of the filter selection switch on the IF board is set such that the output of the 200 kHz filter is input to the receiver. To generate the desired spectral information, the receiver performs an FFT of specified length on the 303.333 kHz complex data stream. The results of the transform are converted to decibels relative the magnitude of the carrier (dBC) and made available to the user. The APT receiver software can provide 1024, 2048, 4096, 8192, 16384 or 32768-point spectra on request.

It should be noted that the relative magnitudes of spectral peaks in periodograms are only approximate. As discussed in Section 5.4.1, errors due to the discrete nature of the DFT and errors due to imperfect complex sampling can significantly affect spectral peak magnitude. Furthermore, the complex sampling errors described in Section 5.2.2 cause signals at frequencies other than 455 kHz to have an image peak in the Fourier domain as shown in Figure 5-24. The magnitude of this image peak is directly proportional to the spectral distance from 455 kHz.

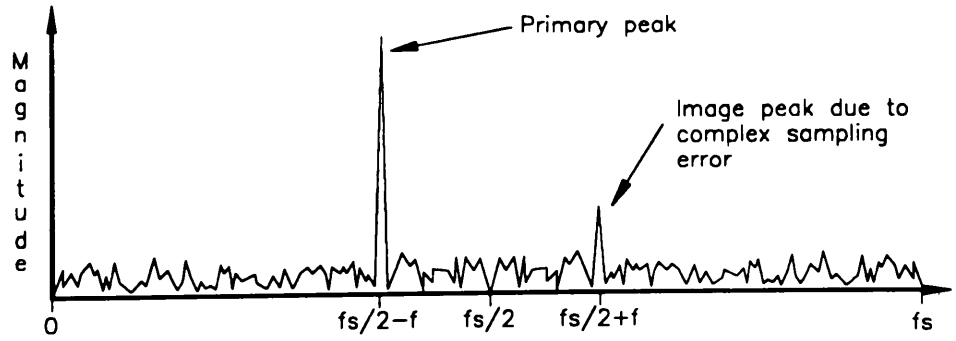


Figure 5-24. Results of a complex DFT. Image peak is due to imperfect complex sampling.

5.4.4.4 Carrier Frequency Estimates

The APT digital receiver calculates the current carrier frequency to remain locked to the signal. These carrier frequency calculations are accurate to within ±0.5 Hz and are available to the user.

5.4.4.5 Track hold

Operation of the carrier frequency tracking algorithm described in Section 5.4.2 can be disabled at the user's request. When a *track hold* command is issued to the receiver, the receiver uses the current carrier frequency estimate to translate the carrier to 455 kHz. No further tracking or out-of-lock detection is performed until another *track hold* command is issued.

The track hold function was implemented to minimize the amount of data lost during APT radiometer system calibrations. The radiometers on the APT system require calibration very frequently. During the calibration process the beacon signal is not available at the receiver's input. Hence, no signal power measurements can be made. By issuing a *track hold* command, the user can prevent the receiver from reentering signal acquisition mode when the carrier signal is removed from the receiver's input. When the radiometer calibration process has been completed and the carrier signal is restored to the receiver's input another *track hold* command must be issued. The receiver will then resume the signal power measurement process. Hence, this procedure reduces system downtime by preventing the receiver from unnecessarily executing the carrier signal acquisition algorithm.

5.4.5 Algorithm Modification

When the first APT receiver was constructed, an amplifier in the analog-to-digital converter circuit created a small dc offset on the receiver's input signal. Unfortunately, when the APT receiver decimates the 303.333 kHz complex sample stream to 1011.111 Hz this dc signal aliases to the same spectral location as the 455 kHz carrier signal. The magnitude of this dc signal became significant at very low signal-to-noise ratios and caused signal power measurements to become nonlinear. To eliminate the dc offset signal, the APT receiver signal power measurement mode algorithm was modified to include a comb filter which removes the dc signal without altering signals in the signal power measurement mode bandwidth. The comb filter in Figure 5-25 is applied to the 303.333 kHz complex data as the complex samples are assembled. As shown, the comb filter has notches at $n \cdot f_s$ and $(n+1) \cdot f_s$ and is flat across the bandwidth of interest. Hence, the comb filter removes the dc offset from the complex sample stream without altering signals in the signal power measurement mode bandwidth.

5.5 Algorithm Selection Summary

This chapter has presented an overview of the digital signal processing techniques considered for use in the APT receiver algorithm. The APT digital receiver algorithm is a hybrid algorithm comprised of the most successful techniques investigated. The algorithm was chosen for its simplicity and use of well known digital signal processing techniques. Additionally, the algorithm is flexible enough to provide all of the supplemental functions discussed in Chapter 2. Simulation of the APT algorithm indicated the receiver would meet all the design goals set by the ACTS Users' Community and the Virginia Tech Satellite Communications Group.

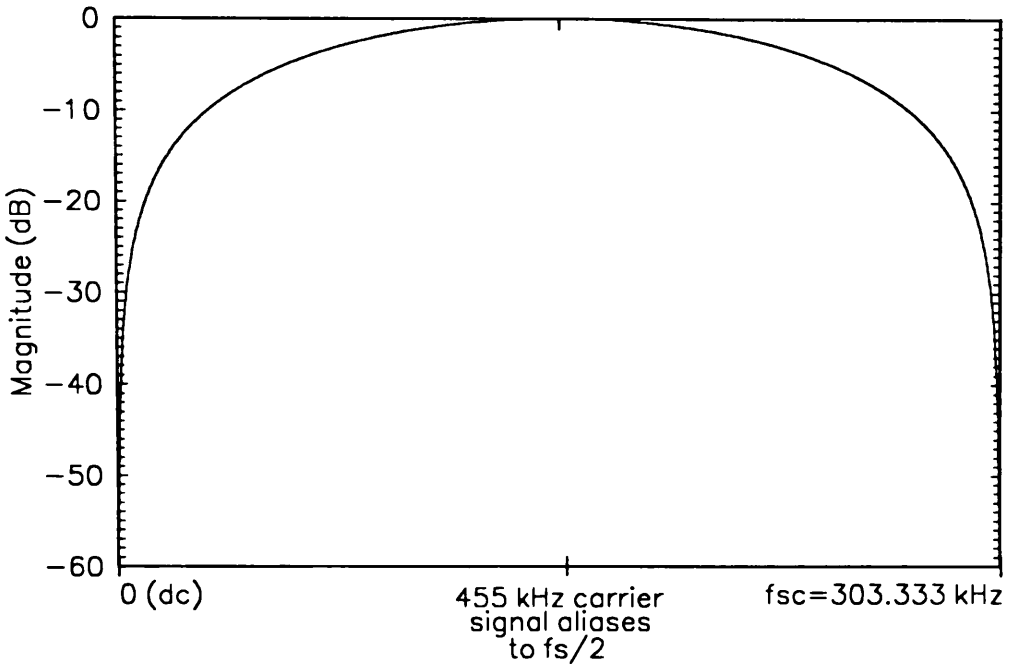


Figure 5-25. Magnitude response of the comb filter added to the APT receiver algorithm to remove dc signals.

Chapter 6

System Development

Custom hardware and software was designed and constructed to implement the APT receiver algorithm described in Chapter 5. A block diagram showing the relationship between receiver hardware and the rest of the APT system is shown in Figure 6-1. The following sections describe receiver system requirements and then present a detailed description of the solution designed to fulfill those requirements.

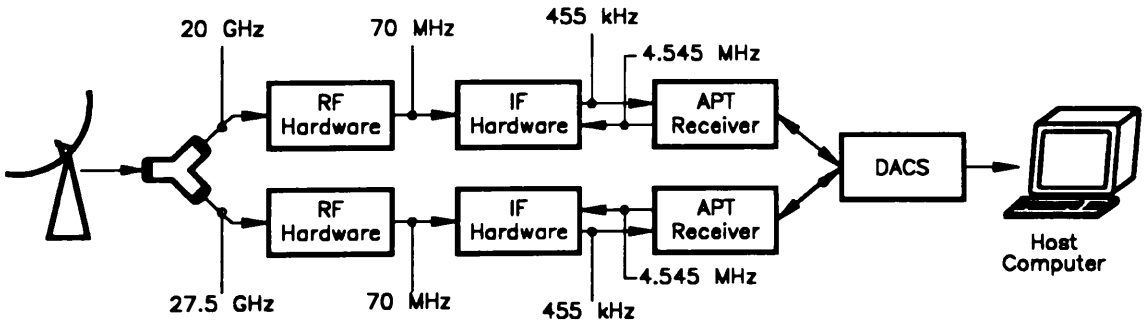


Figure 6-1. APT system hardware block diagram.

6.1 Hardware

Implementation of the APT receiver algorithm requires the following system level facilities:

- The ability to execute approximately 15 MOPS
- 10 or 12 bit analog-to-digital conversion at 606.667 MSPS
- The ability to generate signals with very low phase noise at precise frequencies in the 4.545 MHz ± 100 kHz bandwidth
- Bi-directional communication port
- 3.74 Mbits of RAM memory configured as 32 and 48-bit words

Furthermore, the APT design requires the receiver subsystem to operate at ambient temperatures up to 60° Celsius.

A study of the predominant commercially available DSP platforms indicated that these requirements could not be met with an affordable off-the-shelf system. Hence, custom subsystems were designed to meet each requirement. When integrated, these subsystems form a very flexible, low cost DSP platform capable of implementing the APT receiver algorithm. A block diagram of the APT receiver hardware is shown in Figure 6-2.

The following sections describe the APT receiver hardware in terms of the subsystems shown above. A schematic of the entire APT receiver system is included in Appendix D. This section concludes with descriptions of the APT receiver printed circuit board and the receiver subenclosure.

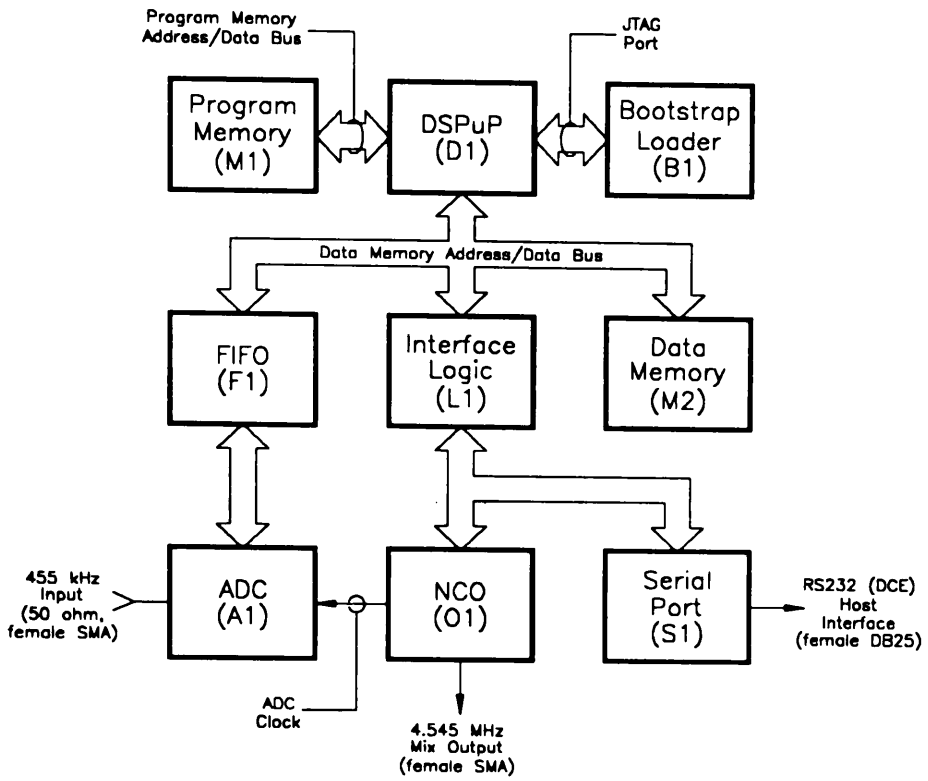


Figure 6-2. APT receiver hardware block diagram.

6.1.1 Receiver Subsystems

6.1.1.1 DSPμP (D1)

The core of the APT receiver design is the digital signal processing microprocessor (DSPμP). The DSPμP

- controls the initialization of all other components in the design
- performs all of the mathematical calculations necessary to implement the APT receiver algorithm
- communicates results and status information to a host system via the interface protocol outlined in Appendix B

Selection criteria for the DSPμP encompassed not only the capabilities and cost of the chip itself but also the cost and availability of development tools and the vendor's reputation for customer support. An extensive investigation was conducted to evaluate available DSPμPs in terms of these criteria. The Analog Devices ADSP21020 was selected for use on the APT receiver based on the results of that investigation shown in Table 6-1.

Table 6-1. Candidate DSPμPs

Vendor	Part #	Price	Data[21] Format	Cycle Time (ns)	FFT Benchmark (ms)	Comments
Analog Devices	ADSP21020-60 ADSP21020-80 ADSP21020-100	$195 $265 $320	32,40 float	60 50 40	1.16 0.96 0.77	ADSP21020-EX-ICE in circuit emulator available, $3000 ADSP21020-DSW-PC assembly language development software (including a simulator) available, $995 ADSP21020-EX-LAB stand-alone evaluation board available, $995 Context switching facilitates efficient interrupt handling All instructions execute in 1 clock cycle High degree of parallelism Recommended by Fujitsu Same chip available at different speeds-exact requirements can be empirically determined Slower external clock simplifies PCB design No on-chip memory

[21] 32=32 bit precision, float=floating point math

Table 6-1 (continued). Candidate DSPμPs

Vendor	Part #	Price	Data Format	Cycle Time (ns)	FFT Benchmark (ms)	Comments
Zoran	ZR4325	$695	32, float	80	1.73	Chip is very expensive Development software available, $5000
NEC	μPD77230	$560	32, float	150	11.78	Very slow
Fujitsu	MB86232	$357	32, float	75	≈1.50	Fujitsu support in the United States is minimal unless purchase exceeds $100,000 Fujitsu field applications engineer recommends ADSP21020
BIT	Only offer discrete *multiply* chips					Hardware required to implement the APT algorithm would be extremely complex
TI	TMS320C30 TMS320C30-27	$219 $163	32, float	60 74	3.04 3.71	2 chips might be required to implement APT algorithm High degree of parallelism University owns development system In-circuit emulator available for rent, $1000/month
AT&T	DSP32C	$300	32, float	80	3.20	Difficult to use
IDT	Only offer discrete *multiply-accumulate* chips					Hardware required to implement the APT algorithm would be extremely complex
TI	TMS320C40	?	32, float	40	1.80	Designed primarily for multiprocessor applications Not yet available
Motorola	DSP96002	$480	32, float	50	1.13	Stack based-no context switching

The ADSP21020 can execute a large number of operations in parallel. The ADSP21020 supports two completely independent memory busses and an on-

chip two way set-associative cache. As a result, the ADSP21020 can fetch multiple combinations of program instructions and data simultaneously. Furthermore, instructions using other resources on the chip can be executed in parallel with memory accesses. In all, the chip can perform an instruction fetch, an addition operation, a subtraction operation, a multiplication operation, two independent memory accesses and test a condition in a single clock cycle. Additionally, as shown in Table 6-1, the ADSP21020 is a floating point microprocessor. The ability to perform floating point calculations facilitated the task of programming the APT algorithm by eliminating the scaling operations required by fixed point DSPμPs.[22] The selection of the ADSP21020 was further supported by Analog Devices' outstanding reputation for customer support and the availability of low cost development tools. A 12-pin header is included in the APT receiver design to permit the use of the ADSP21020 JTAG in-circuit emulator.

6.1.1.2 Memory (M1,M2)

As described in Section 6.1.1.1, the ADSP21020 has two completely independent memory systems. These systems are termed *program memory* and *data memory*. Program memory stores executable code as well as data while data memory stores only data. To fully utilize the capabilities of the ADSP21020, both memory systems are used in the APT receiver design. Moreover, to simplify the design process as well as facilitate system maintenance, both memory systems have been designed using the same components. Memory system requirements include the following:

---

[22] Fixed-point DSPμPs must scale operands in order to reduce coefficient quantization and round-off errors. Conversely, coefficient quantization and round-off errors are usually negligible in floating point math[17].

- 37KW of 48-bit wide *program memory*
- 32KW of 32-bit wide *data memory*[23]
- 35 ns access time (zero wait-state operation at 20 MHz[24])

A study of available memory components quickly showed that a high density system was required. Traditional DIP memory systems would entail a large number of components and, as a result, an intolerable amount of printed circuit board area. The Cypress Semiconductor line of SIMM static ram memory modules was, therefore, selected for use on the APT receiver. The SIMM modules provide a large amount of memory in a compact package and are available in configurations compatible with the APT receiver requirements. Specifically, data memory requirements were fulfilled with a single 64KWx32 module while the program memory requirements were met with two 64KWx32 modules. As shown in Figures 6-3 and 6-4, the 35 ns version of these SIMM modules satisfies the ADSP21020 timing requirements for zero wait state operation at 20 MHz.

Furthermore, this memory system is easily expandable. If future receiver applications require additional memory, existing SIMM sockets can be repopulated with larger SIMM modules.[25]

Finally, it should be noted that there is no ROM in the APT receiver memory system. All executable code is downloaded to the system RAM using the bootstrap loader discussed in Section 6.1.1.6.

---

[23] The ADSP21020 supports the IEEE standard floating point format (32 bits) and the extended floating point format (40 bits). The APT receiver algorithm only requires IEEE standard floating point precision. Hence, only 32 bits of the data bus are utilized.

[24] The system tests described in Chapter 7 indicate the ADSP21020 can execute the APT receiver algorithm in real time running at 15 MHz. The entire system has, however, been designed to operate at 20 MHz in order to make future expansions possible.

[25] SIMM modules up to 256KWx32 are available.

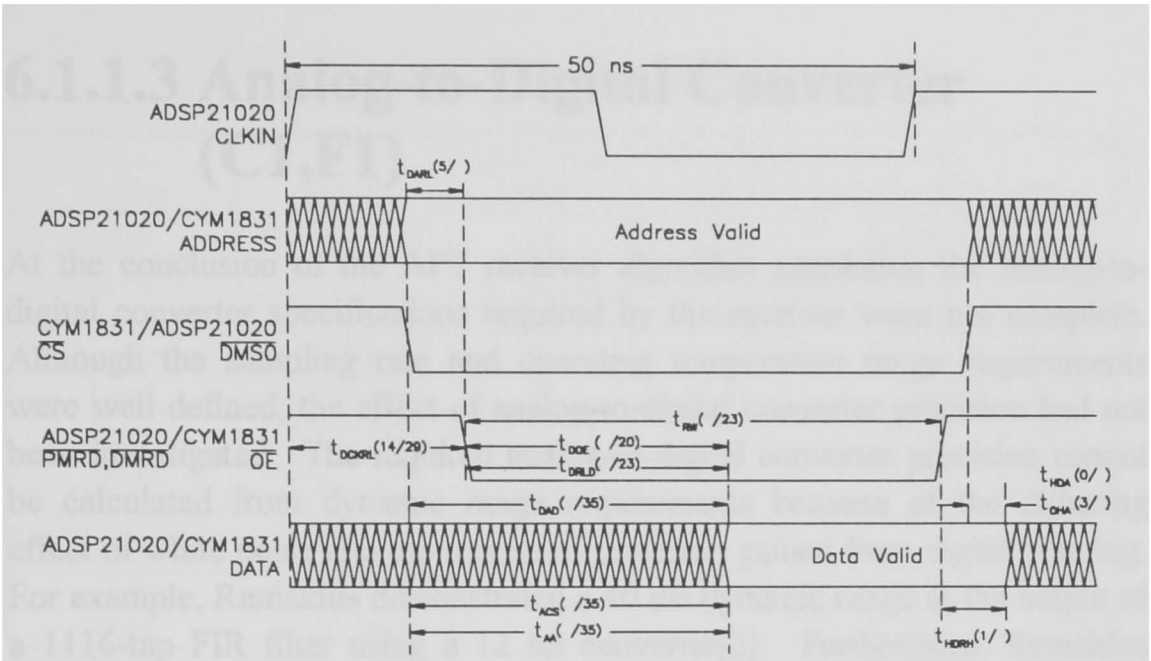
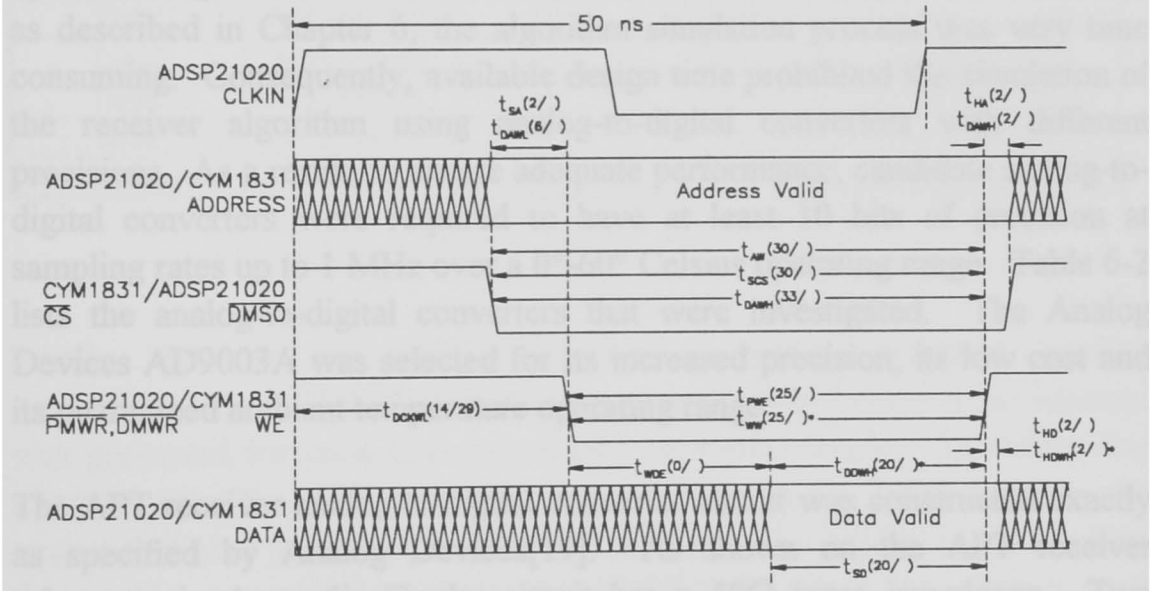


Figure 6-3. ADSP21020-CYM1831PM35C memory read timing diagram.



* These ADSP21020 specifications have changed on the latest version of the ADSP21020 data sheet. These changes have created small worst-case disagreements with the CYM1831 timing specifications. This interface has, however, been tested and was found to be reliable.

Figure 6-4. ADSP21020-CYM1831PM35C memory write timing diagram.

6.1.1.3 Analog-to-Digital Converter (C1,F1)

At the conclusion of the APT receiver algorithm simulation the analog-to-digital converter specifications required by the receiver were not complete. Although the sampling rate and operating temperature range requirements were well defined, the effect of analog-to-digital converter precision had not been investigated. The required analog-to-digital converter precision cannot be calculated from dynamic range requirements because of the dithering effect of white noise and the increased precision gained from digital filtering. For example, Remaklus demonstrated a 40 dB dynamic range at the output of a 1116-tap FIR filter using a 12 bit converter[2]. Furthermore, Remaklus conducted extensive studies but failed to find any quantization effects[2]. To determine the analog-to-digital converter precision required for a given dynamic range and noise level, the system must be simulated. Unfortunately, as described in Chapter 6, the algorithm simulation process was very time consuming. Consequently, available design time prohibited the simulation of the receiver algorithm using analog-to-digital converters with different precisions. As a result, to ensure adequate performance, candidate analog-to-digital converters were required to have at least 10 bits of precision at sampling rates up to 1 MHz over a 0°-60° Celsius operating range. Table 6-2 lists the analog-to-digital converters that were investigated. The Analog Devices AD9003A was selected for its increased precision, its low cost and its guaranteed ambient temperature operating range.

The APT receiver analog-to-digital converter circuit was constructed exactly as specified by Analog Devices[19]. As shown on the APT receiver schematic in Appendix D, the circuit has a 50Ω input impedance. Two operational amplifiers are used to limit the input signal to ±2.53 volts[26] and a third operational amplifier provides a voltage gain of 4.

---

[26] ±2.53 volts is the full-scale voltage range of the AD9003A.

Table 6-2. Candidate analog-to-digital converters.

Vendor	Part #	Resolution (bits)	Throughput (MSPS)	Analog Bandwidth	Cost
TRW	TDC1020J1C	10	20.0	60 MHz	$148
Analog Devices	AD9020	10	60.0	150 MHz	$233
Analog Devices	AD9060	10	75.0	150 MHz	$260
Analog Devices	AD9003A	12	1.0	4 MHz	$143
Datel	ADC500MC	12	1.54	no S/H	$311
Datel	ADS117MC	12	2.0	5 MHz	$299
Datel	ADS132MC	12	2.0	8 MHz	$346
Analog Devices	AD671-500	12	2.0	no S/H	$105
Analog Devices	AD1671-500	12	2.0	not available yet	
Datel	ADC530MC	12	2.22	no S/H	$316
Datel	ADS118MC	12	5.0	30 MHz	$399
Datel	ADS131MC	12	5.0	30 MHz	$549
Datel	ADS130MC	12	10.0	30 MHz	$775
Analog Devices	AD9005	12	10.0	38 MHz	$765
TRW	THC120256B	12	10.0	70 MHz	$500
TRW	THC120153B	12	10.0	70 MHz	$795
Datel	ADS120MC	12	20.0	not available yet	
Datel	ADS942MC	14	2.0	1.75 MHz	$499
Datel	ADS944MC	14	5.0	not available	
Analog Devices	AD9014	14	10.0	50 MHz	$2750

Finally, although the ADSP21020 has exceptional interrupt handing capabilities, the ADSP21020 would be unnecessarily loaded if an interrupt was generated for each analog-to-digital converter sample. To reduce this interrupt rate, the APT receiver design includes a 1K FIFO between the analog-to-digital converter and the ADSP21020. The IDT72225L25J buffers the sample stream and interrupts the ADSP21020 when 896 samples have accumulated. The AD9003A-IDT72225L25J and ADSP21020-IDT72225L25J unidirectional interfaces are described by the timing diagrams in Figures 6-5 and 6-6.

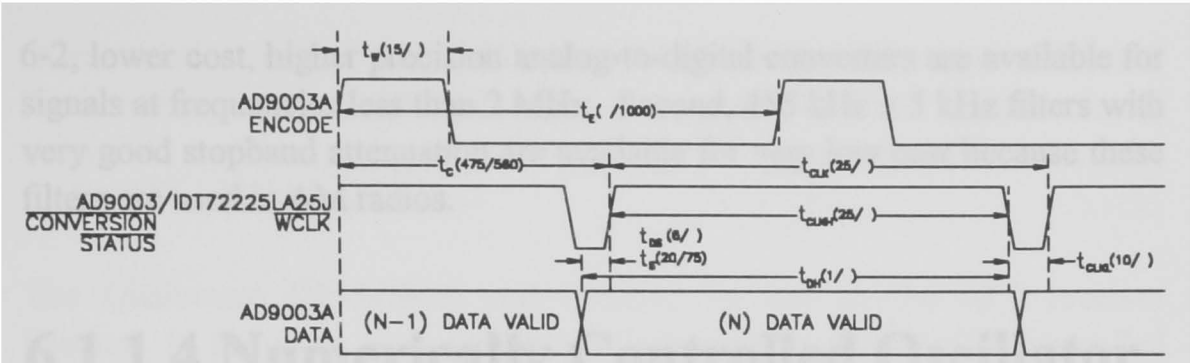


Figure 6-5. AD9003A-IDT72225L25J write timing diagram.

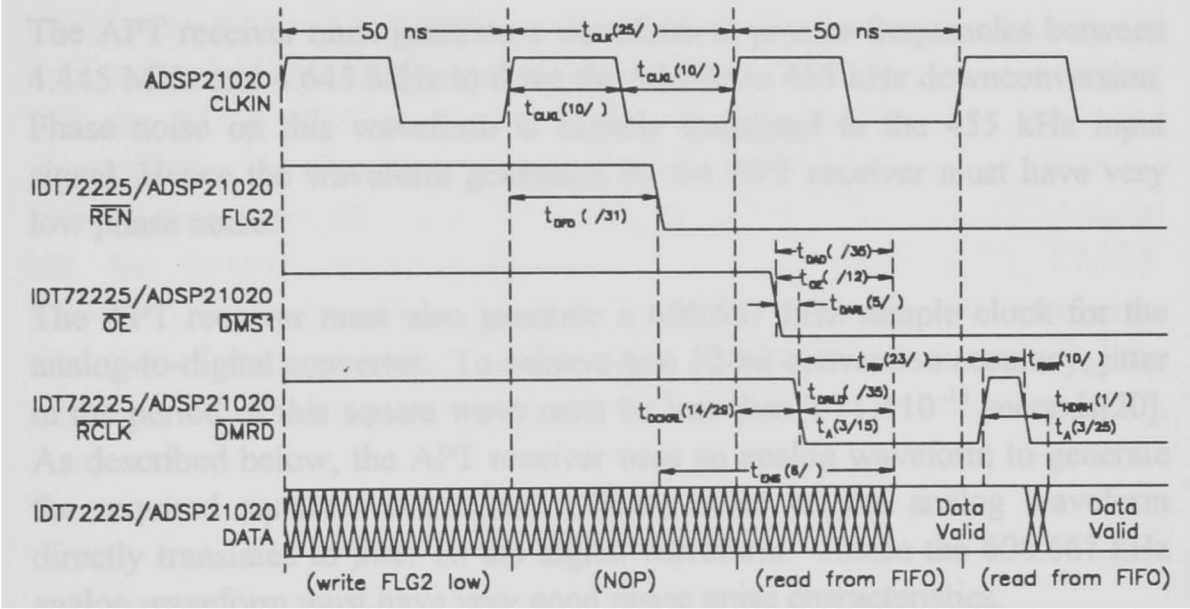


Figure 6-6. ADSP21020-IDT72225L25J read timing diagram.

6.1.1.3.1 Intermediate Frequency Selection

As shown in Figure 5-17, the APT receiver's input frequency is 455 kHz. This input frequency was selected for two reasons. First, as shown in Table

6-2, lower cost, higher precision analog-to-digital converters are available for signals at frequencies less than 2 MHz. Second, 455 kHz ± 5 kHz filters with very good stopband attenuation are available for very low cost because these filters are used in AM radios.

6.1.1.4 Numerically Controlled Oscillator (O1)

The APT receiver must generate a waveform at precise frequencies between 4.445 MHz and 4.645 MHz to drive the 5 MHz to 455 kHz downconversion. Phase noise on this waveform is directly translated to the 455 kHz input signal. Hence the waveform generated by the APT receiver must have very low phase noise.

The APT receiver must also generate a 606.667 kHz sample clock for the analog-to-digital converter. To achieve true 12-bit conversion accuracy, jitter in the period of this square wave must be less than 1.71×10^{-10} seconds[20]. As described below, the APT receiver uses an analog waveform to generate the required square wave signal. Phase noise on the analog waveform directly translates to jitter on the digital waveform. Hence the 606.667 kHz analog waveform must have very good phase noise characteristics.

The APT receiver uses numerically controlled oscillators to generate both of the analog waveforms described above. Numerically controlled oscillators (NCOs) accept a reference frequency as input and use a phase accumulator to compute the amplitude of a desired output signal at every clock cycle. This amplitude information may then be converted to an analog waveform using a digital-to-analog converter and an analog lowpass filter[27]. The nature of numerically controlled oscillators provides output signal phase noise

[27] The analog signal must be low pass filtered to ensure the Nyquist criteria is satisfied.

characteristics which are better than that of the reference oscillator. Furthermore, using a 20 MHz reference oscillator, a 32-bit numerically controlled oscillator can generate waveforms at frequencies up to 8 MHz with 0.00463 Hz resolution.

The Qualcomm Q2334I20N was selected for use on the APT receiver because it provides two completely independent numerically controlled oscillator channels in a single package. A Vectron CO238T oscillator is used as the Q2334I20N's reference oscillator. The CO238T has very good phase noise characteristics. The ADSP21020-Q2334I20N interface is unidirectional. The Q2334I20N must be initialized with mode and frequency information but does not provide any status information. As shown in Figure 6-7, the ADSP21020 is unable to meet the Q2334I20N data hold time requirement. To alleviate this problem, a 74LS573 latch was added to the circuit. Notice, the latch is clocked using the ADSP21020 write ($\overline{WR}$) signal but the latch's output enable is gated with ADSP21020 $\overline{DMS2}$. $\overline{DMS0}$ - $\overline{DMS4}$ are select lines. The state of these lines in preserved longer than that of data lines at the end of each ADSP21020 clock cycle thereby allowing the latch outputs to satisfy the Q2334I20N hold time requirements.

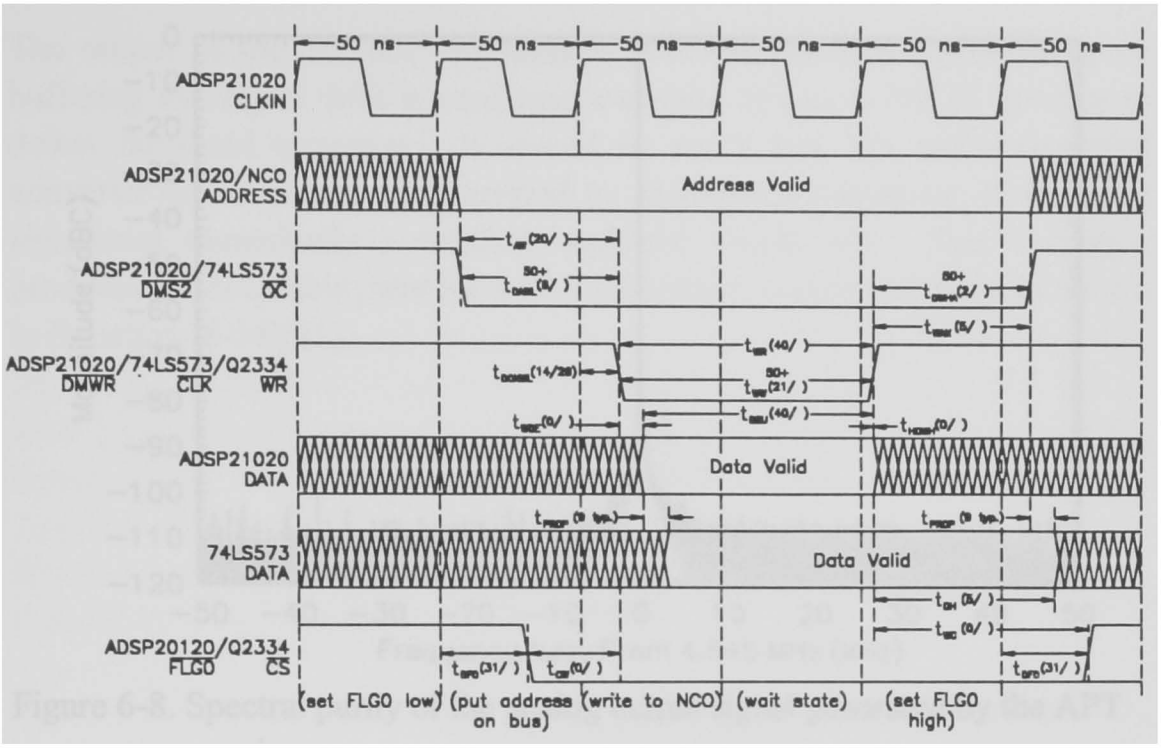


Figure 6-7. ADSP21020-Q2334I20N write timing diagram.

The digital-to-analog converter circuits were constructed according to Qualcomm's specifications using TRW TDC1012N7C1 digital-to-analog converters. As shown in Figures 6-8 and 6-9, the spectral purity and phase noise characteristics of the mix output waveform are exceptional.

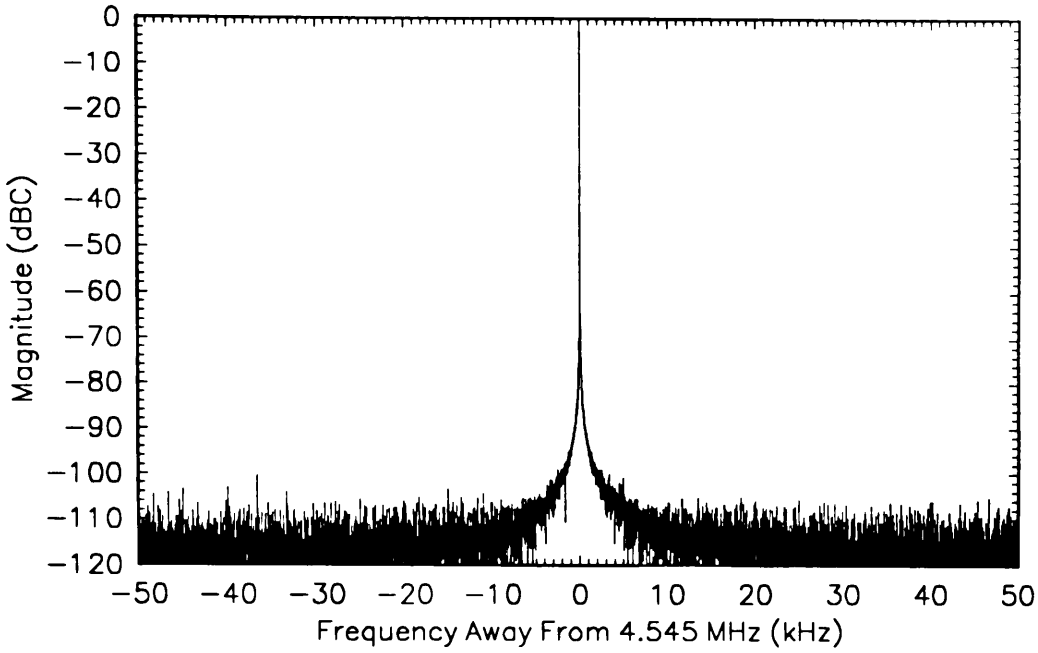


Figure 6-8. Spectral purity of the analog output signal generated by the APT receiver.

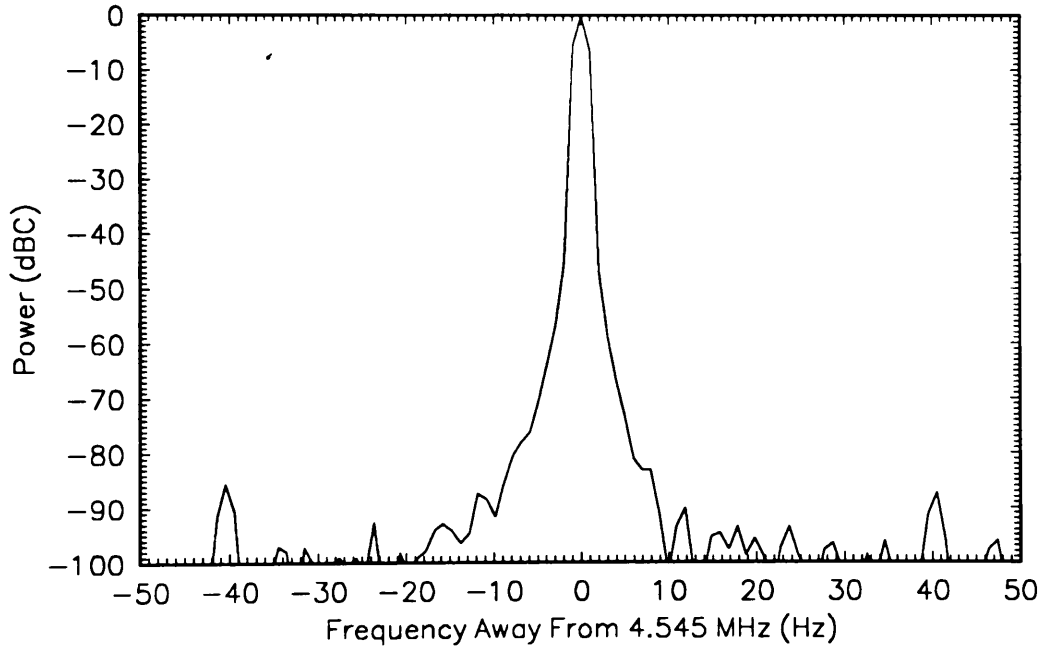


Figure 6-9. Phase noise characteristics of the analog output signal generated by the APT receiver.

The output of the 606.667 Hz channel is converted to a square wave by buffering the signal with a transistor and then driving a 74F04 inverter to detect threshold crossings. It should be noted that the analog-to-digital converter clock cannot be generated by identifying transitions of the most significant numerically controlled oscillator output bit. This technique produces unacceptable jitter for a 12-bit digital-to-analog converter as shown in Equation (6-14)[21].

$$Jitter_{MSB} = \frac{1}{2 \cdot reference\,frequency} \tag{6-14}$$

$$= \frac{1}{2 \cdot 15\,MHz}$$

$$= 33.33 \times 10^{-10}\,seconds$$

6.1.1.5 Serial Port (S1)

Communication with the APT receiver board is accomplished via a serial link which operates at 2400, 4800, 9600 or 19200 baud. Although parallel links typically have higher bandwidths, custom parallel links are less versatile than serial links. For example, the APT receiver board may be connected to any host with a standard DCE RS-232 serial port. Conversely, if communication were via a custom parallel link, the receiver would only be operable with the ACTS data acquisition and control system.

A standard DCE RS-232 port is implemented on the APT receiver board using Maxum's MAX232 RS232-to-TLL level converter, Intel's 8251A USART[28] and a 74LS193 4-bit counter. The MAX232 level converter chip

[28] USART stands for Universal Serial Asynchronous Receiver Transmitter

was selected because it only requires a single 5 volt supply and provides level conversions in both directions. Level converter chips such as the 1488 and 1489 only provide conversion in one direction and require a 10 volt supply which is not available on the APT receiver board. The Intel 8251A was selected because it is the industry standard USART and the designer has considerable experience with the device. Furthermore, the 8251A will detect break signals[29] even in an uninitialized state[23]. The APT receiver takes advantage of this fact by using the 8251A *SYNDET/BD* signal in the system reset logic. Hence, the user may reset the entire system by sending a break signal to the serial port. As shown on the APT receiver schematic in Appendix D, the 74LS193 counter provides jumper selectable baud rates by dividing the output of a 2.4576 MHz TTL oscillator to generate the 8251A transmit/receive clocks.

The ADSP21020-8251A interface was difficult to design because the ADSP21020 operates at a much higher clock speed than the 8251A. As shown in Figure 6-10, the ADSP21020 cannot meet the data hold time requirements of the 8251A.

To extend the ADSP21020 data hold time, a 74LS573 latch was added to the circuit. Notice, the latch is clocked using the ADSP21020 write ($\overline{WR}$) signal but the latch's output enable is gated with ADSP21020 $\overline{DMS2}$. $\overline{DMS0}$ - $\overline{DMS4}$ are select lines. The state of these lines in preserved longer than that of data lines at the end of each ADSP21020 clock cycle thereby allowing the latch outputs to satisfy the 8251A hold time requirements. A 74LS245 was then used to direct the 8251A output to the ADSP21020 data bus during ADSP21020 data memory bank 3 ($\overline{DMS3}$) read operations. The timing diagram for this interface is shown in Figure 6-11.

---

[29] Break signals are issued by pulling the receive (RxD) line low through two consecutive stop bit sequences including start bits, data bits and parity bits.

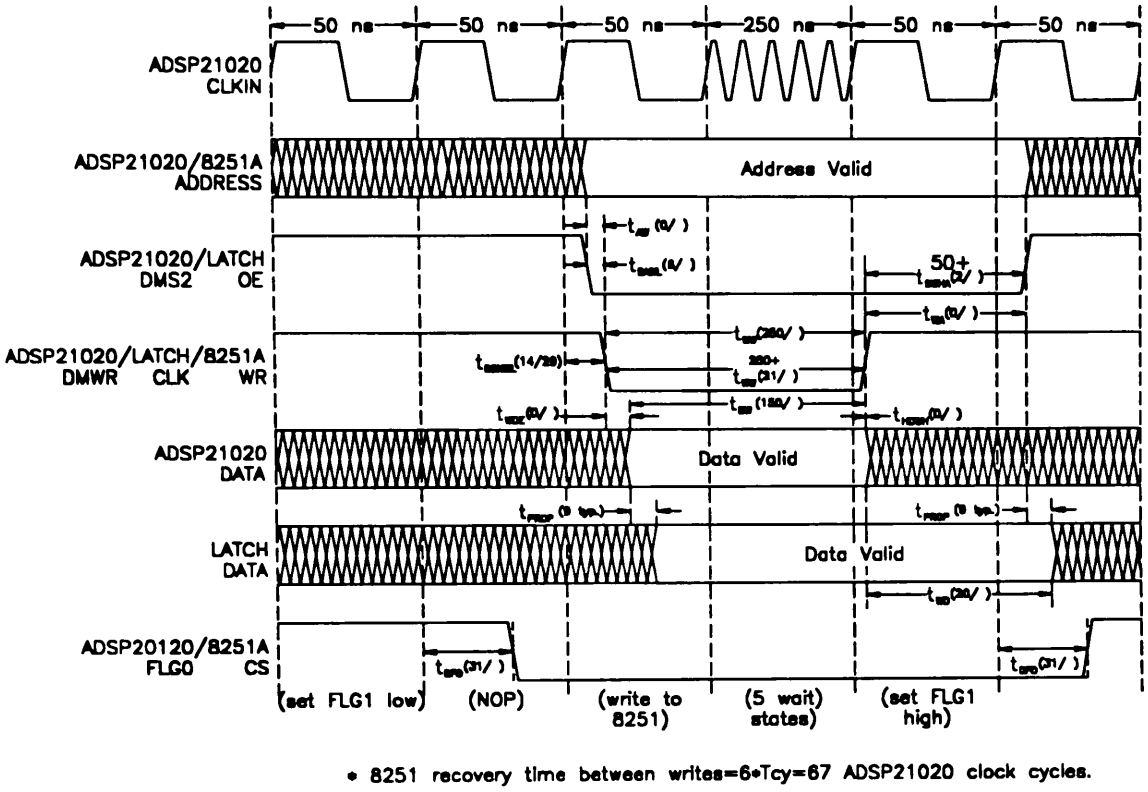


Figure 6-10. ADSP21020-8251A write timing diagram.

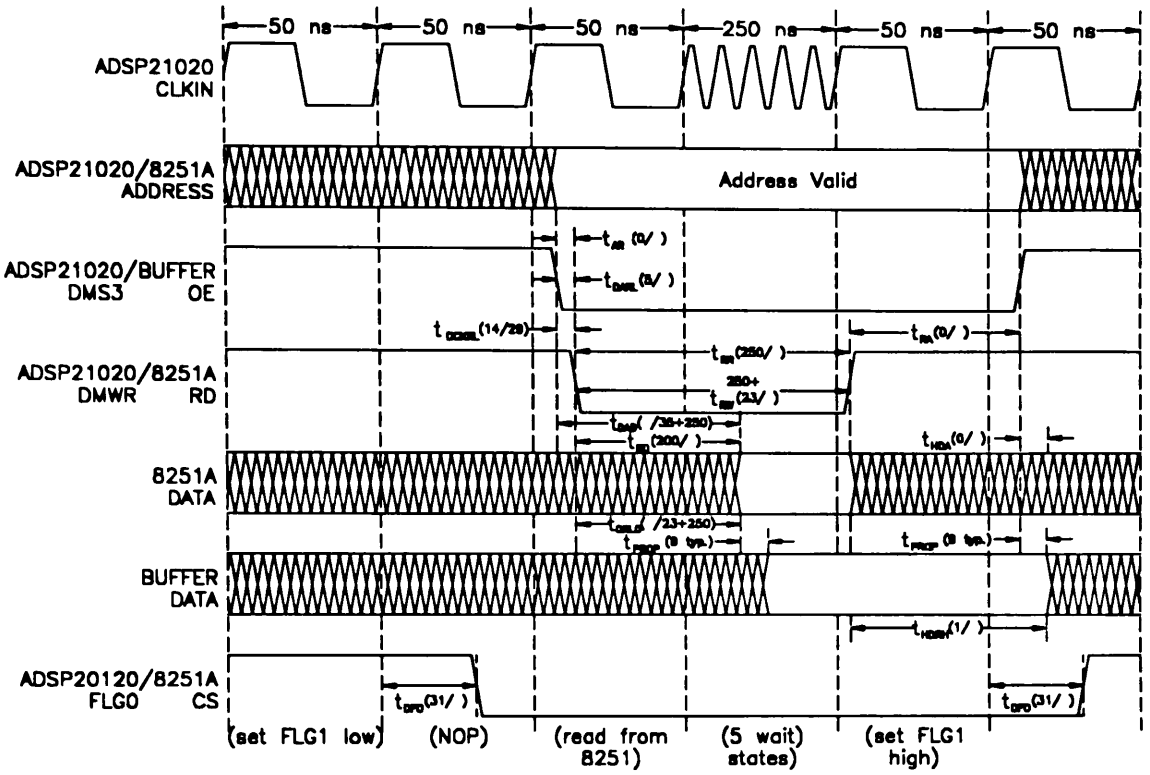


Figure 6-11. ADSP21020-8251A read timing diagram.

6.1.1.6 Bootstrap Loader (B1)

The APT receiver board is equipped with a bootstrap loader which downloads application-specific executable code to system RAM. This feature makes the APT receiver board very versatile. Different applications can be run on the same board simply by downloading a different program. In a prototype environment, this feature makes software development much easier by eliminating the time and cost of burning PROMS. In a production environment, APT operators may install updated versions of the APT receiver software without opening the receiver enclosure or making any hardware modifications.

This bootstrap loader circuit utilizes the JTAG[30] port on the ADSP21020 to force the ADSP21020 to execute a small kernel of instructions. These instructions initialize the serial port on the APT receiver board and allow the user to download executable code through the port very quickly. The hardware design and software required to build and use the loader was obtained from an Analog Devices application note. A complete description of the circuit is available in *A JTAG Boot Downloader For The ADSP21020* by Jim Donahue, Analog Devices, One Technology Way, P.O. Box 9106, Norwood, MA, 02062-9106, (617) 329-4700.

6.1.2 Printed Circuit Board Construction

All of the subsystems described above were integrated as shown on the schematic in Appendix D. Prototype and production versions of the APT receiver hardware were implemented on printed circuit boards. The prototype printed circuit board had two layers. Unfortunately the lack of substantial power and ground planes prohibited the prototype unit from running at clock speeds above 12 MHz. The noise problems encountered on the prototype unit were eliminated on the production receivers by adding a +5 volt power plane as well as separate analog and digital ground planes to the printed circuit board. As a result, the production version of APT receiver printed circuit board has 4 layers. Although the production APT receiver boards operate at 16 MHz, tests indicate they will operate reliably at clock speeds up to 20 MHz. A photograph of a production APT receiver and the production printed circuit board artwork are shown in Figures 6-12 through 6-17.

---

[30] The JTAG port is a boundary scan interface. See IEEE specification 1149.1.

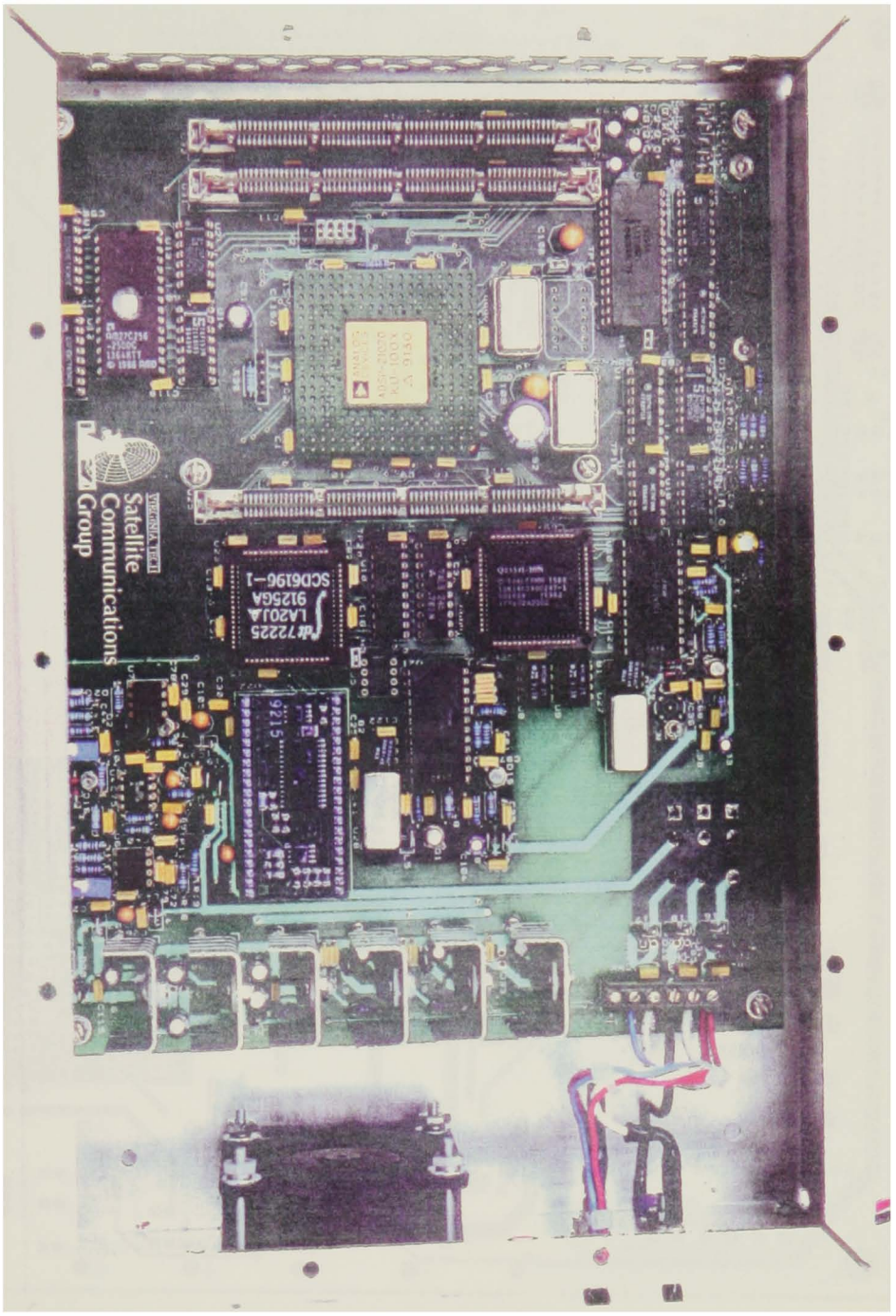


Figure 6-12. Photograph of a production APT receiver system.

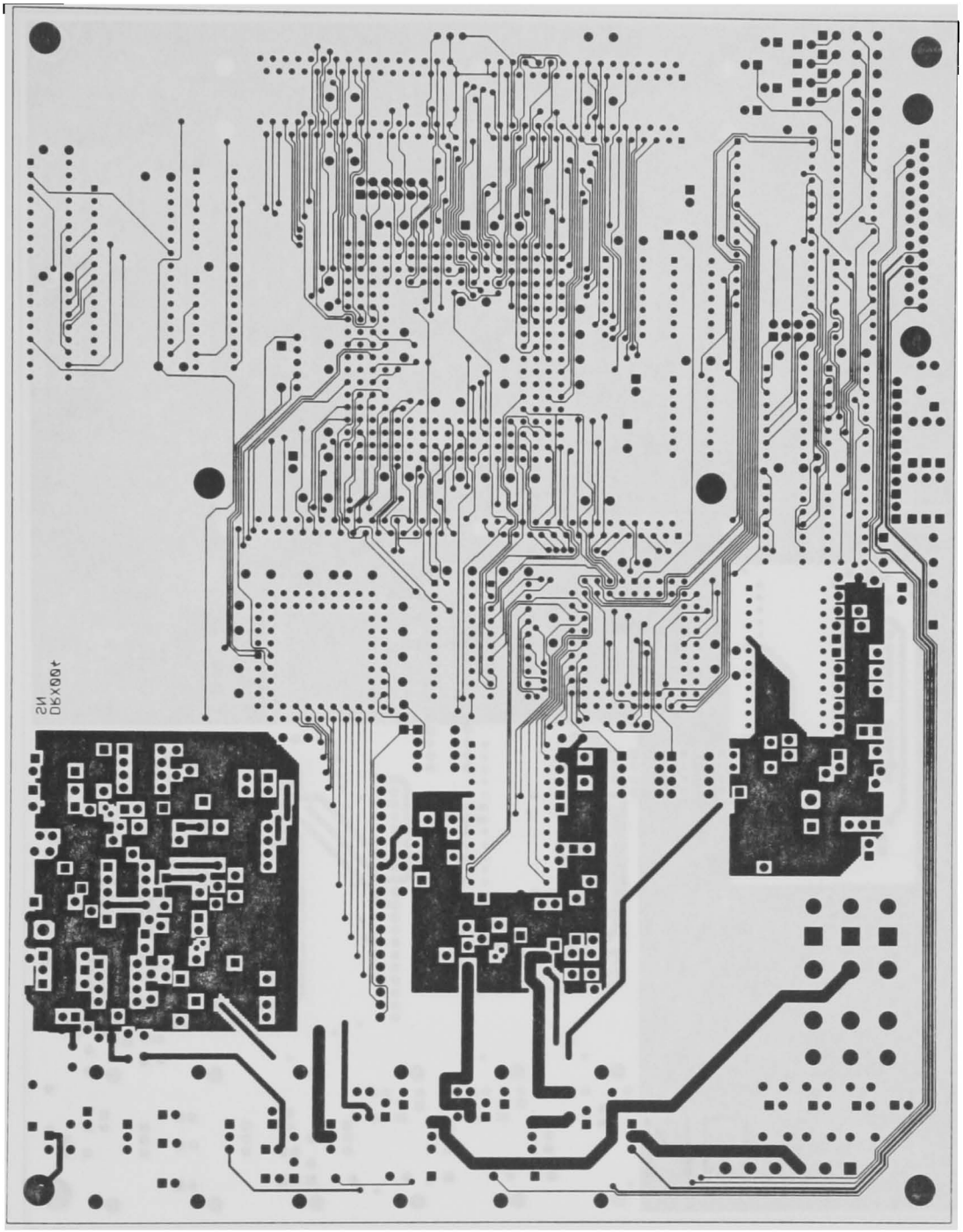


Figure 6-13. APT production receiver printed circuit board artwork (layer 1 of 4, solder side), scale 0.75:1.0.

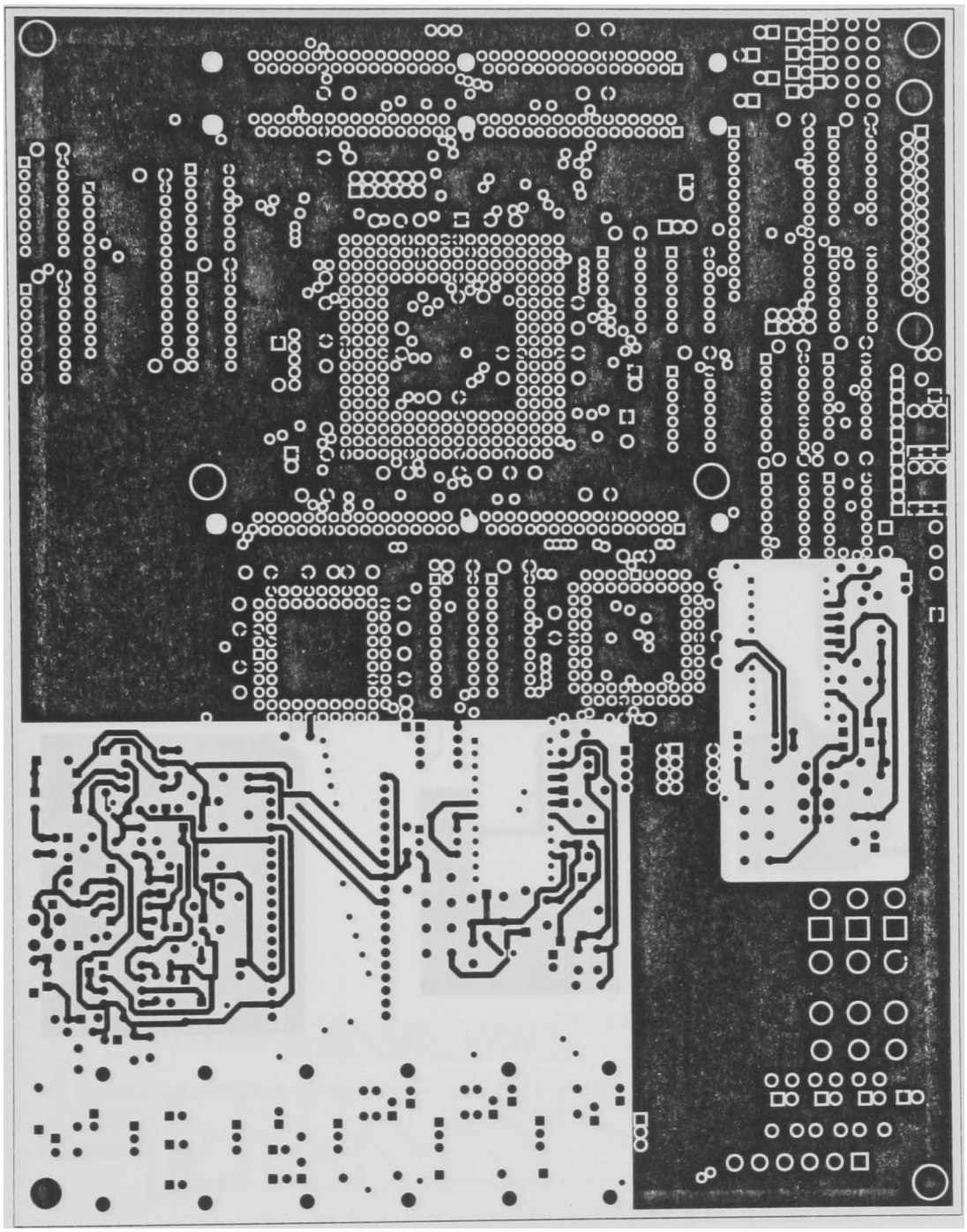


Figure 6-14. APT production receiver printed circuit board artwork (layer 2 of 4), scale 0.75:1.0.

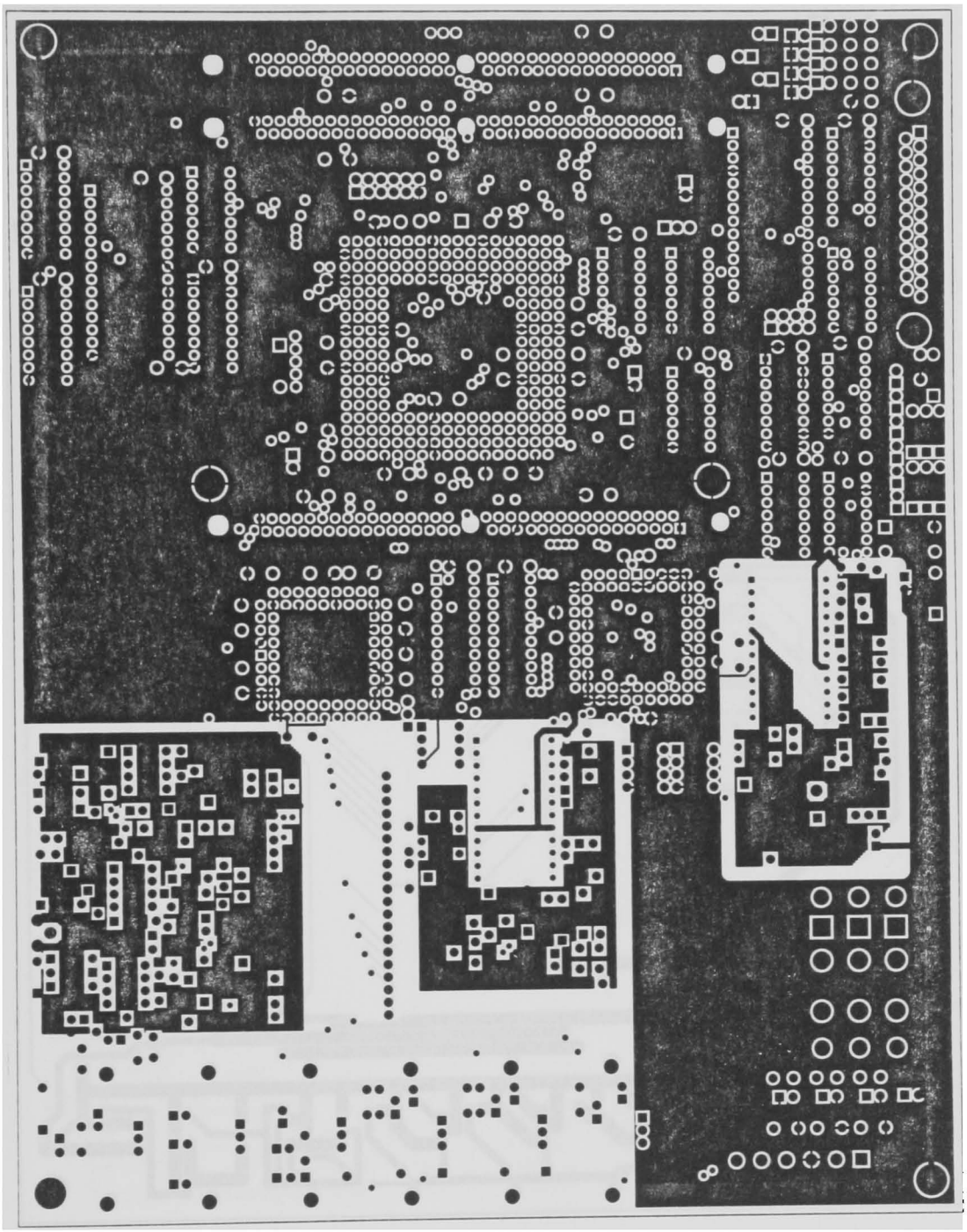


Figure 6-15. APT production receiver printed circuit board artwork (layer 3 of 4), scale 0.75:1.0.

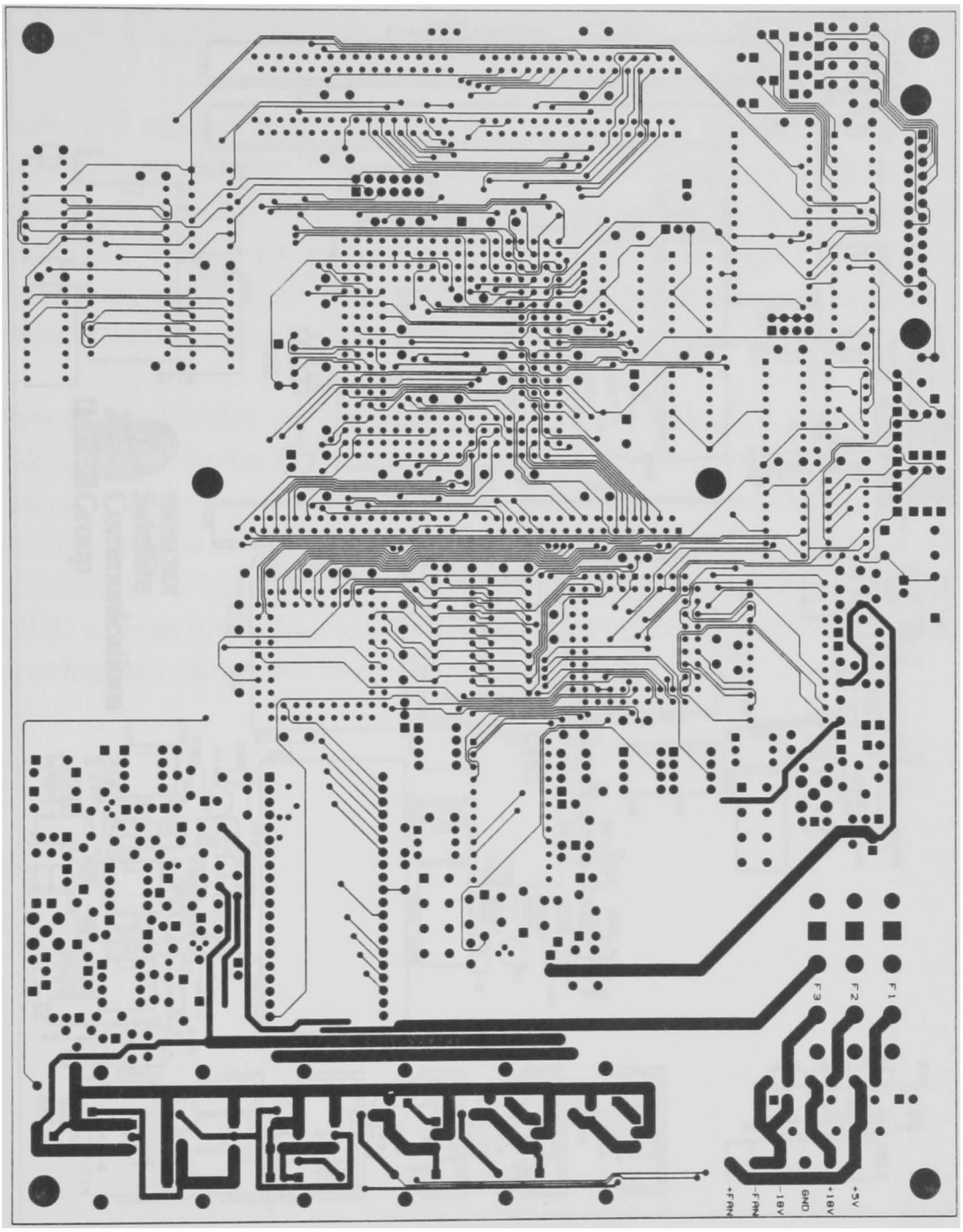


Figure 6-16. APT production receiver printed circuit board artwork (layer 4 of 4, component side), scale 0.75:1.0.

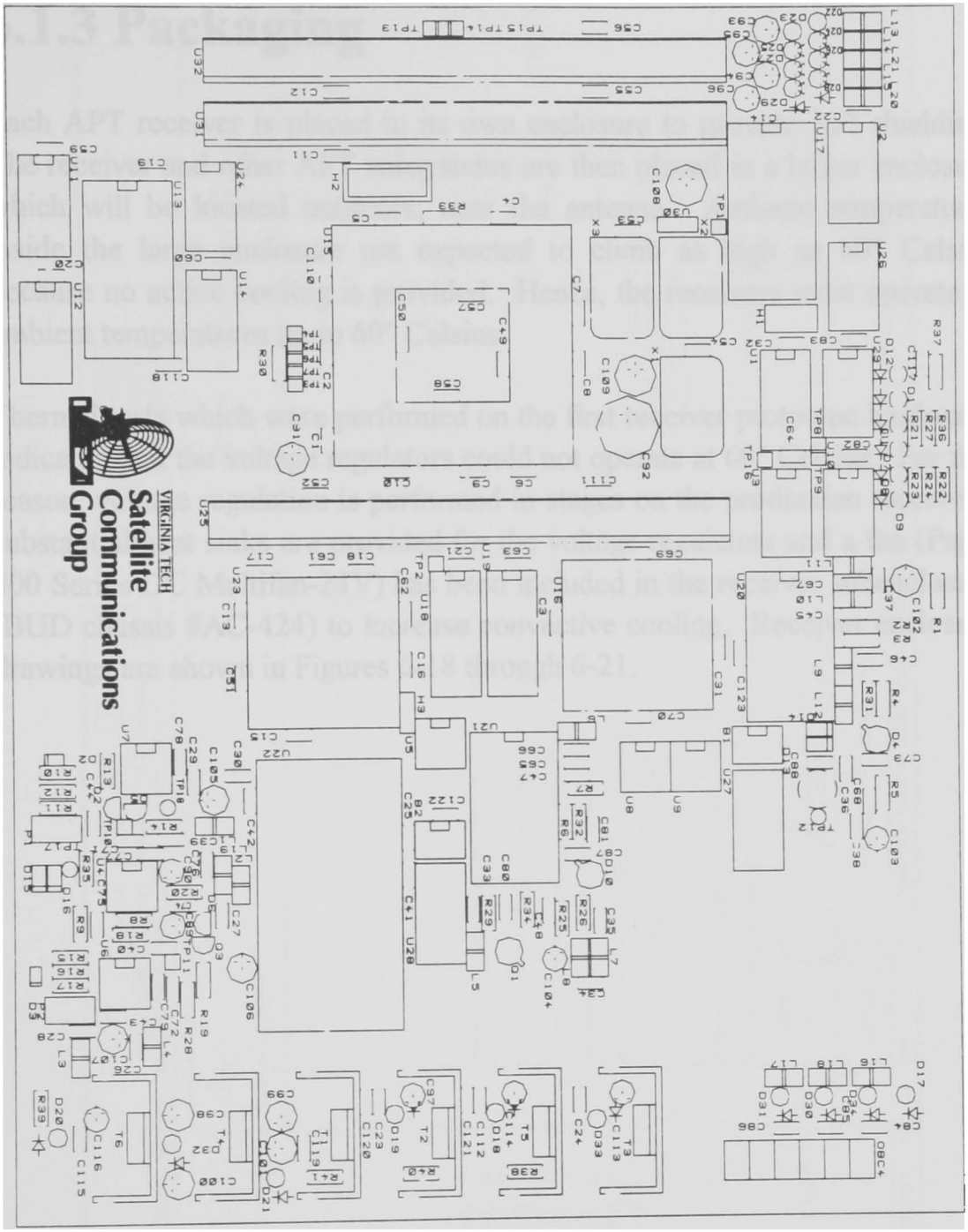


Figure 6-17. APT production receiver printed circuit board artwork (silk screen), scale 0.75:1.0.

6.1.3 Packaging

Each APT receiver is placed in its own enclosure to provide RFI shielding. The receiver and other APT subsystems are then placed in a larger enclosure which will be located outdoors, near the antenna. Ambient temperatures inside the large enclosure are expected to climb as high as 60° Celsius because no active cooling is provided. Hence, the receivers must operate at ambient temperatures up to 60° Celsius.

Thermal tests which were performed on the first receiver prototype hardware, indicated that the voltage regulators could not operate at 60° Celsius. For this reason, voltage regulation is performed in stages on the production receivers, substantial heat sinks are provided for the voltage regulators and a fan (Papst 800 Series DC Multifan-24V) has been included in the receiver subenclosure (BUD chassis #AC-424) to increase convective cooling. Receiver enclosure drawings are shown in Figures 6-18 through 6-21.

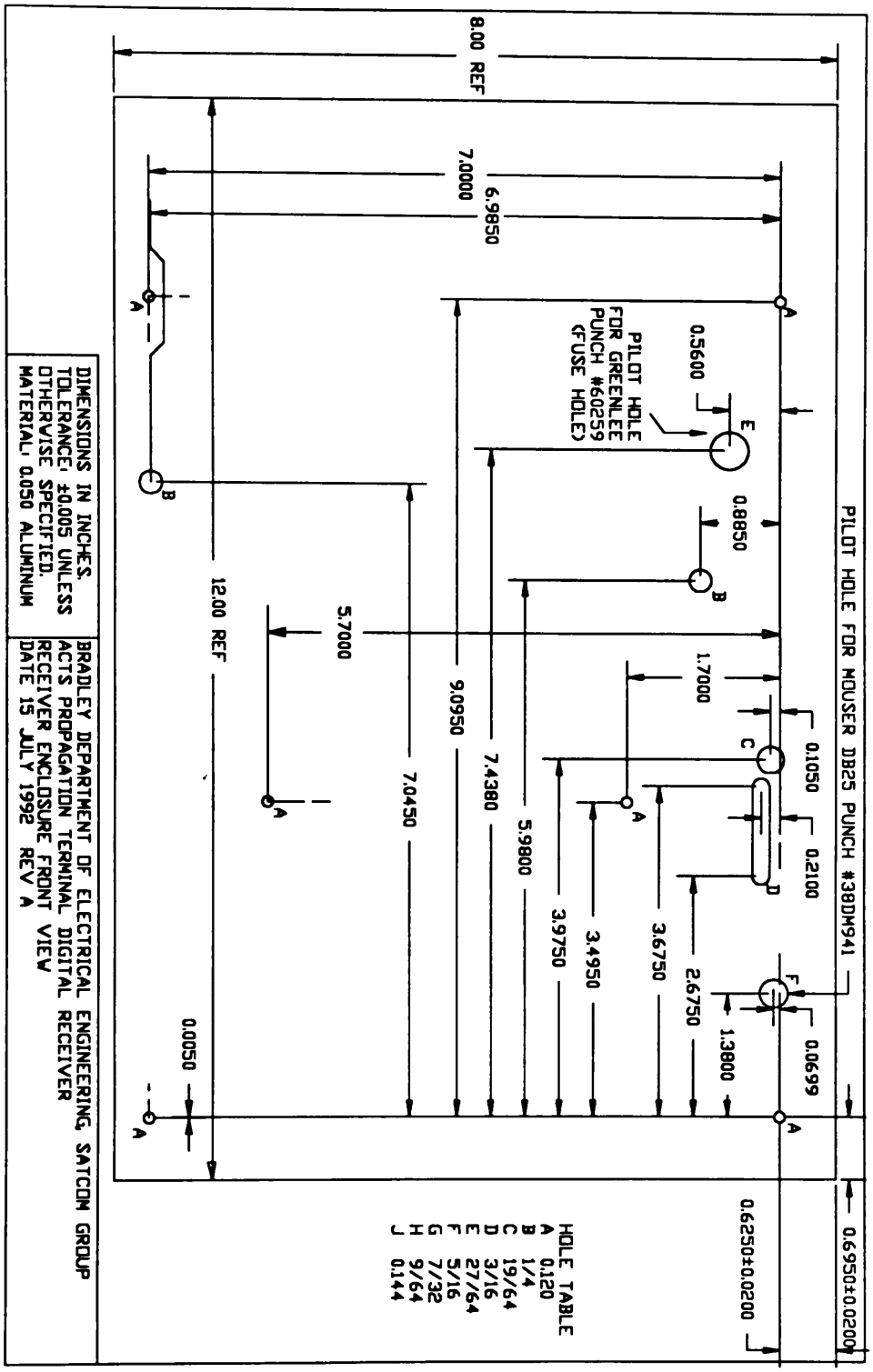


Figure 6-18. APT receiver enclosure, front view.

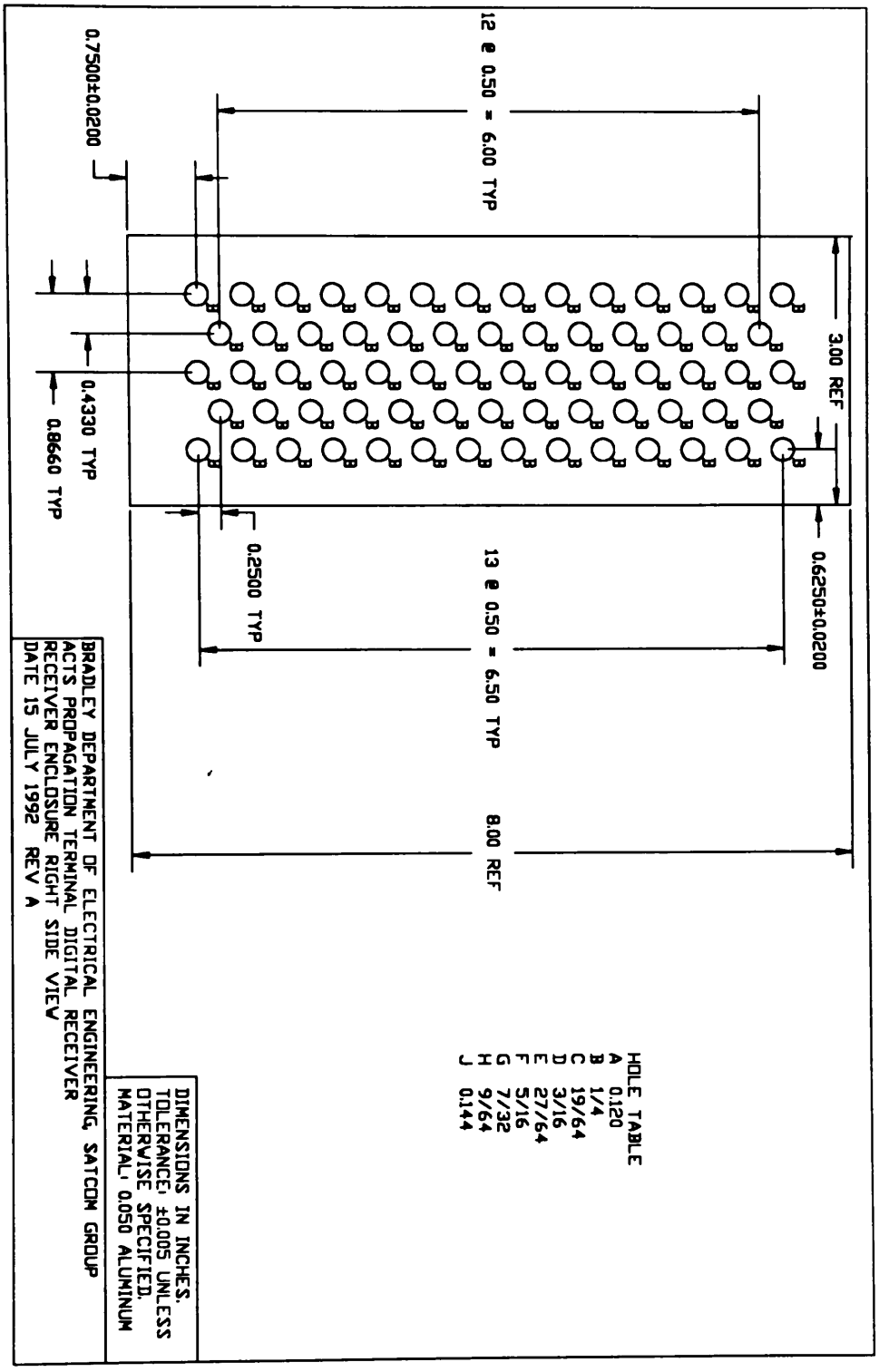


Figure 6-19. APT receiver enclosure, right side view.

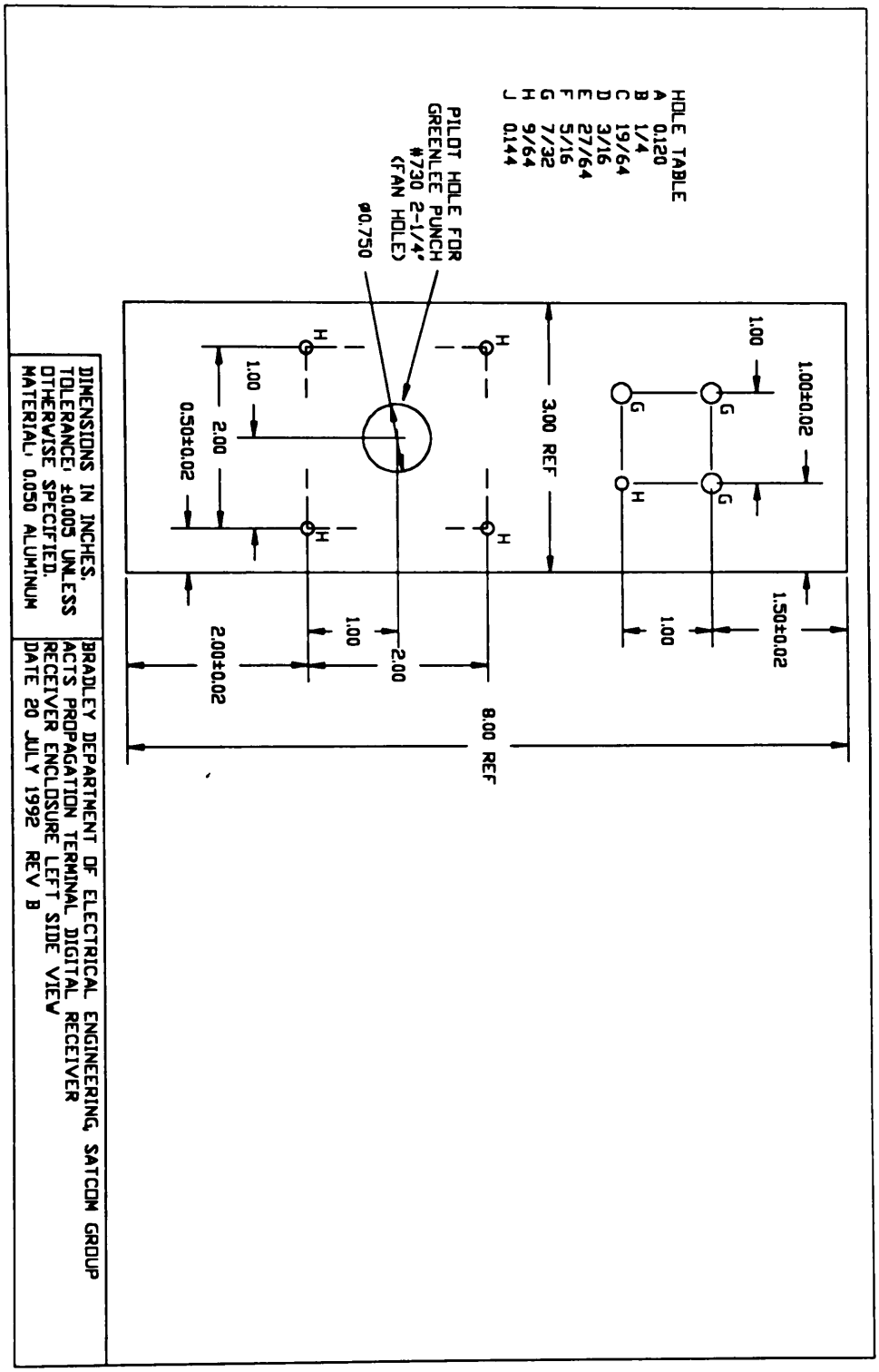


Figure 6-20. APT receiver enclosure, left side view.

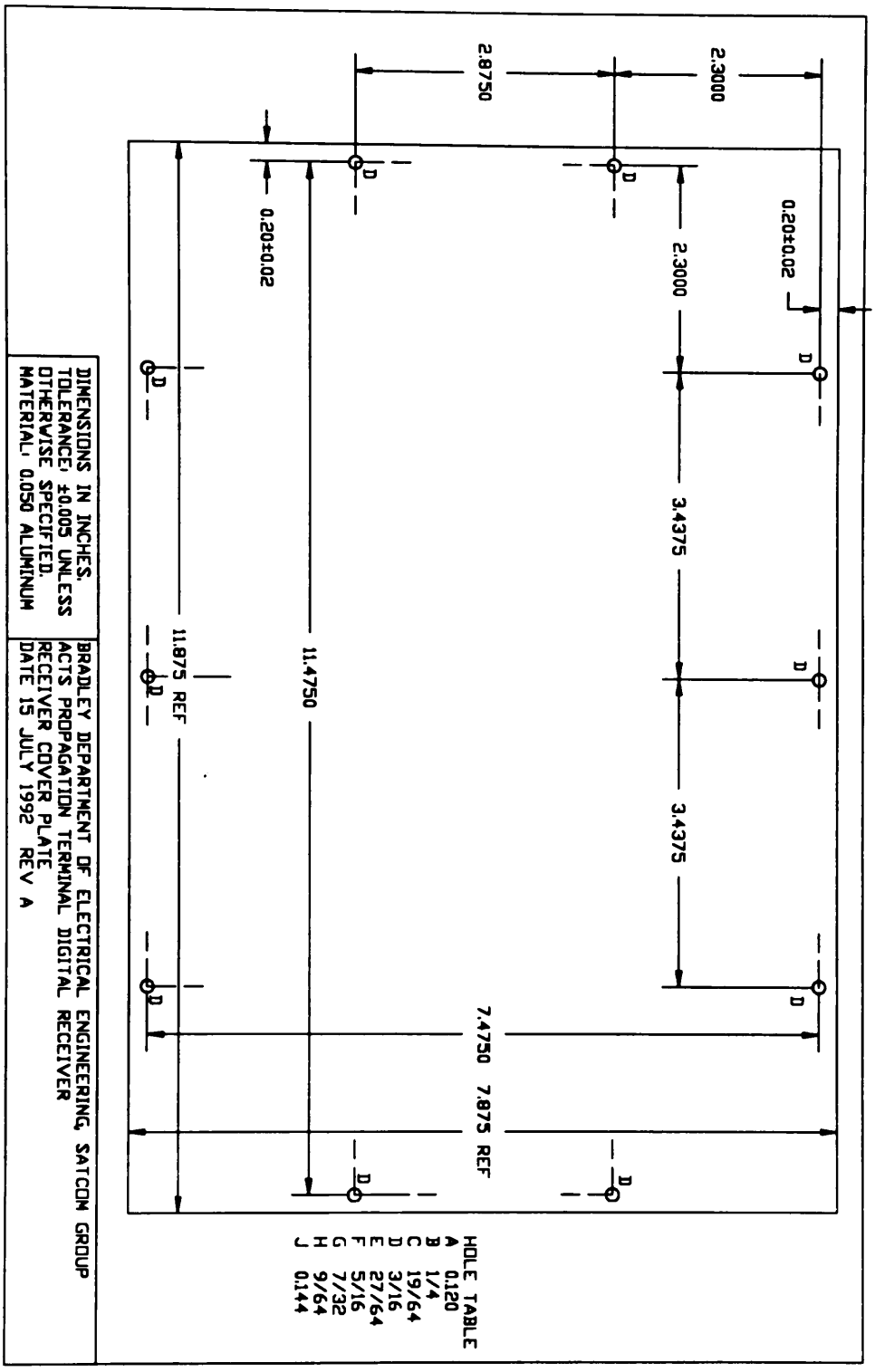


Figure 6-21. APT receiver enclosure, cover.

6.2 Software

The APT receiver algorithm has been implemented in ADSP21020 symbolic assembly language. Although, Analog Devices markets a C compiler for the ADSP21020, this software was not available during the receiver's development stage. The receiver software does utilize several algebraic and transcendental function subroutines provided by Analog Devices. A listing of the APT receiver source code is provided in Appendix C.

Chapter 7

System Tests

A battery of tests were performed on the APT receiver at ambient temperatures up to 60° Celsius. Unfortunately, these tests could not utilize the actual ACTS beacons because ACTS will not be launched until July, 1993. Therefore, extensive bench tests as well as tests using the OLYMPUS 20 GHz beacon were performed. Accuracy and linearity tests were performed using the output of a Hewlett Packard HP3330B signal generator while carrier acquisition and dynamic range tests were carried out utilizing the OLYMPUS 20 GHz beacon. Although the OLYMPUS 20 GHz beacon is not modulated with telemetry information or ranging tones, spectral peaks representing the odd harmonics associated with square wave modulation appear in the OLYMPUS 20 GHz beacon spectrum. The polarization of the OLYMPUS 20 GHz beacon is switched at 933 Hz. The switching polarization appears as square wave modulation at the APT receiver input because the hardware which performs the 20 GHz to 455 kHz downconversion only accepts the co-polar signal. The resulting tones are much closer to the OLYMPUS 20 GHz beacon than corresponding tones will be to the ACTS 20 GHz beacon. Hence the side tones on the OLYMPUS

20 GHz beacon rigorously test the APT receiver carrier signal acquisition algorithm. The data presented in the following sections demonstrate that the APT receiver will reliably acquire the ACTS beacon signal and will provide exceptionally accurate linear signal power measurements. Furthermore, all of the supplemental functions described in Chapter 6 operate properly to produce the expected results.

7.1 Carrier Signal Acquisition

The acquisition characteristic of the receiver was determined by repeatedly attempting to acquire the OLYMPUS 20 GHz beacon at different signal-to-noise ratios. The test system used is shown in Figure 7-1.

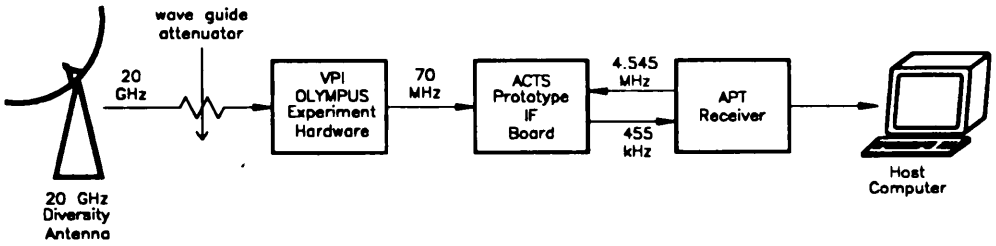


Figure 7-1. System used to test the APT receiver with the OLYMPUS 20 GHz beacon

As shown, the OLYMPUS 20 GHz beacon signal is available at 70 MHz at the output of the Virginia Tech's OLYMPUS Experiment hardware. The 70 MHz signal was downconverted to 455 kHz by the prototype ACTS IF chain and input to the receiver. Different signal-to-noise ratios were generated by adjusting a wave guide attenuator in the front end of the OLYMPUS Experiment hardware. The primary component in the system noise level is thermal noise from the electronic components used to amplify, bandlimit and translate the 20 GHz signal down to 455 kHz. Sky noise contributions to the system noise level are very small. Therefore, adjusting

the front end attenuator alters the signal level but does not significantly affect the system noise level.[31]

The APT receiver was able to reliably identify the carrier signal at signal-to-noise ratios down to 10 dB in a 20 Hz bandwidth (23 dBHz in a 1 Hz bandwidth, 26 dB fade conditions APT system) in 2 seconds.

7.2 Signal Power Measurement

Accuracy and linearity tests were performed on the APT receiver by injecting a synthesized 455 kHz signal into the receiver's input using a Hewlett Packard HP3330B signal generator. As shown in the following sections, the receiver produces exceptionally accurate, linear signal power measurements over the +8 dBm to -35 dBm operating region. These results exceed the goals established by Virginia Tech and those defined by the ACTS Users' Community.

7.2.1 Accuracy

To determine the accuracy of the APT receiver signal power measurements, 100 measurements were made at signal levels from +10 dBm to -35 dBm (1 dBm increments). The measurements taken at several signal levels are plotted versus their expected values on the following pages. Additionally, the standard deviation of the signal power measurements made at each signal level are presented in Table 7-1. Recall, the resolution of all APT receiver signal power measurements is 0.01 dB. This quantization is reflected in the data presented below The data presented below indicates that the APT

[31] The small amount of noise generated by the wave guide attenuator was neglected.

receiver's measurement accuracy is ±0.05 dB. This accuracy exceeds the ACTS Users' Community goal of 0.5 dB specified in Chapter 2.

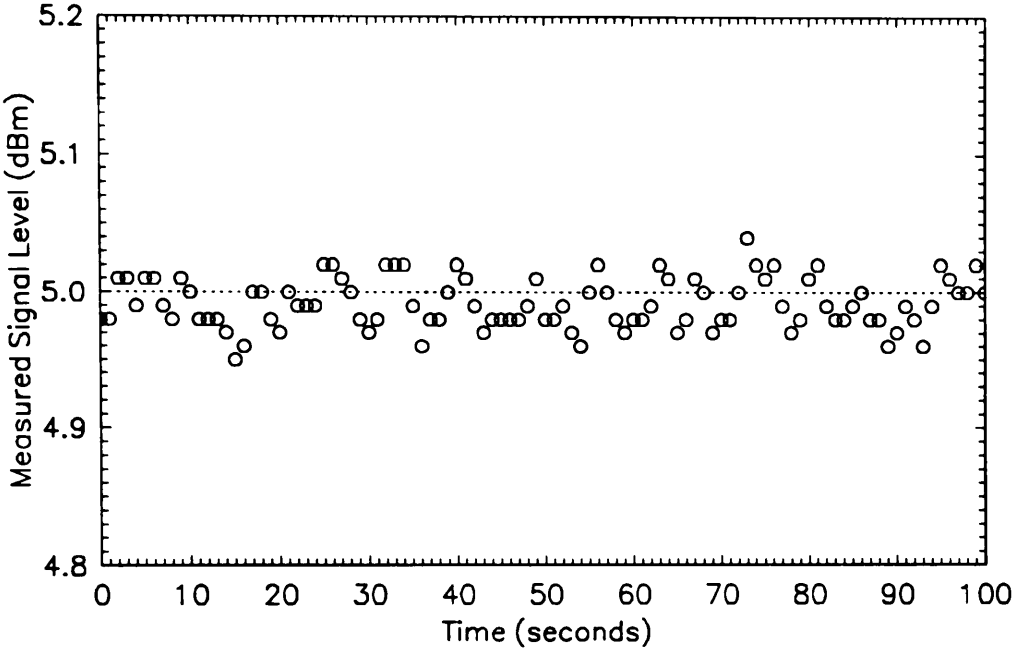


Figure 7-2. APT receiver signal power measurements of a +5 dBm signal.

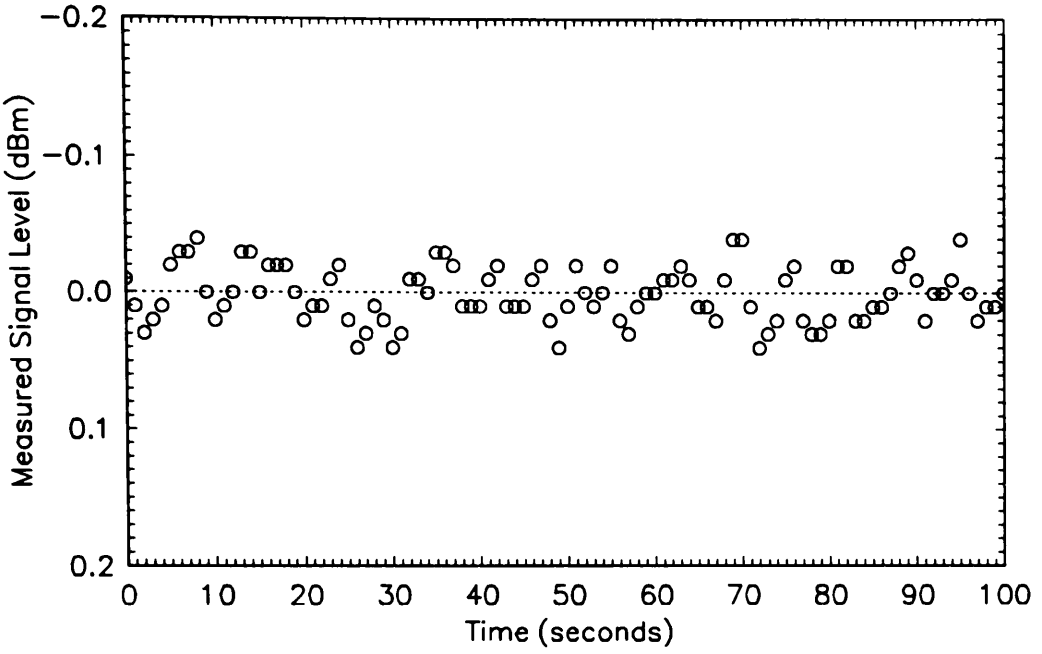


Figure 7-3. APT receiver signal power measurements of a 0 dBm signal.

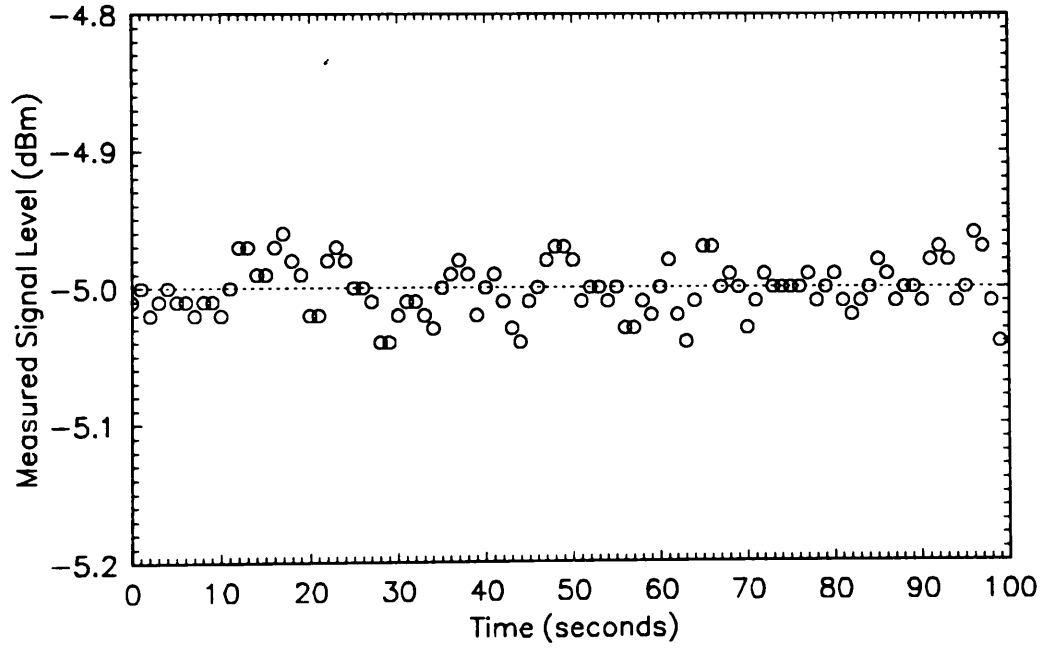


Figure 7-4. APT receiver signal power measurements of a -5 dBm signal.

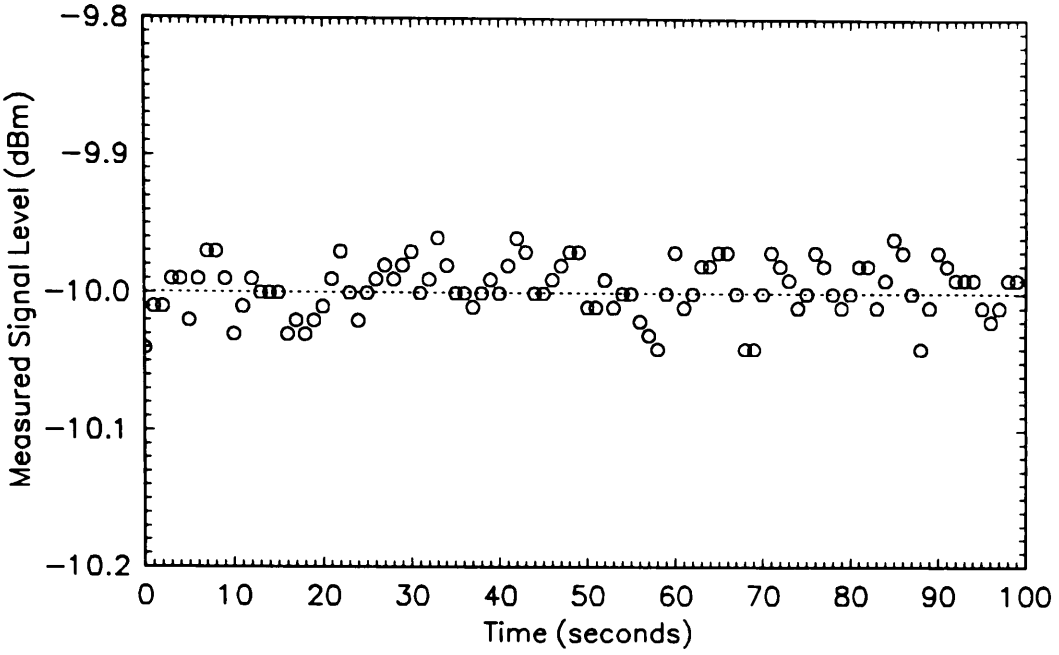


Figure 7-5. APT receiver signal power measurements of a -10 dBm signal.

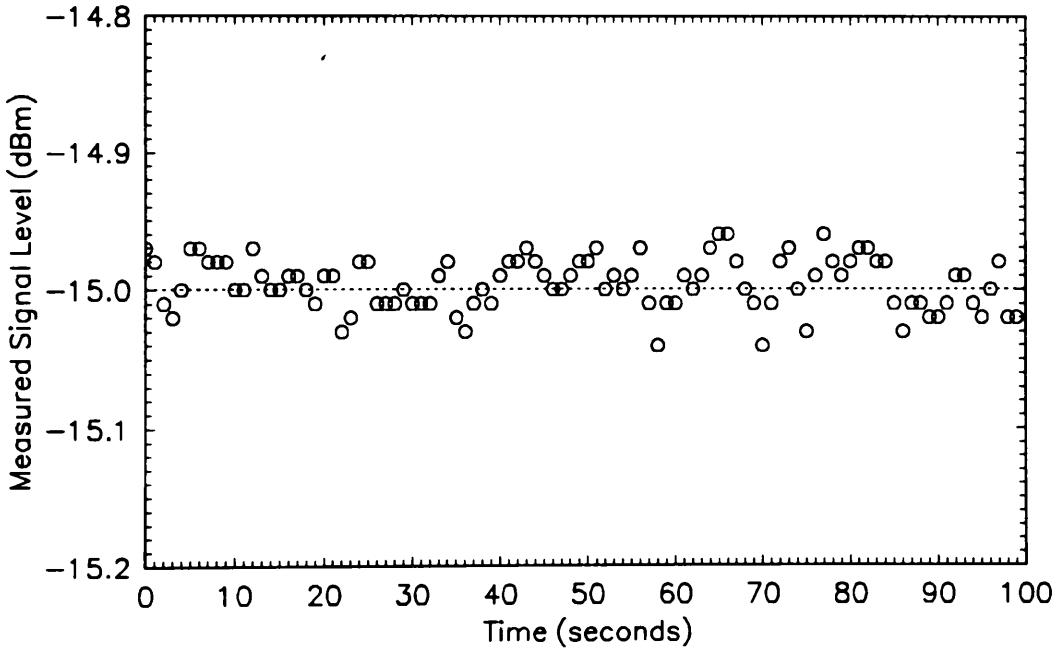


Figure 7-6. APT receiver signal power measurements of a -15 dBm signal.

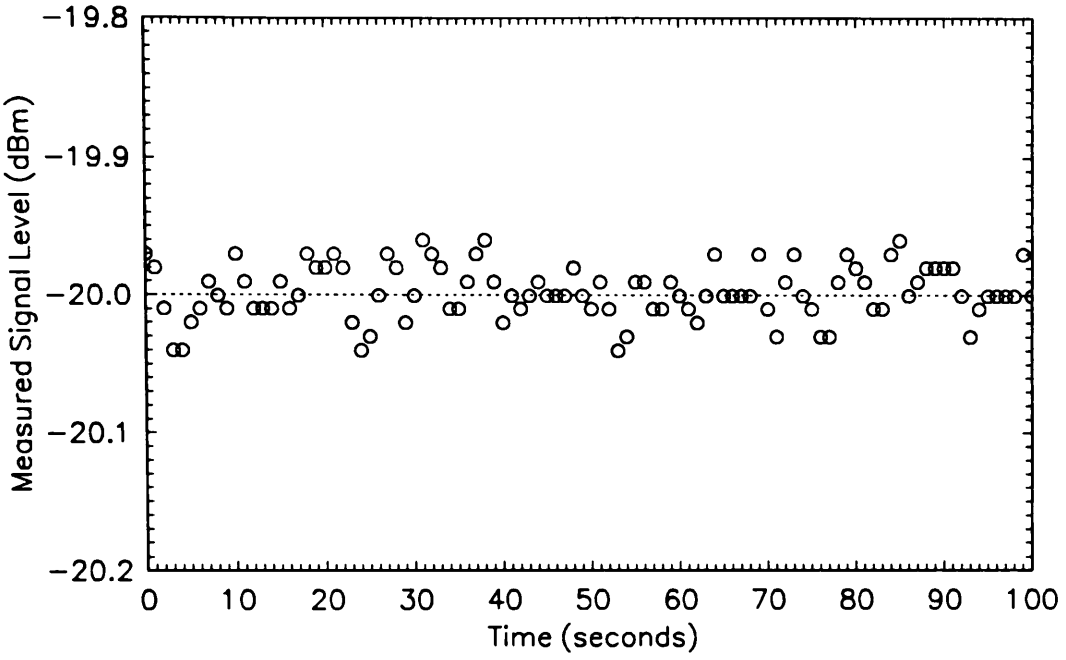


Figure 7-7. APT receiver signal power measurements of a -20 dBm signal.

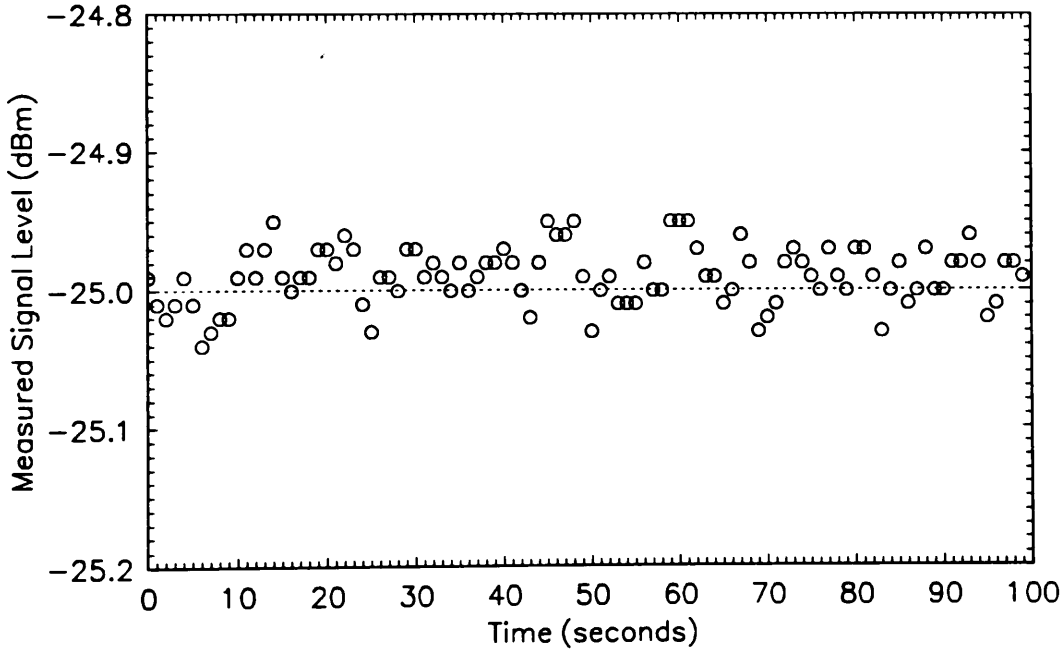


Figure 7-8. APT receiver signal power measurements of a -25 dBm signal.

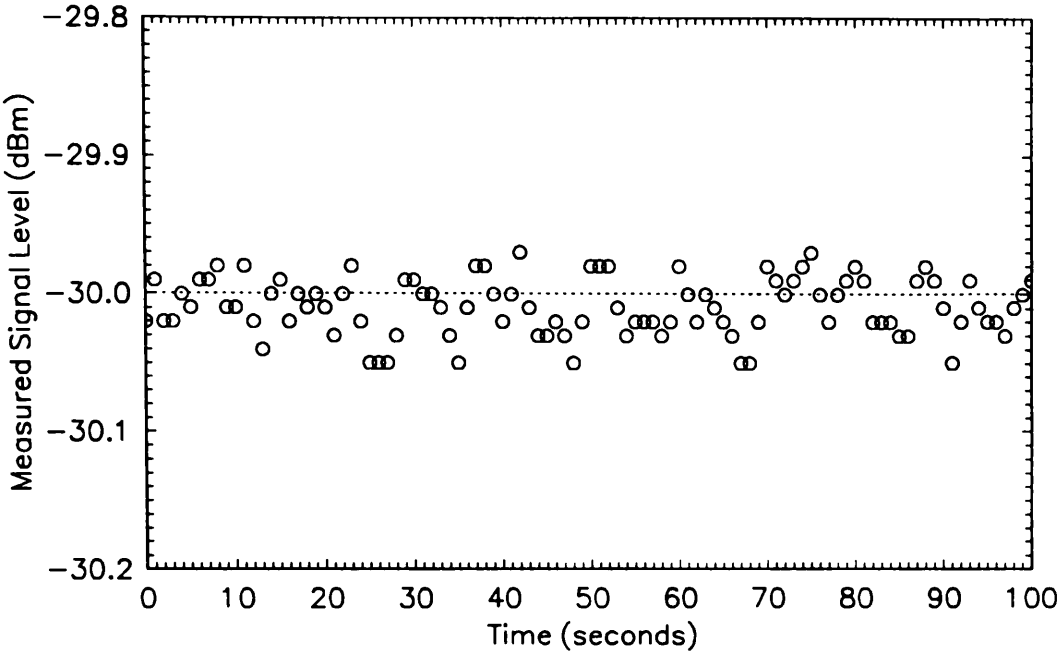


Figure 7-9. APT receiver signal power measurements of a -30 dBm signal.

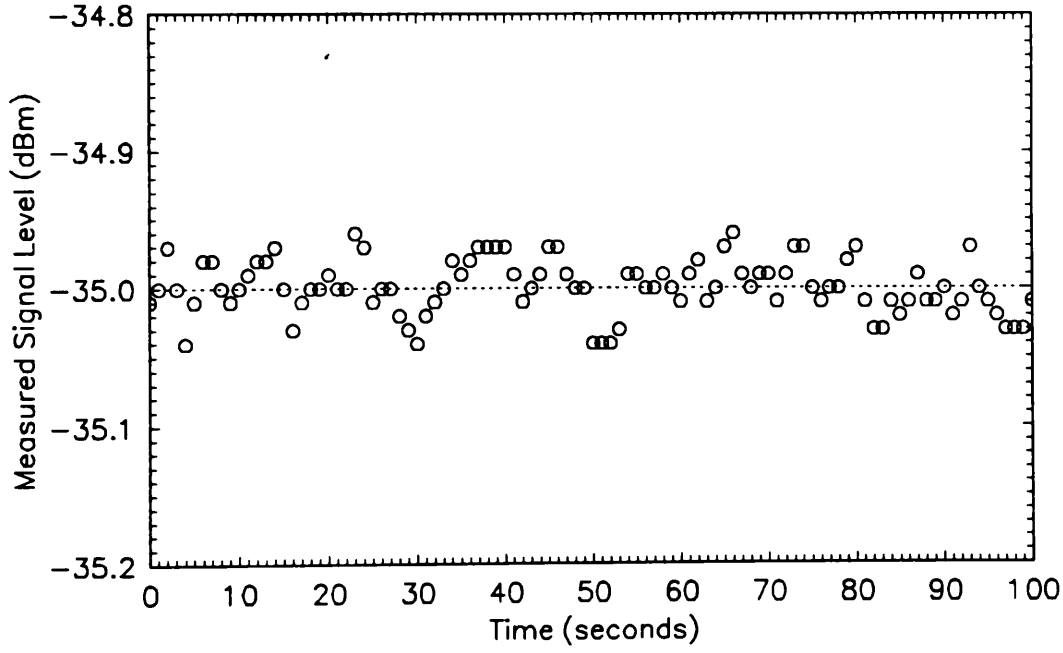


Figure 7-10. APT receiver signal power measurements of a -35 dBm signal.

Note, the results of signal power measurement tests at levels greater than +8 dBm were not presented because the receiver saturates at +8 dBm due to the voltage gain of 4 in the analog-to-digital converter circuit. This voltage gain can be altered to provide a signal power measurement range suited to a specific application.

Table 7-1. Standard deviations of APT receiver signal power measurements.

Signal Level (dBm)	Standard Deviation (dBm)
+10.00	
+9.00	
+8.00	0.0211
+7.00	0.0194
+6.00	0.0197
+5.00	0.0184
+4.00	0.0210
+3.00	0.0200
+2.00	0.0192
+1.00	0.0216
0.00	0.0203
-1.00	0.0188
-2.00	0.0212
-3.00	0.0189
-4.00	0.0190
-5.00	0.0192
-6.00	0.0184
-7.00	0.0217
-8.00	0.0213
-9.00	0.0189
-10.00	0.0196
-11.00	0.0212
-12.00	0.0104

Signal Level (dBm)	Standard Deviation (dBm)
-13.00	0.0210
-14.00	0.0212
-15.00	0.0186
-16.00	0.0219
-17.00	0.0217
-18.00	0.0196
-19.00	0.0224
-20.00	0.0196
-21.00	0.0195
-22.00	0.0201
-23.00	0.0199
-24.00	0.0202
-25.00	0.0209
-26.00	0.0238
-27.00	0.0221
-28.00	0.0201
-29.00	0.0216
-30.00	0.0207
-31.00	0.0208
-32.00	0.0217
-33.00	0.0200
-34.00	0.0204
-35.00	0.0200

7.2.2 Linearity

Linearity tests were performed by injecting a synthesized 455 kHz signal into the receiver's input using an HP3330B signal generator. 100 signal power measurements were obtained at signal levels from +10 dBm to -35 dBm in 1 dBm increments. The geometric mean of the measurements taken at each signal power level are plotted versus their expected values in Figure 7-11. These results are also tabulated in Table 7-2.

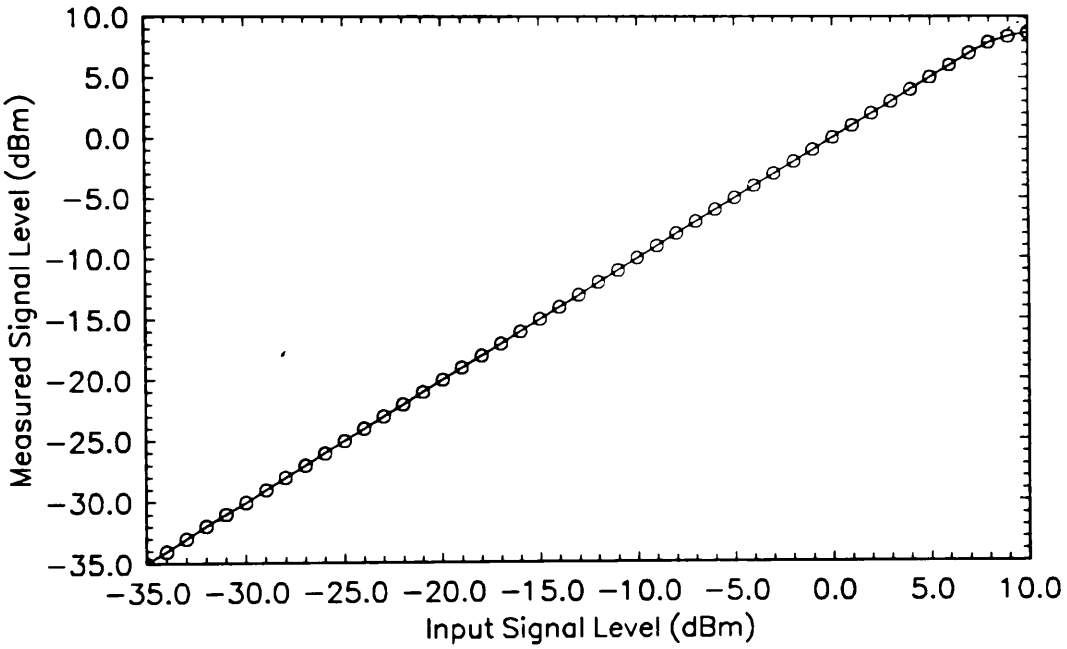


Figure 7-11. Plot of APT receiver's linearity curve. Each point represents the geometric mean of 100 signal power measurements.

As shown, the receiver's output is exceptionally linear over a +8 dBm to -35 dBm operating range.

Table 7-2. APT receiver linearity measurements.

Expected Value (dBm)	Measured Value (dBm)
+10.00	+8.71
+9.00	+8.38
+8.00	+7.90
+7.00	+7.00
+6.00	+6.00
+5.00	+4.99
+4.00	+4.00
+3.00	+3.00
+2.00	+2.00
+1.00	+1.00
0.00	0.00
-1.00	-1.00
-2.00	-2.00
-3.00	-3.00
-4.00	-4.00
-5.00	-5.00
-6.00	-6.00
-7.00	-6.99
-8.00	-8.00
-9.00	-9.00
-10.00	-10.00
-11.00	-11.00
-12.00	-12.00

Expected Value (dBm)	Measured Value (dBm)
-13.00	-13.00
-14.00	-14.00
-15.00	-15.00
-16.00	-16.00
-17.00	-17.00
-18.00	-18.00
-19.00	-19.00
-20.00	-20.00
-21.00	-21.00
-22.00	-22.00
-23.00	-23.00
-24.00	-23.99
-25.00	-25.01
-26.00	-26.01
-27.00	-27.00
-28.00	-27.99
-29.00	-29.00
-30.00	-30.02
-31.00	-30.98
-32.00	-31.97
-33.00	-32.99
-34.00	-34.01
-35.00	-35.00

7.3 Dynamic Range

Dynamic range tests were conducted on the APT receiver using the OLYMPUS 20 GHz beacon test system described in Section 7.1. Recall, in order to make accurate signal power measurements, the receiver must be able to distinguish the carrier signal from spurious noise spikes. To determine its ability to identify the carrier signal, the APT receiver calculates the variance of the difference between consecutive carrier frequency estimates using Equation (5-13). As shown in Figure 7-12, these variance calculations clearly determine the receiver's lock status. The APT receiver is unable to reliably identify the carrier at signal-to-noise ratios below approximately 5 dB in a 20 Hz bandwidth (18 dBHz in a 1 Hz bandwidth, 31 dB fade conditions on the APT system).

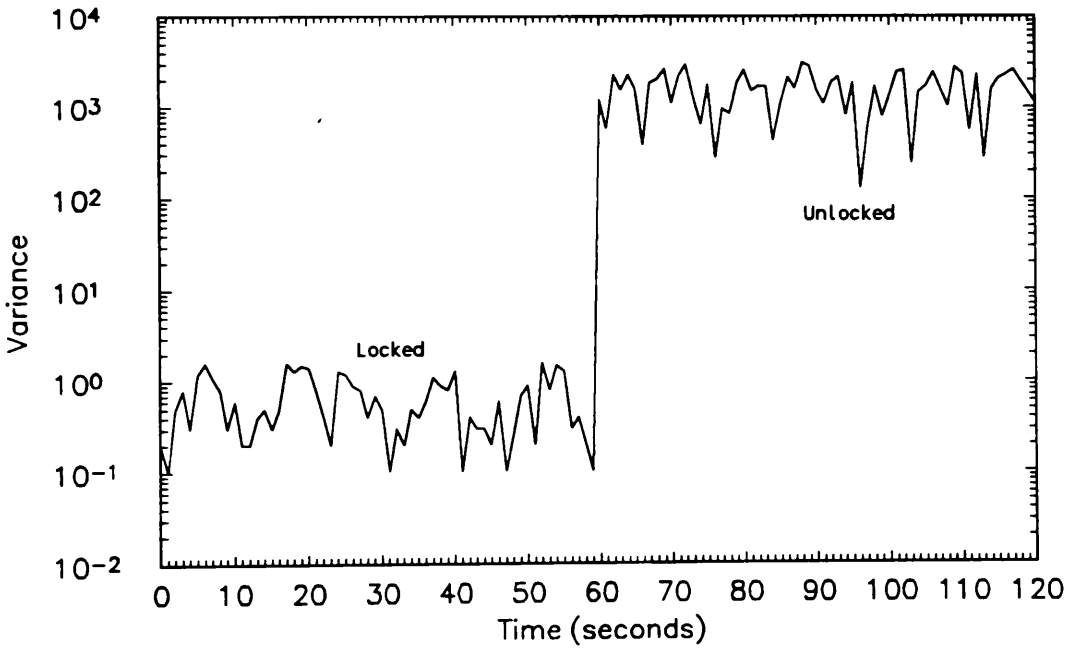


Figure 7-12. Variance of carrier frequency estimates generated by the APT receiver at signal-to-noise ratios near the receiver lock threshold.

7.4 Supplemental Functions

As described in Section 5.4.4, the APT receiver algorithm is able to provide several supplemental functions not specifically required by the ACTS propagation experiment. All of these functions have been implemented and tested. The results of these tests are presented in the following sections.

7.4.1 Spectrum Analyzer Output

The APT receiver can generate approximate spectral data about a 303.333 kHz bandwidth centered on the current carrier frequency. When the user requests this information, the receiver generates 1024, 2048, 4096, 8192, 16384 or 32768-point spectrum (user-specified size) by performing an FFT on the 303.333 kHz complex data stream. It should be noted that due to the discrete nature of the FFT and imperfect complex sampling, these frequency domain representations are only approximate. Refer to Section 5.4.1 for more information. Several spectra generated by the receiver are shown in Figures 7-13 through 7-16[32].

[32] The test system in Figure 7-1 was used to downconvert the OLYMPUS 20 GHz beacon to 455 kHz.

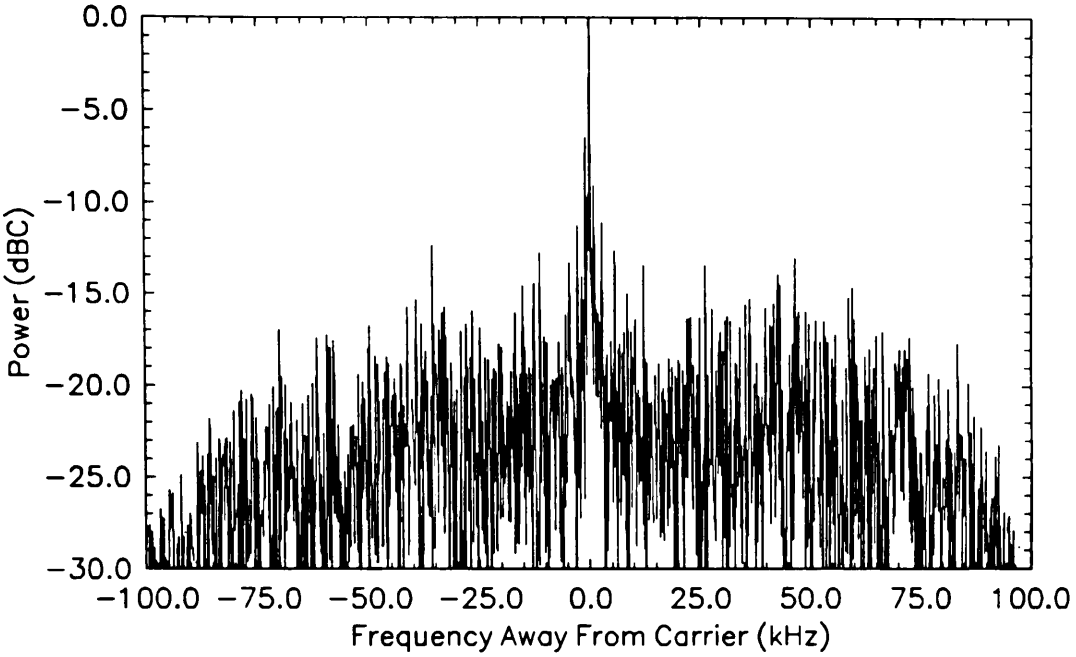


Figure 7-13. 2048-point spectrum of the OLYMPUS 20 GHz beacon.

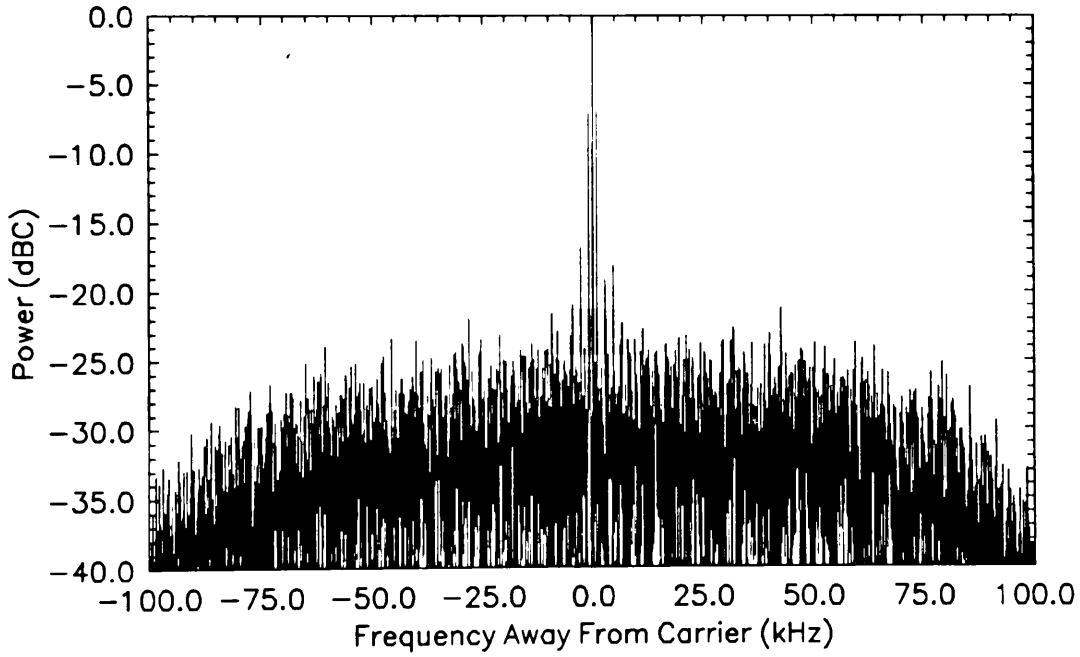


Figure 7-14. 8192-point spectrum of the OLYMPUS 20 GHz beacon.

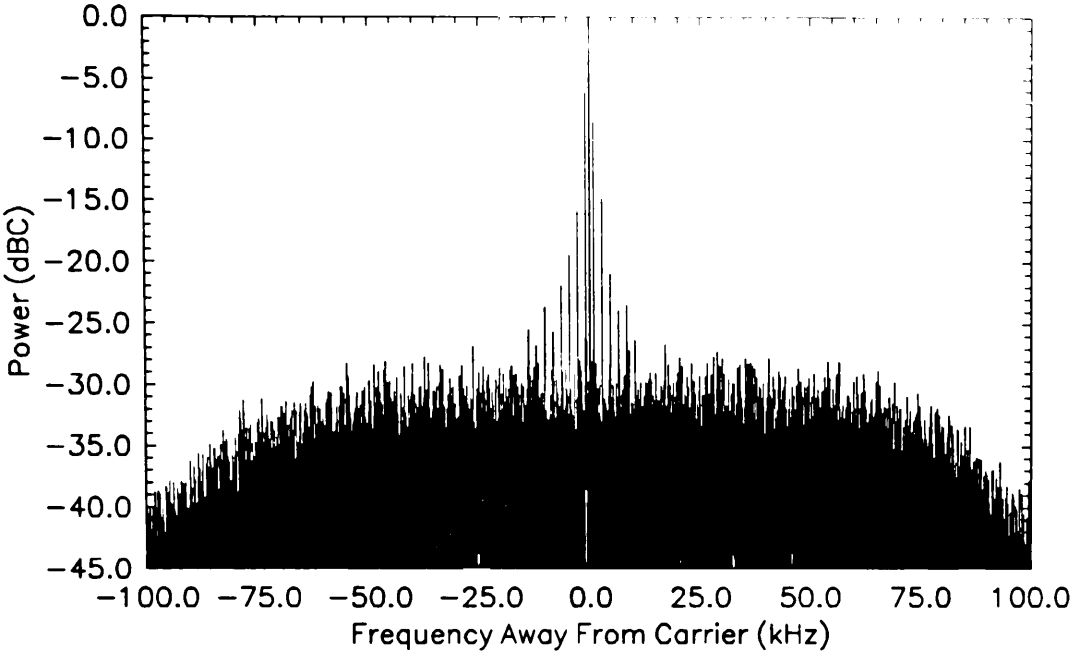


Figure 7-15. 32768-point spectrum of the OLYMPUS 20 GHz beacon.

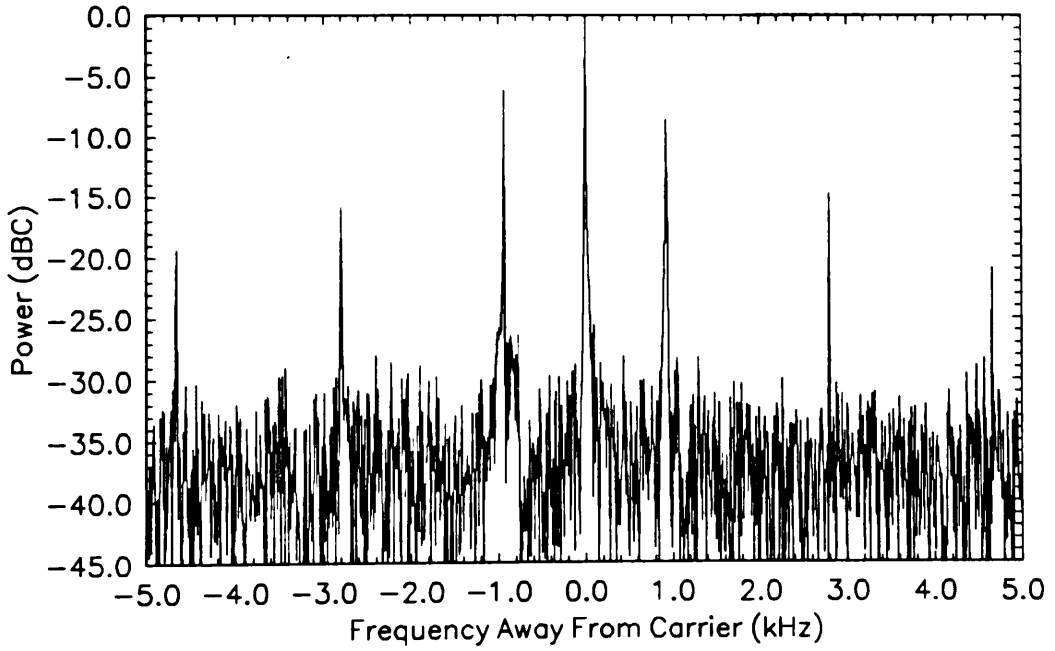


Figure 7-16. Expanded view of the 32768-point spectrum of the OLYMPUS 20 GHz beacon.

7.4.2 Selectable Detection Bandwidth

The APT receiver detection bandwidth can be altered by the user without interrupting the operation of the receiver. The receiver can provide detection bandwidths from 2 Hz to 50 Hz in approximately 1 Hz increments. As shown in Appendix B, a *detection bandwidth* field is included with all signal power measurements to indicate the current detection bandwidth.

This function was implemented because no precise phase noise information is available for the ACTS beacons. As such, it was impossible to calculate an optimum detection filter width. Using the selectable detection bandwidth feature, the Virginia Tech Satellite Communications Group can empirically determine an optimal detection filter width after ACTS is launched. Furthermore, the APT experimenters may update the operation of their receivers to match that of Virginia Tech without any hardware or software modifications.

7.4.3 Force Carrier Reacquisition

The user can force the APT receiver to enter carrier signal acquisition mode at any time. Although the tests described above indicate the APT receiver will reliably acquire the ACTS carrier signals, it is statistically possible for the receiver to incorrectly lock to another tone in the acquisition bandwidth. Should this occur, the operator may use this function to force the receiver to repeat the carrier signal acquisition process.

7.4.4. Track Hold

The user can disable the APT receiver's carrier tracking algorithm at any time by issuing the *track hold* command described in Appendix B. The receiver will not track carrier frequency drift or detect the out-of-lock condition until another *track hold* command is issued. As described in Section 5.4.4.5, this feature was implemented to reduce system downtime during the APT radiometer calibration process.

7.4.5 Optional High Sample Rate

As discussed in Chapter 2, the collective experience of the ACTS Users' Community indicates that a 1 Hz signal power measurement rate is sufficient to characterize the atmospheric phenomena of interest. The 1 Hz sampling rate is also supported by a study conducted by the Virginia Tech Satellite Communications Group[12]. Several users have, however, requested signal power measurements at a higher sampling rate. To satisfy the needs of these users, the APT receiver provides 20 Hz signal power measurements in addition to the required 1 Hz measurements. At discussed in Section 4.2, the 20 Hz measurements are bandlimited to 10 Hz in order to satisfy the Nyquist criteria.

7.4.6 Carrier Frequency Estimates

In order to remain locked to the carrier signal, the APT generates carrier frequency estimates accurate to ±0.5 Hz using the results of the 40 Hz signal power measurement mode FFTs. Figure 7-17 displays the output of a signal power measurement mode FFT generated when the APT receiver was tracking the OLYMPUS 20 GHz beacon.

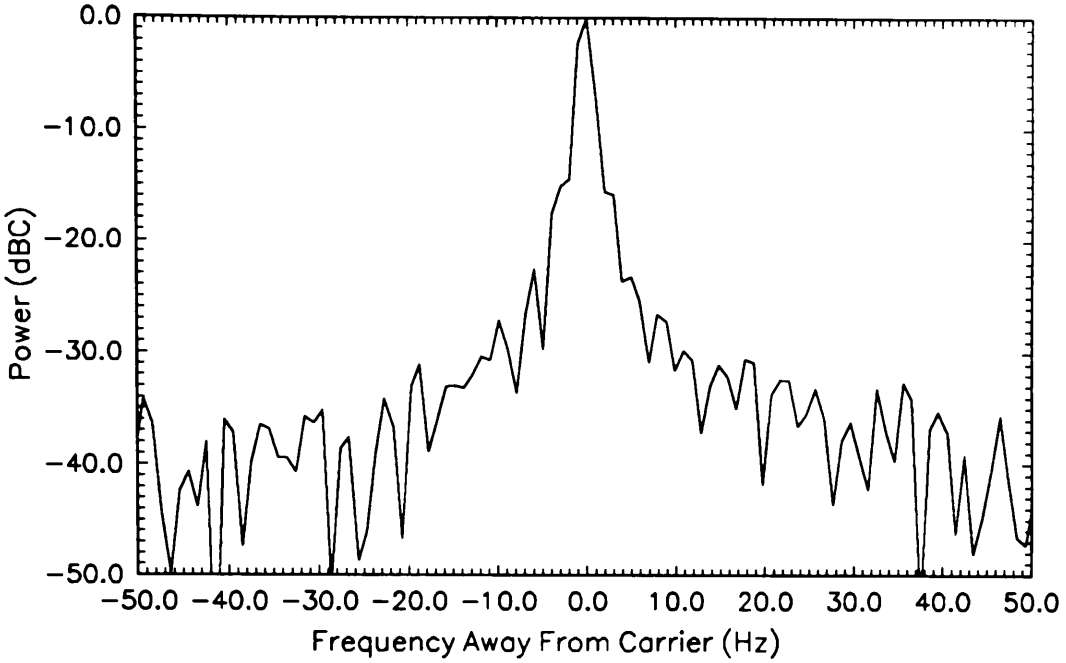


Figure 7-17. APT receiver signal power measurement mode FFT of the OLYMPUS 20 GHz beacon.

To determine the accuracy of the APT receiver carrier frequency estimates, a synthesized signal was injected into the receiver's input using a Hewlett Packard HP3330B signal generator. The synthesized signal was slewed back and forth between 455 kHz±5 Hz at a rate of 0.33 Hz/second. As shown in Figure 7-18, the receiver was able to identify 1 Hz changes in the frequency of the input signal.

7.5 Thermal

Thermal tests were conducted on the APT receiver system to ensure that the receiver will operate at ambient temperatures up to 60° Celsius. The receiver subenclosure was placed in a temperature controlled chamber while the receiver was locked to the OLYMPUS 20 GHz beacon. The ambient

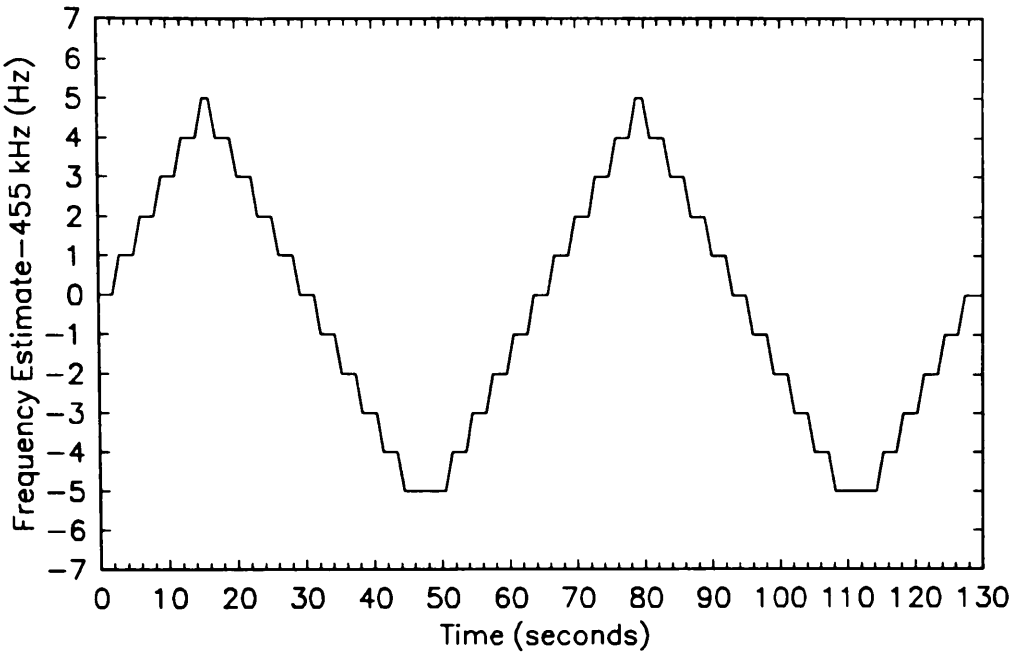


Figure 7-18. Plot demonstrating the APT receiver's frequency detection capability.

temperature in the chamber was then increased to 60° Celsius. No changes in the receiver's operation were detected. Furthermore, at the end of the test, the temperatures of critical components were only slightly higher than the ambient temperature. Hence, the heat sink/fan/subenclosure design is very satisfactory.

Chapter 8

Conclusions and Recommendations

The theory, design and implementation of an accurate, low cost digital beacon receiver has been presented in the context of the ACTS propagation experiments. Chapter 1 described the need for additional statistical data on the effects of atmospheric phenomena on signal propagation at K_a band. Many emerging technologies require new high bandwidth satellite communication links and lower frequency bands are already crowded. The ACTS propagation experiments will generate statistical data which will aid engineers in the design of reliable satellite communication links at K_a band.

The Virginia Tech Satellite Communications Groups is designing, constructing and distributing eight ACTS propagation terminals (APTs) to experimenters across North America. Chapter 2 provided an overview of the APT system and presented receiver subsystem requirements. As shown in Chapter 3, a digital receiver is better suited to the APT requirements than an analog receiver. The APT receiver algorithm is a hybrid algorithm based on the existing digital receiver technology discussed in Chapter 4 and on the simulation of the alternative digital signal processing techniques presented in

Chapter 5. The APT receiver digitizes the unlocked carrier signal at 455 kHz and utilizes multiple fast Fourier transforms to discern the carrier signal amidst neighboring modulation tones. In signal power measurement mode the APT receiver locks the carrier signal at 455 kHz and uses a comb filter, two FIR filters and additional FFTs to make signal power measurements and track carrier frequency drift. The design of the APT algorithm also permits the receiver to provide several supplemental functions including spectrum analyzer type output, carrier frequency estimates accurate to 0.5 Hz and user-selectable detection bandwidths from 2 to 50 Hz in 1 Hz increments.

A versatile digital signal processing platform was designed and constructed to execute the APT receiver algorithm. The components and capabilities of the system presented in Chapter 6 are summarized in Table 8-1. The entire system can be constructed for approximately $1600.

Table 8-1. APT receiver hardware summary.

Component	Capability
DSPμP (D1): Analog Devices ADSP21020-20	• 60 MFLOPS peak, 40 MFLOPS sustained • 32 & 40 bit floating point math (40 bit not supported by APT receiver) • Parallel execution of a floating point addition, a floating point subtraction, a floating point multiplication, three independent memory accesses and condition test in a single 20 MHz clock cycle • Full context switching

Table 8-1 (continued). APT receiver hardware summary.

Component	Capability
ADC (C1): Analog Devices AD9003A	• 12 bit precision • ambient operating temperature range: 0° to 70° Celsius • Maximum clock rate=1 MHz (APT receiver incorporates user selectable clock rates from 1 Hz to 1 MHz in 0.00463 Hz increments) • Analog input bandwidth=4 MHz
Memory (M1): Cypress Semiconductor CYM1831-35PM	• 35 ns access time • program memory: 64KWx48 bits (expandable to 256KWx48 bits) • data memory: 64KWx32 bits (expandable to 256KWx32 bits)
NCO (O1): Qualcomm Q2334I20N	• Two independent channels (APT receiver uses 1 channel to generate the ADC clock and the other to generate the analog output waveform) • Using a 20 MHz clock, output frequency step size=0.00463 Hz. • Spectral purity > 100 dB • Phase noise = -45 dBC at ±2 Hz
USART (S1): Intel 8251A	• APT receiver host interface is standard RS232 (DCE). • Jumper selectable baud rates from 2400 to 19.2K baud. • Break character resets entire APT system.

Table 8-1 (continued). APT receiver hardware summary.

Component	Capability
Bootstrap Loader (B1): Analog Devices Design	• Enables user to download application software to the APT system. Hence, the APT receiver system can execute different algorithms without any hardware modifications.

The APT system tests presented in Chapter 7 and summarized in Table 8-2 indicate that the receiver meets or exceeds all of goals established by the Virginia Tech Satellite Communications Group and the ACTS Users' Community.

Table 8-2. APT receiver performance/function summary.

Parameter	APT Receiver Performance
Acquisition Bandwidth	455 kHz±151.666 kHz
Acquisition Signal Level Threshold (2 seconds)	10 dB C/N in a 20 Hz bandwidth (23 dBHz C/N in a 1 Hz bandwidth, 26 dB fade conditions on the APT system)
Signal Power Measurement Resolution	0.01 dB
Signal Power Measurement Accuracy	0.1 dB
Signal Power Measurement Sample Rate	1 Hz and/or 20 Hz
Detection Bandwidth	Software selectable: 2 Hz to 50 Hz in 1 Hz increments.
Out-of-Lock Signal Level Threshold	5 dB C/N in a 20 Hz bandwidth (18 dBHz C/N in a 1 Hz bandwidth, 31 dB fade conditions on the APT system)
Spectrum Analyzer Mode Output	1024, 2048, 4096, 8192, 16384 and 32768-point, 303.333 kHz-wide spectrums centered on the current carrier frequency available upon request.
Carrier Frequency Estimates	Accurate to ±0.5 Hz, available upon request.
Track Pause	Carrier frequency drift tracking/out-of-lock detection can be temporarily disabled upon request.

8.1 Current Status

The Virginia Tech Satellite Communications Group is currently constructing 16 receivers for use in eight APT which will be distributed to experimenters across North America in early 1993.

8.2 Recommendations for Future Work

The digital receiver system described above is extremely flexible. As a result, many digital-signal-processing projects could utilize the hardware designed to implement the APT receiver. The APT receiver algorithm takes advantage of this flexibility to provide a number of supplemental functions not specifically required by the ACTS propagation experiments. The system's flexibility could be further exploited to provide improved performance in very deep fade conditions and to generate carrier signal phase noise information.

8.2.1 Improved Deep Fade Performance

As described in Chapter 5, the APT receiver does not immediately reenter signal acquisition mode when it is no longer able to reliably identify the carrier signal. Rather, the receiver examines a 100 Hz bandwidth centered on the carrier's last known frequency for a period of time based on the worst case carrier frequency drift rate. By maintaining a narrow bandwidth the receiver will be able to reacquire the carrier signal at lower signal-to-noise ratios if the fade conditions lessen before the receiver reenters signal acquisition mode. The receiver's deep fade performance could be improved by scanning the 100 Hz bandwidth for a period of time based on the carrier's last known actual frequency drift rate instead of the worst case rate. This modification would be very easy to implement because the receiver already calculates the required frequency estimates in order to maintain lock.

8.2.2 Phase Noise Information

As shown in Figure 7-18, the APT receiver is able to resolve the carrier signal to a 1 Hz bandwidth.[33] The APT receiver code would have to be modified only slightly in order to make this information available to the user.

[33] This spectral information will ultimately be used to determine the phase noise characteristics of the ACTS beacons.

Appendix A

A.1 APT 20 GHz System Link Budget

Frequency	20.185 GHz
EIRP Toward Va. Tech	20.70 dBW
Modulation Loss	3.20 dB
Pointing Loss at Blacksburg	0.00 dB
EIRP Toward Blacksburg	17.50 dBW
Path Loss	210.07 dB
Clear Sky Attenuation	0.95 dB
Antenna Efficiency	65.00 %
Antenna Gain (1.22 Meter)	46.36 dB
Nominal Antenna Temperature	100.00 K
Pointing Loss	0.10 dB
Net Losses	164.51 dB
Power Available from Antenna	-147.26 dBW
Switch Insertion Loss	1.00 dB
Diplexer Loss	0.70 dB
Converter Gain	21.00 dB
Converter Noise Figure (SSB)	5.60 dB
IF System Noise Figure	5.00 dB
Net RF Gain	19.30 dB
IF Signal Power at Preamp Output	-127.96 dBW
System Noise Temperature	1645.32 K
G/T	14.19 dB/K
Equivalent Noise Power at Antenna Terminal in 1 1 Hz Bandwidth	-196.44 dBW
Equivalent Noise Power at IF Processor in a 1 Hz Bandwidth	-177.14 dBW
C/N in a 15 Hz Bandwidth	37.42 dB
C/N_0	49.18 dBHz

A.2 APT 27.5 GHz System Link Budget

Frequency	27.505 GHz
EIRP Toward Va. Tech	16.50 dBW
Modulation Loss	0.00 dB
Pointing Loss at Blacksburg	0.00 dB
EIRP Toward Blacksburg	16.50 dBW
Path Loss	212.76 dB
Clear Sky Attenuation	0.70 dB
Antenna Efficiency	65.00 %
Antenna Gain (1.22 Meter)	49.05 dB
Nominal Antenna Temperature	100.00 K
Pointing Loss	0.10 dB
Net Losses	164.51 dB
Power Available from Antenna	-148.01 dBW
Switch Insertion Loss	1.00 dB
Diplexer Loss	0.30 dB
Converter Gain	21.00 dB
Converter Noise Figure (SSB)	6.20 dB
IF System Noise Figure	7.00 dB
Net RF Gain	19.70 dB
IF Signal Power at Preamp Output	-128.31 dBW
System Noise Temperature	1677.49 K
G/T	16.80 dB/K
Equivalent Noise Power at Antenna Terminal in 1 1 Hz Bandwidth	-196.35 dBW
Equivalent Noise Power at IF Processor in a 1 Hz Bandwidth	-176.65 dBW
C/N in a 15 Hz Bandwidth	36.58 dB
C/N_0	48.34 dBHz

Appendix B

APT Receiver Interface Summary

All communication to the APT receiver is via a DCE RS-232 serial port on the APT receiver board. The APT receiver uses eight data bits, one stop bit and no parity. Jumper selectable baud rates of 2400, 4800, 9600 and 19200 are available at H1 (see the APT receiver schematic in Appendix D).

B.1 Host to APT Receiver Interface

The following commands may be issued to the APT receiver by the host. The receiver will respond to certain commands by sending data or commands back to the host. The format of information output by the APT receiver is described later in this appendix.

Generate Spectrum

The *generate spectrum* command will cause the APT receiver to temporarily stop executing the signal power measurement/carrier tracking algorithm to generate a frequency domain representation of the ±151.5 kHz bandwidth centered about the current carrier frequency. After both bytes of the command have been issued, the receiver will direct the host to set the IF filter selection switch so the output of the 200 kHz filter is input to the receiver. When the host acknowledges the filter selection request, the receiver will generate a spectrum of user selectable length and will immediately begin to transmit the results to the host. The spectral information stream will consist of a frequency word followed by *n* spectrum words where *n* is the size of the requested spectrum. The receiver will then ask the host to set the IF filter selection switch so the output of the 10 kHz filter is input to the receiver.

When the host acknowledges the filter selection request the receiver will resume execution of the signal power measurement/carrier signal tracking algorithm.

Note,

- There will be a 4.5 second delay after the host sends the acknowledge signal before the receiver starts to produce accurate 1 Hz signal power measurements.
- If the receiver is prompted for signal power measurements during the spectrum generation process, the receiver will respond with a signal power measurement of -71.92 dBm.
- No error checking is performed on the *S* parameter described below. Invalid *S* values will produce unpredictable results.
- The APT receiver will generate and transmit a *default_spectrum_size* (see the APT receiver software internal documentation) spectrum each time the receiver completes the carrier signal acquisition algorithm.

Command format: 0x53, *S* (comma is not transmitted)

where: *S*=0 to generate a 1024-point spectrum
S=1 to generate a 2048-point spectrum
S=2 to generate a 4096-point spectrum
S=3 to generate a 8192-point spectrum
S=4 to generate a 16384-point spectrum
S=5 to generate a 32768-point spectrum

Force Acquisition

The *force acquisition* command will cause the receiver to stop executing the signal power measurement/carrier signal tracking algorithm and begin the carrier signal acquisition process. The receiver will direct the host to set the IF filter selection switch so the output of the 200 kHz filter is input to the

receiver. When the host acknowledges the filter selection request, the receiver will begin executing the carrier signal acquisition algorithm.

Command Format: 0x48

Change Detection Bandwidth

When a *change detection bandwidth* command is issued to the APT receiver, the receiver will begin to use the new detection bandwidth to calculate signal power measurements. This command may be issued without interrupting the receiver's operation. Note, the current detection bandwidth is indicated in the *detection bandwidth* field of all signal power measurements.

Command Format: 0x42, *S*

where: *S*=new detection bandwidth (Hz). *S* must be is an 8-bit integer between 2 and 50. Note, no error checking is performed on this value. Invalid values of *S* will cause unpredictable results.

Track Hold

The track hold command will cause the receiver to stop tracking the carrier frequency drift until another track hold command is issued. Note, receiver will not check for an out-of-lock condition while tracking is disabled.

Command Format: 0x54

Acknowledge

The APT receiver will direct the host to alter the position of the filter selection switch on the IF board as appropriate for the current receiver

operation. After the host has set the switch position, the host must send an *acknowledge* command to the receiver before the receiver will proceed.

Command Format: 0x41

Send Current Carrier Frequency Estimate

The host may obtain an estimate of the current carrier frequency by issuing a *send current carrier frequency estimate* command to the APT receiver. The receiver will respond by sending a frequency word which contains the most recent carrier frequency estimate.

Command Format: 0x46

Send 1 Hz Signal Power Measurement Sample

The host may obtain the most recent 1 Hz signal power measurement sample by issuing a *send 1 Hz signal power measurement sample* command to the APT receiver. The receiver will respond by sending a signal power word which contains the most recent 1 Hz signal power measurement. Note, all signal power measurements contain a *time stamp* field. This 7-bit field is incremented by 1 each time a new signal power measurement sample is generated. Hence, it is possible for the host to detect missing or duplicate measurements.

Command Format: 0x31

Send 20 Hz Signal Power Measurement Sample

The host may obtain the most recent 20 Hz signal power measurement sample by issuing a *send 20 Hz signal power measurement sample* command to the APT receiver. The receiver will respond by sending a signal power word which contains the most recent 20 Hz signal power measurement. Note, all

signal power measurements contain a *time stamp* field. This 7-bit field is incremented by 1 each time a new signal power measurement sample is generated. Hence, it is possible for the host to detect missing or duplicate measurements.

Command Format: 0x32

Reset

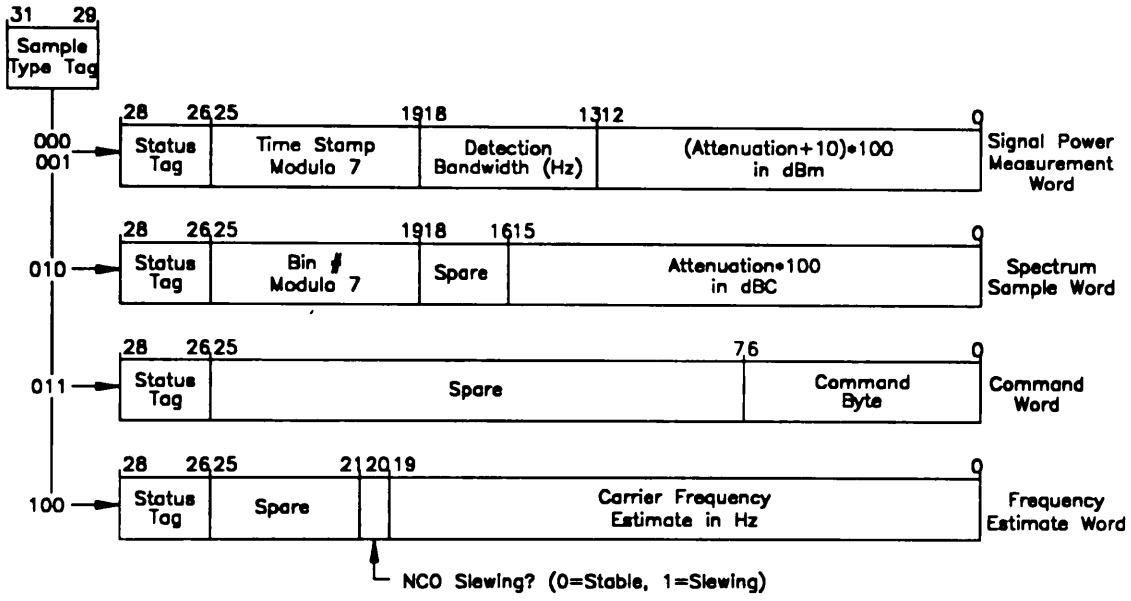
The user may reset the entire APT receiver system by sending a break signal to the APT receiver serial port. The receiver will respond by directing the host to download new executable code. Note, all words output by the APT receiver are 4 bytes long. In order to synchronize communication with the receiver, the host must execute at least one reset command.

Command Format: Send a break signal for 0.5 seconds.

B.2 APT Receiver to Host Interface

The APT receiver will output data and/or commands in response to commands issued by the host. The format of these command and data words is described below. Note, the receiver transmits the low byte of all command and data words first; the remaining bytes follow in ascending order.

The type and format of data contained in each word output by the receiver is marked in the *sample type tag.* As shown below different types of data are formatted differently.



Sample Type Tag Definitions	
1 Hz Signal Power Measurement Word	000
20 Hz Signal Power Measurement Word	001
Spectrum Word	010
Frequency Estimate Word	011
Command Word	100

Status Tag Definitions	
<Acquisition Mode>: • The receiver is currently attempting to acquire the carrier signal. • Signal power measurements are invalid.	000
<Locked>: • The receiver is locked to the carrier signal. • Signal power measurements are valid.	001
<Tracking Disabled>: • Tracking has been temporarily disabled using the *Track Pause* command. • Signal power measurements are invalid.	010
<Generating Spectrum>: • The receiver is currently generating spectrum data. • Signal power measurements are invalid.	011
<Low SNR>: • The receiver is unable to reliably identify the carrier signal but has not re-entered carrier signal acquisition mode yet. (The receiver is attempting to reacquire the carrier in the signal power measurement mode bandwidth). • Signal power measurements may or may not be valid.	100

Command Byte Definitions	
Switch IF filter selection switch so the output of the 200 kHz filter is input to the receiver.	0x57
Switch IF filter selection switch so the output of the 10 kHz filter is input to the receiver.	0x4E
Download executable code.	0x43

As shown above, all data output by the receiver is 4 bytes long. To synchronize communication with the receiver, the host must issue a *reset* command. The next byte output by the receiver will be the low byte of a *send executable code* command.

Appendix C

APT Receiver Source Code Listings

C.1 Algorithm Source Code Listing

```
/*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/
/*!!!                         APT RECEIVER SOFTWARE                         !!!*/
/*!!!                         ---------------------                         !!!*/
/*!!!                         ALGORITHM SOURCE CODE                         !!!*/
/*!!!                                                                       !!!*/
/*!!!                       William R. Sylvester Jr.                        !!!*/
/*!!!                    Satellite Communications Group                     !!!*/
/*!!!                         621 Whittemore Hall                           !!!*/
/*!!!          Virginia Polytechnic Institute and State University          !!!*/
/*!!!                         Blacksburg, VA  24061                         !!!*/
/*!!!                            (703) 231-6834                             !!!*/
/*!!!                                                                       !!!*/
/*!!!                            August, 1992                               !!!*/
/*!!!                                                                       !!!*/
/*!!!=======================================================================!!!*/
/*!!!                                                                       !!!*/
/*!!!   Commands From Host To DRX                                           !!!*/
/*!!!   -------------------------                                           !!!*/
/*!!!   Generate Spectrum:              <S,#> where:                        !!!*/
/*!!!                                      #=0->1024-point spectrum         !!!*/
/*!!!                                        1->2048-point spectrum         !!!*/
/*!!!                                        2->4096-point spectrum         !!!*/
/*!!!                                        3->8192-point spectrum         !!!*/
/*!!!                                        4->16384-point spectrum        !!!*/
/*!!!                                        5->32768-point spectrum        !!!*/
/*!!!   Forced Acquisition:             <H>                                 !!!*/
/*!!!   Detection Bandwidth:            <B,size (Hz)=8 bit integer>         !!!*/
/*!!!   Track Pause (toggle on/off):    <T>                                 !!!*/
/*!!!   IF Filter Position Acknowledge: <A>                                 !!!*/
/*!!!   Send 1 Hz sample:               <1>                                 !!!*/
/*!!!   Send 20 Hz sample:              <2>                                 !!!*/
/*!!!                                                                       !!!*/
/*!!!   Commands From DRX to Host                                           !!!*/
/*!!!   -------------------------                                           !!!*/
/*!!!   Use 200 KHz IF Filter:          <W>                                 !!!*/
/*!!!   Use 10 KHz IF Filter:           <N>                                 !!!*/
/*!!!   Send DRX executable code:       <C>                                 !!!*/
/*!!!                                                                       !!!*/
/*!!!     31       29                                                       !!!*/
/*!!!    |==========|                                                       !!!*/
/*!!!    |  Sample  |                                                       !!!*/
/*!!!    | Type Tag |                                                       !!!*/
/*!!!    |==========|                                                       !!!*/
/*!!!    |                                                                  !!!*/
/*!!!    |      28      26 25         19 18          13 12             0  !!!*/
/*!!!    |      |========|============|===============|================|  !!!*/
/*!!!   000 --> | Status | Time Stamp |   Detection   | (Atten+10)*100 |  !!!*/
/*!!!   001 --> |  Tag   |  modulo 7  | Bandwidth (Hz)|     in dBm     |  !!!*/
/*!!!    |      |========|============|===============|================|  !!!*/
```

```
/*!!!   |                                                                 !!!*/
/*!!!   |                                                                 !!!*/
/*!!!   |      28     26 25         19 18   16 15                  0      !!!*/
/*!!!   |      |=======|===========|======|=======================|      !!!*/
/*!!!  010 --> | Status |    Bin #   | Spare |    Attenuation*100       |  !!!*/
/*!!!   |      |  Tag   |  modulo 7  |       |         in dBC           |  !!!*/
/*!!!   |      |=======|===========|======|=======================|      !!!*/
/*!!!   |                                                                 !!!*/
/*!!!   |      28     26 25                             7 6        0      !!!*/
/*!!!   |      |=======|==================================|========|      !!!*/
/*!!!  011 --> | Status |              Spare               | Command |    !!!*/
/*!!!   |      |  Tag   |                                  |  Word   |    !!!*/
/*!!!   |      |=======|==================================|========|      !!!*/
/*!!!   |                                                                 !!!*/
/*!!!   |      28     26 25   21   20   19                       0        !!!*/
/*!!!   |      |=======|======|=====|==========================|          !!!*/
/*!!!  100 --> | Status | Spare | NCO  |     Carrier Frequency      |     !!!*/
/*!!!          |  Tag   |       | Slew |      Estimate in Hz        |     !!!*/
/*!!!          |=======|======|=====|==========================|          !!!*/
/*!!!                                                                     !!!*/
/*!!!  Status Tag                          Sample Type Tag                !!!*/
/*!!!  ----------                          ---------------                !!!*/
/*!!!  Acquisition Mode:        000        1 Hz Data Sample:       000    !!!*/
/*!!!  Locked:                  001        20 Hz Data Sample:      001    !!!*/
/*!!!  Tracking Disabled:       010        Spectrum Sample:        010    !!!*/
/*!!!  Generating Spectrum:     011        Command:                011    !!!*/
/*!!!  Low SNR:                 100        Frequency:              100    !!!*/
/*!!!                                                                     !!!*/
/*!!!  * NCO Slew = 0 if NCO output frequency is stable, 1 is the output  !!!*/
/*!!!              frequency is slewing                                   !!!*/
/*!!!                                                                     !!!*/
/*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/
/****************************************************************************/
/**** Include the following files to make system routines and constants ****/
/**** available.                                                         ****/
/****************************************************************************/
#include "def21020.h"   /* Included to define symbolic names for 21020 system */
                        /* register bits. (This file provided by Analog       */
                        /* Devices Inc.)                                      */

/****************************************************************************/
/**** External declarations inform the compiler that certain routines will ****/
/**** be compiled separately.                                             ****/
/****************************************************************************/
.EXTERN sine,cosine,divide,log10,logb2;  /* These routines reside in the     */
                                         /* Analog Devices assembly library. */

/****************************************************************************/
/**** Compiler directives tailor program compilation to DRX hardware.    ****/
/****************************************************************************/
.PRECISION=32;          /* The DRX hardware supports 32 bit data storage (not */
                        /* 40 bit).  All ADSP21020 arithmetic operands should */
                        /* be rounded to 32 bits.                             */

.ROUND_NEAREST;         /* Although floating point data representation should */
                        /* cause coefficient/computation quantization error to */
                        /* be negligible, quantization error is minimized when */
                        /* symmetric rounding is used.                        */

/****************************************************************************/
/****                     INTERRUPT VECTOR TABLE                         ****/
/****************************************************************************/
.segment/pm IRQtable;
Reserved_1:     rti;           nop; nop; nop; nop; nop; nop; nop;
Chip_Reset:     JUMP start;    nop; nop; nop; nop; nop; nop; nop;
Reserved_2:     rti;           nop; nop; nop; nop; nop; nop; nop;
Stack_Error:    rti;           nop; nop; nop; nop; nop; nop; nop;
Timer_Hi_Prior: rti;           nop; nop; nop; nop; nop; nop; nop;
```

```
IRQ3_Asserted:  JUMP rxrdy;     nop; nop; nop; nop; nop; nop; nop;
IRQ2_Asserted:  rti;            nop; nop; nop; nop; nop; nop; nop;
IRQ1_Asserted:  JUMP txrdy;     nop; nop; nop; nop; nop; nop; nop;
IRQ0_Asserted:  JUMP fifofull; nop; nop; nop; nop; nop; nop; nop;
Reserved_3:     rti;            nop; nop; nop; nop; nop; nop; nop;
Reserved_4:     rti;            nop; nop; nop; nop; nop; nop; nop;
Circ_Overfl_7:  rti;            nop; nop; nop; nop; nop; nop; nop;
Circ_Overfl_15: rti;            nop; nop; nop; nop; nop; nop; nop;
Reserved_5:     rti;            nop; nop; nop; nop; nop; nop; nop;
Timer_Lo_Prior: rti;            nop; nop; nop; nop; nop; nop; nop;
Fix_Pt_Overfl:  rti;            nop; nop; nop; nop; nop; nop; nop;
Flt_Pt_Overfl:  rti;            nop; nop; nop; nop; nop; nop; nop;
Flt_Pt_Underfl: rti;            nop; nop; nop; nop; nop; nop; nop;
Flt_Pt_InvOp:   rti;            nop; nop; nop; nop; nop; nop; nop;
Reserved_6:     rti;            nop; nop; nop; nop; nop; nop; nop;
Reserved_7:     rti;            nop; nop; nop; nop; nop; nop; nop;
Reserved_8:     rti;            nop; nop; nop; nop; nop; nop; nop;
Reserved_9:     rti;            nop; nop; nop; nop; nop; nop; nop;
Reserved_10:    rti;            nop; nop; nop; nop; nop; nop; nop;
User_Softw_1:   rti;            nop; nop; nop; nop; nop; nop; nop;
User_Softw_2:   rti;            nop; nop; nop; nop; nop; nop; nop;
User_Softw_3:   rti;            nop; nop; nop; nop; nop; nop; nop;
User_Softw_4:   rti;            nop; nop; nop; nop; nop; nop; nop;
User_Softw_5:   rti;            nop; nop; nop; nop; nop; nop; nop;
User_Softw_6:   rti;            nop; nop; nop; nop; nop; nop; nop;
User_Softw_7:   rti;            nop; nop; nop; nop; nop; nop; nop;
User_Softw_8:   rti;            nop; nop; nop; nop; nop; nop; nop;
.ENDSEG;

/*****************************************************************************/
/****                        Executable Code Begins Here                  ****/
/*****************************************************************************/
.SEGMENT/PM pmcode;
/*---------------------------------------------------------------------------*/
/* The DRX always operates in one of three modes: acquisition mode, power    */
/* measurement mode or spectrum mode.  Each mode is described in detail below.*/
/* The contents of the variable <DRX_mode> reflects which mode the receiver  */
/* is in as follows:                                                         */
/*---------------------------------------------------------------------------*/
#define mode_acquisition 0   /* Possible contents of the variable DRX_mode    */
#define mode_power_meas  1   /* representing different modes in which the DRX */
#define mode_spectrum    2   /* operates.                                     */

/*---------------------------------------------------------------------------*/
/* DRX status tag definitions.  One of these flags appears in the status tag */
/* of each word output by the DRX.                                           */
/*---------------------------------------------------------------------------*/
#define status_acq_mode  0   /* The DRX is in carrier signal acquisition mode.*/
#define status_locked    1   /* The DRX is locked to the carrier signal       */
                             /*    (signal power measurement mode).           */
#define status_track_dis 2   /* The DRX was locked to the carrier signal      */
                             /*    but carrier signal tracking has been       */
                             /*    disabled.                                  */
#define status_gen_spect 3   /* The DRX is currently generating spectrum      */
                             /*    output.                                    */
#define status_low_SNR   4   /* The DRX has lost lock within the past 29.6    */
                             /*    seconds and is still trying to reacquire   */
                             /*    the carrier signal in signal power         */
                             /*    measurement mode.                          */

/*---------------------------------------------------------------------------*/
/* DRX sample type tag definitions.  One of these flags appears in the       */
/* sample type tag field of each word output by the DRX.                     */
/*---------------------------------------------------------------------------*/
#define stype_1Hz        0   /* Sample is a 1 Hz signal power measurement.    */
#define stype_20Hz       1   /* Sample is a 20 Hz signal power measurement.   */
#define stype_spectrum   2   /* Sample is a spectral power measurement.       */
#define stype_command    3   /* Sample is a command from the DRX to the host. */
```

```
#define stype_frequency  4   /* Sample is a carrier frequency estimate       */

/*---------------------------------------------------------------------------*/
/* Define the default DRX operating conditions.                              */
/*---------------------------------------------------------------------------*/
#define default_detection_bandwidth 20    /* Signal power will be measured in */
                                          /*    this bandwidth.  The value of */
                                          /*    the bandwidth must be an      */
                                          /*    integer between 2 and 50      */
                                          /*    (inclusive).                  */
#define default_spectrum_size       1024  /* The DRX generates a spectrum of  */
                                          /*    this size after acquiring the */
                                          /*    the carrier signal.           */

/*---------------------------------------------------------------------------*/
/* DRX signal power measurements are in relative dB.  This offset is added   */
/* to each measurement to translate the measurement to dBm.  This value      */
/* must be updated for each individual receiver because the ADC op-amp gain  */
/* will be slightly different for each receiver.  To determine the correct   */
/* value for this variable, set the variable value to 0, inject a -10 dBm,   */
/* 455 kHz signal into the receiver.  The correct value for this variable    */
/* is then:                                                                  */
/*                        dBm_offset = -10 - DRX output                      */
/*---------------------------------------------------------------------------*/
#define dBm_offset                  34.75

/*---------------------------------------------------------------------------*/
/* Define the ASCII constants for the DRX-to-host commands.                  */
/*---------------------------------------------------------------------------*/
#define wide_filter                 0x57  /* "W" */
#define narrow_filter               0x4E  /* "N" */
#define send_code                   0x43  /* "C" */

/*---------------------------------------------------------------------------*/
/* Define the Pascal "Boolean" data type values.                             */
/*---------------------------------------------------------------------------*/
#define TRUE  1
#define FALSE 0

/*****************************************************************************/
/****             ADSP21020 RELATED DEFINITIONS/INITIALIZATIONS           ***/
/*****************************************************************************/
start:   BIT SET MODE2 0x10;     /* An ADSP21020 silicon flaw prevents the PM */
         NOP;                    /*    cache from being correctly initialized */
         READ CACHE 0;           /*    on power-up/reset.  This code was      */
         BIT CLR MODE2 0x10;     /*    supplied by Analog Devices to fix the  */
                                 /*    problem.  It must be the first code    */
                                 /*    executed at power-up/reset.            */

#define default_ws 0x21          /* ADSP21020 default memory configuration:   */
         DMWAIT=default_ws;      /*    neither PM or DM require wait states.  */
         PMWAIT=default_ws;
                                 /* Initialize the ALU:                       */
         BIT SET MODE1 ALUSAT;   /*   -> Fixed point overflows are set to the */
                                 /*      maximum representable number, under- */
                                 /*      flows are set to zero.               */
         BIT SET MODE1 RND32;    /*   -> Use 32 bit operands.                 */
         BIT CLR MODE1 TRUNC;    /*   -> Round results to nearest 32 bit      */
                                 /*      boundary.                            */

         DMBANK1=0x10000;        /* Configure DM bank select registers:       */
         DMBANK2=0x20000;        /*    Bank #0 is a 64Kx32 bit SIMM mapped to */
         DMBANK3=0x30000;        /*    0x00000 to 0x0ffff.  The select lines  */
                                 /*    for banks #1, #2 and #3 are used only  */
                                 /*    to select various peripherals.         */

                                 /* FLAG pins are used as peripheral select   */
                                 /*    lines.  Initialize the pins to be      */
```

```
                              /*    output pins.                              */
        BIT SET MODE2 FLG0O|FLG1O|FLG2O|FLG3O;
        BIT SET ASTAT FLG0|FLG1|FLG2;         /*Deselect all peripherals.    */

                              /*Initialize the interrupt structure:          */
                              /*  -> Define all interrupt lines as edge-     */
                              /*     sensitive or level sensitive.           */
        BIT CLR MODE2 IRQ0E|IRQ1E;
        BIT SET MODE2 IRQ2E|IRQ3E;
                              /*  -> Disable all interrupts.                 */
        BIT CLR IMASK IRQ0I|IRQ1I|IRQ2I|IRQ3I;

        BIT CLR IRPTL 0x0;    /*  -> Clear all pending interrupts.           */

        BIT CLR MODE1 NESTM;  /*  -> Disable lower priority interrupts       */
                              /*     interrupts during IRQ servicing.        */

        BIT SET MODE1 IRPTEN; /*  -> Allow interrupt processing.             */

/*****************************************************************************/
/**** Miscellaneous system initializations.                               ****/
/*****************************************************************************/
        /*-------------------------------------------------------------------*/
        /* Initialize default detection bandwidth.                           */
        /*-------------------------------------------------------------------*/
        R0=default_detection_bandwidth;
        PM(detection_bandwidth)=R0;

        /*-------------------------------------------------------------------*/
        /* Initialize default (after carrier signal acquisition) spectrum    */
        /* size.                                                             */
        /*-------------------------------------------------------------------*/
        R0=default_spectrum_size;
        PM(spectrum_size)=R0;

        /*-------------------------------------------------------------------*/
        /* All words output by the DRX are 4 bytes long.  <shift_count>      */
        /* counts how many bytes of the current word have been output.       */
        /*-------------------------------------------------------------------*/
        R0=0;
        PM(shift_count)=R0;

        /*-------------------------------------------------------------------*/
        /* Each signal power measurement contains a 7-bit time stamp which   */
        /* is incremented each time a new measurement is calculated.  These  */
        /* time stamps are initialized to 0.                                 */
        /*-------------------------------------------------------------------*/
        PM(time_stamp_1Hz)=R0;
        PM(time_stamp_20Hz)=R0;

        /*-------------------------------------------------------------------*/
        /* Several Host-to-DRX commands are more than 1 byte long.  The      */
        /* following variables indicate if the DRX is waiting for the second */
        /* byte of a command.                                                */
        /*-------------------------------------------------------------------*/
        R0=FALSE;
        PM(wait_spectrum_size)=R0;
        PM(wait_detection_bandwidth)=R0;

        /*-------------------------------------------------------------------*/
        /* The host must acknowledge all DRX-to-host commands.               */
        /* <acknowledge> indicates whether the host has sent the acknowledge */
        /* command.                                                          */
        /*-------------------------------------------------------------------*/
        PM(acknowledge)=R0;

/*****************************************************************************/
/**** Compute the Hanning window coefficients which are applied to signal  ****/
```

```
/**** power measurement mode FFTs.                                          ****/
/*****************************************************************************/
         F1=0.0;
         F11=2.0;
         F15=1.0;
         B8=hanning; L8=0; M8=1;
         LCNTR=1024, DO calc_han UNTIL LCE;
            F0=PM(pi);
            CALL divide (DB); F0=F0*F1; F12=1023.0;
            CALL sine (DB); L11=0; NOP;
            F0=F0*F0;
calc_han:   F1=F1+F15, PM(I8,M8)=F0;

/*****************************************************************************/
/**** Compute the 30 dB/octave window coefficients which are used to       ****/
/**** bandlimit 20 Hz signal power measurements to 10 Hz (13 pt. window).  ****/
/**** This window is applied to 40 Hz signal power measurements.           ****/
/*****************************************************************************/
         F1=0.0;
         F11=2.0;
         F15=1.0;
         B8=dB30octave_20_coefs; L8=0; M8=1;
         LCNTR=13, DO dB30_20 UNTIL LCE;
            F0=PM(pi);
            CALL divide (DB); F0=F0*F1; F12=13.0;
            CALL sine (DB); L11=0; NOP;
            F0=F0*F0;
dB30_20:    F1=F1+F15, PM(I8,M8)=F0;

/*****************************************************************************/
/**** Compute the 30 dB/octave window coefficients which are used to       ****/
/**** bandlimit 1 Hz signal power measurements to 0.5 Hz (140 pt. window). ****/
/**** This window is applied to 40 Hz signal power measurements.           ****/
/*****************************************************************************/
         F1=0.0;
         F11=2.0;
         F15=1.0;
         B8=dB30octave_1_coefs; L8=0; M8=1;
         LCNTR=140, DO dB30_1 UNTIL LCE;
            F0=PM(pi);
            CALL divide (DB); F0=F0*F1; F12=140.0;
            CALL sine (DB); L11=0; NOP;
            F0=F0*F0;
dB30_1:     F1=F1+F15, PM(I8,M8)=F0;

/*****************************************************************************/
/**** COMMUNICATIONS INTERFACE (8251A) RELATED DEFINITIONS/INITIALIZATIONS ****/
/*****************************************************************************/
#define r_stat_8251 0x00030001   /* Read status register address.            */
#define w_mc_8251   0x00020001   /* Write mode-command register address.     */
#define w_dat_8251  0x00020000   /* Write data register address.             */
#define r_dat_8251  0x00030000   /* Read data register address.              */
#define dm_ws_8251  0x000AD421   /* Exchanges between the ADSP-21020 and the */
                                 /*    8251A require 5 ADSP-21020 DM wait    */
                                 /*    states.                               */

#define scomm_xmit_q_len 3500    /* Length of the serial data transmission   */
                                 /*    queue in words.  The queue must be    */
                                 /*    long enough to hold an entire 1K      */
                                 /*    spectrum and a small amount of control*/
                                 /*    information.   No overrun checking is */
                                 /*    performed on the queue.               */

i8251:   DMWAIT=dm_ws_8251;          /* Initialize the ADSP-21020 DM bus to  */
                                     /*    interact with the 8215A.          */
         BIT SET ASTAT FLG0|FLG2;    /* Deselect all other peripherals.      */
         BIT CLR ASTAT FLG1;         /* Select the 8251A.                    */
         R0=0x00;                    /* Force 8251A to the internal idle     */
```

```
        DM(w_mc_8251)=R0;           /*    state by sending 4 0x00s to the   */
        CALL wait8251;              /*    mode-command register.            */
        DM(w_mc_8251)=R0;
        CALL wait8251;
        DM(w_mc_8251)=R0;
        CALL wait8251;              /* The 8251A has an 8*Tcy write         */
        DM(w_mc_8251)=R0;           /*    recovery time.  "wait8251"        */
        CALL wait8251;              /*    pauses for 8*Tcy=67 ADSP-21020    */
        R0=0x40;                    /*    clock cycles.                     */
        DM(w_mc_8251)=R0;
        CALL wait8251;
        R0=0x4f;              /* Send the 8251A mode word: one stop bit,     */
        DM(w_mc_8251)=R0;     /*    no parity, eight bit chars., baud rate   */
                              /*    factor=64X for 2400 Hz, 4800 Hz, 9600 Hz */
                              /*    or 19.2 kHz.                             */

        CALL wait8251;        /* Send the 8251A command word: transmit       */
        R0=0x37;              /*    enabled, DTR* true, receive enabled,     */
        DM(w_mc_8251)=R0;     /*    send no break character, reset all       */
                              /*    errors, RTS* FALSE, do not reset, do not */
                              /*    enter hunt mode.                         */

        BIT SET ASTAT FLG1;   /* Deselect the 8251A.                         */
        DMWAIT=default_ws;    /* Return the ADSP-21020 DM bus to the default */
                              /*    configuration.                           */

        B15=squeue;           /* Address register #15 controls writes into   */
        M15=1;                /*    the serial communications transmit queue.*/
        L15=scomm_xmit_q_len; /*    I15 always points to the next empty      */
                              /*    queue slot.                              */

        B14=squeue;           /* Address register #14 controls reads from    */
        M14=1;                /*    the serial communications transmit queue.*/
        L14=scomm_xmit_q_len; /*    I14 always points to the slot whose      */
                              /*    contents are to be output next.          */

        BIT SET IMASK IRQ3I;  /* Enable the receive interrupt.               */

/*****************************************************************************/
/****          NUMERICALLY CONTROLLED OSCILLATOR (Q2334I20N) RELATED      ****/
/****                      DEFINITIONS/INITIALIZATIONS                    ****/
/*****************************************************************************/
#define dm_ws_NCO 0x0009421     /* Exchanges between the ADSP-21020 and the    */
                                /*    NCO require 1 ADSP-21020 DM wait         */
                                /*    state.                                   */

#define NCO_ref 16.0E6;         /* The Q2334I20N is driven with a 19.9 MHz     */
                                /*    reference frequency.  The reference      */
                                /*    frequency is required to calculate       */
                                /*    delta phi which determines the frequency */
                                /*    generated by the NCO.                    */

#define clk_NCO_smc       0x00020008 /* Address of the sample clock NCO smc   */
                                     /*    register.                          */
#define clk_NCO_amc       0x0002000A /* Address of the sample clock NCO amc   */
                                     /*    register.                          */
#define clk_NCO_pira_base 0x00020000 /* Address of the sample clock NCO pira  */
                                     /*    register, LSB.                     */
#define clk_NCO_hop_clk   0x0002000E /* Address of the sample clock NCO       */
                                     /*    asynchronous hop clock register.   */
#define mix_NCO_smc       0x00020018 /* Address of the mix NCO smc register.  */
#define mix_NCO_amc       0x0002001A /* Address of the mix NCO amc register.  */
#define mix_NCO_pira_base 0x00020010 /* Address of the mix NCO pira register, */
                                     /*    LSB.                               */
#define mix_NCO_hop_clk   0x0002001E /* Address of the mix NCO asynchronous   */
                                     /*    hop clock register.                */

iNCO:    BIT SET ASTAT FLG1|FLG2;    /* Deselect all other peripherals.       */
```

```
BIT CLR ASTAT FLG0;          /* Select the NCO.                          */
DMWAIT=dm_ws_NCO;            /* Initialize the ADSP-21020 DM bus to      */
                             /*    interact with the NCO.                */

/* The NCO requires a longer address set up time than the   */
/* ADSP21020 can provide.  To compensate, a read is performed from   */
/* the pending write address before the write is performed.  This    */
/* places the pending write address on the bus a full clock cycle    */
/* before the write occurs.                                          */

R0=DM(clk_NCO_smc);          /* Set the sample clock NCO (#1) to         */
R0=0x00;                     /*    operate in simple oscillator mode     */
DM(clk_NCO_smc)=R0;          /*    by disabling all other modes.         */

R0=DM(clk_NCO_amc);          /* Initialize NCO #1 to output in offset */
R0=0x0E;                     /*    binary format and use 12-bit NRC.  */
DM(clk_NCO_amc)=R0;

R0=DM(mix_NCO_smc);          /* Set the mix NCO (#2) to operate in       */
R0=0x00;                     /*    simple oscillator mode by             */
DM(mix_NCO_smc)=R0;          /*    disabling all other modes.            */

R0=DM(mix_NCO_amc);          /* Initialize NCO #2 to output in offset */
R0=0x0E;                     /*    binary format and use 12-bit NRC.  */
DM(mix_NCO_amc)=R0;

BIT SET ASTAT FLG0;          /* Deselect the NCO.                        */
DMWAIT=default_ws;           /* Return the ADSP-21020 DM bus to the      */
                             /*    default configuration.                */

/* The DRX performs complex sampling of the 455 KHz signal using a   */
/* single ADC by sampling at a rate which satisfies the following    */
/* equation:                                                         */
/*            Fs=455 KHz / (N +/- 0.25) where N = integer            */
/*                                                                   */
/* Complex sample pairs are then formed by pairing time adjacent     */
/* samples. N should be selected so the resulting complex sampling   */
/* rate is at least equal to the bandwidth of the input signal.  The */
/* DRX samples at;                                                   */
/*                                                                   */
/*        Fs(real)=455 KHz / (1 - 0.25) = 606.67 KHz                 */
/*        Fs(complex)=Fs(real)/2=606.667 kHz/2=303.333 kHz.          */
/*                                                                   */
/* This sampling rate is adequate to characterize the 200 kHz        */
/* acquisition bandwidth provided by the APT IF hardware.            */
/*                                                                   */
/* The following code initializes the appropriate channel of the     */
/* NCO to produce a 606.67 KHz sinusoid.  This sinusoid is converted */
/* to a square wave by additional hardware and used to clock the ADC.*/

/* The NCO is programmed to generate a given frequency by writing a  */
/* value to the appropriate phase increment register (PIR) and then  */
/* writing any value to the asynchronous hop clock register.  The    */
/* value that must be written to the PIR register to generate        */
/* frequency Fg may be computed as:                                  */
/*                                                                   */
/*            delta phi (write to PIR) = (Fg * 2^32) / Fs            */
/*                                                                   */
/*            where Fs is the NCO reference frequency.               */

F0=455000.0/0.75;            /* Compute sample rate (Fg).                */
F11=2.0; F12=4294967296.0;   /* Compute delta phi.                       */
CALL divide (DB); F0=F0*F12; F12=NCO_ref;
R0=FIX F0;                   /* Convert delta phi to fixed point rep.*/
B1=clk_NCO_pira_base; L1=0;  /* Write the new information to the NCO.*/
B3=clk_NCO_hop_clk;  L3=0;   /*              .          .             */
CALL alterNCO;               /*              .          .             */
```

```
/*****************************************************************************/
/****                FIFO RELATED DEFINITIONS/INITIALIZATIONS                ****/
/*****************************************************************************/
#define r_fifo 0x10000                /* Fifo read address.                    */

/*****************************************************************************/
/*****************************************************************************/
/****                              MAIN PROGRAM                             ****/
/*****************************************************************************/
/*****************************************************************************/
/*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/
/*!!!                        Signal Acquisition Mode                        !!!*/
/*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/
/*===========================================================================*/
/* In acquisition mode the DRX attempts to determine the frequency of the    */
/* carrier signal.  The carrier signal is identified as the strongest signal */
/* in the a 200 kHz bandwidth about 455 kHz.  The DRX signal acquisition      */
/* algorithm is outlined below.                                              */
/*                                                                           */
/*  1) Switch IF filter to 200 KHz bandwidth.  (Note, the sampling rate is   */
/*     set to 303.333 KHz to ensure that the stop bands of this filter have  */
/*     rolled off significantly before aliasing occurs).                     */
/*  2) Adjust the mix frequency such that a 455 KHz signal would pass through */
/*     the center of the 200 KHz IF filter.  (Fmix=5 MHz-455 kHz=4.545 MHz).  */
/*  3) Accumulate 32K complex sample pairs.                                  */
/*  4) Perform 32K complex FFT.                                              */
/*  5) Convert resulting I,Q information to magnitude squared results.       */
/*  6) Locate and record DFT bin containing the most energy (largest         */
/*     magnitude #1).                                                        */
/*  7) Adjust mix frequency such that a 455 KHz + 4.5 Hz (1/2 bin width)     */
/*     signal would pass through the center of the 200 KHz IF filter.        */
/*     (Fmix=5 MHz-455 KHz-4.5 Hz=4.545 MHz-4.5 Hz).                         */
/*  8) Accumulate 32K complex sample pairs.                                  */
/*  9) Perform 32K complex FFT.                                              */
/* 10) Convert resulting I,Q information to magnitude squared results.       */
/* 11) Locate and record DFT bin containing the most energy (largest         */
/*     magnitude #2).                                                        */
/* 12) Determine which DFT bin contains the most energy (compare largest     */
/*     magnitude #1 and largest magnitude #2).                               */
/* 13) The bin with the most energy corresponds to the carrier because the   */
/*     carrier is the strongest signal in the bandwidth.  Compute an         */
/*     estimate of the carrier frequency based on the spectral location      */
/*     of the DFT bin containing the most energy.                            */
/* 14) Repeat steps 2 through 13 two more times in order to generate a       */
/*     total of three independent carrier frequency estimates.               */
/* 15) If all three carrier frequency estimates match, the DRX has acquired  */
/*     the carrier signal.  If the estimates do not match:                   */
/*     a) discard the oldest estimate.                                       */
/*     b) repeat steps 2 through 13 to generate another independent          */
/*        estimate.                                                          */
/*     c) repeat step 15.                                                    */
/*                                                                           */
/* Notes:                                                                    */
/* 1) Why is the spectrum shifted by 1/2 bin width and the FFT repeated?     */
/*                                                                           */
/*    Signal amplitude estimates derived from the results of DFTs are only  */
/*    approximate.  For example, if the signal lies exactly between two DFT  */
/*    points, both bins will contain half of the energy in the signal.  This */
/*    results in a 3 dB signal power measurement error.  Shifting the        */
/*    spectrum by 1/2 bin width and repeating the FFT process effectively    */
/*    halves the spectral distance between adjacent DFT points thereby       */
/*    yielding an improved spectral estimate.                                */
/*                                                                           */
/* 2) Why is the entire procedure repeated 3 times?                          */
/*                                                                           */
/*    It is crucial that the DRX reliably identify the carrier signal among  */
/*    noise and modulation tones.  Repeating the acquisition procedure       */
/*    multiple times reduces the chance of the DRX locking onto a spurious   */
```

```
/*    noise spike because noise spikes are not persistent and will therefore  */
/*    not show up in all three FFTs.                                           */
/*                                                                             */
/* If the DRX is polled for a signal power sample while it is attempting to    */
/* acquire the carrier signal, the DRX outputs a signal power measurement of   */
/* -71.92 dB.  The sample status tag will indicate that the DRX is in          */
/* acquisition mode and therefore unable to provide an accurate signal power   */
/* sample.                                                                     */
/*=============================================================================*/

/*-----------------------------------------------------------------------------*/
/* Consecutive correct carrier frequency estimates may not match precisely     */
/* because                                                                     */
/*    1) acquisition mode carrier frequency estimates are quantized to 4.5 Hz.*/
/*    2) the carrier frequency can drift during the acquisition process.       */
/*                                                                             */
/* For these reasons, if the carrier frequency estimates are within            */
/* <acq_freq_diff_thresh> of one another, the DRX will assume they match.      */
/*                          4.6 Hz       2 second      1.69 Hz/sec.           */
/* acq_freq_diff_thresh = quantization + acquisition * maximum carrier         */
/*                          noise          time         drift rate             */
/*                                                                             */
/*                    = 8.0 Hz.                                                */
/*-----------------------------------------------------------------------------*/
#define acq_freq_diff_thresh   8.0

/*-----------------------------------------------------------------------------*/
/* When the DRX is in acquisition mode, it is unable to provide accurate       */
/* signal power measurements.  If the DRX is polled for a power measurement    */
/* sample during this time, a signal power measurement value of -71.92 is      */
/* transmitted.  The DRX status tag will indicate that the DRX is in carrier   */
/* signal acquisition mode and consequently unable to generate the desired     */
/* measurement.  The following code places a signal power measurement value    */
/* of -71.92 with the appropriate DRX status tag into program memory           */
/* locations <sample_1Hz> and <sample_20Hz>.  (These memory locations          */
/* always contain the latest signal power measurement information).            */
/*-----------------------------------------------------------------------------*/
acq_mode:R0=8191;                    ,          /* <sample_1Hz> and <sample_20Hz>       */
         R1=status_acq_mode;                    /* memory locations always contain the  */
         R0=R0 OR FDEP R1 BY 26:3;              /* latest signal power measurement      */
         R1=PM(detection_bandwidth);            /* information.  The contents of the    */
         R0=R0 OR FDEP R1 BY 13:6;              /* appropriate memory location are      */
         R1=stype_1Hz;                          /* output whenever the DRX is polled    */
         R0=R0 OR FDEP R1 BY 29:3;              /* for a signal power measurement.      */
         PM(sample_1Hz)=R0;                     /* Place dummy -71.92 signal power      */
         R1=stype_20Hz;                         /* measurements with status tags        */
         R0=R0 OR FDEP R1 BY 29:3;              /* indicating the DRX is in carrier     */
         PM(sample_20Hz)=R0;                    /* signal acquisition mode in these     */
                                                /* memory locations.  Note, all other   */
                                                /* fields in the output word are        */
                                                /* also initialized with nominal values.*/

/*-----------------------------------------------------------------------------*/
/* The <current_fc> memory location always contains the latest carrier         */
/* frequency estimate.  Carrier frequency estimates cannot be generated        */
/* until the receiver acquires the carrier.  Initialize the <current_fc>       */
/* memory location to 0 to indicate no carrier frequency estimates are         */
/* available.                                                                  */
/*-----------------------------------------------------------------------------*/
         F0=0.0;
         PM(current_fc)=F0;

/*-----------------------------------------------------------------------------*/
/* The user may force the DRX to enter carrier signal acquisition mode by      */
/* sending the "force acquisition" command.  The contents of the <force_acq>   */
/* memory location indicate whether the "force acquisition" command has        */
/* been issued.                                                                */
/*-----------------------------------------------------------------------------*/
```

```
        R0=FALSE;
        PM(force_acq)=R0;

/*-------------------------------------------------------------------------*/
/* The DRX slews the mix output frequency by changing the NCO output       */
/* frequency by 1 NCO tick (NCO tick=NCO reference frequency/2^32) at      */
/* a 1.011 kHz rate.  The mix output frequency can be slewed up at a       */
/* rate of 3.73 Hz/second by setting <mix_NCO_delta> to +1.  The mix        */
/* output frequency can be slewed down at the same rate by setting         */
/* <mix_NCO_delta> to -1.  <mix_NCO_delta> should be set to 0 for no slew. */
/*-------------------------------------------------------------------------*/
        PM(mix_NCO_delta)=R0;

/*-------------------------------------------------------------------------*/
/* Set the DRX_mode flag to indicate that the receiver is in signal        */
/* acquisition mode.                                                       */
/*-------------------------------------------------------------------------*/
        R0=mode_acquisition; PM(DRX_mode)=R0;

/*-------------------------------------------------------------------------*/
/* There are two analog bandpass filters in front of the DRX (on the IF    */
/* board): one filter is 200 KHz wide, the other is 10 KHz wide. During    */
/* acquisition the 200 KHz filter is used.  Select the 200 KHz filter by   */
/* sending the appropriate command to the host and waiting for the host    */
/* to send an acknowledge indicating that the desired filter has been      */
/* selected.  (Acquisition algorithm step 1).                              */
/*-------------------------------------------------------------------------*/
        R0=wide_filter;         /* Send the "select 200 kHz filter"        */
        CALL send_cmd;          /*   command.                              */
        CALL acknow;            /* Wait for the host to acknowledge the    */
                                /*   command.                              */

/*-------------------------------------------------------------------------*/
/* Carrier frequency estimates are stored in a circular buffer of length 3 */
/* (i.e. the most recent 3 carrier frequency estimates are stored in the   */
/* buffer).  The ADSP21020 automatically performs all of the calculations  */
/* necessary to maintain the buffer.  The following code initializes address */
/* registers 8 and 9 to access the circular buffer and fills the buffer with */
/* zeros.  Address register 8 is used to enter new estimates into the buffer. */
/* Address register 9 is used to read entries from the buffer to determine */
/* whether all carrier frequency estimates in the buffer match.            */
/*                                                                         */
/* The acquisition code below adds a carrier frequency estimate to the buffer */
/* each time algorithm steps 2 through 13 are executed.  Each time an entry */
/* is added to the buffer, the contents of the buffer are checked to see if */
/* all carrier frequency estimates match.  The buffer is initialized with  */
/* zeros so that after one carrier frequency estimate is added to the buffer, */
/* all buffer entries cannot match until at least two more estimates are   */
/* entered (the receiver will not generate carrier frequency estimates of  */
/* 0 Hz).                                                                  */
/*-------------------------------------------------------------------------*/
        B8=fc_est; L8=3;                /* Define the starting address and */
        B9=fc_est; L9=3;                /* length of the buffer for address */
                                        /* registers 8 and 9.              */
        R0=0;
        LCNTR=3, DO init_acq UNTIL LCE; /* Fill the buffer with zeros.     */
init_acq:   PM(I8,1)=R0;

/*-------------------------------------------------------------------------*/
/* The following code implements the DRX signal acquisition algorithm      */
/* described above until DRX acquires the carrier.                         */
/*-------------------------------------------------------------------------*/
                        /* Adjust the mix frequency such that a 455 KHz    */
                        /* signal would pass through the center of the     */
                        /* 200 KHz IF filter.  (Acquisition algorithm      */
                        /* step 2).                .       .               */
rept_acq:F0=4545000.0;                          /*      .       .          */
        F11=2.0; F12=4294967296.0;              /*      .       .          */
```

```
CALL divide (DB); F0=F0*F12; F12=NCO_ref;   /*       .        .      */
R0=FIX F0;                                  /*       .        .      */
B1=mix_NCO_pira_base; L1=0;      /* The procedure to alter the       */
B3=mix_NCO_hop_clk;  L3=0;       /* frequency output by the NCO is   */
CALL alterNCO;                   /* explained in detail in the NCO   */
                                 /* initialization section above.    */

F1=455000.0/0.75/2.0;  /* Compute the frequency which will be        */
PM(freq_base)=F1;      /* represented by the first DFT point (#0)    */
                       /* computed in the 32K complex FFT computed   */
                       /* below.                                     */

CALL acq_fft;     /* The acq_fft subroutine acquires 32K complex     */
PM(I8,0)=R0;      /* sample points, performs a 32K complex FFT,      */
PM(temp2)=F1;     /* converts the resulting I,Q information to       */
                  /* magnitude information, and returns the location */
                  /* of the DFT bin containing the greatest amount   */
                  /* of energy (in R0) and the energy level (in F1). */
                  /* (Acquisition algorithm steps 3,4,5 and 6).      */


                  /* Adjust the mix frequency such that a            */
                  /* 455 KHz + 4.5 Hz (1/2 bin width) signal would   */
                  /* pass through the center of the 200 KHz IF filter */
                  /* (Acquisition algorithm step 7).   .       .     */
                                                 /*       .       .   */
F0=4545000.0+455000.0/0.75/2.0/32768.0/2.0;      /*       .       .   */
F11=2.0; F12=4294967296.0;                       /*       .       .   */
CALL divide (DB); F0=F0*F12; F12=NCO_ref;        /*       .       .   */
R0=FIX F0;                                       /*       .       .   */
B1=mix_NCO_pira_base; L1=0;      /* The procedure to alter the      */
B3=mix_NCO_hop_clk;  L3=0;       /* frequency output by the NCO is  */
CALL alterNCO;                   /* explained in detail in the NCO  */
                                 /* initialization section above.   */

                /* Compute the frequency which will be represent    */
                /* by the first DFT point (#0) computed in the      */
              , /* 32K complex FFT of the shifted spectrum computed*/
                /* below.                                           */
F1=455000.0/0.75/2.0+455000.0/0.75/2.0/32768.0/2.0;
PM(freq_base)=F1;

CALL acq_fft;     /* Again, the acq_fft subroutine acquires 32K     */
                  /* complex sample points, performs a 32K complex  */
                  /* FFT, converts the resulting I,Q information to */
                  /* magnitude information, and returns the location */
                  /* of the DFT bin containing the greatest amount  */
                  /* of energy (in R0) and the energy level (in F1). */
                  /* This time the FFT operation is performed on the */
                  /* shifted spectrum.  (Acquisition algorithm steps */
                  /* 8,9,10 and 11).                                */

F2=PM(temp2);     /* Compare the DFT bin maximum signal levels from */
COMP(F2,F1);      /* the FFT of the "normal" and shifted spectra.   */
IF GT F0=PM(I8,0);/* Record the carrier frequency estimate associated*/
PM(I8,1)=F0;      /* with the DFT bin which contains the most energy.*/
                  /* (Acquisition algorithm steps 12 and 13).       */

                  /* Check to see if the last three (independent)   */
                  /* carrier frequency estimates are within         */
                  /* "acq_freq_diff_thresh" Hz of one another.  If  */
                  /* not, repeat acquisition algorithm steps 2      */
                  /* through 13 above to generate another independent*/
                  /* carrier frequency estimate and then repeat this */
                  /* comparison step.  If all three carrier frequency*/
F3=acq_freq_diff_thresh;       /* estimates match, the DRX has      */
LCNTR=3,DO chk_freq UNTIL LCE; /* acquired the carrier signal.      */
   F1=PASS F0,F0=PM(I9,1);     /* Continue to the steps below.      */
```

```
            F2=ABS(F0-F1);              /* (This is acquisition algorithm    */
            COMP(F2,F3);                /* step 15).         .         .     */
chk_freq:   IF GT JUMP rept_acq (LA);   /*         .         .         .       */

                            /* Copy the latest carrier frequency estimate to  */
          MODIFY(I8,2);     /* program memory location <current_fc>.  The     */
          F0=PM(I8,0);      /* signal power measurement algorithm will begin  */
          PM(current_fc)=F0; /* tracking the carrier from this frequency.     */

/*-------------------------------------------------------------------------*/
/* Compute the mix frequency required to translate the carrier signal to   */
/* 455 KHz.                                                                */
/*              mix frequency=(5 MHz-455 KHz)-(fc-455 KHz)                 */
/*                                                                         */
/* Update the NCO to generate the required mix frequency.  The procedure to*/
/* alter the frequency output by the NCO is explained in detail in the NCO */
/* initialization section above.                                           */
/*-------------------------------------------------------------------------*/
          F1=455000.0;              /* Compute the mix frequency required  */
          F0=F0-F1;                 /* to translate the carrier signal to  */
          F1=4545000.0;             /* 455 KHz.                            */
          F0=F0+F1;
                                    /* Update the NCO to generate the      */
          B1=mix_NCO_pira_base;     /* required mix frequency.  The        */
          L1=0;                     /* procedure to alter the NCO output   */
          B3=mix_NCO_hop_clk;       /* frequency is described in the NCO   */
          L3=0;                     /* initialization section above.       */
          F11=2.0; F12=4294967296.0;
          CALL divide (DB); F0=F0*F12; F12=NCO_ref;
          R0=FIX F0;
          PM(mix_NCO_dphi)=R0;
          CALL alterNCO;

          R0=default_spectrum_size; /* Send a spectrum (default size) to   */
          PM(spectrum_size)=R0;     /* the DACS.                           */
          R0=TRUE;
          PM(generate_spectrum)=R0;

          PM(tracking)=R0;          /* Enable carrier frequency tracking.  */

/*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/
/*!!!                    Signal Power Measurement Mode                  !!!*/
/*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/
/*=========================================================================*/
/* In signal power measurement mode the DRX measures the amount of power in*/
/* the carrier signal.  The signal power measurement algorithm is described*/
/* below.                                                                  */
/*                                                                         */
/*   1) Update the mix frequency such that the carrier signal is translated*/
/*      to 455 KHz.                                                        */
/*   2) Utilize the available 10 KHz filter (455 KHz +/- 5 KHz) on the IF   */
/*      board to obtain better SNR at the ADC input.                       */
/*   3) Filter the 303.333 kHz complex sample stream with a comb filter    */
/*      which has a notch at 0 and fs and is flat at fs/2 in order to      */
/*      remove any dc offset at the output of the ADC op-amp.              */
/*   4) Decimate the 303.333 KHz complex sample stream to 25.278 KHz.      */
/*      (Recall, the DRX must sample at a complex rate greater than or equal*/
/*      to the bandwidth of the signal (10 kHz).  Here, the DRX samples at */
/*      25.278 KHz and can therefore characterize a 25.278 KHz bandwidth.  */
/*      The larger bandwidth is used to ensure the stopbands of the 10 kHz */
/*      filter have rolled off significantly before aliasing occurs).      */
/*   5) Use 2 FIR filter/decimate stages to further band limit and decimate*/
/*      the complex sample stream to 1.011 kHZ.                            */
/*   6) Perform a 1K FFT on a sliding window of 1.011 KHz complex data.  An*/
/*      FFT is performed whenever 25 new samples have accrued (approximately*/
/*      40 Hz).                                                            */
/*   7) Convert the FFT results to power magnitude results.                */
/*   8) Pass the magnitude results of the transforms through a frequency   */
```

```
/*      domain bandpass filter of user selectable width.  This filter is      */
/*      implemented by summing the contents of DFT bins surrounding the       */
/*      carrier's center frequency (DFT bin containing the most energy).      */
/*   9) Derive an updated carrier frequency estimate by examining the location*/
/*      of the spectral peak representing the carrier in the FFT results.     */
/*  10) If the carrier has drifted more than 25 Hz from 455 kHz, begin to      */
/*      slew the mix output frequency to drive the carrier signal back        */
/*      toward 455 KHz.  If the carrier signal is within 10 Hz of 455 kHz,    */
/*      stop any slew of the mix frequency.                                   */
/*                                                                            */
/*      In signal power measurement mode, the DRX updates the mix frequency   */
/*      output 1011 times per second.  To slew the mix frequency the DRX      */
/*      alters dphi by +/- 1 NCO tick (0.00372 Hz) each time the NCO is       */
/*      updated.  Hence, the mix frequency is slewed at a rate of             */
/*      3.72 Hz/second.                                                       */
/*  11) Compute the variance of the difference between the last 40 carrier    */
/*      frequency estimates (the DRX generates 40 carrier frequency estimates */
/*      per second).  If the result is greater than 2.0, the receiver is not  */
/*      able to reliably identify the carrier signal; increment the low_SNR   */
/*      counter and set the low_SNR flag.  If the result is less than 2.0,    */
/*      clear the low_SNR counter and the low_SNR flag.                       */
/*  12) If the DRX is unable to identify the carrier signal for more than     */
/*                                                                            */
/*                    +/- FIR                                                 */
/*                    passband                                                */
/*                     width        50 Hz                                     */
/*                   ---------- = --------- = 29.6 sec.                       */
/*                    maximum          Hz                                    */
/*                    carrier     1.69 ----                                  */
/*                   drift rate        sec.                                  */
/*                                                                            */
/*      the carrier could drift out of the FIR filter passband.  Hence,       */
/*      the DRX re-enters carrier signal acquisition mode.  If the low_SNR    */
/*      counter > 40*29.6, jump to acq_mode.                                  */
/*  13) Go to step 6 and wait for more complex samples to accrue.             */
/*============================================================================*/
/*----------------------------------------------------------------------------*/
/* Set the DRX_mode flag to indicate that the receiver is in signal power     */
/* measurement mode.                                                          */
/*----------------------------------------------------------------------------*/
op_mode: R0=mode_power_meas; PM(DRX_mode)=R0;

/*----------------------------------------------------------------------------*/
/* Initialize the FIR filters which bandlimit the input signal to 1.011 KHz.  */
/* The memory for each filter is configured as a circular buffer.  The        */
/* ADSP21020 performs all of the indexing computations required to maintain   */
/* these buffers.  The following code initializes the buffers by identifying  */
/* their start addresses and lengths.  The status of each buffer (i.e.        */
/* next slot pointer) is then stored in memory because all filter routines    */
/* use the same address registers.  Hence, to update a filter, the status of  */
/* the filter buffer must be retrieved from memory, the filter operations     */
/* performed, and the buffer status written back to memory.                   */
/*----------------------------------------------------------------------------*/
#define len_25K  22                     /* Length of the filter which         */
                                        /*    bandlimits the 25.278 KHz       */
                                        /*    complex data to 5.506 KHz.      */

                                        /* Initialize the 25.278 KHz filter   */
                                        /*    buffer and store buffer status  */
        B7=r25K_dat; PM(r25K_I)=I7;     /*    (i.e. next slot pointer) in     */
        B13=i25K_dat; PM(i25K_I)=I13;   /*    program memory.                 */

#define len_5K   30                     /* Length of the filter which         */
                                        /*    bandlimits the 5.056 KHz        */
                                        /*    complex data to 1.011 KHz.      */

                                        /* Initialize the 5.056 KHz filter    */
                                        /*    buffer and store buffer status  */
```

```
        B7=r5K_dat;  PM(r5K_I)=I7;     /*     (i.e. next slot pointer) in      */
        B13=i5K_dat;  PM(i5K_I)=I13;  /*     program memory.                    */

/*-------------------------------------------------------------------------*/
/* Power measurement information and tracking information are extracted from  */
/* the results of 1K complex FFTs performed on a sliding window of 1.011 KHz  */
/* complex data.  The latest 1024 complex sample points are stored in a       */
/* circular buffer.  The ADSP21020 performs all of the indexing calculations  */
/* required to maintain the buffer.  The following code initializes the buffer*/
/* by defining its start address and storing it in program memory.            */
/*-------------------------------------------------------------------------*/
        B7=trkfft;  PM(trkfft_I)=I7;

/*-------------------------------------------------------------------------*/
/* 1 Hz power measurement samples must be bandlimited to 0.5 Hz.  20 Hz power */
/* measurement samples must be bandlimited to 10 Hz.  2 30 dB/octave windows  */
/* are used to bandlimit the signal power measurements.  The window used to   */
/* bandlimit 40 Hz signal power measurements to 10 Hz is 13 points long.  The */
/* window used to bandlimit 40 Hz signal power measurement to 0.5 Hz is 140   */
/* points long.  Note, the windows were not cascaded in order to minimize     */
/* delay through the system.  Window memory is implemented in a circular      */
/* buffer.  The ADSP-21020 performs all of the indexing calculations required */
/* to maintain the buffer.  The following code initializes the buffer by      */
/* defining its start address and storing it in program memory.  (Recall, all */
/* filters use the same address registers so the status of each must be       */
/* recorded in memory)                                                        */
/*-------------------------------------------------------------------------*/
        B9=dB30octave_20_dat;  PM(dB30octave_20_I)=I9;
        B9=dB30octave_1_dat;  PM(dB30octave_1_I)=I9;

/*-------------------------------------------------------------------------*/
/* Define some miscellaneous address modifier constants.  Having these        */
/* constants available in registers permits the use of more multifunction     */
/* instructions which yields more efficient, compact code.                    */
/*-------------------------------------------------------------------------*/
        M7=1; M6=-1; M13=1; M12=-1; M5=0; M11=0;

/*-------------------------------------------------------------------------*/
/* As the IF signal is passed through the aforementioned FIR filters, the     */
/* bandwidth of the signal is reduced.  Hence the sampling rate may be        */
/* decreased without loss of information.  The sampling rate is decreased     */
/* by decimating the sample stream.  Initialize the counter variables which   */
/* control this decimation.                                                   */
/*-------------------------------------------------------------------------*/
        R0=0;
        PM(s5K)=R0;              /*5.056 KHz to 1.011 KHz decimation counter.*/
        PM(s20)=R0;              /*20.22 Hz to 1.011 Hz decimation counter.  */
        PM(s1)=R0;

/*-------------------------------------------------------------------------*/
/* As described in the signal power measurement algorithm (step 10), the mix  */
/* frequency should not be significantly altered instantaneously.  Rather,    */
/* the DRX should slew the mix frequency from one value to another when the   */
/* mix frequency must be altered.  Hence, the DRX writes new frequency        */
/* information (delta phi) to the NCO at a rate of 1 KHz.  If the DRX does    */
/* not need to update the NCO, the same delta phi value is written to the     */
/* NCO repeatedly.  When the DRX does need to alter the mix frequency, it     */
/* does so by changing delta phi by 1 increment (=0.00372 Hz) each time it    */
/* writes to the NCO.  Hence, the NCO output may be slewed as fast as         */
/* 3.72 Hz/sec.  Initialize the delta phi increment to 0 because the mix      */
/* frequency does not need to be altered.                                     */
/*-------------------------------------------------------------------------*/
        PM(mix_NCO_delta)=R0;

/*-------------------------------------------------------------------------*/
/* Clear the fifo of all old samples.                                         */
/*-------------------------------------------------------------------------*/
        BIT SET ASTAT FLG0|FLG1;
```

```
        BIT CLR ASTAT FLG2;
        LCNTR=2048,DO clrfifo2 UNTIL LCE;
clrfifo2:   R0=DM(r_fifo);
        BIT SET ASTAT FLG2;

/*------------------------------------------------------------------------*/
/* The fifo generates an interrupt pulse when it is 7/8ths full.  The DRX */
/* must respond to this interrupt by reading ADC samples from the fifo before */
/* the fifo overflows.  Enable the "fifo almost full" interrupt.          */
/*------------------------------------------------------------------------*/
        BIT SET IMASK IRQ0I;

/*------------------------------------------------------------------------*/
/* The DRX computes the variance of the differences between consecutive   */
/* carrier frequency estimates in order to determine its ability to reliably */
/* identify the carrier signal.  <old_fc> contains the most recent carrier */
/* frequency estimate.  Initialize <old_fc> to -1 to indicate that no     */
/* carrier frequency estimates have been generated yet.                   */
/*------------------------------------------------------------------------*/
        F0=-1.0;
        PM(old_fc)=F0;

/*------------------------------------------------------------------------*/
/* To calculate the variance of the past 40 carrier frequency estimates, the */
/* DRX must calculate the average of these estimates.  Hence, the past    */
/* 40 carrier frequency estimates are stored in a circular buffer.  The   */
/* following code initializes this buffer with 0s.                        */
/*------------------------------------------------------------------------*/
        B8=delta_fc_avg; PM(delta_fc_I)=I8; L8=40;
        F0=0.0;
        LCNTR=40, DO ideltas UNTIL LCE;
ideltas:   PM(I8,1)=F0;

/*------------------------------------------------------------------------*/
/* <low_SNR_counter> is used to count the amount of time the DRX is unable */
/* to reliably identify the carrier signal.  If <low_SNR_counter> is greater */
/* than 29.6 seconds*40 carrier frequency estimates/second=1184, the DRX will */
/* re-enter carrier signal acquisition mode.                              */
/*------------------------------------------------------------------------*/
        R0=0; PM(low_SNR_counter)=R0;

/*------------------------------------------------------------------------*/
/* The following code implements the signal power measurement algorithm   */
/* described above.                                                       */
/*------------------------------------------------------------------------*/
wait_opm:BIT CLR ASTAT FLG3;     /* Turn off the LED connected to FLAG3 to  */
                                 /* indicate the receiver is idle while     */
                                 /* it waits for ADC samples to accrue.     */

        R0=PM(generate_spectrum);       /* Has the host issued a "generate  */
        R1=TRUE;                        /* spectrum" command?  If so, call  */
        COMP(R0,R1);                    /* the routine which generates      */
        IF EQ CALL spectrum;            /* spectra.  If not, continue       */
                                        /* executing the signal power       */
                                        /* measurement algorithm.           */

        R0=PM(force_acq);               /* Has the host issued a "force     */
        R1=TRUE;                        /* acquisition" command?  If so,    */
        COMP(R0,R1);                    /* jump to the carrier signal       */
        IF EQ JUMP acq_mode;            /* acquisition algorithm.  If not,  */
                                        /* continue executing the signal    */
                                        /* power measurement algorithm.     */

                                 /* 1K FFTs are performed at a rate of 40 Hz */
                                 /* on a sliding window of the complex data. */
                                 /* An FFT is performed when 25 new samples  */
                                 /* enter the sliding window.  If 25 new     */
                                 /* samples have accrued since the last FFT  */
```

```
                              /* was performed, do another FFT (below).    */
                              /* If not, continue to wait.  (Note, when    */
                              /* the DRX initially enters operational      */
                              /* mode, <sfft> is initialized to            */
                              /*     -(1024+sum of FIR filter lengths)     */
        R0=PM(sfft); R1=25;   /* so that new signal power measurements     */
        COMP(R0,R1);          /* are not output until the FIR filter/FFT   */
        IF NE JUMP wait_opm;  /* pipeline has filled up.                   */

                              /* Turn on the LED connected to FLAG3 to     */
                              /* indicate the DRX is performing            */
        BIT SET ASTAT FLG3;   /* calculations.                             */

        R0=0; PM(sfft)=R0;    /* Clear the number of new complex samples   */
                              /* in the sliding window to zero.            */

        /* The following code applies a Hanning window to the time domain   */
        /* data as it is copied to scratch memory in bit reversed order.    */

                              /* Retrieve index of oldest sample in the    */
                              /* sliding window.  (Real components are     */
                              /* stored in the circular buffer identified  */
                              /* by the retrieved information.  The complex*/
                              /* counterpart of each real sample is stored */
                              /* in circular buffer offset from the first  */
                              /* by 1024 samples.                          */
bitrvs: B1=trkfft; I1=PM(trkfft_I); L1=2048;
        MODIFY(I1,-1024);
                              /* Initialize an address register to the     */
                              /* scratch memory which will contain the     */
        B2=scratch; L2=1024;  /* real data components.                     */

                              /* Initialize an address register to the     */
                              /* scratch memory which will contain the     */
        B3=scratch+1024;      /* complex data components.                  */
        L3=1024;

        B11=hanning; L11=1024; /* Initialize an address register to the    */
                              /* Hanning window indices.                   */

        R0=0;                 /* R0 counts the number of complex data      */
                              /* points processed.  Bit reversed address   */
                              /* locations are computed as                 */
                              /*    address=base addr+[BITREV(R0) >> 22].  */
                              /* The bit reversed offsets must be shifted  */
                              /* because the BITREV command bit reverses   */
                              /* the entire 32-bit address.  We want to bit*/
                              /* reverse only the 10 LSBs.                 */
                              /*   ex: we want to bit reverse the offset   */
                              /*       0x001 to obtain 0x800.  BITREV      */
                              /*       operates on 32 bit data:            */
                              /*       BITREV(0x00000001b)=0x10000000.     */
                              /*       Hence, we must shift right by       */
                              /*       32-log2(1024)=22 bits.              */

                              /* Loop through all 1024 complex data points */
                              /* in the sliding window.                    */
        LCNTR=1024, DO br UNTIL LCE;
           F5=PM(I11,1);      /* Retrieve the next Hanning window index.   */

           F1=DM(2048,I1);    /* Retrieve the complex component of the     */
                              /*    next sample.                           */
           F1=F1*F5;          /* Apply the Hanning window index to the     */
                              /*    complex sample component.              */
           F2=DM(I1,1);       /* Retrieve the real component of the next   */
                              /*    sample.                                */
           F2=F2*F5;          /* Apply the Hanning window index to the     */
                              /*    real sample component.                 */
```

```
                                /* Compute the bit reversed offset into the  */
                                /*    scratch memory                         */
            I4=R0;              /*    1) bit reverse the offset              */
            BITREV(I4,0);       /*            .       .                      */
            R3=I4;              /*            .       .                      */
            R3=LSHIFT R3 BY -22; /*    2) shift the result and store it in    */
            M3=R3;              /*       an address modifier register.       */
            DM(M3,I2)=F2;       /* Store the complex component.              */
            DM(M3,I3)=F1;       /* Store the real component.                 */
br:         R0=R0+1;            /* Increment the offset (number of complex   */
                                /*    samples processed.                     */

dofft:    LCNTR=10;             /* Compute a 1024-point FFT on the new data. */
          B12=pm_vars; L12=0;
          R6=1024;
          B0=scratch; L0=0;
          B1=scratch+1024; L1=0;
          CALL fft_dm;
                                /* Convert the FFT I,Q results to magnitude  */
                                /* results and locate the bin containing the */
                                /* most energy.                              */
          B1=scratch; L1=0;             /* Initialize an address register to */
                                        /*    the real component of the first*/
                                        /*    DFT point.                     */
          F2=-1.0;                      /* F2 records the maximum power found*/
                                        /*    in a DFT bin thus far.         */
                                        /*    Initialize F2 to contain a     */
                                        /*    number < 0.  All DFT bin power */
                                        /*    values are > 0 (sum of squares)*/
                                        /*    Hence, F2 contains a value     */
                                        /*    lower than the maximum bin     */
                                        /*    power.  The ensuing comparisons*/
                                        /*    will accurately identify the   */
                                        /*    DFT bin containing the most    */
                                        /*    power.                         */
          R3=0;       R6=0;             /* R3 counts the number of DFT bins  */
                                        /*    evaluated thus far (R3         */
                                        /*    contains the number of the     */
                                        /*    DFT bin currently being        */
                                        /*    evaluated).                    */

          LCNTR=1024,DO fm_sn UNTIL LCE;  /*Loop through all 1024 DFT bins.    */

             F0=DM(1024,I1);            /* Retrieve the complex component of */
                                        /*    the current DFT bin (Q).       */
             F0=F0*F0,F1=DM(I1,0);      /* Compute F0 = Q^2 and retrieve the */
                                        /*    real component of the current  */
                                        /*    DFT bin.                       */
             F1=F1*F1;                  /* Compute F1 = I^2.                 */
             F0=F0+F1;                  /* Compute F0 = I^2 + Q^2.           */

             DM(I1,1)=F0;               /* Store the result back in scratch  */
                                        /* memory for later use.             */

             COMP(F0,F2);               /* Is the power in the current bin   */
                                        /*    greater than the maximum power */
                                        /*    in any bin thus far?           */
             IF LE JUMP fm_sn;          /*    * if not, jump to increment the*/
                                        /*      bin count.                   */
             R6=R3;                     /*    * if so, record the bin number */
             F2=F0;                     /*      and the power value.         */


fm_sn:       R3=R3+1;                   /* Increment the number of bins      */
                                        /*    processed.                     */

          /*-------------------------------------------------------------------*/
```

```
        /* The carrier is represented by the DFT bin with the most energy    */
        /* the carrier signal is the strongest signal in bandwidth of        */
        /* interest.  Compute the current carrier frequency based on the     */
        /* spectral location of the this DFT bin:                            */
        /*           /                                                       */
        /*          | 4.545 MHz-fmix+455 kHz+1011.111*max_bin#/1024,         */
        /*          |                                        max_bin#<513    */
        /*       fc=|                                                        */
        /*          | 4.545 MHz-fmix+455 kHz+1011.111*(max_bin#-1024)/1024,  */
        /*          |                                        max_bin#>513    */
        /*           \                                                       */
        /*                                                                   */
        /* Note: 455 kHz aliases to DFT bin #0 at a complex sampling rate    */
        /* of 1011.111 kHz:                                                  */
        /*                                                                   */
        /*                  DFT             fc=455 kHz                       */
        /*                  bin  =  1024 * ------------ % 1.0 = 0            */
        /*                 index           fsc=1011.111                      */
        /*                                                                   */
        /* Hence, frequencies slightly greater than 455 kHz are represented  */
        /* by bins 1..512.  Frequencies slightly less than 455 kHz are       */
        /* represented by bins 513..1024.                                    */
        /*-------------------------------------------------------------------*/
        PM(temp2)=R6;         /* R6 contains the index of the DFT bin with   */
                              /*   the most energy.  Store this value for    */
                              /*   later use.                                */

                              /*-------------------------------------------*/
                              /* Compute the spectral location of the max DFT */
                              /* bin.                                      */
                              /*-------------------------------------------*/
        R1=512;                           /* Is the max DFT bin index<512?   */
        COMP(R6,R1);                      /* If not, subtract 1024 from the  */
        IF LE JUMP g512;                  /* index.                          */
        R1=1024;
g512:   F6=float R6;                      /* F6=(corrected)max bin#*1011.111 */
        F0=1011.111;
        F0=F0*F6; F12=1024.0;             /*    (corrected)   1011.111       */
        CALL divide (DB); F11=2.0; L11=0; /* F0=    max     * -------- = F2  */
        F2=F0;                            /*       bin #       1024.0       */

                              /*-------------------------------------------*/
                              /* Compute the current mix output frequency. */
                              /*                                           */
                              /*            NCO reference frequency        */
                              /*       fmix=----------------------- * dphi  */
                              /*                     2^32                  */
                              /*-------------------------------------------*/
        F0=NCO_ref;
        F12=4294967296.0;
        CALL divide (DB); F11=2.0; L11=0; /* F0=NCO_ref/2^32.                */
        R1=PM(mix_NCO_dphi);              /* F1=dphi.                        */
        F1=float R1;
        F0=F0*F1;                         /* F0=mix frequency output.        */

                              /*-------------------------------------------*/
                              /* Compute the current carrier frequency.    */
                              /*-------------------------------------------*/
        F1=4545000.0;
        F0=F0-F1;             /* F0=4.545 MHz-fmix.                          */
        F1=455000.0;
        F0=F0+F1;             /* F0=4.545 MHz-fmix+455 kHz.                  */
        F0=F0+F2;             /* F0=current carrier frequency.               */
        PM(current_fc)=F0;

        /*-------------------------------------------------------------------*/
        /* Compute the variance of the differences between the past 40       */
        /* consecutive carrier frequency estimates in order to determine     */
```

```
        /* whether the DRX is reliably identifying the carrier frequency.    */
        /*-------------------------------------------------------------------*/
        F1=-1.0;            /* If the current carrier frequency estimate is  */
        F2=PM(old_fc);      /* the first estimate to be calculated, set the  */
        PM(old_fc)=F0;      /* past carrier frequency estimate variable to 0.*/
        COMP(F1,F2);        /* This is done so the difference between the    */
        IF EQ F2=F0;        /* "latest two" carrier frequency estimates is 0.*/
                            /* which does not artificially drive up the      */
                            /* variance.                                     */
        F1=F2-F0;                                   /* F1=new delta fc.      */

                            /*-----------------------------------------------*/
                            /* Compute the average of the differences between*/
                            /* the past 40 carrier frequency estimates.      */
                            /*-----------------------------------------------*/
        B8=delta_fc_avg;                  /* Initialize address register #8 to */
        I8=PM(delta_fc_I);                /*    the circular buffer containing */
        L8=40;                            /*    the differences between the 40 */
                                          /*    most recent fc estimates.      */
        PM(I8,1)=F1;                      /* Store the new delta fc value over */
                                          /*    the oldest delta in the buffer.*/
        PM(delta_fc_I)=I8;                /* Store the new buffer status.      */

        F0=0.0;                           /* Compute the sum of the deltas in  */
        LCNTR=40; DO cdelta UNTIL LCE; /*    the buffer (F0).                  */
           F2=PM(I8,1);
cdelta:      F0=F0+F2;
                                          /* F0=average.                       */
        CALL divide (DB); F12=40.0; F11=2.0;

                            /*-----------------------------------------------*/
                            /* Compute the variance of the buffer contents.  */
                            /*                                               */
                            /*               SUM{ delta(i)-average }^2       */
                            /*   variance = ------------------------ = F2.    */
                            /*                        40                     */
                            /*-----------------------------------------------*/
        F2=0.0;         ,
        LCNTR=20,DO compvar UNTIL LCE;
           F1=PM(I8,1);
           F1=F0-F1;
           F1=F1*F1;
compvar:   F2=F2+F1;

        F0=F2;
        CALL divide (DB); F12=19.0; F11=2.0;
        PM(variance)=F0;

                            /*-----------------------------------------------*/
                            /* If the DRX is able to reliably identify the   */
                            /* carrier signal, the variance of the past 40   */
                            /* consecutive carrier frequency estimates will be*/
                            /* small (<2.0) because the carrier drifts       */
                            /* relatively slowly.  If the DRX cannot reliably*/
                            /* identify the carrier signal, the variance     */
                            /* will be large because the receiver will attempt*/
                            /* to track the white noise floor.  A variance   */
                            /* value of 2.0 has been empirically determined  */
                            /* as the in-lock/out-of-lock threshold.  If the */
                            /* receiver is locked to the carrier signal,     */
                            /* jump to tracking algorithm code.  If the DRX  */
                            /* is not locked to the signal, make sure the    */
                            /* DRX is not slewing the mix output frequency,  */
                            /* set the low_SNR status flag and increment the */
                            /* low_SNR_counter.  If the DRX has been         */
                            /* out-of-lock for more than 29.6 seconds, jump  */
                            /* to carrier signal acquisition mode.           */
                            /*-----------------------------------------------*/
```

```
        F1=2.0;                         /* If the new variance is less than   */
        COMP(F0,F1);                    /* 2.0 jump to the carrier tracking   */
        IF LE JUMP trackit;             /* algorithm.                         */

        R0=0;                           /* If the DRX is out-of-lock, stop    */
        PM(mix_NCO_delta)=R0;           /* any slew of the mix output         */
                                        /* frequency.                         */

        F0=status_low_SNR;              /* Set the low_SNR status flag.       */
        PM(DRX_status)=F0;

        R0=PM(low_SNR_counter);         /* Increment the low_SNR_counter.     */
        R0=R0+1;
        PM(low_SNR_counter)=R0;
        R1=1184; /* =40*29.6 */         /* If the DRX has been out-of-lock    */
        COMP(R0,R1);                    /* more than 29.6 seconds, jump to    */
        IF GT JUMP acq_mode;            /* carrier signal acquisition mode.   */

        JUMP sumit;                     /* If the DRX has been out-of-lock    */
                                        /* for less than 29.6 seconds,        */
                                        /* continue to execute the signal     */
                                        /* power measurement algorithm except */
                                        /* for the tracking stage.            */

        /*-------------------------------------------------------------------*/
        /* The following code tracks the carrier drift according to:         */
        /*                                                                   */
        /*     dNCO=    dNCO=      dNCO=      dNCO=      dNCO=      dNCO=    */
        /*      -1      dNCO         0          0        dNCO        +1      */
        /*       |        |          |          |          |          |      */
        /*   <------- ---------- ---------- ---------- ---------- -------->  */
        /*           |          |          |          |          |           */
        /*       -25 Hz      -10 Hz      0 Hz     +10 Hz     +25 Hz         */
        /*           |          |          |          |          |           */
        /*      (bin 998)  (bin 1013)  455 KHz    (bin 10)   (bin 25)       */
        /*                             (bin 0)                               */
        /*                                                                   */
        /* where: dNCO=change in delta phi each time the mix NCO is updated. */
        /*        Recall, the output of the mix NCO is updated at a rate of  */
        /*        1 kHz.  Changing delta phi by +/- 1 alters the mix NCO     */
        /*        mix NCO frequency output +/- 3.72 mHz.  Hence, the mix NCO */
        /*        frequency output may be slewed at 3.72 Hz/second.          */
        /*-------------------------------------------------------------------*/
trackit: R0=0.0;                /* The DRX is locked to the carrier signal   */
        PM(low_SNR_counter)=R0; /* so reset the low_SNR counter and set the  */
        R0=status_locked;       /* "locked" status flag.                     */
        PM(DRX_status)=R0;

        R0=PM(tracking);        /* Recall, the user may disable carrier      */
        R1=TRUE;                /* frequency tracking.  If tracking has been */
        COMP(R0,R1);            /* disabled, jump to compute the new signal  */
        IF NE JUMP sumit;       /* power measurement.                        */

        R6=PM(temp2);           /* Retrieve the index of DFT point with the  */
                                /* most energy.                              */

                                /* Recall, the complex sampling rate = 1 kHz.*/
                                /* Aliasing maps the frequency ranges into   */
                                /* the digital domain as follows:            */
                                /*                                           */
                                /*          DFT bin       DFT bin            */
                                /*            #0    .....  #1024             */
                                /*          ---------------------            */
                                /*          453 KHz ..... 454 KHz            */
                                /*          454 KHz ..... 455 KHz            */
                                /*          455 KHz ..... 456 KHz            */
                                /*          456 KHz ..... 457 KHz            */
                                /*                                           */
```

```
                                /* The DRX assumes that the frequency of the */
                                /* translated carrier is between 454.5 KHz   */
                                /* and 455.5 KHz.                            */
                                /*                                           */
        R3=512;                 /* Determine if the carrier less than or     */
        COMP(R6,R3);            /* greater than 455 KHz.  If the carrier     */
        IF LE JUMP driftup;     /* frequency is greater than 455 KHz,        */
                                /*                                           */
                                /*            # of the DFT bin               */
                                /*      1 <= representing the <= 512         */
                                /*               carrier                     */
                                /*                                           */
                                /* Otherwise the carrier frequency is less   */
                                /* than 455 KHz.                             */

                                /*-------Carrier frequency < 455 kHz---------*/
        R3=1014;                /* If the carrier frequency is within 10 Hz  */
        COMP (R6,R3);           /* of 455 KHz, do not alter the output of    */
        IF LT JUMP trk1;        /* the NCO (set dNCO=0) and jump to compute  */
        R3=0;                   /* a carrier power measurement.  Otherwise,  */
        PM(mix_NCO_delta)=R3;   /* check to see if the carrier frequency     */
        JUMP sumit;             /* is more than 25 Hz away from 455 KHz.  If */
trk1:   R3=999;                 /* not, jump to compute a carrier power      */
        COMP(R6,R3);            /* measurement.  If so, change dNCO to -1 so */
        IF GE JUMP sumit;       /* the mix NCO output frequency is slewed    */
        R3=-1;                  /* downward at a 3.72 Hz/second in order to  */
        PM(mix_NCO_delta)=R3;   /* slew the translated carrier frequency     */
        JUMP sumit;             /* back toward 455 KHz.  Jump to compute a   */
                                /* carrier power measurement.                */

                                /*-------Carrier frequency > 455 KHZ---------*/
driftup: R3=25;                 /* If the carrier frequency is more than     */
        COMP(R6,R3);            /* 25 Hz away from 455 KHz, change dNCO to   */
        IF LT JUMP trk2;        /* +1 so the mix NCO output frequency is     */
        R3=1;                   /* slewed upward at a rate of 3.72 Hz/second */
        PM(mix_NCO_delta)=R3;   /* in order to slew the translated carrier   */
        JUMP sumit;             /* frequency back toward 455 KHz.  Jump to   */
trk2:   R3=10;                  /* compute a carrier power measurement.  If  */
        COMP(R6,R3);            /* the carrier frequency is not more than    */
        IF GE JUMP sumit;       /* 25 Hz away from 455 KHz, check to see if  */
        R3=0;                   /* it is more than 10 Hz away.  If so, jump  */
        PM(mix_NCO_delta)=R3;   /* to compute a carrier power measurement.   */
                                /* If the carrier frequency is less than     */
                                /* 10 Hz away from 455 KHz, change dNCO to 0 */
                                /* so the output frequency of the mix NCO is */
                                /* not altered.                              */

        /*-------------------------------------------------------------------*/
        /* Compute a 40 Hz power measurement sample by summing the power in  */
        /* the DFT bins containing the carrier signal.  Recall, the          */
        /* detection bandwidth is selected by the user.  Hence, the number   */
        /* of bins summed will vary (=detection bandwidth).                  */
        /*-------------------------------------------------------------------*/
sumit:  R2=PM(detection_bandwidth);     /* Retrieve the detection bandwidth  */
                                        /* specified by the user.            */

        R2=LSHIFT R2 BY -1;             /* Divide the desired detection      */
                                        /* bandwidth by 2.  The mean carrier */
                                        /* frequency (DFT bin with the most  */
                                        /* energy) will be centered in the   */
                                        /* frequency domain filter.  The     */
                                        /* filter passband will extend R2 Hz */
                                        /* on either side of the mean        */
                                        /* carrier frequency.                */

        R6=R6-R2;                       /* Compute the DFT bin number of the */
        M0=R6;                          /* first DFT bin to fall into the    */
                                        /* frequency domain filter passband. */
```

```
                                    /* Store the result in address       */
                                    /* modifier register M0 which will   */
                                    /* be used to retrieve each DFT bin  */
                                    /* power.                            */

        R2=PM(detection_bandwidth); /* Again, retrieve the detection     */
                                    /* bandwidth specified by the user.  */
                                    /* Each DFT bin is approximately     */
                                    /* 1 Hz wide.  Hence <detection      */
                                    /* bandwidth> bins will be summed    */
                                    /* to produce the desired signal     */
                                    /* power measurement.                */

        B1=scratch; L1=1024;        /* Initialize address register #1 to */
        MODIFY (I1,M0);             /* the memory location containing the*/
                                    /* first DFT bin in the passband of  */
                                    /* the frequency domain filter.      */

        F0=0.0;                     /* F0 is used to maintain a running  */
                                    /* sum of the signal power as the    */
                                    /* power in each DFT bin in the      */
                                    /* passband of the frequency domain  */
                                    /* filter is retrieved from memory.  */

        LCNTR=R2, DO sumpwr UNTIL LCE; /* Compute the new 40 Hz power    */
           F1=DM(I1,1);             /* measurement.                      */
           F5=F5-F1;                /*                    2   2          */
sumpwr:    F0=F0+F1;                /*       P=sqrt(sum(I + Q ))         */
        CALL sqrt (DB); F8=3.0; F1=0.5; /*                   i   i        */
                                    /*   where i=1..detection bandwidth. */

        /*-------------------------------------------------------------------*/
        /* Place the new 40 Hz power measurement into to the filter memory   */
        /* of the 30 dB/octave windows used to bandlimit power measurements  */
        /* to 10 Hz and 0.5 Hz.                                              */
        /*-------------------------------------------------------------------*/
        B8=dB30octave_20_dat; I8=PM(dB30octave_20_I); L8=13;       /* 20 Hz  */
        PM(I8,1)=F0;
        PM(dB30octave_20_I)=I8;

        B8=dB30octave_1_dat; I8=PM(dB30octave_1_I); L8=140;        /* 1 Hz   */
        PM(I8,1)=F0;
        PM(dB30octave_1_I)=I8;

        /*-------------------------------------------------------------------*/
        /* As noted above, 20 Hz and 1 Hz power measurements are derived     */
        /* by bandlimiting and decimating the 40 Hz power measurement        */
        /* stream.  The 30 dB/octave windows used to bandlimit the 40 Hz     */
        /* power measurement stream to produce 20 Hz and 1 Hz signal         */
        /* power measurements are only executed 20 and 1 times per second,  */
        /* respectively.  That is, signal power measurements that would      */
        /* be discarded in the decimation process are not computed.          */
        /*-------------------------------------------------------------------*/
        R0=PM(s20);           /* The 30 dB/octave window used to bandlimit   */
        R0=R0+1;              /* the 40 Hz sample stream to 10 Hz is         */
        PM(s20)=R0;           /* executed each time 2 new 40 Hz measurements */
        R1=2;                 /* have accrued (i.e. the window outputs       */
        COMP(R0,R1);          /* measurements at 20 Hz).  If 2 new 40 Hz     */
        IF NE JUMP chk_1Hz;   /* measurements have not accrued jump to       */
                              /* see if 40-40 Hz measurements have accrued   */
                              /* since the last 1 Hz measurement.            */

        R0=0; PM(s20)=R0;     /* If 2 new 40 Hz measurements have accrued,   */
                              /* reset the sample counter to 0 and           */
                              /* generate a new 20 Hz power measurement...   */

        F0=0;                           /* F0 will contain the window      */
                                        /*    output.                      */
```

```
        B9=dB30octave_20_dat;              /* Retrieve the window memory     */
        I9=PM(dB30octave_20_I);            /*    status.                     */
        L9=13;
        B8=dB30octave_20_coefs;
        L8=13;
        LCNTR=13,DO compavg1 UNTIL LCE;    /* Execute the 10 Hz,30 dB/octave */
                                           /*    window.                     */
           F2=PM(I8,1);                    /*   -> retrieve next window      */
                                           /*      coefficient.             */
           F1=PM(I9,1);                    /*   -> retrieve next power       */
                                           /*      measurement.             */
           F1=F1*F2;                       /*   -> F1=coeff.*measurement.   */
compavg1:  F0=F0+F1;                       /*   -> Add result to running sum.*/

        CALL log10 (DB); L11=0; NOP;       /* Convert the signal power       */
        F1=20.0;                           /* measurement to relative dB.    */
        F0=F0*F1;

        F1=dBm_offset;                     /* Convert the power measurement  */
        F0=F0+F1;                          /* from relative dB to dBm by     */
                                           /* adding the offset discussed    */
                                           /* above.                         */

        F1=10.0;                           /* Format the new 20 Hz power     */
        F0=F0-F1;                          /* measurement for output by      */
        F1=-100.0;                         /* converting it to               */
        F0=F0*F1;                          /*                                */
        R0=FIX F0;                         /*         (  signal       )      */
        R1=0x1FFF;                         /* output= (  power   + 10 )*100. */
        R0=R0 AND R1;                      /*         (measurement    )      */
        R1=PM(detection_bandwidth);        /*                                */
        R0=R0 OR FDEP R1 BY 13:6;          /* Convert the result to fixed    */
        R1=PM(DRX_status);                 /* point format and insert the    */
        R0=R0 OR FDEP R1 BY 26:3;          /* sample type, status and        */
        R1=stype_1Hz;                      /* detection bandwidth fields.    */
        R0=R0 OR FDEP R1 BY 29:3;

        R1=PM(tracking);                   /* When carrier tracking has      */
        R2=TRUE;                           /* been disabled, the DRX         */
        COMP(R1,R2);                       /* outputs signal power           */
        IF NE JUMP chk_1Hz;                /* measurements of -71.92 dBm.    */
        PM(sample_20Hz)=R0;                /* If tracking is enabled,        */
                                           /* update <sample_20Hz> to        */
                                           /* reflect the new signal power   */
                                           /* measurement.  If tracking is   */
                                           /* disabled, do not disturb       */
                                           /* the contents of <sample_20Hz>. */

chk_1Hz: R0=PM(s1);          /* The 30 dB/octave window used to bandlimit   */
        R0=R0+1;             /* the 40 Hz sample stream to 0.5 Hz is        */
        PM(s1)=R0;           /* executed each time 40 new 40 Hz measurements*/
        R1=20;               /* have accrued (i.e. the window outputs       */
        COMP(R1,R0);         /* measurements at 1 Hz).  If 40 new 40 Hz     */
        IF NE JUMP wait_opm; /* measurements have not accrued jump back to  */
                             /* signal power measurement algorithm step #6. */

        R0=0; PM(s1)=R0;     /* If 40 new 40 Hz measurements have accrued,  */
                             /* reset the sample counter to 0 and           */
                             /* generate a new 1 Hz power measurement...    */

        F0=0;                              /* F0 will contain the window     */
                                           /*    output.                     */

        B9=dB30octave_1_dat;               /* Retrieve the window memory     */
        I9=PM(dB30octave_1_I);             /*    status.                     */
        L9=140;
        B8=dB30octave_1_coefs;
        L8=140;
```

```
                                            /* Execute the 10 Hz,30 dB/octave */
       LCNTR=140,DO compavg2 UNTIL LCE;     /*    window.                     */
          F2=PM(I8,1);                      /*   -> retrieve next window      */
                                            /*      coefficient.              */
          F1=PM(I9,1);                      /*   -> retrieve next power       */
                                            /*      measurement.              */
          F1=F1*F2;                         /*   -> F1=coeff.*measurement.    */
compavg2: F0=F0+F1;                         /*   -> Add result to running sum.*/

       CALL log10 (DB); L11=0; NOP;         /* Convert the signal power       */
       F1=20.0;                             /* measurement to relative dB.    */
       F0=F0*F1;

       F1=dBm_offset;                       /* Convert the power measurement  */
       F0=F0+F1;                            /* from relative dB to dBm by     */
                                            /* adding the offset discussed    */
                                            /* above.                         */

       F1=10.0;                             /* Format the new 20 Hz power     */
       F0=F0-F1;                            /* measurement for output by      */
       F1=-100.0;                           /* converting it to               */
       F0=F0*F1;                            /*                                */
       R0=FIX F0;                           /*          (  signal        )    */
       R1=0x1FFF;                           /* output= (  power    + 10 )*100. */
       R0=R0 AND R1;                        /*          (measurement     )    */
       R1=PM(detection_bandwidth);          /*                                */
       R0=R0 OR FDEP R1 BY 13:6;            /* Convert the result to fixed    */
       R1=PM(DRX_status);                   /* point format and insert the    */
       R0=R0 OR FDEP R1 BY 26:3;            /* sample type, status and        */
       R1=stype_1Hz;                        /* detection bandwidth fields.    */
       R0=R0 OR FDEP R1 BY 29:3;

       R1=PM(tracking);                     /* When carrier tracking has      */
       R2=TRUE;                             /* been disabled, the DRX         */
       COMP(R1,R2);                         /* outputs signal power           */
       IF NE JUMP wait_opm;                 /* measurements of -71.92 dBm.    */
       PM(sample_1Hz)=R0;                   /* If tracking is enabled,        */
       JUMP wait_opm; ,                     /* update <sample_1Hz> to         */
                                            /* reflect the new signal power   */
                                            /* measurement.  If tracking is   */
                                            /* disabled, do not disturb       */
                                            /* the contents of <sample_1Hz>.  */

/*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/
/*!!!                              Subroutines                             !!!*/
/*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/
/**********************CFIR (Complex FIR Filter) Subroutine******************/
/*                                                                          */
/* This subroutine executes a complex FIR filter of specified (even) length. */
/*                                                                          */
/* FIR filter coefficients are symmetrical about the filter's midpoint.  To */
/* reduce the number of multiply operations required to execute the filter, */
/* the ith and (N-i)th data points are summed and then multiplied by the    */
/* appropriate filter coefficient (N=filter length).  i.e. the filter       */
/* output is calculated as:                                                 */
/*                                                                          */
/*                        (N/2)                                             */
/*                output = SUM [d(i)+d(N-i)]*f(i);                          */
/*                        i=0                                               */
/*                                                                          */
/*                where: d(i)=ith data point                                */
/*                       f(i)=ith filter coefficient                        */
/*                       N=filter length                                    */
/*                                                                          */
/* Input Requirements:                                                      */
/*       <I7> = address of oldest real sample in the delay line (DM).       */
/*       <L7> = filter length (must be an even integer).                    */
/*       <B7> = base address of real data.                                  */
```

```
/*        <I13> = address of oldest complex sample in the delay line (PM).    */
/*        <L13> = filter length (must be an even integer).                    */
/*        <B13> = base address of imaginary data.                             */
/*        <LCNTR> = filter length/2.                                          */
/*        <I10> = address of first filter coefficient (PM).                   */
/*                                                                            */
/* Output Conditions:                                                         */
/*        <F9> = real component of filter output.                             */
/*        <F10> = complex component of filter output.                         */
/*                                                                            */
/******************************************************************************/
CFIR:    B6=B7;              /* Each circular filter buffer contains data     */
         I6=I7;              /* points in the following order:                */
         L6=L7;              /*                                               */
         MODIFY (I6,-1);     /*              newest-      -oldest             */
                             /*                        |    |                 */
         B12=B13;            /*    ... (3) (2) (1) (0) (N-1) (N-2)  ...       */
         I12=I13;            /*                  |              |             */
         L12=L13;            /*      2nd newest-                -2nd oldest   */
         MODIFY (I12,-1);    /*                                               */
                             /* One address register is required to traverse  */
                             /* the buffer in each direction in order to      */
                             /* retrieve the ith and (N-i)th data points      */
                             /* quickly.  Initialize I7 to traverse the       */
                             /* buffer back into the real components of the   */
                             /* newer data samples.  I6 will be incremented   */
                             /* to retrieve the real components of the older  */
                             /* data samples.  Address registers 12 and 13    */
                             /* are used in a similar fashion to retrieve the */
                             /* complex data sample components.               */

                             /* Execute the filter.                           */
         F9=0.0;
         F10=PASS F9, F3=DM(I7,M7), F11=PM(I13,M13);
                      F4=DM(I6,M6), F12=PM(I12,M12);
         DO macs UNTIL LCE;
                      F3=F3+F4,                   F4=PM(I10,1);
            F12=F3*F4, F3=F11+F12;
            F12=F3*F4, F9=F9+F12,   F3=DM(I7,M7), F11=PM(I13,M13);
macs:                  F10=F10+F12, F4=DM(I6,M6), F12=PM(I12,M12);
         RTS;

/************************FFT_DM (Complex FFT) Subroutine************************/
/*                                                                            */
/* This subroutine performs an N point, in-place, radix-2,                    */
/* decimation-in-time FFT.  The assembly language code below implements the   */
/* following C program.                                                       */
/*                                                                            */
/*    #define pi 3.141592654                                                  */
/*    struct cmplx                                                            */
/*     {                                                                      */
/*      float re;     * 're' is the real part of a complex number.        *   */
/*      float im;     * 'im' is the imaginary part of a complex number.   *   */
/*     };                                                                     */
/*    ************************************************************************/
/*    * Performs a complex multiplication.                                 *  */
/*    ************************************************************************/
/*    void cmult (struct cmplx num1, struct cmplx num2,                       */
/*                struct cmplx *result)                                       */
/*       {                                                                    */
/*        result->re=num1.re*num2.re-num1.im*num2.im;                         */
/*        result->im=num1.re*num2.im+num1.im*num2.re;                         */
/*       }                                                                    */
/*    ************************************************************************/
/*    * Performs a complex subtraction.                                    *  */
/*    ************************************************************************/
/*    void csub (struct cmplx num1, struct cmplx num2,                        */
/*               struct cmplx *result)                                        */
```

```
/*      {                                                                  */
/*       result->re=num1.re-num2.re;                                       */
/*       result->im=num1.im-num2.im;                                       */
/*      }                                                                  */
/*   ********************************************************************  */
/*   * Performs a complex addition.                                     *  */
/*   ********************************************************************  */
/*   void cadd (struct cmplx num1, struct cmplx num2,                      */
/*              struct cmplx *result)                                      */
/*      {                                                                  */
/*       result->re=num1.re+num2.re;                                       */
/*       result->im=num1.im+num2.im;                                       */
/*      }                                                                  */
/*   ********************************************************************  */
/*   * FFT subroutine.                                                  *  */
/*   ********************************************************************  */
/*   void fft (struct cmplx *indata, unsigned long length)                 */
/*      {                                                                  */
/*       unsigned long stages;                                             */
/*       unsigned long newindx;    * (newindx) is the new index of a data *  */
/*                                 *    sample in the bit-reversal    *  */
/*                                 *    process.                      *  */
/*       struct cmplx temp;        * (temp) is temporary storage used in  *  */
/*                                 *    the bit-reversal process.     *  */
/*       unsigned long j,k,i,l;    * (j),(k),(i) & (l) are loop control   *  */
/*                                 *    variables                     *  */
/*       unsigned long le,le1,ip; * (le),(le1),(ip),(u) & (w) are        *  */
/*                                 *    miscellaneous                 *  */
/*       struct cmplx u,w;         *   variables used in computing the    *  */
/*                                 *    FFT                           *  */
/*                                                                         */
/*       stages=(unsigned long)(log(length)/log(2.0)+0.5);                 */
/*       *-----------------------------------------------------------*     */
/*       * Bit reversal.                                           *     */
/*       *-----------------------------------------------------------*     */
/*       for (j=0;j<length;j++)                                            */
/*          {                                                              */
/*           newindx=0;,                                                   */
/*           for (k=0;k<stages;k++)                                        */
/*              if (j&(1<<k)) newindx|=(length>>(k+1));                    */
/*           if (newindx>j)                                                */
/*              {                                                          */
/*               memcpy(&temp,&indata[j],sizeof(struct cmplx));            */
/*               memcpy(&indata[j],&indata[newindx],sizeof(struct cmplx)); */
/*               memcpy(&indata[newindx],&temp,sizeof(struct cmplx));      */
/*              }                                                          */
/*          }                                                              */
/*       *-----------------------------------------------------------*     */
/*       * Fast Fourier transform.                                 *     */
/*       *-----------------------------------------------------------*     */
/*       for (l=1;l<=stages;l++)                                           */
/*          {                                                              */
/*           le=(unsigned long)pow(2.0,l);                                 */
/*           le1=le/2;      '                                              */
/*           u.re=(float)1.0;                                              */
/*           u.im=(float)0.0;                                              */
/*           w.re=(float)cos(pi/le1);                                      */
/*           w.im=-(float)sin(pi/le1);                                     */
/*           for (j=1;j<=le1;j++)                                          */
/*              {                                                          */
/*               i=j;                                                      */
/*               while (i<=length)                                         */
/*                  {                                                      */
/*                   ip=i+le1;                                             */
/*                   cmult(indata[ip-1],u,&temp);                          */
/*                   csub(indata[i-1],temp,&indata[ip-1]);                 */
/*                   cadd(indata[i-1],temp,&indata[i-1]);                  */
/*                   i=i+le;                                               */
```

```
/*                    }                                                        */
/*                 cmult(u,w,&u);                                              */
/*                }                                                            */
/*            }                                                                */
/*        }                                                                    */
/*                                                                             */
/* Assembly Languge                                                            */
/*  Routine Input                                                              */
/*   Requirements: <LCNTR> = base-2 logarithm of the FFT length.               */
/*                 <I12> points to program memory variable space.              */
/*                 <R6> = FFT length.                                          */
/*                 <I0> = data memory address of real input data.              */
/*                 <I1> = data memory address of imaginary input data.         */
/*                                                                             */
/* Output Conditions: Complex FFT results are stored where the complex         */
/*                    input data used to be.                                   */
/*******************************************************************************/
#define pm_pi   0               /* Initialize indices into the program memory  */
#define pm_temp 1               /* data space.                                 */

fft_dm: R15=1;                  /* R15=le1.                                    */

levells:DO levelle until LCE;

          /*-------------------------------------------------------------------*/
          /* Compute twiddle factor argument, F0=pi/le1.                       */
          /*-------------------------------------------------------------------*/
          CALL divide (DB);
          F12=FLOAT R15, F0=PM(pm_pi,I12);
          F11=2.0;

          /*-------------------------------------------------------------------*/
          /* Compute imaginary twiddle factor, F1=sine(pi/le1)=w.im.           */
          /*-------------------------------------------------------------------*/
          CALL sine (DB);
          L11=0;
          PM(pm_temp,I12)=F0; /* Store twiddle factor arg. for later use.      */
          F1=-F0;       ,     /* Store negative result in F1.                  */

          /*-------------------------------------------------------------------*/
          /* Compute real twiddle factor, F0=cosine(pi/le1)=w.re.              */
          /*-------------------------------------------------------------------*/
          CALL cosine (DB);
          L11=0;              /* Required by "cosine."                         */
          F0=PM(pm_temp,I12); /* Retrieve twiddle factor argument.             */

          /*-------------------------------------------------------------------*/
          /* Initialize Temporary Computational Variable.                      */
          /*-------------------------------------------------------------------*/
          F2=1.0;             /* F2=u.re.                                      */
          F3=0.0;             /* F3=u.im.                                      */

level2s:  LCNTR=R15,DO level2e UNTIL LCE;  /* for j=0;j<le1;j++.              */
             R8=CURLCNTR;     /* Compute R8=i=j from loop status.              */
             R8=R15-R8;
level3s:     M1=R8;           /* While i<length, M1=i-index.                   */
             R11=R8+R15;      /* Compute R11=ip=i+le1, (NOP if ref. DAG1).     */
             M0=R11;          /* M0=ip-index.                                  */
             R7=LSHIFT R15 BY 1;  /* R7=le, NOP if ref. DAG1).                 */

             /*----------------------------------------------------------------*/
             /* Perform complex arithmetic:                                    */
             /*  temp = data[ip]*u = (a+bj)*(c+di) = (ac-bd)+(bc+ad)j          */
             /*  data[ip] = data[i]-temp = (e+fj)-[(ac-bd)+(bc+ad)j]           */
             /*                          = (e-ac+bd)+(f-bc-ad)j                */
             /*  data[i]  = data[i]+temp = (e+fi)+[(ac-bd)+(bc+ad)j]           */
             /*                          = (e+ac-bd)+(f+bc+ad)j                */
             /*----------------------------------------------------------------*/
```

```
                /*:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
                /* F4=a                                                           */
                /*:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
                                                    F4=DM(M0,I0);

                /*:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
                /* F11=e                                                          */
                /*:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
                                                    F11=DM(M1,I0);

                /*:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
                /* F12=ac, F5=b                                                   */
                /*:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
                F12=F2*F4,                          F5=DM(M0,I1);

                /*:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
                /* F13=bd, F9=e+ac, F10=e-ac, F11=f                               */
                /*:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
                F13=F3*F5, F9=F11+F12, F10=F11-F12,     F11=DM(M1,I1);

                /*:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
                /* F12=ad, F10=e-ac+bd                                            */
                /*:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
                F12=F3*F4, F10=F10+F13;

                /*:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
                /* F9=e+ac-bd, data[i].re=e-ac+bd                                 */
                /*:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
                                    F9=F9-F13,      DM(M0,I0)=F10;

                /*:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
                /* F12=bc, F9=f+ad, F11=f-ad, data[i].re=e+ac-bd                  */
                /*:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
                F12=F2*F5, F9=F11+F12, F11=F11-F12,     DM(M1,I0)=F9;

                /*:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
                /* F9=f+ad+bc                                                     */
                /*:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
                           F9=F9+F12;

                /*:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
                /* F11=f-ad-bc, data[i].im=f+bc+ad                                */
                /*:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
                                        F11=F11-F12, DM(M1,I1)=F9;

                /*:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
                /* Update loop counter(i=i+le), data[ip].im=f-ad-bc               */
                /*:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
                           R8=R8+R7,                DM(M0,I1)=F11;

                COMP (R6,R8);       /*End of i-loop, compare R8=i to R6=FFT len.*/
level3e:        IF GT JUMP level3s; /*length and jump if i<length.              */

                /*---------------------------------------------------------------*/
                /* Perform complex arithmetic:                                   */
                /*  u=w*u = (a+bj)*(c+dj) = (ac-bd)+(bc+ad)j                     */
                /*---------------------------------------------------------------*/
                F12=F0*F2;          /* F12=ac.                                   */
                F10=F1*F2;          /* F10=bc.                                   */
                F13=F1*F3;          /* F13=bd.                                   */
                F9=F0*F3;           /* F9=ad.                                    */
                F2=F12-F13;         /* u.re=ac-bd.                               */
level2e:        F3=F10+F9;          /* u.im=bc+ad.                               */
levelle:R15=LSHIFT R15 BY 1;        /* Update le1=2^loop count.                  */
        RTS;

/************************ACQ_FFT (Acquisition FFT) Subroutine*******************/
/* This subroutine performs a 32k complex FFT and evaluates the results for   */
```

```
/* the carrier signal acquisition algorithm.                                  */
/*                                                                            */
/* Input Requirements: none                                                   */
/* Output Conditions: <F0>=carrier frequency estimate.                        */
/*                    <F1>=power value in DFT bin representing the carrier    */
/******************************************************************************/
        /*--------------------------------------------------------------------*/
        /* Clear the fifo of any old samples.                                 */
        /*--------------------------------------------------------------------*/
acq_fft: BIT SET ASTAT FLG0|FLG1;
        BIT CLR ASTAT FLG2;
        LCNTR=2048,DO clrfifo1 UNTIL LCE;
clrfifo1:   R0=DM(r_fifo);
        BIT SET ASTAT FLG2;

        /*--------------------------------------------------------------------*/
        /* The following code performs initializations which cause the        */
        /* "fifo almost full" interrupt service routine to place the next     */
        /* 32k complex data points into data memory in bit reversed order     */
        /* (real components are stored from address 0 to address 32k,         */
        /* complex components are stored from address 32k to address 64k).    */
        /* A complete description of the bit reversal storage process is      */
        /* included in the "fifo almost full" interrupt service routine       */
        /* documentation.                                                     */
        /*--------------------------------------------------------------------*/
        R0=0;                       /* R0 counts the number of samples placed */
                                    /*    in memory so far.                   */

        R1=32768;                   /* R0 is compared to R1 to check if 32k   */
                                    /*    complex samples have accrued.       */

        B0=0; L0=0;                 /* Address register #0 is used as an      */
                                    /*    index register in the bit           */
                                    /*    reversed storage process.           */

        M0=131072;                  /* M0=2^(32-logb2(FFT size)).  M0 is used */
                                    /*    as a bit-reversed increment by 1.   */

        M2=32768;                   /* M2 is used as the offset from the      */
                                    /*    real component storage locations    */
                                    /*    to the complex component storage    */
                                    /*    locations.                          */

        /*--------------------------------------------------------------------*/
        /* Enable the "fifo almost full" interrupt and wait for 32k complex   */
        /* samples to accrue.  (The interrupt service routine increments R0   */
        /* to indicate the number of samples stored so far).                  */
        /*--------------------------------------------------------------------*/
        BIT SET IMASK IRQ0I;
acqwait: COMP (R0,R1);              /* Have 32k samples accrued? If not,      */
        IF NE JUMP acqwait;         /* continue to wait here.                 */
        BIT CLR IMASK IRQ0I;

        /*--------------------------------------------------------------------*/
        /* Perform a 32k complex FFT.                                         */
        /*--------------------------------------------------------------------*/
initfft: LCNTR=15;
        B12=pm_vars; L12=0;
        R6=32768;
        B0=0; L0=0;
        B1=32768; L1=0;
        CALL fft_dm;

        /*--------------------------------------------------------------------*/
        /* Convert the complex FFT results to power (P=I^2+Q^2) to locate     */
        /* the spectral peak with the greatest magnitude.  Note, only the     */
        /* DFT points whose spectral location falls in the passband of the    */
        /* 200 kHz IF filter are examined.  These bins are:                   */
```

```
        /*                                                                */
        /*      filter passband       DFT bin #     (    355 kHz    )    */
        /*         starts at     ->  representing = (   -------     )%1.0 */
        /*      455 kHz-100 kHz        355 kHz      (fsc=303.333 kHz)    */
        /*         =355 kHz                                               */
        /*                                    = 5581                      */
        /*                                                                */
        /*      filter passband       DFT bin #     (    555 kHz    )    */
        /*         ends at       ->  representing = (   -------     )%1.0 */
        /*      455 kHz+100 kHz        555 kHz      (fsc=303.333 kHz)    */
        /*         =555 kHz                                               */
        /*                                    = 27167                     */
        /*                                                                */
        /* The entire 303.333 kHz bandwidth is not checked to save time and */
        /* to avoid erroneously identifying the spectral peak representing */
        /* any dc offset present at the output of the ADC op-amp as the   */
        /* carrier signal.                                                */
        /*----------------------------------------------------------------*/
        B0=5581; L0=0;      /* The real component of the 5581th DFT bin is */
                            /* stored at address 5581.  The corresponding */
                            /* complex component is stored at address     */
                            /* 5581+32k.  The remaining FFT results of    */
                            /* interest are stored in ascending locations. */
                            /* Initialize address register #0 to index    */
                            /* through the DFT bins of interest.          */


        F3=-999999.0;       /* F3 is used to record the maximum DFT bin power */
                            /* found as the bins of interest are examined. */

                            /* Examine the contents of 27167-5581=21606 DFT */
                            /* bins starting at bin #5581 to locate the   */
                            /* spectral peak with the greatest magnitude. */
        LCNTR=21606,DO findmax UNTIL LCE;
           F0=DM(32768,I0);       /* Retrieve the complex component of the */
                                  /*    next DFT bin (Q).                */
           F0=F0*F0,F1=DM(I0,1);  /* Retrieve the real component of the next */
                                  /*    DFT bin (I) and compute F0=Q^2.  */
           F1=F1*F1;              /* F1=I^2.                             */
           F0=F0+F1;              /* F0=I^2+Q^2=power.                   */
           COMP(F0,F3);           /* Is the power in the current DFT bin */
                                  /*    greater than that contained in any */
                                  /*    bin examined so far?             */
           IF LE JUMP findmax;    /* If not, jump to the end of the loop to */
                                  /*    examine the contents of the next bin.*/
           R4=I0;                 /* If so, record the bin# in R4 and update */
           F3=F0;                 /*    F3 to reflect the new maximum power. */
findmax:   NOP;

        /*----------------------------------------------------------------*/
        /* Compute a carrier frequency estimate based on the spectral     */
        /* location of DFT bin with the greatest magnitude.               */
        /*                                                                */
        /*               max                          frequency represented */
        /*             DFT bin * (fsc=303.333 kHz)    by DFT bin index #0 */
        /*              index                         (based on current mix */
        /*          fc=------------------------- +    frequency...see calling */
        /*                      32768                 routine for this     */
        /*                                            calculation.         */
        /*----------------------------------------------------------------*/
        F0=FLOAT R4;
        F1=32768.5;
        F0=F1-F0;
        CALL divide (DB); F12=32767.0; F11=2.0;
        F1=455000.0/0.75/2.0;
        F0=F0*F1;
        F1=PM(freq_base);
        F0=F0+F1;
```

```
        F1=F3;
        RTS;

/*****************************SPECTRUM Subroutine*****************************/
/* This subroutine performs a complex FFT of user-specified size, converts    */
/* the results to dBC, outputs the results and the returns the APT system     */
/* to the signal power measurement configuration.                             */
/*                                                                            */
/* Input Requirements: none                                                   */
/* Output Conditions: The DRX/APT system will begin to make signal power      */
/*                    measurements when the FIR/FFT pipeline refills.         */
/******************************************************************************/
        /*-------------------------------------------------------------------*/
        /* The DRX is unable to compute signal power measurements while it   */
        /* is performing spectrum calculations.  If the DRX is polled for    */
        /* a signal power measurement during this time, the receiver will    */
        /* output a signal power measurement of -71.92 with a status tag     */
        /* indicating that the DRX is in spectrum analyzer mode and unable   */
        /* to provide the desired power measurement.  Place the -71.92 dB    */
        /* signal power measurement word in the <sample_1Hz> and             */
        /* <sample_20Hz> memory locations.  (The power measurement           */
        /* information contained in these memory locations are output when   */
        /* the DRX is polled for power measurement information).             */
        /*-------------------------------------------------------------------*/
spectrum:R0=8191;                    /* -(-71.92-10.00)*100=8192.            */
        R1=status_gen_spect;         /* Add the "spectrum mode" status tag to */
        R0=R0 OR FDEP R1 BY 26:3;    /*    the output word.                  */
        R1=PM(detection_bandwidth); /* Add the current detection bandwidth   */
        R0=R0 OR FDEP R1 BY 13:6;    /*    tag to the output word.           */
        R1=stype_1Hz;                /* Add the 1 Hz power measurement sample */
        R0=R0 OR FDEP R1 BY 29:3;    /*    type tag to the output word.      */
        PM(sample_1Hz)=R0;           /* Store the -71.92 dB, 1 Hz output word.*/
        R1=stype_20Hz;               /* Add the 20 Hz power measurement       */
        R0=R0 OR FDEP R1 BY 29:3;    /*    sample type tag to the output word.*/
        PM(sample_20Hz)=R0;          /* Store the -71.92 dB, 20 Hz output     */
                                     /*    word.                              */

        /*-------------------------------------------------------------------*/
        /* Reset the generate spectrum flag used to indicate that this       */
        /* subroutine should be called.                                      */
        /*-------------------------------------------------------------------*/
        R0=FALSE;
        PM(generate_spectrum)=R0;

        /*-------------------------------------------------------------------*/
        /* Request that the host switch the IF filter switch to the 200 kHz  */
        /* bandwidth and wait for the host to acknowledge the request.       */
        /*-------------------------------------------------------------------*/
        R0=wide_filter;
        CALL send_cmd;
        CALL acknow;

        /*-------------------------------------------------------------------*/
        /* Output a frequency estimate word indicating the current carrier   */
        /* frequency.                                                        */
        /*-------------------------------------------------------------------*/
        F0=PM(current_fc);           /* Retrieve the latest carrier          */
                                     /*    frequency estimates.              */
        R0=FIX F0;                   /* Convert to fixed point format.       */
        R1=0xFFFFF;
        R0=R0 AND R1;
        R1=stype_frequency;          /* Add the "frequency word" sample type */
        R0=R0 OR FDEP R1 BY 29:3;    /*    tag.                              */
        R1=status_gen_spect;         /* Add the "generating spectrum" DRX    */
        R0=R0 OR FDEP R1 BY 26:3;    /*    status tag.                       */
        CALL senddata;               /* Place the frequency word in the      */
                                    /*    transmit queue.                    */
```

```
        /*-------------------------------------------------------------------*/
        /* Clear the fifo of any old samples.                                */
        /*-------------------------------------------------------------------*/
        BIT SET ASTAT FLG0|FLG1;
        BIT CLR ASTAT FLG2;
        LCNTR=2048,DO clrfifo3 UNTIL LCE;
clrfifo3:   R0=DM(r_fifo);
        BIT SET ASTAT FLG2;

        /*-------------------------------------------------------------------*/
        /* Set the DRX mode flag to indicate the DRX is in spectrum analyzer */
        /* mode.                                                             */
        /*-------------------------------------------------------------------*/
        R0=status_gen_spect;
        PM(DRX_mode)=R0;

        /*-------------------------------------------------------------------*/
        /* The following code performs initializations which cause the       */
        /* "fifo almost full" interrupt service routine to place the next    */
        /* N complex data points into data memory in bit reversed order      */
        /* (real components are stored from address 0 to address N, complex  */
        /* components are stored from address 32k to address 32k+N).  A      */
        /* complete description of the bit reversal storage process is       */
        /* included in the "fifo almost full" interrupt service routine      */
        /* documentation.                                                    */
        /*-------------------------------------------------------------------*/
        L11=0;
        CALL logb2 (DB); R0=PM(spectrum_size); F0=FLOAT R0;
        R0=FIX F0;
        PM(temp1)=R0;               /* M0 is used as a bit-reversed          */
        R1=32;                      /* increment by 1.  Compute              */
        R1=R1-R0;                   /* M0=2^(32-logb2(FFT size)).            */
        R0=1;
        R0=LSHIFT R0 BY R1;
        M0=R0;

        R0=0;                       /* R0 counts the number of samples placed */
                                    /*    in memory so far.                  */

        R1=PM(spectrum_size);       /* R0 is compared to R1 to check if 32k   */
                                    /*    complex samples have accrued.      */

        M2=R1;                      /* M2 is used as the offset from the      */
                                    /*    real component storage locations   */
                                    /*    to the complex component storage   */
                                    /*    locations.                         */

        B0=0; L0=0; L1=0;           /* Address register #0 is used as an      */
                                    /*    index register in the bit          */
                                    /*    reversed storage process.          */

        /*-------------------------------------------------------------------*/
        /* Enable the "fifo almost full" interrupt and wait for N complex    */
        /* samples to accrue.  (The interrupt service routine increments R0  */
        /* to indicate the number of samples stored so far).                 */
        /*-------------------------------------------------------------------*/
        BIT SET IMASK IRQ0I;
specwait:COMP(R0,R1);
        IF NE JUMP specwait;
        BIT CLR IMASK IRQ0I;

        /*-------------------------------------------------------------------*/
        /* Perform a N-point complex FFT.                                    */
        /*-------------------------------------------------------------------*/
initfft2:LCNTR=PM(temp1);
        B12=pm_vars; L12=0;
        R6=PM(spectrum_size);
        B0=0; L0=0;
```

```
        B1=R6; L1=0;
        CALL fft_dm;

        /*--------------------------------------------------------------------*/
        /* Convert the complex results of the FFT to magnitude (dB) results  */
        /* and find the magnitude of the largest spectral peak.  The         */
        /* magnitude of the largest spectral peak (assumed to represent the  */
        /* the carrier signal) are used later to convert the FFT results     */
        /* from relative dB to dBC.                                          */
        /*--------------------------------------------------------------------*/
        B1=0;  L0=0;                /* The real components of the FFT results */
                                    /* are stored at memory locations 0 to N. */
                                    /* The complex components of the FFT      */
                                    /* results are stored at memory locations */
                                    /* 32k to 32k+N.  Address register #0 is  */
                                    /* used to index through the FFT results. */

        F9=-9999999.0;              /* F9 records the maximum DFT bin power   */
                                    /* located thus far.                      */

        F2=10.0;                    /* The constant 10 is used to convert the */
                                    /* the FFT magnitude results to dB. i.e.  */
                                    /* power=(10)*log10(I^2+Q^2).             */

        R6=PM(spectrum_size);       /* Retrieve the size of the FFT executed. */

        M2=R6;                      /* M2 is used as the offset from the      */
                                    /*    real component storage locations    */
                                    /*    to the complex component storage    */
                                    /*    locations.                          */

                                    /* Convert the contents of each DFT bin   */
                                    /* to magnitude (dB) results and locate   */
                                    /* the DFT bin with the maximum power.    */
        LCNTR=R6,DO comp_mag UNTIL LCE;
           F0=DM(M2,I1);                    /* Retrieve the complex component */
                                            /*    of the next DFT bin (Q).    */
           F0=F0*F0, F1=DM(I1,0);           /* Compute F0=Q^2 and retrieve    */
                                            /*    the real component of the   */
                                            /*    next DFT bin.               */
           F1=F1*F1;                        /* F1=I^2.                        */
           F0=F0+F1;                        /* F0=I^2+Q^2.                    */
           CALL log10 (DB); L11=0; NOP;     /* F0=log10(I^2+Q^2).             */
           F0=F0*F2;                        /* F0=10*log10(I^2+Q^2)=power(dB).*/
           COMP(F0,F9);                     /* Is the power in the current    */
                                            /*    DFT bin greater than the    */
                                            /*    contents of the other DFT   */
                                            /*    bins examined so far.       */
           IF LE JUMP save_it;              /* If not, jump to the end of the */
                                            /*    loop to save the current    */
                                            /*    DFT bin power and process   */
                                            /*    the next DFT bin.           */
           F9=F0;                           /* If so, record the new power    */
                                            /*    maximum.                    */
save_it:   DM(I1,1)=F0;                     /* Record the current DFT bin     */
           NOP;                             /*    power.                      */
           NOP;
comp_mag:  NOP;

        /*--------------------------------------------------------------------*/
        /* Convert the FFT results from relative dB to dBC, format the       */
        /* results for output and place the results in the transmit queue.   */
        /*--------------------------------------------------------------------*/
        R2=0x7F;                    /* Each spectrum word output contains a   */
                                    /*    a "bin # counter" field.  This      */
                                    /*    field is 7 bits wide and contains   */
                                    /*    the low seven bits of the DFT       */
                                    /*    bin index.  R2 is used to AND       */
```

```
                                /*    off the high bits of the indices.   */

        F3=-100.0;              /* DFT bin powers are output as:          */
                                /*    (attenuation*100) in dBC.           */
                                /*    F3 stores the constant -100 for     */
                                /*    use in calculations performed       */
                                /*    to prepare DFT bin power values     */
                                /*    for output.                         */

        R5=0xFFFF;              /* The DFT bin power output field is 16   */
                                /*    bits wide.  R5 is used to limit     */
                                /*    the magnitude of the attn. value    */
                                /*    to 655.35 dB to ensure the          */
                                /*    measurement can be represented by   */
                                /*    16 bits.                            */

        B1=0;                   /* Address register #1 is used to step    */
                                /* through the FFT results (now in        */
                                /* relative magnitude (dB) form).  These  */
                                /* results are stored in memory locations */
                                /* 0 through N.                           */

        R6=PM(spectrum_size);   /* Retrieve the size of the FFT executed  */
                                /* above.                                 */

                                /* Process all DFT points.                */
        LCNTR=R6,DO send_mag UNTIL LCE;
           R6=I1; R6=R6 AND R2;         /* The magnitude of DFT bin 0 is   */
                                        /*    stored at address 0, the     */
                                        /*    magnitude of DFT bin 1 is    */
                                        /*    stored at address 1...       */
                                        /*    Use the low seven bits of    */
                                        /*    the address of the current   */
                                        /*    DFT bins as the "bin #       */
                                        /*    counter" field of the next   */
                                        /*    spectrum output word.        */
           F0=DM(I1,1);                 /* Retrieve the magnitude of the   */
                                        /*    next DFT point.              */
           F0=F0-F9;                    /* Subtract the magnitude of the   */
                                        /*    DFT bin representing the     */
                                        /*    carrier signal to convert    */
                                        /*    the magnitude of the current */
                                        /*    DFT bin from relative dB to  */
                                        /*    dBC.                         */
           F0=F0*F3;                    /* Convert the magnitude of the    */
                                        /*    the current DFT bin from dBC */
                                        /*    to attenuation*100 (dB).     */
           R0=FIX F0;                   /* Convert the attenuation*100     */
                                        /*    measurement to fixed point   */
                                        /*    format.                      */
           R0=MIN(R0,R5);               /* Limit the attenuation           */
                                        /*    measurement to 655 dB to     */
                                        /*    ensure the measurement can   */
                                        /*    be represented by 16 bits.   */
           R1=stype_spectrum;           /* Add the "spectrum sample"       */
           R0=R0 OR FDEP R1 BY 29:3;    /*    sample type tag to the       */
                                        /*    output word.                 */
           R1=status_gen_spect;         /* Add the "Generating Spectrum"   */
           R0=R0 OR FDEP R1 BY 26:3;    /*    DRX status tag to the        */
                                        /*    output word.                 */
           R0=R0 OR FDEP R6 BY 19:7;    /* Add the "bin # counter" field   */
                                        /*    to the output word.          */
send_mag:  CALL senddata;               /* Place the output word in the    */
                                        /*    transmit queue.              */

        /*-------------------------------------------------------------------*/
        /* Prepare the DRX/APT system to re-enter signal power measurement   */
        /* mode.                                                             */
```

```
        /*-------------------------------------------------------------------*/
        R0=-1078;                   /* Reset the complex sample counter used  */
        PM(sfft)=R0;                /* to determine when the next signal      */
                                    /* power measurement mode FFT is executed.*/
                                    /* A signal power measurement mode FFT    */
                                    /* is executed when the value of this     */
                                    /* counter is 25.  By initializing this   */
                                    /* counter to                             */
                                    /*                                        */
                                    /*         length  length   length  length*/
                                    /* counter=  of   + of FIR + of FIR +  of  */
                                    /*          comb    filter   filter   FFT  */
                                    /*         filter     #1      #2          */
                                    /*                                        */
                                    /*        =2+22+30+1024=1078              */
                                    /*                                        */
                                    /* the DRX will not execute a signal      */
                                    /* power measurement mode FFT until       */
                                    /* the filter/FFT pipeline has filled     */
                                    /* up.                                    */

        R0=narrow_filter;           /* Request the host to set the IF filter  */
        CALL send_cmd;              /* switch to the 10 kHz position and wait */
        CALL acknow;                /* for the host to acknowledge the        */
                                    /* request.                               */

                                    /* Set the DRX mode flag to indicate the  */
                                    /* receiver is re-entering signal power   */
                                    /* measurement mode.                      */
        R0=mode_power_meas; PM(DRX_mode)=R0;

        BIT SET ASTAT FLG0|FLG1;            /* Clear any old samples from the */
        BIT CLR ASTAT FLG2;                 /* FIFO.                          */
        LCNTR=2048,DO clrfifo4 UNTIL LCE;
clrfifo4:   R0=DM(r_fifo);
        BIT SET ASTAT FLG2;

        BIT SET IMASK IRQ0I;        /* Enable the "fifo almost full"          */
                                    /* interrupt.                             */

        RTS;                        /* Return to the signal power measurement */
                                    /* mode algorithm.                        */

/*****************************WAIT8251 Subroutine*****************************/
/* One of the 8251A timing specifications mandates that writes to the 8251A   */
/* must be separated by 8*Tcy=67 ADSP21020 clock cycles.  This subroutine     */
/* counts 70 ADSP21020 clock cycles.                                          */
/*                                                                            */
/* Input Requirements: none.                                                  */
/* Output Conditions: none.                                                   */
/******************************************************************************/
wait8251:LCNTR=70, DO waitmore UNTIL LCE;
waitmore:   NOP;
        RTS;
        NOP;
        NOP;

/*****************************SEND_CMD Subroutine******************************/
/* This subroutine inserts the specified command word at the front of the     */
/* transmits queue so the command will be the next word transmitted.          */
/*                                                                            */
/* Input Requirements: R0=command word (7 LSBs).                              */
/* Output Conditions: Command word inserted at the front of the transmit      */
/*                    queue.                                                  */
/******************************************************************************/
send_cmd:R1=stype_command;          /* Insert the "command word" tag into     */
        R0=R0 OR FDEP R1 BY 29:3;   /* the appropriate field of the output    */
                                    /* word.                                  */
```

```
        R1=I14;                     /* If the transmit queue is empty, add    */
        R2=I15;                     /* output word to the queue in the normal */
        COMP(R1,R2);                /* fashion (call SENDDATA).               */
        IF EQ JUMP senddata;

                                /* If the transmit queue is not empty,        */
        R1=PM(I14,-1);          /*   -> move the current word being sent back */
        PM(I14,1)=R1;           /*      one queue slot.                       */
        PM(I14,-1)=R0;          /*   -> store the command word in the current */
                                /*      queue slot and move current slot      */
                                /*      pointer back one to the word than was */
                                /*      being transmitted when command was    */
                                /*      issued.                               */
        BIT SET IMASK IRQ1I;    /* Enable the 8251A transmit interrupt.       */
        RTS;

/*****************************SENDDATA Subroutine*****************************/
/* This subroutine adds the specified word to the end of the transmit queue.  */
/* Note, no error checking is performed for queue overrun.                    */
/*                                                                            */
/* Input Requirements: R0=output word (32 bits).                              */
/* Output Conditions: Output word added to the end of the transmit queue.     */
/******************************************************************************/
senddata:PM(I15,1)=R0;          /* Place the data to be transmitted at the    */
                                /*   end of the transmit queue.               */
        BIT SET IMASK IRQ1I;    /* Enable the 8251A transmit interrupt.       */
        RTS;

/*********************ACKNOW (acknowledge) Subroutine**************************/
/* This subroutine waits for the host to transmit an "acknowledge" command    */
/* word.                                                                      */
/*                                                                            */
/* Input Requirements: none.                                                  */
/* Output Conditions: "acknowledge" command received from host.               */
/******************************************************************************/
acknow:  R1=TRUE;               /* The receive_data interrupt service routine */
wait_ack:R0=PM(acknowledge);    /* will store TRUE in memory location         */
        COMP(R0,R1);            /* <acknowledge> whenever an "acknowledge"    */
        IF NE JUMP wait_ack;    /* command is received from the host.  If the */
        R0=FALSE;               /* <acknowledge> memory location does not     */
        PM(acknowledge)=R0;     /* contain TRUE, wait here.  When the         */
        RTS;                    /* <acknowledge> memory location does contain */
                                /* TRUE exit this subroutine.                 */

/*******************ALTERNCO (change NCO output) Subroutine********************/
/* This subroutine writes a new "delta phi" value to the PIR register of the  */
/* specified NCO channel to change the frequency of the corresponding output  */
/* waveform.  "delta phi" calculations are described in the NCO initialization*/
/* section of this software.                                                  */
/*                                                                            */
/* Input Requirements: <R0>=new delta phi value.                              */
/*                     <I1>=address of the LSB of the PIR register of the     */
/*                          desired NCO channel.                              */
/*                     <I3>=address of the hop clock register of the desired  */
/*                          NCO channel.                                      */
/*                     <L1>=<I1>=0 to prevent circular buffer operation.      */
/* Output Conditions: New "delta phi" value written to the specified PIR reg. */
/******************************************************************************/
alterNCO:BIT SET ASTAT FLG1|FLG2;     /* Deselect all peripherals except the  */
                                      /* NCO.                                 */

        BIT CLR ASTAT FLG0;           /* Select the NCO.                      */

        DMWAIT=dm_ws_NCO;             /* The ADSP21020-NCO interface requires */
                                      /* 1 wait state. Change the ADSP21020   */
                                      /* configuration to add a software      */
                                      /* wait state.                          */
```

```
        R10=-8;                         /* The ADSP21020-NCO interface is only  */
                                        /* 8 bits wide.  Hence, outputting the  */
                                        /* required 32 bit delta phi value      */
                                        /* requires 4 writes.  R10 is used to   */
                                        /* shift the desired byte of new delta  */
                                        /* phi value (R0) to the 8 LSBs of the  */
                                        /* output word.                         */

                                        /* Loop to output all 4 bytes of the    */
                                        /* new delta phi value.                 */
        LCNTR=4, DO wrt_pira UNTIL LCE;

           R1=DM(I1,0);                 /* The NCO requires a longer            */
                                        /* address set up time than the         */
                                        /* ADSP21020 can provide.  To           */
                                        /* compensate, a read is performed      */
                                        /* from the pending write address       */
                                        /* before the write is performed.       */
                                        /* This places the pending write        */
                                        /* address on the bus a full clock      */
                                        /* cycle before the write occurs.       */

wrt_pira:   R0=LSHIFT R0 BY R10,DM(I1,1)=R0;  /* Write the next byte of the   */
                                              /* new delta phi value to the   */
                                              /* next PIR register byte       */
                                              /* address and increment the    */
                                              /* address pointer.             */

        R0=DM(I3,0);                    /* Write any value to the NCO hop clock */
        DM(I3,0)=R0;                    /* register to force the NCO to use the */
                                        /* new delta phi value in the PIR       */
                                        /* register.                            */

        BIT SET ASTAT FLG0;             /* Deselect the NCO and return the      */
        DMWAIT=default_ws;              /* ADSP21020 data memory bus to its     */
        RTS;                            /* default configuration.               */

/***********TXRDY (serial port transmit) Interrupt Service Subroutine**********/
/* This subroutine processes the transmit ready interrupt generated by the    */
/* 8251A which indicates that the 8251A is ready to transmit another byte.    */
/* This subroutine outputs the next byte in the transmit queue to the 8251A.  */
/*                                                                            */
/* Input Requirements: <I14>=address of the front of the transmit queue.      */
/* Output Conditions: Next byte in the transmit queue output to the 8251A.    */
/******************************************************************************/
txrdy:  PM(TxStore_R0)=R0;              /* Store the contents of all registers  */
        PM(TxStore_R1)=R1;              /* altered by this subroutine in memory */
        PM(TxStore_R2)=R2;              /* so the register values can be        */
        PM(TxStore_R3)=R3;              /* restored at the completion of this   */
        PM(TxStore_ASTAT)=ASTAT;        /* routine.                             */

        BIT CLR MODE1 IRPTEN;           /* Disable all interrupts to prevent    */
                                        /* interrupt nesting.                   */

        DMWAIT=dm_ws_8251;              /* The ADSP21020-8251A interface        */
                                        /* requires 5 wait states. Change the   */
                                        /* ADSP21020 configuration to add 5     */
                                        /* software wait states.                */

        BIT SET ASTAT FLG0|FLG2;        /* Deselect all other peripherals and   */
        BIT CLR ASTAT FLG1;             /* select the 8251A.  (The ADSP21020    */
        NOP; NOP; NOP;                  /* FLG? outputs lag the ASTAT register  */
                                        /* values by 1 clock cycle.  Execute    */
                                        /* NOPs to wait for the FLG? output     */
                                        /* pins to change state.                */

        R1=DM(r_stat_8251);             /* At higher clock speeds, the          */
```

```
        R3=0x1;                         /* ADSP21020 executes and exits this    */
        R1=R1 AND R3;                   /* interrupt service routine before     */
        IF EQ JUMP qfull;               /* the 8251 TXRDY pin changes state     */
                                        /* to acknowledge the latest byte       */
                                        /* output by the ADSP21020.  This       */
                                        /* causes the ADSP21020 to regenerate   */
                                        /* the TXRDY interrupt and execute      */
                                        /* this routine again to output the     */
                                        /* next transmit byte before the 8251A  */
                                        /* can transmit the first byte.  To     */
                                        /* ensure the ADSP21020 does not        */
                                        /* overwrite the 8251A, poll the        */
                                        /* status of the 8251A to verify that   */
                                        /* the 8251A transmit buffer is empty.  */
                                        /* If the buffer is empty, continue to  */
                                        /* execute this service routine.  If    */
                                        /* the buffer is still full, exit this  */
                                        /* routine without changing the status  */
                                        /* of the transmit queue.               */

        R1=PM(I14,0);                   /* Read the word at the front of the    */
                                        /* transmit queue.                      */

        DM(w_dat_8251)=R1;              /* Output the 8 LSBs of the output word */
                                        /* to the 8251A.                        */

        R3=PM(shift_count);             /* <shift_count> counts the number of   */
        R3=R3+1;                        /* bytes in the current word that have  */
        R2=4;                           /* been transmitted.  If all 4 bytes of */
        COMP(R3,R2);                    /* the current word have been           */
        IF EQ JUMP doneshft;            /* jump to see if the transmit queue    */
        PM(shift_count)=R3;             /* is empty.  If not, increment         */
        R1=LSHIFT R1 BY -8;             /* <shift_counter>, shift the current   */
        PM(I14,0)=R1;                   /* output word right by 8 bits and      */
        JUMP qfull;                     /* store the shifted output word at the */
                                        /* front of the transmit queue.  Jump   */
                                        /* to the end of the service routine.   */

doneshft:R3=0;                          /* All 4 bytes of the current output    */
        PM(shift_count)=R3;             /* word have been transmitted.  Reset   */
        MODIFY(I14,1);                  /* the shift counter and increment the  */
                                        /* font-of-queue pointer.               */
doneyet: R1=I14;                        /* If the front- and end-of-queue       */
        R2=I15;                         /* pointers match, the queue is empty   */
        COMP(R2,R1);                    /* so disable the transmit interrupt.   */
        IF NE JUMP qfull;               /* If the queue is not empty jump to    */
        BIT CLR IMASK IRQ1I;            /* the end of the service routine.      */

qfull:  DMWAIT=default_ws;              /* Restore the state of the ADSP21020.  */
        R0=PM(TxStore_R0);
        R1=PM(TxStore_R1);
        R2=PM(TxStore_R2);
        R3=PM(TxStore_R3);
        ASTAT=PM(TxStore_ASTAT);
        BIT SET MODE1 IRPTEN;           /* Enable interrupts.                   */
        RTI;                            /* Exit the interrupt service routine.  */

/***********RXRDY (serial port receive) Interrupt Service Subroutine**********/
/* This subroutine processes the receive ready interrupt generated by the     */
/* 8251A which indicates the 8251A has received a byte transmitted by the     */
/* host.  This subroutine responds to the host commands by placing requested  */
/* information in the transmit queue or setting flags which will cause the    */
/* DRX to perform the requested function.                                     */
/*                                                                            */
/* Input Requirements: none.                                                  */
/* Output Conditions: received command processed.                             */
/******************************************************************************/
rxrdy:  PM(RxStore_ASTAT)=ASTAT;        /* Store the contents of all registers  */
```

```
        PM(RxStore_R0)=R0;          /* altered by this subroutine in memory */
        PM(RxStore_R1)=R1;          /* so the register values can be        */
        PM(RxStore_R2)=R2;          /* restored at the completion of this   */
        PM(RxStore_I0)=I0;          /* routine.                             */
        PM(RxStore_R7)=R7;
        PM(RxStore_R10)=R10;
        PM(RxStore_R11)=R11;
        PM(RxStore_R12)=R12;
        PM(RxStore_B1)=B1;
        PM(RxStore_I1)=I1;
        PM(RxStore_L1)=L1;
        PM(RxStore_B3)=B3;
        PM(RxStore_I3)=I3;
        PM(RxStore_L3)=L3;
        PM(RxStore_MODE1)=MODE1;

        BIT CLR MODE1 IRPTEN;       /* Disable all interrupts to prevent    */
                                    /* interrupt nesting.                   */

        DMWAIT=dm_ws_8251;          /* The ADSP21020-8251A interface        */
                                    /* requires 5 wait states. Change the   */
                                    /* ADSP21020 configuration to add 5     */
                                    /* software wait states.                */

        BIT SET ASTAT FLG0|FLG2;    /* Deselect all other peripherals and   */
        BIT CLR ASTAT FLG1;         /* select the 8251A.  (The ADSP21020    */
        NOP; NOP; NOP;              /* FLG? outputs lag the ASTAT register  */
                                    /* values by 1 clock cycle.  Execute    */
                                    /* NOPs to wait for the FLG? output     */
                                    /* pins to change state.                */

        R0=DM(r_dat_8251);          /* Read the contents of the 8251A       */
        R1=0xFF;                    /* receive buffer.  This buffer is only */
        R0=R0 AND R1;               /* 8 bits wide so clear the high 24     */
                                    /* bits of the data word.               */


        R1=PM(wait_spectrum_size);  /* The "generate spectrum" command is   */
        R2=TRUE;                    /* 2 bytes long.  If the first byte of  */
        COMP(R1,R2);                /* this command has been received,      */
        IF NE JUMP check_db;        /* <wait spectrum size> will contain    */
        R1=1;                       /* TRUE and the current byte will be    */
        R0=LSHIFT R1 BY R0;         /* the spectrum size code.  If the      */
        PM(spectrum_size)=R0;       /* DRX is not waiting for the second    */
        R0=FALSE;                   /* byte of the "generate spectrum" cmd, */
        PM(wait_spectrum_size)=R0;  /* check to see if the DRX is waiting   */
        R0=TRUE;                    /* for the second byte of the "change   */
        PM(generate_spectrum)=R0;   /* detection bandwidth" command.  If    */
        JUMP donerx;                /* the current byte is the second byte  */
                                    /* of the "generate spectrum " command, */
                                    /* store the spectrum size code in      */
                                    /* <spectrum_size>, clear the <wait_    */
                                    /* spectrum_size> flag and set the      */
                                    /* <generate_spectrum> flag to force    */
                                    /* the DRX to generate a spectrum of    */
                                    /* specified length upon returning      */
                                    /* to signal power measurement mode.    */
                                    /* Jump to the end of the interrupt     */
                                    /* service routine.                     */

check_db:R1=PM(wait_detection_bandwidth);  /* Check to see if the DRX is    */
        R2=TRUE;                    /* waiting for the second byte of the   */
        COMP(R1,R2);                /* "change detection bandwidth" command */
        IF NE JUMP check_S;         /* (<wait_detection_bandwidth>=TRUE?).  */
        PM(detection_bandwidth)=R0; /* If so, the current byte is the new   */
```

```
        R0=FALSE;                       /* detection bandwidth.  Store this     */
        PM(wait_detection_bandwidth)=R0;  /* value in <detection_bandwidth>, */
        JUMP donerx;                    /* clear the <wait_detection_bandwidth> */
                                        /* flag and jump to the end of the      */
                                        /* interrupt service routine.  If not,  */
                                        /* jump to see if the new byte is the   */
                                        /* start of a "generate_spectrum"       */
                                        /* command."                            */

check_S: R1=0x53;                       /* If the new byte='S' ("generate       */
        COMP(R0,R1);                    /* spectrum" command, set the <wait     */
        IF NE JUMP check_T;             /* spectrum size> flag to indicate that */
        R0=TRUE;                        /* the next byte received will be the   */
        PM(wait_spectrum_size)=R0;      /* spectrum size code and jump to the   */
        JUMP donerx;                    /* end of the interrupt service         */
                                        /* routine.  If not, jump to see if     */
                                        /* the new byte is the "track pause"    */
                                        /* command.                             */

check_T: R1=0x54;                       /* If the new byte='T' ("track pause"   */
        COMP(R0,R1);                    /* command), check to the current       */
        IF NE JUMP check_F;             /* status to the <tracking> flag to see */
        R0=PM(tracking);                /* if tracking is on or off.  If not,   */
        R1=TRUE;                        /* jump to see if the new byte is the   */
        COMP(R0,R1);                    /* "send current carrier frequency      */
        IF EQ JUMP trck_off;            /* estimate" command.                   */

trck_on: PM(tracking)=R1;               /* If tracking is currently disabled,   */
        R1=-1087;                       /* the new command should re-enable     */
        PM(sfft)=R1;                    /* tracking.  Hence, set the <tracking> */
        JUMP donerx;                    /* flag, and reset the complex sample   */
                                        /* counter <sfft> so the DRX will       */
                                        /* continue to make signal power        */
                                        /* measurements when the FIR filter/FFT */
                                        /* pipeline fills up.  A signal power   */
                                        /* measurement mode 40 kHz FFT is       */
                                        /* executed when <sfft>=25.  Initialize */
                                        /* <sfft> to -(length of first FIR      */
                                        /* filter stage+length of second FIR    */
                                        /* filter stage+length of comb filter+  */
                                        /* length of FFT)=-(30+22+2+1024)=-1078.*/
                                        /* Jump to the end of the interrupt     */
                                        /* service routine.                     */

                                        /* If tracking is currently enabled,    */
                                        /* the current command should disable   */
                                        /* tracking, the carrier should be      */
                                        /* shifted to 455 kHz to minimize the   */
                                        /* possibility of the carrier drifting  */
                                        /* out of the +/- 50 Hz FIR filter      */
                                        /* passbands while tracking is disabled */
                                        /* and place holder -71.92 dB signal    */
                                        /* power measurements should be         */
                                        /* generated.                           */

trck_off:R0=PM(current_fc);             /* Compute the mix frequency required   */
        F1=455000.0;                    /* to translate the carrier signal to   */
        F0=F0-F1;                       /* 455 KHz.                             */
        F1=4545000.0;
        F0=F0+F1;
                                        /* Update the NCO to generate the       */
        B1=mix_NCO_pira_base;           /* required mix frequency.  The         */
        L1=0;                           /* procedure to alter the NCO output    */
        B3=mix_NCO_hop_clk;             /* frequency is described in the NCO    */
        L3=0;                           /* initialization section above.        */
        F11=2.0; F12=4294967296.0;
        CALL divide (DB); F0=F0*F12; F12=NCO_ref;
        R0=FIX F0;
```

```
        PM(mix_NCO_dphi)=R0;
        CALL alterNCO;

        R0=FALSE;                   /* Clear the <tracking> flag to        */
        PM(tracking)=R0;            /* indicate that carrier tracking is   */
                                    /* disabled.                           */

                                    /* Place a "dummy" -71.92 dB signal    */
                                    /* power measurement word in the       */
                                    /* <sample_1Hz> and <sample_20Hz>      */
                                    /* memory locations.  (The power       */
                                    /* measurement information contained   */
                                    /* in these memory locations are       */
                                    /* output when the DRX is polled for   */
                                    /* power measurement information).     */

        R0=8191;                    /* -(-71.92-10.00)*100=8192.           */
        R1=status_track_dis;        /* -> Add the "tracking disabled DRX   */
        R0=R0 OR FDEP R1 BY 26:3;   /*    status tag to the output word.   */
        R1=PM(detection_bandwidth); /* -> Add the current detection        */
        R0=R0 OR FDEP R1 BY 13:6;   /*    bandwidth tag to the output word.*/
        R1=stype_1Hz;               /* -> Add the 1 Hz power measurement   */
        R0=R0 OR FDEP R1 BY 29:3;   /*    sample type tag to the output    */
                                    /*    word.                            */
        PM(sample_1Hz)=R0;          /* -> Store the -71.92 dB, 1 Hz output */
                                    /*    word.                            */
        R1=stype_20Hz;              /* -> Add the 20 Hz power measurement  */
        R0=R0 OR FDEP R1 BY 29:3;   /*    sample type tag to the output    */
                                    /*    word.                            */
        PM(sample_20Hz)=R0;         /* -> Store the -71.92 dB, 20 Hz output*/
                                    /*    word.                            */
        JUMP donerx;                /* Jump to the end of the interrupt    */
                                    /* service routine.                    */

                                    /* If the new byte='F' ("send current  */
                                    /* carrier frequency estimate" command),*/
                                    /* format the latest carrier frequency */
                                    /* estimate for output.  If not, jump  */
                                    /* to see if the new byte is an        */
check_F: R1=0x46;                   /* acknowledge command.                */
        COMP(R0,R1);                /*                                     */
        IF NE JUMP check_A;         /* Format the carrier frequency        */
        F0=PM(current_fc);          /* estimate word for output:           */
        R0=FIX F0;                  /*   -> convert to fixed point format  */
        R1=0xFFFFF;                 /*   -> clear bits above the actual    */
        R0=R0 AND R1;               /*      carrier frequency estimate     */
                                    /*      field                          */
        R1=PM(mix_NCO_delta);       /*   -> if the NCO mix output frequency*/
        R2=0;                       /*      is currently being slewed,     */
        COMP(R1,R2);                /*      place a 1 in the "slew NCO"    */
        IF EQ JUMP notslew;         /*      field.                         */
        R2=1;                       /*                                     */
notslew: R0=R0 OR FDEP R2 BY 20:1;  /*                                     */
        R2=stype_frequency;         /*   -> deposit sample type tag to     */
        R0=R0 OR FDEP R2 BY 29:3;   /*      identify word as a carrier     */
                                    /*      frequency estimate             */
        R2=PM(DRX_status);          /*                                     */
        R0=R0 OR FDEP R2 BY 26:3;   /*   -> fill the DRX status tag field  */
                                    /*      with the current DRX status    */

        CALL senddata;              /* Place the carrier frequency estimate*/
        JUMP donerx;                /* in the transmit queue and jump to   */
                                    /* the end of the interrupt service    */
                                    /* routine.                            */

check_A: R1=0x41;                   /* If the new byte='A' ("acknowledge   */
        COMP(R0,R1);                /* command") set the <acknowledge> flag*/
        IF NE JUMP check_B;         /* to indicate that the "acknowledge   */
```

```
        R0=TRUE;                        /* command was received and jump to     */
        PM(acknowledge)=R0;             /* the end of the interrupt service     */
        JUMP donerx;                    /* routine.  If not, jump to see if the */
                                        /* new byte is the start of a "change   */
                                        /* detection bandwidth command."        */

check_B: R1=0x42;                       /* If the new byte='B' ("change         */
        COMP(R0,R1);                    /* detection bandwidth command"), set   */
        IF NE JUMP check_H;             /* the <wait_detection bandwidth> flag  */
        R0=TRUE;                        /* to indicate that the DRX is waiting  */
        PM(wait_detection_bandwidth)=R0;  /* for the second byte of the         */
        JUMP donerx;                    /* command and jump to the end of the   */
                                        /* interrupt service routine.  If not,  */
                                        /* jump to see if the new byte is a     */
                                        /* "force reacquisition" command.       */

check_H: R1=0x48;                       /* If the new byte='H' ("force          */
        COMP(R0,R1);                    /* reacquisition" command), set the     */
        IF NE JUMP check_1;             /* <force reacquisition> flag to        */
        R0=TRUE;                        /* indicate that the DRX should         */
        PM(force_acq)=R0;               /* re-enter carrier signal acquisition  */
        JUMP donerx;                    /* mode after exiting the interrupt     */
                                        /* service routine.  If not, jump to    */
                                        /* see if the new byte is a "send       */
                                        /* 1 Hz power measurement sample"       */
                                        /* command.                             */

                                        /* If the new byte is a "send 1 Hz      */
                                        /* signal power measurement sample      */
                                        /* command, format the latest           */
                                        /* 1 Hz signal power measurement for    */
                                        /* output and place it in the data      */
                                        /* transmission queue.  If not, jump    */
                                        /* to see if the new byte is a "send    */
                                        /* 20 Hz signal power measurement       */
                                        /* sample" command.                     */

check_1: R1=0x31;                       /* Format the output word:              */
        COMP(R0,R1);                    /*   -> retrieve the latest 1 Hz signal */
        IF NE JUMP check_2;             /*      power measurement  (note, this  */
        R0=PM(sample_1Hz);              /*      data is already in fixed point  */
                                        /*      format, ready to be output).    */
        R1=PM(time_stamp_1Hz);          /*   -> add the time stamp field to the */
        R0=R0 OR FDEP R1 BY 19:7;       /*      output word, and place the      */
        CALL senddata;                  /*      output word in the data         */
                                        /*      transmission queue.             */
        R1=R1+1;                        /*   -> increment the time stamp        */
        R0=0x7F;                        /*      (modulo 7) and store it for     */
        R1=R1 AND R0;                   /*      later use                       */
        PM(time_stamp_1Hz)=R1;
        JUMP donerx;                    /* Jump to the end of the interrupt     */
                                        /* service routine.                     */

                                        /* If the new byte is a "send 20 Hz     */
                                        /* signal power measurement sample      */
                                        /* command, format the latest           */
                                        /* 20 Hz signal power measurement for   */
                                        /* output and place it in the data      */
                                        /* transmission queue.  If not, the     */
                                        /* command is invalid so ignore it.     */

check_2: R1=0x31;                       /* Format the output word:              */
        COMP(R0,R1);                    /*   -> retrieve the latest 1 Hz signal */
        IF NE JUMP donerx;              /*      power measurement  (note, this  */
        R0=PM(sample_20Hz);             /*      data is already in fixed point  */
                                        /*      format, ready to be output).    */
        R1=PM(time_stamp_20Hz);         /*   -> add the time stamp field to the */
        R0=R0 OR FDEP R1 BY 19:7;       /*      output word, and place the      */
```

```
        CALL senddata;                  /*      output word in the data         */
                                        /*      transmission queue.             */
        R1=R1+1;                        /*   -> increment the time stamp        */
        R0=0x7F;                        /*      (modulo 7) and store it for     */
        R1=R1 AND R0;                   /*      later use                       */
        PM(time_stamp_20Hz)=R1;
        JUMP donerx;                    /* Jump to the end of the interrupt     */
                                        /* service routine.                     */

donerx: DMWAIT=default_ws;              /* Restore the state of the ADSP21020.  */
        ASTAT=PM(RxStore_ASTAT);
        R0=PM(RxStore_R0);
        R1=PM(RxStore_R1);
        R2=PM(RxStore_R2);
        I0=PM(RxStore_I0);
        R7=PM(RxStore_R7);
        R10=PM(RxStore_R10);
        R11=PM(RxStore_R11);
        R12=PM(RxStore_R12);
        B1=PM(RxStore_B1);
        I1=PM(RxStore_I1);
        L1=PM(RxStore_L1);
        B3=PM(RxStore_B3);
        I3=PM(RxStore_I3);
        L3=PM(RxStore_L3);
        MODE1=PM(RxStore_MODE1);
        BIT SET MODE1 IRPTEN;           /* Enable interrupts.                   */
        RTI;                            /* Exit the interrupt service routine.  */

/***********FIFOFULL (FIFO almost full) Interrupt Service Subroutine**********/
/* This subroutine processes the "almost full" interrupt generated by the     */
/* FIFO when it is 7/8ths full.  The routine reads ADC samples from the FIFO  */
/* and processes them according what mode the DRX is in.  If the DRX is in    */
/* carrier signal acquisition mode or spectrum analyzer mode, new complex     */
/* samples are placed into memory in bit reversed order (real components of   */
/* complex samples at addresses 0 to N-1, complex components at addresses N   */
/* to 2N-1).  If the DRX is in signal power measurement mode, new complex     */
/* samples are passed through a "dc kill" comb filter, and two FIR            */
/* filter/decimate stages which bandlimit and decimate the complex sample     */
/* stream to 1.011 kHz.  This process generates a 1.011 kHz complex samples   */
/* (which is added to a circular buffer) each time this subroutine is         */
/* executed.                                                                  */
/*                                                                            */
/* Input Requirements:                                                        */
/*    (carrier signal acquisition and spectrum analyzer modes):               */
/*                     <I0>=<B0>=base address for all data storage=0 at the   */
/*                              start of the storage process.                 */
/*                     <M2>=size of FFT to be performed                       */
/*                     <M0>=2^(32-logb2(<M2>))                                */
/*                     <R0>=complex sample counter=0 at the start of the      */
/*                          storage process.                                  */
/*                                                                            */
/*    (signal power measurement mode):                                        */
/*                     <s5K>,<sfft>=decimation/sample counters=0 at the       */
/*                          start of the data storage process.                */
/*                     FIR filter memory (circular buffer) status stored in   */
/*                          the memory locations identified in the code below.*/
/*                                                                            */
/* Output Conditions: -> new ADC samples read from the FIFO and processed     */
/*                       according to the DRX status.                         */
/*                     -> R0 or <sfft> updated to reflect the number of       */
/*                       complex samples added to the particular output       */
/*                       stream.                                              */
/******************************************************************************/
fifofull:PM(FifoStore_ASTAT)=ASTAT;     /* Store the contents of all registers  */
         PM(FifoStore_R9)=R9;           /* altered by this subroutine in memory */
         PM(FifoStore_R13)=R13;         /* so the register values can be        */
         PM(FifoStore_R14)=R14;         /* restored at the completion of this   */
```

```
        PM(FifoStore_R15)=R15;        /* routine.                              */

        BIT CLR MODE1 IRPTEN;         /* Disable all interrupts to prevent     */
                                      /* interrupt nesting.                    */

        R14=0x000000000FFF;           /* ADC data is 12 bits wide and is in    */
        R15=2047;                     /* offset linear format.  The constants  */
                                      /* 0xFFF and 2047 are used to convert    */
                                      /* ADC data to IEEE fixed point format.  */

        R13=PM(DRX_mode);             /* If the DRX is in signal power         */
        R9=mode_power_meas;           /* measurement mode jump to the section  */
        COMP(R9,R13);                 /* of this routine which filters the     */
        IF EQ JUMP fifo_pm;           /* new samples as described above.       */

        /*-------------------------------------------------------------------*/
        /* Service routine for carrier signal acquisition and spectrum       */
        /* analyzer modes.                                                   */
        /*-------------------------------------------------------------------*/
fifo_asm:PM(FifoStore_I1)=I1;         /* Store the contents of all registers   */
        PM(FifoStore_MODE1)=MODE1;    /* altered by this subroutine in memory  */
        PM(FifoStore_R12)=R12;        /* so the register values can be         */
        PM(FifoStore_R11)=R11;        /* restored at the completion of this    */
        PM(FifoStore_R10)=R10;        /* routine.                              */

        BIT SET ASTAT FLG0|FLG1;      /* Deselect all other peripherals and    */
        BIT CLR ASTAT FLG2;           /* select the FIFO.                      */

        BIT SET MODE1 BR0;            /* Enable ADSP21020 automatic bit        */
                                      /* reversed addressing.                  */

        /*...................................................................*/
        /* Read 256 ADC samples (128 complex samples) from the FIFO and      */
        /* store them in bit reversed order in the memory ranges denoted     */
        /* above.                                                            */
        /*                                                                   */
        /* The bit reversed addresses for new samples are computed in the    */
        /* following fashion: This simple example assumes the DRX will be    */
        /* executing an 16 point FFT.                                        */
        /*                                                                   */
        /*   -> initial conditions: I0=B0=0                                  */
        /*                          M2=16                                    */
        /*                          M0=2^(32-logb2(16))=2^28                 */
        /*                            =0x10000000                            */
        /*                          automatic bit reversed addressing        */
        /*                             using I0 enabled                      */
        /*                                                                   */
        /*   -> Recall, BITREV operates on 32 bit addresses                  */
        /*                                                                   */
        /*   * imaginary component of 1st complex sample must be stored      */
        /*     at address:                                                   */
        /*         bit reverse(0x00)=0x00+16=BITREV(I0)+M2                   */
        /*                                  =0x00+16                         */
        /*                                        ^                          */
        /*                                        |                          */
        /*          offset to complex data space -                           */
        /*                                                                   */
        /*   * real component of 1st complex sample must be stored           */
        /*     at address                                                    */
        /*         bit reverse(0x00)=0x00=(auto)BITREV(I0)=0x00              */
        /*                                                                   */
        /*   * I0 is then updated by M0 -> I0=I0+M0=0x10000000               */
        /*                                                                   */
        /*   * imaginary component of 2nd complex sample must be stored      */
        /*     at address:                                                   */
        /*         bit reverse(0x01)=0x80+8=BITREV(I0)+M2                    */
        /*                                 =0x80+8                           */
        /*                                                                   */
```

```
        /*   * real component of 2nd complex sample must be stored          */
        /*     at address                                                    */
        /*         bit reverse(0x01)=0x80=(auto)BITREV(I0)=0x80              */
        /*                                                                   */
        /*   * I0 is then updated by M0 -> I0=I0+M0=0x20000000               */
        /*                          .                                        */
        /*                          .                                        */
        /*                          .                                        */
        /* This addressing process is repeated until 16 new complex data     */
        /* points have been stored                                           */
        /*.....................................................................*/
        LCNTR=256, DO rdfifo1 UNTIL LCE;
           R13=DM(r_fifo);                 /* Read the next ADC sample.      */
                                           /*     (imaginary component of    */
                                           /*     the next complex sample)   */
           R13=R13 AND R14;                /* The output of the ADC is 12    */
                                           /*     bits wide.  Clear all bits */
                                           /*     above bits 0 through 11    */
                                           /*     of the new sample.         */
           R13=R13-R15;                    /* The output of the ADC is in    */
                                           /*     offset linear format.      */
                                           /*     Subtracting 2047 (R15)     */
                                           /*     converts the ADC output    */
                                           /*     to a +/- 2048 span         */
                                           /*     representing the +/- 2.53v */
                                           /*     ADC input range.           */
           F13=FLOAT R13;                  /* Convert to floating point      */
                                           /*     format.                    */
           I1=I0;                          /* Store the imaginary component  */
           BITREV(I1,0);                   /*     at the bit reversed        */
           DM(M2,I1)=F13;                  /*     address described above.   */
           R13=DM(r_fifo);                 /* Read the next ADC sample.      */
                                           /*     (imaginary component of    */
                                           /*     the next complex sample)   */
           R13=R13 AND R14;                /* The output of the ADC is 12    */
                                           /*     bits wide.  Clear all bits */
                                           /*     above bits 0 through 11    */
                                           /*     of the new sample.         */
           R13=R13-R15;                    /* The output of the ADC is in    */
                                           /*     offset linear format.      */
                                           /*     Subtracting 2047 (R15)     */
                                           /*     converts the ADC output    */
                                           /*     to a +/- 2048 span         */
                                           /*     representing the +/- 2.53v */
                                           /*     ADC input range.           */
           F13=FLOAT R13;                  /* Convert to floating point      */
                                           /*     format.                    */
           DM(I0,M0)=F13;                  /* Store the imaginary component  */
                                           /*     at the bit reversed        */
                                           /*     address described above.   */
rdfifo1:   R0=R0+1;                        /* Increment the number of        */
                                           /*     complex samples stored.    */

        MODE1=PM(FifoStore_MODE1);   /* Restore the state of the ADSP21020.  */
        ASTAT=PM(FifoStore_ASTAT);
        R12=PM(FifoStore_R12);
        R11=PM(FifoStore_R11);
        R10=PM(FifoStore_R10);
        R9=PM(FifoStore_R9);
        R13=PM(FifoStore_R13);
        R14=PM(FifoStore_R14);
        R15=PM(FifoStore_R15);
        I1=PM(FifoStore_I1);
        BIT SET MODE1 IRPTEN;        /* Enable interrupt processing.         */
        RTI;                         /* Exit the interrupt service routine.  */

        /*---------------------------------------------------------------------*/
        /* Service routine for signal power measurement mode.                  */
```

```
        /*-------------------------------------------------------------------*/
fifo_pm: PM(FifoStore_B1)=B1;          /* Store the contents of all registers  */
        PM(FifoStore_I1)=I1;           /* altered by this subroutine in memory */
        PM(FifoStore_L1)=L1;           /* so the register values can be        */
        PM(FifoStore_B3)=B3;           /* restored at the completion of this   */
        PM(FifoStore_I3)=I3;           /* routine.                             */
        PM(FifoStore_L3)=L3;
        PM(FifoStore_B6)=B6;
        PM(FifoStore_I6)=I6;
        PM(FifoStore_L6)=L6;
        PM(FifoStore_B7)=B7;
        PM(FifoStore_I7)=I7;
        PM(FifoStore_L7)=L7;
        PM(FifoStore_B10)=B10;
        PM(FifoStore_I10)=I10;
        PM(FifoStore_L10)=L10;
        PM(FifoStore_B12)=B12;
        PM(FifoStore_I12)=I12;
        PM(FifoStore_L12)=L12;
        PM(FifoStore_B13)=B13;
        PM(FifoStore_I13)=I13;
        PM(FifoStore_L13)=L13;
        PM(FifoStore_R0)=R0;
        PM(FifoStore_R1)=R1;
        PM(FifoStore_R3)=R3;
        PM(FifoStore_R4)=R4;
        PM(FifoStore_R10)=R10;
        PM(FifoStore_R11)=R11;
        PM(FifoStore_R12)=R12;

        R9=-12;                        /* R9 stores an empirically determined  */
                                       /* scaling factor used in the fixed     */
                                       /* floating point conversions performed */
                                       /* on each new complex data sample.     */

                                       /* Retrieve the status of the FIR       */
                                       /* filter used to bandlimit the         */
                                       /* incoming (25.278 kHz) complex data   */
                                       /* stream to 5 kHz.                     */
        B7=r25K_dat; I7=PM(r25K_I); L7=len_25K;
        B13=i25K_dat; I13=PM(i25K_I); L13=len_25K;

        BIT SET ASTAT FLG0|FLG1;       /* Deselect all other peripherals and   */
        BIT CLR ASTAT FLG2;            /* select the FIFO.                     */

                                       /* 5 new 25.278 kHz complex samples     */
                                       /* are accumulated and then the 5K      */
                                       /* bandlimiting filter is executed on   */
                                       /* the 22 most recent complex samples   */
                                       /* to produce a new 5.055k complex      */
                                       /* sample                               */
        LCNTR=5, DO gen1_5K UNTIL LCE;

                                       /* The analog IF waveform is            */
                                       /* bandlimited to 10 kHz by the analog  */
                                       /* filter on the IF board.  Decimate    */
                                       /* the incoming 303.333 kHz sample      */
                                       /* stream by 12 to 25.278 kHz (a        */
                                       /* slightly higher than theoretically   */
                                       /* necessary sample rate is maintained  */
                                       /* to ensure the stop bands of the      */
                                       /* 10 kHz IF filter have rolled off     */
                                       /* significantly before aliasing        */
                                       /* occurs).  The decimation is          */
                                       /* performed by discarding 20 ADC       */
                                       /* samples, pairing the next 4 time-    */
                                       /* adjacent ADC samples to form 2       */
                                       /* complex samples and executing a      */
```

```
                                    /* comb filter with 2 coefficients      */
                                    /* to produce a 25.278 kHz complex      */
                                    /* sample.                              */

            LCNTR=20, DO decimate UNTIL LCE;   /* Discard 20 ADC samples.     */
decimate:      R13=DM(r_fifo);

            R12=DM(r_fifo);         /* Read the next ADC sample (real       */
                                    /* component of the 1st complex sample) */

            R12=R12 AND R14;        /* The output of the ADC is 12 bits     */
                                    /* wide. Clear all bits above bits 0    */
                                    /* through 11 of the new sample.        */

                                    /* The output of the ADC is in offset   */
                                    /* linear format.  Subtracting 2047     */
                                    /* converts the ADC output to a         */
                                    /* +/- 2048 span representing the       */
                                    /* +/- 2.53v ADC input range.  This     */
                                    /* This operation is not performed      */
                                    /* however because the offsets cancel   */
                                    /* on another in the comb filtering     */
                                    /* process described below.             */

            R11=DM(r_fifo);         /* Read the next ADC sample (imaginary  */
                                    /* component of the 1st complex sample) */

            R11=R11 AND R14;        /* The output of the ADC is 12 bits     */
                                    /* wide. Clear all bits above bits 0    */
                                    /* through 11 of the new sample.        */

            R13=DM(r_fifo);         /* Read the next ADC sample (real       */
                                    /* component of the 2nd complex sample) */

            R13=R13 AND R14;        /* The output of the ADC is 12 bits     */
                                    /* wide. Clear all bits above bits 0    */
                                    /* through 11 of the new sample.        */

                                    /* The comb filter used to kill any     */
                                    /* dc offset in the input signal has    */
                                    /* two coefficients: +1, -1.            */

            R13=-R13;               /* Execute the comb filter on the real  */
            R13=(R12+R13)/2;        /* components of the latest 2           */
                                    /* 303.333 kHz complex samples.         */

            F13=FLOAT R13 BY R9;    /* Convert to floating point format.    */

            DM(I7,1)=F13;           /* Store the real component of the new  */
                                    /* 25.278 kHz complex data sample       */
                                    /* in the FIR filter (25.278 kHz->      */
                                    /* 5.0555 kHz) memory.                  */

            R13=DM(r_fifo);         /* Read the next ADC sample (imaginary  */
                                    /* component of the 2nd complex sample) */

            R13=R13 AND R14;        /* The output of the ADC is 12 bits     */
                                    /* wide. Clear all bits above bits 0    */
                                    /* through 11 of the new sample.        */

            R13=-R13;               /* Execute the comb filter on the       */
            R13=(R11+R13)/2;        /* imaginary components of the latest 2 */
                                    /* 303.333 kHz complex samples.         */

            F13=FLOAT R13 BY R9;    /* Convert to floating point format.    */

gen1_5K:    PM(I13,1)=F13;          /* Store the imaginary component of the */
                                    /* new 25.278 kHz complex data sample   */
```

```
BIT SET ASTAT FLG2;          /* in the FIR filter (25.278 kHz->      */
                             /* 5.055 kHz) memory.                   */

                             /* Store the status of the 25.278 kHz-> */
                             /* 5.055 kHz filter memory.             */
PM(r25K_I)=I7; PM(i25K_I)=I13;

L10=0;                       /* Execute the 25.278 kHz -> 5.055 kHz  */
CALL CFIR (DB);  B10=cfs_25K;  LCNTR=len_25K/2;   /* filter on the   */
                             /* latest 22 25.278 kHz complex data    */
                             /* samples to produce a new 5.055 kHz   */
                             /* complex sample.                      */

                             /* Retrieve the status of the 5.055 kHz */
                             /* -> 1.011 kHz filter memory.          */
B7=r5K_dat; I7=PM(r5K_I); L7=len_5K;

                             /* Store the new 5.055 kHz complex      */
                             /* sample in the filter memory.         */
B13=i5K_dat; I13=PM(i5K_I); L13=len_5K;
DM(I7,M7)=F9, PM(I13,M13)=F10;

PM(r5K_I)=I7; PM(i5K_I)=I13; /* Store the status of the 5.055 kHz -> */
                             /* 1.011 kHz filter memory.             */

R9=PM(s5K);                  /* If 5 new 5.055 kHz complex samples   */
R9=R9+1;                     /* have not accrued since the last      */
PM(s5K)=R9;                  /* time the 5.055 kHz -> 1.011 kHz      */
R10=5;                       /* filter was executed, must wait for   */
COMP(R9,R10);                /* more points to accrue so jump to     */
IF NE JUMP doneopm;          /* the end of the service routine. If   */
R10=0;                       /* 5 new 5.055 kHz complex samples      */
PM(s5K)=R10;                 /* have been generated, reset the       */
                             /* decimation counter and execute the   */
                             /* 5.0555 kHz -> 1.011 kHz filter       */
                             /* to produce a new 1.011 kHz complex   */
                             /* sample.                              */
CALL CFIR (DB);, B10=cfs_5K;  LCNTR=len_5K/2;

B7=trkfft; I7=PM(trkfft_I); L7=2048;
DM(2048,I7)=F10;             /* Store the new 1.011 kHz complex      */
DM(I7,1)=F9;                 /* sample in the signal power           */
PM(trkfft_I)=I7;             /* measurement mode FFT memory.  Note,  */
                             /* the complex components are stored    */
                             /* in a circular buffer offset from     */
                             /* the real components circular         */
                             /* buffer by 2048.                      */

R9=PM(sfft);                 /* Increment the 1.011 kHz complex      */
R9=R9+1;                     /* sample counter to reflect the        */
PM(sfft)=R9;                 /* new addition to the FFT memory.      */

/*..................................................................*/
/* To maintain the mix output frequency, the DRX writes the         */
/* delta phi parameter to the NCO at a rate of 1.011 kHz.  When the */
/* NCO output should be stable, the DRX writes the same delta phi   */
/* value to the NCO repeatedly.  When the mix output frequency is   */
/* being slewed to maintain lock, the delta phi parameter is        */
/* incremented by 1 NCO tick (reference freq/2^32) each time a      */
/* a write to the NCO occurs.  Compute and write the current        */
/* delta phi value to the NCO.                                      */
/*..................................................................*/
R0=PM(mix_NCO_dphi);         /* Retrieve the current value of delta */
                             /* phi.                                 */

R1=PM(mix_NCO_delta);        /* Retrieve any increment that should   */
                             /* applied to delta phi.                */
```

```
        R0=R0+R1;                       /* Apply the increment.                */

        PM(mix_NCO_dphi)=R0;            /* Write the new delta phi value to    */
        B1=mix_NCO_pira_base;           /* the NCO mix channel.                */
        L1=0;
        B3=mix_NCO_hop_clk;
        L3=0;
        CALL alterNCO;

doneopm: ASTAT=PM(FifoStore_ASTAT);     /* Restore the state of the ADSP21020. */
        R9=PM(FifoStore_R9);
        R13=PM(FifoStore_R13);
        R14=PM(FifoStore_R14);
        R15=PM(FifoStore_R15);
        B1=PM(FifoStore_B1);
        I1=PM(FifoStore_I1);
        L1=PM(FifoStore_L1);
        B3=PM(FifoStore_B3);
        I3=PM(FifoStore_I3);
        L3=PM(FifoStore_L3);
        B6=PM(FifoStore_B6);
        I6=PM(FifoStore_I6);
        L6=PM(FifoStore_L6);
        B7=PM(FifoStore_B7);
        I7=PM(FifoStore_I7);
        L7=PM(FifoStore_L7);
        B10=PM(FifoStore_B10);
        I10=PM(FifoStore_I10);
        L10=PM(FifoStore_L10);
        B12=PM(FifoStore_B12);
        I12=PM(FifoStore_I12);
        L12=PM(FifoStore_L12);
        B13=PM(FifoStore_B13);
        I13=PM(FifoStore_I13);
        L13=PM(FifoStore_L13);
        R0=PM(FifoStore_R0);
        R1=PM(FifoStore_R1);
        R3=PM(FifoStore_R3);
        R4=PM(FifoStore_R4);
        R10=PM(FifoStore_R10);
        R11=PM(FifoStore_R11);
        R12=PM(FifoStore_R12);
        BIT SET MODE1 IRPTEN;           /* Enable interrupt processing.        */
        RTI;                            /* Exit the interrupt service routine. */

/*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/
/*!!!                 Variable Data Space Declarations                  !!!*/
/*!!!                                                                   !!!*/
/*!!!  Note, two different PM data segments have been defined to minimize !!!*/
/*!!!  download time to the APT receiver hardware.  The ADSP21020      !!!*/
/*!!!  assembler fills all uninitialized variable space with 0s.  Hence, !!!*/
/*!!!  all of these 0s would be transmitted to the receiver.  By placing !!!*/
/*!!!  all of the uninitialized variables in a separate segment and    !!!*/
/*!!!  preventing that segment from being downloaded, the time required !!!*/
/*!!!  for the download process is significantly reduced.              !!!*/
/*!!!                                                                   !!!*/
/*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/
/*------------------Initialized Program Memory Variables--------------------*/
/* All initialized variables are stored in this segment.  This is the only */
/* data segment included in the download process.                          */
/*-------------------------------------------------------------------------*/
pm_vars:
.VAR pi=3.14159265358979323;
.VAR temp;
.VAR cfs_25K[len_25K/2]=3.669023E-003, 7.221078E-003, 1.330527E-002,
                        2.158396E-002, 3.186949E-002, 4.363192E-002,
                        5.602615E-002, 6.796902E-002, 7.828580E-002,
                        8.588281E-002, 8.991291E-002;
```

```
.VAR cfs_5K[len_5K/2]=-3.606767E-003, -4.670348E-003, -6.507331E-003,
                      -7.541097E-003, -6.894052E-003, -3.690187E-003,
                       2.793725E-003,  1.292283E-002,  2.654671E-002,
                       4.292708E-002,  6.076767E-002,  7.835168E-002,
                       9.377586E-002,  1.052435E-001,  1.113584E-001;
.ENDSEG;

/*------------------Uninitialized Program Memory Variables--------------------*/
/* This segment is not included in the download process.                      */
/*----------------------------------------------------------------------------*/
.SEGMENT/PM pmdata;
.VAR hanning[1024];
.VAR temp1;
.VAR temp2;
.VAR temp3;
.VAR temp4;
.VAR squeue [scomm_xmit_q_len];
.VAR fc_est[3];
.VAR freq_base;
.VAR DRX_mode;
.VAR i25K_dat[len_25K];
.VAR i5K_dat[len_5K];
.VAR mix_NCO_dphi;
.VAR mix_NCO_delta;
.VAR sample_1Hz;
.VAR sample_20Hz;
.VAR generate_spectrum;
.VAR spectrum_size;
.VAR current_fc;
.VAR detection_bandwidth;
.VAR shift_count;
.VAR wait_spectrum_size;
.VAR acknowledge;
.VAR time_stamp_1Hz;
.VAR time_stamp_20Hz;

.VAR FifoStore_ASTAT;
.VAR FifoStore_MODE1;
.VAR FifoStore_R9;
.VAR FifoStore_R13;
.VAR FifoStore_R14;
.VAR FifoStore_R15;
.VAR FifoStore_B1;
.VAR FifoStore_I1;
.VAR FifoStore_L1;
.VAR FifoStore_B3;
.VAR FifoStore_I3;
.VAR FifoStore_L3;
.VAR FifoStore_B6;
.VAR FifoStore_I6;
.VAR FifoStore_L6;
.VAR FifoStore_B7;
.VAR FifoStore_I7;
.VAR FifoStore_L7;
.VAR FifoStore_B10;
.VAR FifoStore_I10;
.VAR FifoStore_L10;
.VAR FifoStore_B12;
.VAR FifoStore_I12;
.VAR FifoStore_L12;
.VAR FifoStore_B13;
.VAR FifoStore_I13;
.VAR FifoStore_L13;
.VAR FifoStore_R0;
.VAR FifoStore_R1;
.VAR FifoStore_R3;
.VAR FifoStore_R4;
```

```
.VAR FifoStore_R10;
.VAR FifoStore_R11;
.VAR FifoStore_R12;

.VAR RxStore_ASTAT;
.VAR RxStore_MODE1;
.VAR RxStore_R0;
.VAR RxStore_R1;
.VAR RxStore_R2;
.VAR RxStore_R7;
.VAR RxStore_R10;
.VAR RxStore_R11;
.VAR RxStore_R12;
.VAR RxStore_I0;
.VAR RxStore_B1;
.VAR RxStore_I1;
.VAR RxStore_L1;
.VAR RxStore_B3;
.VAR RxStore_I3;
.VAR RxStore_L3;

.VAR TxStore_R0;
.VAR TxStore_R1;
.VAR TxStore_R2;
.VAR TxStore_R3;
.VAR TxStore_ASTAT;
.VAR wait_detection_bandwidth;
.VAR sfft;
.VAR r25K_I;
.VAR i25K_I;
.VAR r5K_I;
.VAR i5K_I;
.VAR trkfft_I;
.VAR s20;
.VAR s5K;
.VAR force_acq;
.VAR tracking;
.VAR old_fc;
.VAR variance;
.VAR delta_fc_sum;
.VAR delta_fc_avg[40];
.VAR delta_fc_I;
.VAR low_SNR_counter;
.VAR DRX_status;
.VAR dB30octave_20_coefs[13];
.VAR dB30octave_20_dat[13];
.VAR dB30octave_1_coefs[140];
.VAR dB30octave_1_dat[140];
.VAR dB30octave_20_I;
.VAR dB30octave_1_I;
.VAR s1;
.ENDSEG;

/*--------------------Uninitialized Data Memory Variables--------------------*/
/* This segment is not included in the download process.                      */
/*----------------------------------------------------------------------------*/
.SEGMENT/DM dmdata;
.VAR trkfft[4096];
.VAR scratch[2048];
.VAR r25K_dat[len_25K];
.VAR r5K_dat[len_5K];
.ENDSEG;
```

C.2 Application Code Architecture File Source Code Listing

```
/*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/
/*!!!                         APT RECEIVER SOFTWARE                         !!!*/
/*!!!                         ---------------------                         !!!*/
/*!!!                    APT RECEIVER ARCHITECTURE FILE                     !!!*/
/*!!!                                                                       !!!*/
/*!!!                        William R. Sylvester Jr.                       !!!*/
/*!!!                     Satellite Communications Group                    !!!*/
/*!!!                          621 Whittemore Hall                          !!!*/
/*!!!          Virginia Polytechnic Institute and State University          !!!*/
/*!!!                        Blacksburg, VA  24061                          !!!*/
/*!!!                            (703) 231-6834                             !!!*/
/*!!!                                                                       !!!*/
/*!!!                             August, 1992                              !!!*/
/*!!!=======================================================================!!!*/
/*!!! * The APT receiver configuration listed below is:                     !!!*/
/*!!!                        0-64KW program memory                          !!!*/
/*!!!                        0-64KW data memory                             !!!*/
/*!!! * Note: Use this architecture file with application code. Do not use !!!*/
/*!!!   this file to compile the bootstrap loader source code!             !!!*/
/*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/
.SYSTEM drx;
.PROCESSOR=ADSP21020;

.SEGMENT /ROM /BEGIN=0x0000 /END=0x00ff /PM IRQtable;
.SEGMENT /ROM /BEGIN=0x0100 /END=0x0fff /PM pmcode;
.SEGMENT /RAM /BEGIN=0x1000 /END=0xffff /PM pmdata;
.SEGMENT /RAM /BEGIN=0x0000 /END=0xffff /DM dmdata;

.ENDSYS;
```

C.3 Bootstrap Loader Source Code Listing

```
/*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/
/*!!!                         APT RECEIVER SOFTWARE                        !!!*/
/*!!!                         ---------------------                        !!!*/
/*!!!                      BOOT STRAP LOADER SOURCE CODE                   !!!*/
/*!!!                                                                      !!!*/
/*!!!                         William R. Sylvester Jr.                     !!!*/
/*!!!                      Satellite Communications Group                  !!!*/
/*!!!                           621 Whittemore Hall                        !!!*/
/*!!!            Virginia Polytechnic Institute and State University       !!!*/
/*!!!                          Blacksburg, VA  24061                       !!!*/
/*!!!                             (703) 231-6834                           !!!*/
/*!!!                                                                      !!!*/
/*!!!                              August, 1992                            !!!*/
/*!!!======================================================================!!!*/
/*!!! * Do the following to create output files in Motorola Exorcisor     !!!*/
/*!!!   format suitable for burning the loader EPROM:                      !!!*/
/*!!!       1) asm21k loader.asm                                           !!!*/
/*!!!       2) ld21k -a loader loader.obj                                  !!!*/
/*!!!       3) spl21k -a loader.ach -f B -pm loader.exe                    !!!*/
/*!!!       4) pub21k loader 100000, output = loader.s0 (EEPROM file).     !!!*/
/*!!! * The loader jumps to address 0x100 at the end of the downloading    !!!*/
/*!!!   process.                                                           !!!*/
/*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/

/******************************************************************************/
/*** Include the following files to make system routines and constants     ***/
/*** available.                                                            ***/
/******************************************************************************/
#include "def21020.h"    /Included to define symbolic names for 21020 system  */
                         /register bits.                                      */

/******************************************************************************/
/* Define the interrupt vector table.  Only the system interrupt vector       */
/* should appear in the table (i.e. the bootstrap loader code cannot          */
/* utilize interrupt driven processes) because the interrupt vector table     */
/* from the new target executable code will overwrite this table during the   */
/* download process.                                                          */
/******************************************************************************/
.SEGMENT /PM IRQ_tbl1;
rst_svc: BIT SET MODE2 0x10;       /* An ADSP-21020 silicon flaw prevents the  */
         NOP;                      /* PM cache from being correctly            */
         READ CACHE 0;             /* initialized on power-up/reset.  This     */
         BIT CLR MODE2 0x10;       /* code was supplied by Analog Devices to   */
         JUMP loader;              /* fix the problem.  It must be the first   */
.ENDSEG;                           /* code executed at power-up/reset.         */

/******************************************************************************/
/****                ADSP21020 RELATED DEFINITIONS/INITIALIZATIONS        ****/
/******************************************************************************/
.SEGMENT /PM loader;
#define default_ws 0x21            /* ADSP-21020 default memory configuration: */
loader:  DMWAIT=default_ws;        /* neither PM or DM require wait states.    */
         PMWAIT=default_ws;

         DMBANK1=0x10000;          /* Configure DM bank select registers:      */
         DMBANK2=0x20000;          /* Bank #0 is a 64Kx32 bit SIMM mapped to   */
         DMBANK3=0x30000;          /* 0x00000 to 0x0ffff.  The select lines    */
                                   /* for banks #1, #2 and #3 are used only    */
```

```
                                /* to select various peripherals.          */

                                /* FLAG pins are used as peripheral select  */
                                /* lines.  Initialize the pins to be        */
                                /* output pins.                             */
        BIT SET MODE2 FLG0O|FLG1O|FLG2O|FLG3O;

        BIT CLR MODE1 IRPTEN;   /* Disallow interrupt processing.           */

                                /* Wait to ensure all peripherals have been */
                                /* reset and the system is stable.          */
        LCNTR=285716; DO wait0 UNTIL LCE;
wait0:      CALL wait8251;

/*************************************************************************/
/****       8251A USART (serial communications) INITIALIZATIONS       ****/
/*************************************************************************/
#define w_mc_8251   0x00020001     /* Write mode-command register address.  */
#define w_dat_8251  0x00020000     /* Write data register address.          */
#define r_dat_8251  0x00030000     /* Read data register address.           */
#define r_stat_8251 0x00030001     /* Read status register address.         */

#define dm_ws_8251  0x000AD421     /* Exchanges between the ADSP-21020 and  */
                                   /* the 8251A require 5 ADSP-21020 DM     */
                                   /* wait states.                          */

i8251:  DMWAIT=dm_ws_8251;         /* Initialize the ADSP-21020 DM bus to   */
                                   /* interact with the 8215A.              */

        BIT CLR ASTAT FLG1;            /* Select the 8251A.                 */
        BIT SET ASTAT FLG0|FLG2|FLG3;  /* Deselect all other peripherals.   */

        CALL wait8251;             /* The state of FLG? output pins lags    */
                                   /* contents of the ASTAT register by 1   */
                                   /* clock cycle.  Wait to ensure the 8251A*/
                                   /* is selected.                          */

        R0=0x00;                   /* Force 8251A to the internal idle state*/
        DM(w_mc_8251)=R0;          /* state by sending 4 0x00s to the       */
        CALL wait8251;             /* mode-command register.                */
        DM(w_mc_8251)=R0;
        CALL wait8251;             /* The 8251A has an 8*Tcy write recovery */
        DM(w_mc_8251)=R0;          /* time.  "wait8251" pauses for 8*Tcy=67 */
        CALL wait8251;             /* ADSP21020 clock cycles.               */
        DM(w_mc_8251)=R0;
        CALL wait8251;
        R0=0x40;                   /* Reset the 8251A.                      */
        DM(w_mc_8251)=R0;
        CALL wait8251;

        R0=0x4f;                   /* Send the 8251A mode word: one stop bit,*/
        DM(w_mc_8251)=R0;          /* no parity, eight bit chars., baud rate */
                                   /* factor=64X for 2400 Hz, 4800 Hz, 9600 Hz*/
                                   /* or 19.2 kHz.                          */

        CALL wait8251;             /* Send the 8251A command word: transmit */
        R0=0x37;                   /* enabled, DTR* true, receive enabled,  */
        DM(w_mc_8251)=R0;          /* send no break character, reset all    */
        CALL wait8251;             /* errors, RTS* FALSE, do not reset, do not*/
                                   /* enter hunt mode.                      */

/*************************************************************************/
/****         Send the "download executable" command to the host.     ****/
/*************************************************************************/
        R0=0x43;
        DM(w_dat_8251)=R0;                 /* Low byte.                   */
        LCNTR=10000, DO wait1 UNTIL LCE;
wait1:      CALL wait8251;
```

```
        R0=0x00;
        DM(w_dat_8251)=R0;
        LCNTR=10000, DO wait2 UNTIL LCE;
wait2:     CALL wait8251;
        R0=0x00;
        DM(w_dat_8251)=R0;
        LCNTR=10000, DO wait3 UNTIL LCE;
wait3:     CALL wait8251;
        R0=0x60;                              /* High byte.                    */
        DM(w_dat_8251)=R0;

/*****************************************************************************/
/****          Receive and store the information sent by the host.        ****/
/*****************************************************************************/
/****                                                                     ****/
/**** The information sent by the host will be in the following format:   ****/
/****                                                                     ****/
/**** <PM/DM Flag byte (0x80=PM,0x00=DM)>                                 ****/
/**** <Start address to store ensuing program/data words, 4 bytes,        ****/
/****       MSB first>                                                    ****/
/**** <Number of words of program/data to follow, 4 bytes, MSB first>     ****/
/**** <Code word #1 (4 or 6 bytes depending on PM/DM)>                    ****/
/**** <Code word #2 (4 or 6 bytes depending on PM/DM)>                    ****/
/****                          .                                          ****/
/****                          .                                          ****/
/****                          .                                          ****/
/**** <PM/DM Flag byte (0x80=PM,0x00=DM)>                                 ****/
/**** <Start address to store ensuing program/data words, 4 bytes,        ****/
/****       MSB first>                                                    ****/
/**** <Number of words of program/data to follow, 4 bytes, MSB first>     ****/
/**** <Code word #1 (4 or 6 bytes depending on PM/DM)>                    ****/
/**** <Code word #2 (4 or 6 bytes depending on PM/DM)>                    ****/
/****                          .                                          ****/
/****                          .                                          ****/
/****        until <num words to follow>=0                                ****/
/****                                                                     ****/
/*****************************************************************************/
        /*-------------------------------------------------------------------*/
        /* The "getword" subroutine requires that address register B8        */
        /* contain the address of "temp" scratch memory.                     */
        /*-------------------------------------------------------------------*/
        B8=temp; L8=6; M8=1;

        /*-------------------------------------------------------------------*/
        /* Address registers #1 and #9 will be initialized with the start    */
        /* addresses for code to be downloaded (I1=data memory, I9=program   */
        /* memory) and will be used to perform the addressing the program/   */
        /* data storage process.  Initialize the corresponding L registers   */
        /* to 0 to prevent circular buffering.  M9=1 is used to increment    */
        /* the current address each time a new program memory word is        */
        /* received (M8=1 is used for data memory).                          */
        /*-------------------------------------------------------------------*/
        M9=1; L9=0; L1=0;

        /*-------------------------------------------------------------------*/
        /* Miscellaneous Initializations.                                    */
        /*-------------------------------------------------------------------*/
        R15=0;                      /* Used in comparison to determine if    */
                                    /* "number of words to follow"=0.        */

        R3=0x02;                    /* ANDed with 8251A status byte to       */
                                    /* determine if 8251A receive buffer is  */
                                    /* full (rxrdy bit in status word).      */

        BIT SET ASTAT FLG1;         /* Deselect the 8251A and wait to ensure */
        NOP; NOP; NOP;              /* it is deselected.                     */

        /*-------------------------------------------------------------------*/
```

```
        /* Receive the segment header information.                          */
        /*------------------------------------------------------------------*/
acceptin:CALL getword (DB); R0=1; NOP;    /* Get the PM/DM flag byte and    */
        PM(PMDMFlag)=R0;                  /* store it in <PMDMFlag>.        */

        CALL getword (DB); R0=4; NOP;     /* Get the start address (4 Bytes) */
        R1=R1 OR LSHIFT R2 BY 16;         /* and store it in <Address>.     */
        PM(Address)=R1;

        CALL getword (DB); R0=4; NOP;     /* Get the number of words to     */
        R1=R1 OR LSHIFT R2 BY 16;         /* follow (4 Bytes) and store it  */
        PM(WordCount)=R1;                 /* in <WordCount>.                */

        COMP(R1,R15);                     /* If the number of words to      */
        IF EQ JUMP 0x100;                 /* follow=0, the download process */
                                          /* is complete.  Jump to begin    */
                                          /* executing the target code at   */
                                          /* address 0x100.                 */

        R2=PM(PMDMFlag);                  /* Check to see if the segment to */
        COMP(R2,R15);                     /* be downloaded is a program or  */
        IF EQ JUMP dmload;                /* data memory segment.           */

        /*------------------------------------------------------------------*/
        /* Receive and store the information in a new program memory        */
        /* segment.                                                         */
        /*------------------------------------------------------------------*/
pmload:  B9=PM(Address);                  /* B9=start address.              */
        LCNTR=R1; DO pmlp UNTIL LCE;      /* Receive <WordCount> words.     */
           CALL getword (DB); R0=6; NOP; /* Get next word from host.       */
pmlp:      PM(I9,M9)=PX;                  /* Store next word.               */
        JUMP acceptin;                    /* Jump to get next header segment. */

        /*------------------------------------------------------------------*/
        /* Receive and store the information in a new data memory segment.  */
        /*------------------------------------------------------------------*/
dmload:  B1=PM(Address);                  /* B1=start address.              */
        LCNTR=R1; DO dmlp UNTIL LCE;      /* Receive <WordCount> words.     */
           CALL getword (DB); R0=4; NOP; /* Get next word from host.       */
           R1=R1 OR LSHIFT R2 BY 16;      /* Store next word.               */
dmlp:   /* DM(I1,1)=R1; */                /* Jump to get next header segment. */
        JUMP acceptin;

/*************************************************************************/
/****     Receive a program/data word of specified length from the host.  ****/
/*************************************************************************/
/****                                                                    ****/
/**** This subroutine waits to receive the specified number of bytes     ****/
/**** through the serial port and then assembles the received information ****/
/**** into a program or data word.                                       ****/
/****                                                                    ****/
/**** Input conditions: <R0>=number of bytes in the word.                ****/
/**** Output conditions: word assembled in PX register.                  ****/
/*************************************************************************/
getword: BIT CLR ASTAT FLG1;              /* Select the 8251A and wait for  */
        NOP; NOP; NOP;                    /* the FLG1 output bin to change  */
                                          /* state.                         */

        I8=temp;                          /* Clear the temporary memory     */
        LCNTR=R0, DO clrreg UNTIL LCE;    /* which will store the received  */
clrreg:    PM(I8,M8)=R15;                 /* information                    */

                /* Because the ADSP21020 runs so much faster than the     */
                /* 8251A, the ADSP21020 must look for a transition of the */
                /* 8251 rxrdy bit to identify new receive words (the      */
                /* ADSP21020 can read the 8251A status word, identify     */
                /* the status of the 8251A, read the byte and check the   */
                /* status of the 8251A again before the 8251A status      */
```

```
                /* can reflect the latest read.  Hence, each word is        */
                /* read by the ADSP21020 multiple times.                    */

waitlow: R1=DM(r_stat_8251);            /* Wait for rxrdy low.              */
         R1=R1 AND R3;
         IF NE JUMP waitlow;

waithigh:R1=DM(r_stat_8251);            /* Wait for rxrdy high.             */
         R1=R1 AND R3;
         IF EQ JUMP waithigh;

         R1=DM(r_dat_8251);             /* Read the new byte.               */

         R0=R0-1,PM(I8,M8)=R1;          /* Decrement the number of bytes    */
                                        /* bytes that must be read to       */
                                        /* build the new word and store     */
                                        /* the new byte.                    */

         IF NE JUMP waitlow;            /* If still need more bytes, go     */
                                        /* wait for the next byte.          */

         BIT SET ASTAT FLG1;            /* Deselect the 8251A.              */
         NOP; NOP; NOP;

                                        /* Assemble the bytes of the new    */
                                        /* word in the PX register.         */
                                 R0=PM(I8,M8);
         R2=FDEP R0 BY 24:8,     R0=PM(I8,M8);
         R2=R2 OR FDEP R0 BY 16:8, R0=PM(I8,M8);
         R2=R2 OR FDEP R0 BY 8:8,  R0=PM(I8,M8);
         R2=R2 OR FDEP R0 BY 0:8,  R0=PM(I8,M8);
         PX2=R2;
         R1=FDEP R0 BY 8:8,      R0=PM(I8,M8);
         RTS(DB);
         R1=R1 OR FDEP R0 BY 0:8;
         PX1=R1;

/*****************************************************************************/
/****                        WAIT8251A Subroutine                         ****/
/*****************************************************************************/
/****                                                                     ****/
/**** One of the 8251A timing specifications mandates that writes to      ****/
/**** the 8251A must be separated by 8*Tcy=67 ADSP21020 clock cycles.     ****/
/**** This subroutine counts 70 ADSP21020 clock cycles.                   ****/
/****                                                                     ****/
/**** Input Requirements: none.                                           ****/
/**** Output Conditions: none.                                            ****/
/*****************************************************************************/
wait8251:LCNTR=70, DO waitmore UNTIL LCE;
waitmore:   NOP;
         RTS;

/*****************************************************************************/
/****             Program Memory Variable Data Space Declarations         ****/
/*****************************************************************************/
.VAR temp[6];
.VAR PMDMFlag;
.VAR Address;
.VAR WordCount;
.ENDSEG;
```

C.4 Bootstrap Loader Architecture File Source Code Listing

```
/*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/
/*!!!                        APT RECEIVER SOFTWARE                     !!!*/
/*!!!                        ---------------------                     !!!*/
/*!!!                 BOOTSTRAP DOWNLOADER ARCHITECTURE FILE           !!!*/
/*!!!                                                                  !!!*/
/*!!!                        William R. Sylvester Jr.                  !!!*/
/*!!!                      Satellite Communications Group              !!!*/
/*!!!                          621 Whittemore Hall                     !!!*/
/*!!!            Virginia Polytechnic Institute and State University   !!!*/
/*!!!                        Blacksburg, VA  24061                     !!!*/
/*!!!                            (703) 231-6834                        !!!*/
/*!!!                                                                  !!!*/
/*!!!                             August, 1992                         !!!*/
/*!!!==================================================================!!!*/
/*!!! * The APT receiver configuration listed below implies their is ROM  !!!*/
/*!!!   on the APT receiver system.  This is not true.  All segments must !!!*/
/*!!!   be declared as ROM, however, to force the SPL21K program to create !!!*/
/*!!!   the code used as input to PUB21K.  Furthermore, the loader        !!!*/
/*!!!   software is placed in high memory so that applications code may   !!!*/
/*!!!   be placed in low memory without corrupting the loader.            !!!*/
/*!!! * Note: Use this architecture file to compile the bootstrap loader  !!!*/
/*!!!   source code. Do not use this file to compile application code!    !!!*/
/*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/
.SYSTEM drx;
.PROCESSOR=ADSP21020;

.SEGMENT /ROM /BEGIN=0x0008 /END=0x000f /PM IRQ_tbl1;
.SEGMENT /ROM /BEGIN=0x1f00 /END=0x1fff /PM loader;

.ENDSYS;
```

C.5 Bootstrap Loader File Converter Source Code Listing

```
/*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/
/*!!!                           APT RECEIVER SOFTWARE                      !!!*/
/*!!!                           --------------------                       !!!*/
/*!!! THIS PROGRAM CONVERTS A MOTOROLA EXORCISOR FORMAT FILE OF THE DRX    !!!*/
/*!!! EXECUTABLE CODE TO THE FORMAT EXPECTED BY THE APT RECEIVER BOOTSTRAP !!!*/
/*!!! DOWNLOADER.  (INPUT=DRX.STK, OUTPUT=DRX.BSL)                         !!!*/
/*!!!                                                                      !!!*/
/*!!!                         William R. Sylvester Jr.                     !!!*/
/*!!!                      Satellite Communications Group                  !!!*/
/*!!!                           621 Whittemore Hall                        !!!*/
/*!!!             Virginia Polytechnic Institute and State University      !!!*/
/*!!!                          Blacksburg, VA  24061                       !!!*/
/*!!!                              (703) 231-6834                          !!!*/
/*!!!                                                                      !!!*/
/*!!!                               August, 1992                           !!!*/
/*!!!======================================================================!!!*/
/*!!! The binary output file will be in the following format:              !!!*/
/*!!!                                                                      !!!*/
/*!!! <PM/DM Flag byte (0x80=PM,0x00=DM)>                                  !!!*/
/*!!! <Start address to store ensuing program/data words, 4 bytes,         !!!*/
/*!!!       MSB first>                                                     !!!*/
/*!!! <Number of words of program/data to follow, 4 bytes, MSB first>     !!!*/
/*!!! <Code word #1 (4 or 6 bytes depending on PM/DM)>                     !!!*/
/*!!! <Code word #2 (4 or 6 bytes depending on PM/DM)>                     !!!*/
/*!!!                           .                                          !!!*/
/*!!!                           .                                          !!!*/
/*!!!                           .                                          !!!*/
/*!!! <PM/DM Flag byte (0x80=PM,0x00=DM)>                                  !!!*/
/*!!! <Start address to store ensuing program/data words, 4 bytes,         !!!*/
/*!!!       MSB first>                                                     !!!*/
/*!!! <Number of words of program/data to follow, 4 bytes, MSB first>     !!!*/
/*!!! <Code word #1 (4 or 6 bytes depending on PM/DM)>                     !!!*/
/*!!! <Code word #2 (4 or 6 bytes depending on PM/DM)>                     !!!*/
/*!!!                           .                                          !!!*/
/*!!!                           .                                          !!!*/
/*!!! <num words to follow>=0 (signifies end of program)                   !!!*/
/*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/
/*---------------------------------------------------------------------------*/
/* Include the following files to make system constants and subroutines     */
/* available.                                                                */
/*---------------------------------------------------------------------------*/
#include "process.h"
#include "string.h"
#include "stdio.h"
#include "types.h"
#include "stat.h"
#include "fcntl.h"
#include "io.h"

/*****************************************************************************/
/*                          MAIN PROGRAM BEGINS HERE                        */
/*****************************************************************************/
void main ()
   {
    FILE *stkfile;                 /* Input file stream.                      */
    int drxhandle,                 /* Output file handle.                     */

    unsigned long b_NumBytes,      /* Stores the number of bytes to follow    */
```

```
                                /*    in the current segment (binary)      */
            b_StartAddress,     /* Stores the start address of the         */
                                /*    current segment (binary)             */
            b_NumWords,         /* Stores the number of words to follow    */
                                /*    in the current segment (binary)      */
                                /*    =b_NumBytes/WordSize                 */
            WordSize;           /* Size (in bytes) of the words in the     */
                                /*    current segment (data memory=4       */
                                /*    program memory=6). (binary)          */
            i,j,k,              /* Loop control variables.                 */
char a_FieldWidth[3],           /* Used to read the "field width" field in */
                                /*    the segment headers.  This           */
                                /*    information is not used in building  */
                                /*    the output file.                     */
     a_Reserved[3],             /* Used to read the "reserved" field in    */
                                /*    the segment headers.  This           */
                                /*    information is not used in building  */
                                /*    the output file.                     */
     a_PMDMFlags[3],            /* Used to read the PM/DM flag which       */
                                /*    identifies the current segment       */
                                /*    as a data memory or program          */
                                /*    memory segment. (ASCII)              */
     a_UserFlags[3],            /* Used to read the "user flags" field in  */
                                /*    the segment headers.  This           */
                                /*    information is not used in building  */
                                /*    the output file.                     */
     a_StartAddress[9],         /* Stores the start address of the         */
                                /*    current segment (ASCII)              */
     a_NumBytes[9],             /* Stores the number of bytes to follow    */
                                /*    in the current segment (ASCII)       */
     a_OneByte[3],              /* Used to read the next byte from the     */
                                /*    input file.                          */
     CRLF[5],                   /* Used to read (and discard) a carriage   */
                                /*    return/line feed pair from the       */
                                /*    input file.                          */
     b_PMDMFlags,               /* Used to store the PM/DM flag which      */
                                /*    identifies the current segment       */
                                /*    as a data memory or program          */
                                /*    memory segment. (binary)             */
     temp,                      /* Scratch variable.                       */
     b_OneByte;                 /* Used to a single byte to the binary     */
                                /*    output file.                         */

/*-------------------------------------------------------------------------*/
/* Open the input file.  Exit the program if the file is not available.    */
/*-------------------------------------------------------------------------*/
if ((stkfile=fopen ("drx.stk","rt"))==NULL)
   {
    printf ("\nCould NOT open input file.");
    _exit(0);
   }
/*-------------------------------------------------------------------------*/
/* Open the output file.  Exit the program if the disk is full.            */
/*-------------------------------------------------------------------------*/
if ((drxhandle=open("drx.bsl",O_BINARY|O_CREAT|O_RDWR|O_TRUNC,
                              S_IREAD|S_IWRITE))<0)
   {
    printf ("\nCould NOT open output file.");
    _exit(0);
   }
/*-------------------------------------------------------------------------*/
/* Read header records and process segments all segments in the input      */
/* file.                                                                   */
/*-------------------------------------------------------------------------*/
for (;;)
   {
    /*---------------------------------------------------------------------*/
    /* Read the new header information.                                    */
```

```
/*-------------------------------------------------------------------*/
printf ("\nNew Header Record Located");
printf ("\n-------------------------");          /* Read (discard)  */
fgets(a_FieldWidth  ,3,stkfile);                /* the "field      */
printf("\n   Field Width String=%s",a_FieldWidth); /* width" field.  */

fgets(a_Reserved    ,3,stkfile);                /* Read (discard)  */
printf("\n   Reserved String=%s",a_Reserved);   /* the "reserved"  */
                                                /* field.          */

                /* Read the "PM/DM flag" field.  Set the word size  */
                /* for the next segment based on the contents of    */
                /* this field (4 bytes for data memory, 6 bytes for */
                /* program memory).  Write the PM/DM flag to the    */
                /* output file (in binary format).                  */

fgets(a_PMDMFlags   ,3,stkfile);
printf("\n   PMDMFlags String (80=PM,00=DM)=%s",a_PMDMFlags);
if (strcmp(a_PMDMFlags,"80")==0) WordSize=6; else WordSize=5;
sscanf(a_PMDMFlags,"%lx",&b_PMDMFlags);
write (drxhandle,&b_PMDMFlags,1);

fgets(a_UserFlags   ,3,stkfile);                /* Read (discard)  */
printf("\n   UserFlags String=%s",a_UserFlags);  /* the "User flags"*/
                                                /* field.          */

                /* Read the "start address" field from the input    */
                /* file, convert it binary format and write it to   */
                /* the output file (MSB first).                     */

fgets(a_StartAddress,9,stkfile);
sscanf(a_StartAddress,"%lx",&b_StartAddress);
printf("\n   Start Address=%lx",b_StartAddress);
for (k=3;k>=0;k--)
   {
    temp=(b_StartAddress>>(k*8))&0xFF;
    write(drxhandle,&temp,1);
   }

                /* Read the "number of bytes to follow" field from  */
                /* the input file, compute the number of words to   */
                /* follow (=number of bytes/word size) and write    */
                /* the number of words to follow to the output file.*/

fgets(a_NumBytes    ,9,stkfile);
sscanf(a_NumBytes,"%lx",&b_NumBytes);
b_NumWords=b_NumBytes/WordSize;
for (k=3;k>=0;k--)
   {
    temp=(b_NumWords>>(k*8))&0xFF;
    write(drxhandle,&temp,1);
   }
if (b_NumBytes==0L)       /* If the "number of bytes to follow"=0, all */
   {                      /* segments have been processed.  Close all  */
    fclose(stkfile);      /* files and exit the program.               */
    close (drxhandle);
    _exit(0);
   }
printf("\n   Bytes to Follow=%lx (hex), %lu (decimal)\n",
       b_NumBytes,b_NumBytes);

fgets(CRLF,5,stkfile);    /* Read and discard a carriage return-       */
                          /* line feed pair from the input file.       */

                /* Read "number of bytes" from the input file,       */
                /* convert them to binary format and write them      */
                /* to the output file.                               */
i=0;
```

```
    while (i<b_NumBytes)
       {
        for (j=0;j<WordSize;j++)
           {                              /* Read each byte of the next     */
            fgets(a_OneByte,3,stkfile); /*    word from the input file.   */
            i++;                          /* Increment the number of        */
                                          /*    # of bytes read.            */
                                          /* Convert to binary format.      */
            sscanf(a_OneByte,"%2x",&b_OneByte);                              */
                                          /* Write the binary information   */
                                          /*    to the output file.         */
            write (drxhandle,&b_OneByte,1);
           }
        fgets(CRLF,5,stkfile);    /* Discard the CRLF pair at the end      */
                                  /* of each word.                         */
       }
    }
}
```

C.6 Downloadable Code Generation Batch File Source Code Listing

```
/*!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*/
/*!!!                        APT RECEIVER SOFTWARE                       !!!*/
/*!!!                        ---------------------                       !!!*/
/*!!! THIS BATCH FILE PRODUCES DOWNLOADABLE CODE TO BE USED WITH THE     !!!*/
/*!!! BOOTSTRAP LOADER FROM AN .ASM FILE (OUTPUT IS A .BSL FILE)         !!!*/
/*!!!                                                                    !!!*/
/*!!!                       William R. Sylvester Jr.                     !!!*/
/*!!!                    Satellite Communications Group                  !!!*/
/*!!!                         621 Whittemore Hall                        !!!*/
/*!!!          Virginia Polytechnic Institute and State University       !!!*/
/*!!!                        Blacksburg, VA  24061                       !!!*/
/*!!!                           (703) 231-6834                           !!!*/
/*!!!                                                                    !!!*/
/*!!!                            August, 1992                            !!!*/
/*!!!====================================================================!!!*/
asm21k drx.asm                    /* Assemble the code.                      */

ld21k -a drx drx.obj adsp.lib     /* Link with Analog Devices run-time       */
                                  /* library.                                */

spl21k -a drx -pm -f b drx        /* Create ASCII EEPROM files.  NOTE:       */
                                  /* NO EEPROMS SHOULD BE BURNED WITH        */
                                  /* THE OUTPUT OF THIS COMMAND!  This       */
                                  /* command is only executed to produce     */
                                  /* files for input to CONV_BSL!            */

conv_bsl                          /* Convert the EEPROM files to the         */
                                  /* format expected by the downloader       */
                                  /* on the APT receiver                     */
```

Appendix D

APT Receiver Schematic

(The schematic is in the pocket on the back cover)

References

[1] Y. Ninomiya, Y. Ohtsuka, Y. Izumi, S. Goshi, Y. Iwadate, "An HDTV Broadcasting System Utilizing a Bandwidth Compression Technique-MUSE," *IEEE Transactions on Broadcasting*, Vol. BC-33(4), December, 1987.

[2] P. W. Remaklus Jr, *Data Acquisition and Control System for the OLYMPUS Propagation Experiments*, Master of Science Thesis, Virginia Polytechnic Institute and State University, April, 1991.

[3] L.L. Ippolito, R.D. Kaul, R.G. Wallace, *Propagation Effects Handbook for Satellite System Design*, NASA, June, 1983.

[4] T. Pratt, C. W. Bostian, *Satellite Communications*, John Wiley and Sons, New York, 1986

[5] C.W. Bostian, J.C. McKeeman, T. Pratt. A. Safaai-Jazi, W.L. Stutzman, *Communications and Propagation Experiments Using the OLYMPUS and ACTS Spacecraft: Outline of Planned Final Report for Final Year of Data Collection*, Virginia Tech Satellite Communications Group, August, 1990.

[6] Virginia Tech Satellite Communications Group, *Annual Report: Support of the NASA Propagation Studies and Measurement Program; Volume 2: ACTS/OLYMPUS Experiments*, Report EE SATCOM 88-4, September, 1988.

[7] European Space Agency, *OLYMPUS User's Guide UG-6-1 Part 1 Propagation Package*, March, 1989.

[8] P.A. Lowry, *Advanced Communications Technology Satellite (ACTS) Beacon Transmitters*, NASA Lewis Research Center, March 1989

[9] D. Chakraborty, J. Gevargiz, 20 GHz ACTS Beacon Spectral Analysis, *Proceedings of the Fifteenth NASA Propagation Experimenters Meeting (NAPEX XV) and the Advanced Communications Technology Satellite (ACTS) Propagation Studies Miniworkshop*, June 28-29, 1991, London, Ontario, Canada

[10] Telephone conversation between W.L. Stutzman (VPI&SU) and L. Cashman (G.E. Astro Space Division) on January 2, 1992, (data taken from G.E. internal memorandum dated August, 1991). Also documented in a letter from R. Baur (NASA Lewis Research Center) dated January 24, 1992.

[11] Proceedings of the 1991 ACTS Conference, San Jose, CA, August 29-30, 1991.

[12] Virginia Tech Satellite Communications Group, *Communications and Propagation Experiments Using the OLYMPUS Spacecraft, Report on the First Year of Data Collection*, Report EE SATCOM 91-4, October, 1991.

[13] G.R. Runyon, *Parallel Processor Architecture for a Digital Beacon Receiver*, Master of Science Thesis, Virginia Polytechnic Institute and State University, July, 1990.

[14] L.B. Jackson, *Digital Filters and Signal Processing, Second Edition*, Kluwer Academic Publishers, Boston, MA, 1990.

[15] W.H Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, *Numerical Recipes in C*, Cambridge University Press, New York, 1990.

[16] M.I. Skolnik, *Introduction of Radar Systems, Second Edition*, McGraw-Hill Book Company, New York, 1980.

[17] A.V. Oppenheim, R.W. Schafer, *Digital Signal Processing*, Prentice Hall Inc., Englewood Cliffs, NJ, 1975.

[18] W.P. Robins, *Phase Noise in Signal Sources (Theory and Applications)*, Peter Peregrinus Ltd., London, UK, 1982.

[19] Analog Devices, Inc., *Data Converter Reference Manual, Volume 2*, Analog Devices, 1992.

[20] Analog Devices Inc., *Analog-Digital Conversion Handbook, Third Edition*, Prentice-Hall, Englewood Cliffs, NJ, 1986.

[21] Qualcomm, *Direct Digital Synthesis, 21 Questions and Answers for RF Engineers*, Qualcomm, 1991.

[22] Qualcomm, *Q2334 Dual Direct Synthesizer Technical Data Sheet*, Qualcomm, 1991.

[23] Intel Corporation, *Microcommunications, Volume 1*, Intel Corporation, Mt. Prospect, IL, 1990.

[24] Virginia Tech Satellite Communications Group, *Annual Report, Volume 3, ACTS/OLYMPUS Equipment*, Report EE SATCOM 88-4, Virginia Polytechnic Institute and State University, Blacksburg, Va, September, 1988.

[25] D.G. Sweeney and J.C. McKeeman, *Design of a Hybrid Analog/Digital Beacon Receiver*, Electronics Letters, June 21, 1990.

[26] G. Aspin, *Digital Beacon Receiver Unit: Feasibility Study*, Signal Processors Limited, Cambridge, England, March 6, 1986.

[27] J. Ostergaard, *OLYMPUS Beacon Receiver*, Proceedings of NAPEX XII, June 1988.

[28] P. Ransome, *A Digital Beacon Receiver*, Proceedings of NAPEX XII, June 1988.

[29] J. Tsui, *Digital Microwave Receivers: Theory and Concepts*, Artech House, 1989.

[30] J.S. Milton, J.C. Arnold, *Introduction to Probability and Statistics: Principles and Applications for the Engineering and Computing Sciences, Second Edition*, McGraw-Hill Book Co., New York, 1990.

[31] A.B. Carlson, Communications Systems, *An Introduction to Signals and Noise in Electrical Communications, Third Edition*, McGraw-Hill Book Co., New York, 1986.

[32] J. Millman, *Microelectronics, Digital and Analog Circuits and Systems*, McGraw Hill Book Co., New York, 1979.

[33] R.S. Sandige, *Modern Digital Design*, McGraw-Hill Book Co., New York, 1990.

[34] G.F. Franklin, J.D. Powell, M.L. Workman, *Digital Control of Dynamic Systems*, Addison-Wesley Publishing Company Inc., New York, 1990.

[35] L.C. Ludeman, *Fundamentals of Digital Signal Processing*, Harper and Row Publishers, New York, 1986.

[36] Analog Devices Inc., ADSP21020 data sheet, Analog Devices Inc., Norwood, MA, 1992.

[37] Analog Devices Inc., *ADSP21020 User's Manual*, Analog Devices Inc., Norwood. MA, 1991.

[38] Analog Devices Inc., *ADSP21000 Family Development Software Tutorial*, Analog Devices Inc., Norwood, MA, 1991.

[39] Integrated Device Technology Inc., *Specialized Memories Databook*, Integrated Device Technologies, Santa Clara, CA, 1990-91.

[40] Cypress Semiconductor Inc., *BiCMOS/CMOS Databook,* Cypress Semiconductor, Inc., San Jose, CA, 1991.

[41] Mini-Circuits Inc., *RF/IF Signal Processing Guide*, Mini-Circuits Inc., Brooklyn, NY, 1991-92.

[42] Texas Instruments Inc., *TTL Logic Data Book, Standard TTL, Schottky, Low-Power Schottky*, Texas Instruments Inc., Dallas, TX, 1988.

[43] Coilcraft Inc., *RF Inductors*, Coilcraft Inc., Cary, IL, 1991.

[44] Motorola Inc., *Linear and Interface Integrated Circuits Databook*, Motorola Inc., Phoenix, AZ, 1985.

[45] Intel Corporation, *Memory Component Handbook*, Intel Corporation, Santa Clara, CA, 1988.

[46] Vectron Laboratories Inc., *Crystal Oscillators*, Vectron Laboratories Inc., Norwalk, CN, 1990-1991.

[47] TRW LSI Products Inc., *Data Converters-DSP Products Databook*, TRW LSI Products Inc., La Jolla, CA, 1991.

[48] Maxim Inc., *Maxim Integrated Products, Volume 1*, Maxum Inc., Sunnyvale, CA, 1991.

[49] Rhombus Industries Inc., *Delay Line Short Form Catalog*, Rhombus Industries Inc., Huntington Beach, CA, 1991.

[50] Analog Devices Inc., *Linear Products Databook*, Analog Devices Inc., Norwood, CN, 1992.

[51] Motorola Inc., *Small-Signal Transistors, FETs and Diodes Databook*, Motorola, Inc., Phoenix, AZ, 1991.

Addendum

APT Receiver Tests On the Actual ACTS Satellite

After this document was submitted, the APT receiver was tested on the actual ACTS beacons. On August 4, 1992, ACTS was undergoing system tests at the Astro Space Division of General Electric in Princeton, NJ. General Electric permitted members of the Virginia Tech Satellite Communications Group to assemble the APT RF, IF and receiver subsystems in the ACTS test bay next to the satellite as shown in Figure A.

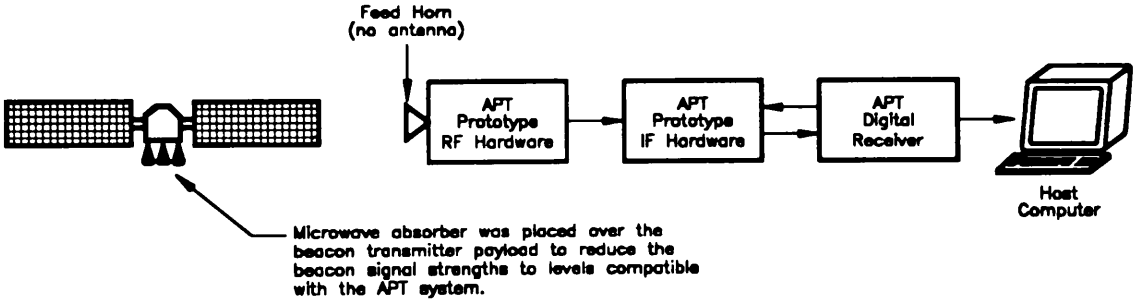


Figure A. System used to test the APT receiver on the actual ACTS beacons.

As ACTS was undergoing system tests which utilized both the 20 and 27.5 GHz beacons, the APT system was able to receive the beacon signals. The APT receiver reliably acquired both the 20 and 27.5 GHz beacons. The resulting spectra generated by the APT receiver are shown in Figures B, C, D and E. Note, the spectra of the 20 GHz beacon show several of the possible modulation formats.

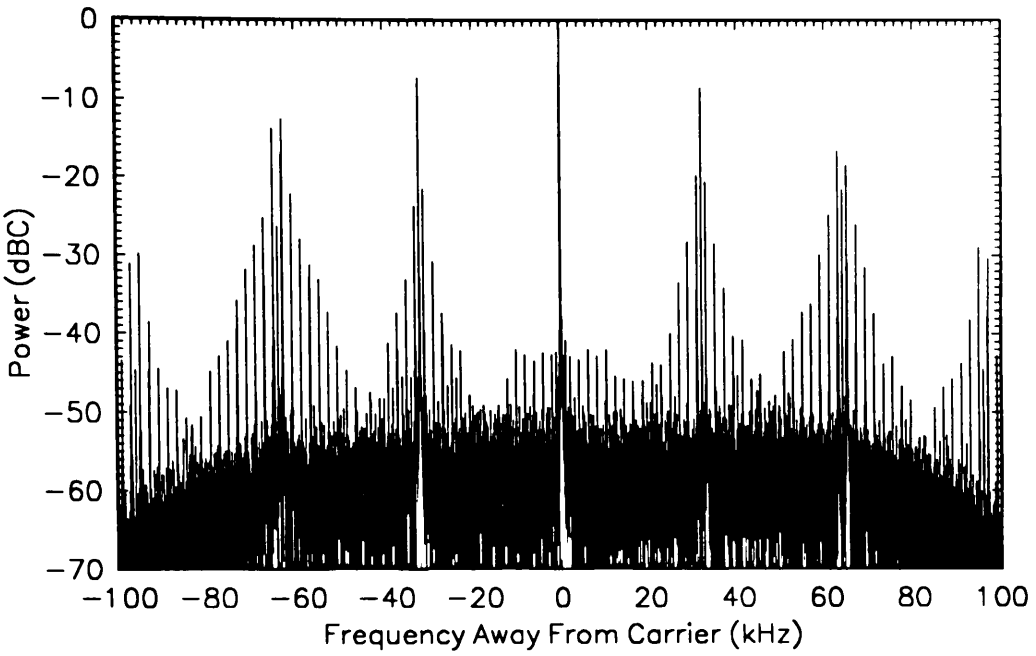


Figure B. 32768-point spectrum of the ACTS 20 GHz beacon in *Digital PCM Telemetry Mode*.

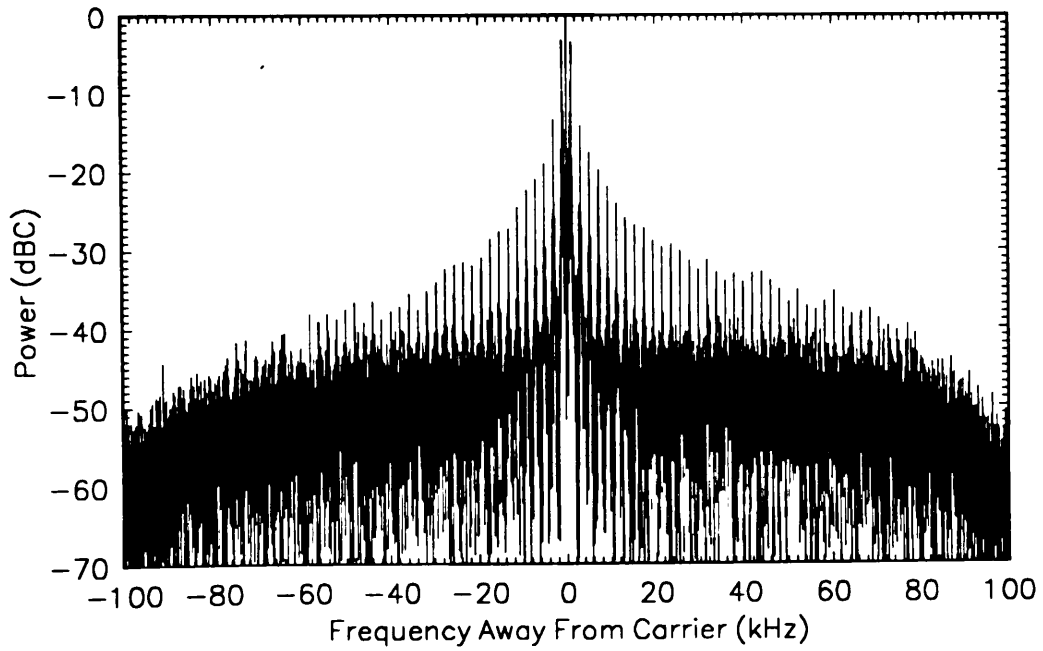


Figure C. 32768-point spectrum of the ACTS 20 GHz beacon in *PCM Direct Mode*.

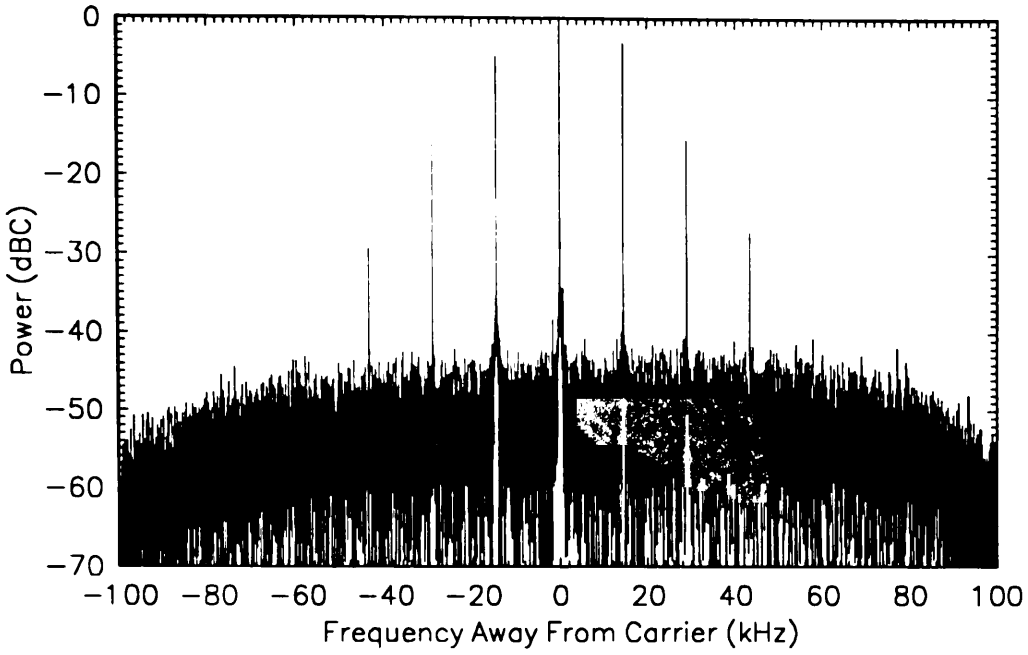


Figure D. 32768-point spectrum of the ACTS 20 GHz beacon in *Attitude Monitor Mode*.

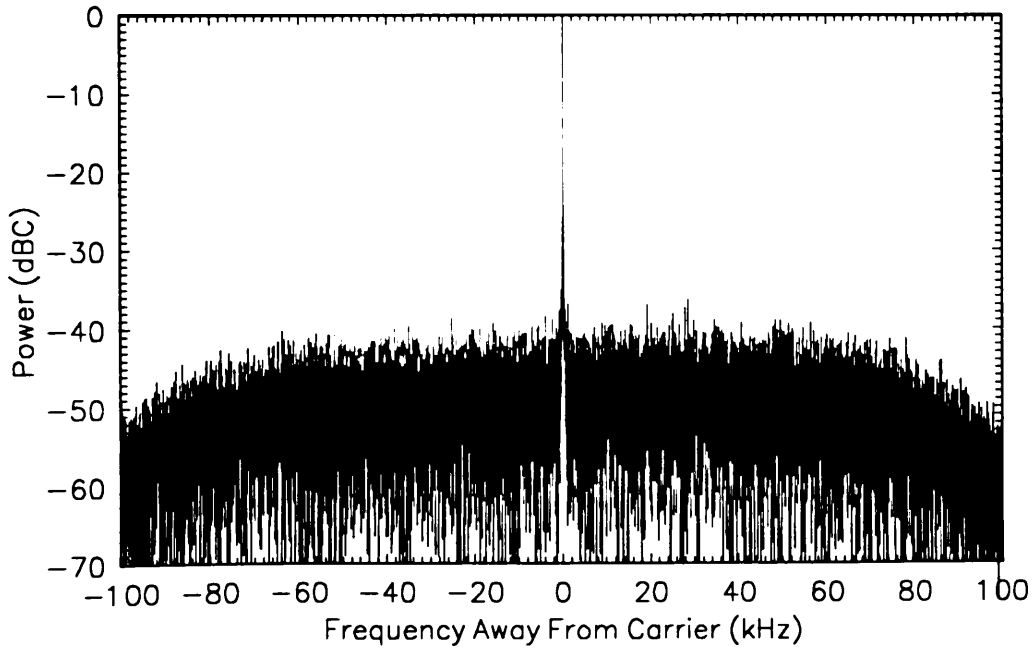


Figure E. 32768-point spectrum of the ACTS 27.5 GHz beacon.

Furthermore, 10 signal power measurement mode spectra of each carrier signal were saved in an attempt to determine each signal's phase noise characteristics. The geometric mean of the 10 spectra from each channel was computed using Equation (1). The results of these calculations are plotted in Figures F and G.

$$\begin{pmatrix} Average \\ DFT\ bin \\ value\ (dB) \end{pmatrix}_i = 10 \cdot log_{10}\left(\frac{\sum_{j=0}^{9} (I_j^{\,2} + Q_j^{\,2})_i}{10} \right) \qquad (1)$$

$$where: i = 0..1023$$

$$(I_j^{\,2} + Q_j^{\,2})_i = power\ contained\ in\ the\ i^{th}\ DFT\ bin\ of\ spectra(j)$$

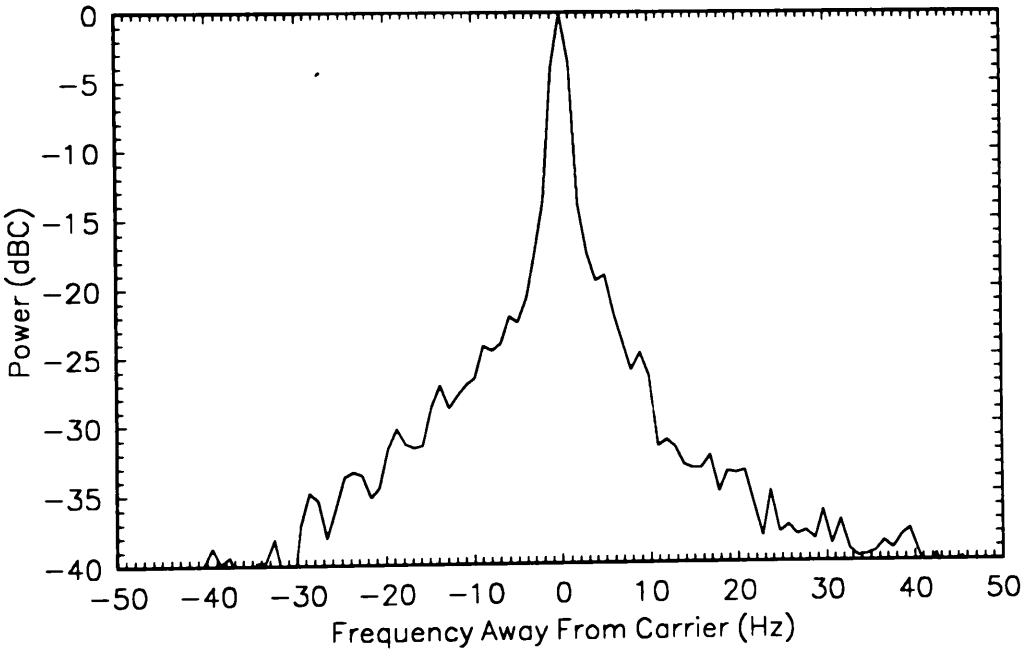


Figure F. Geometric mean of 10 signal power measurement mode spectra of the ACTS 20 GHz carrier.

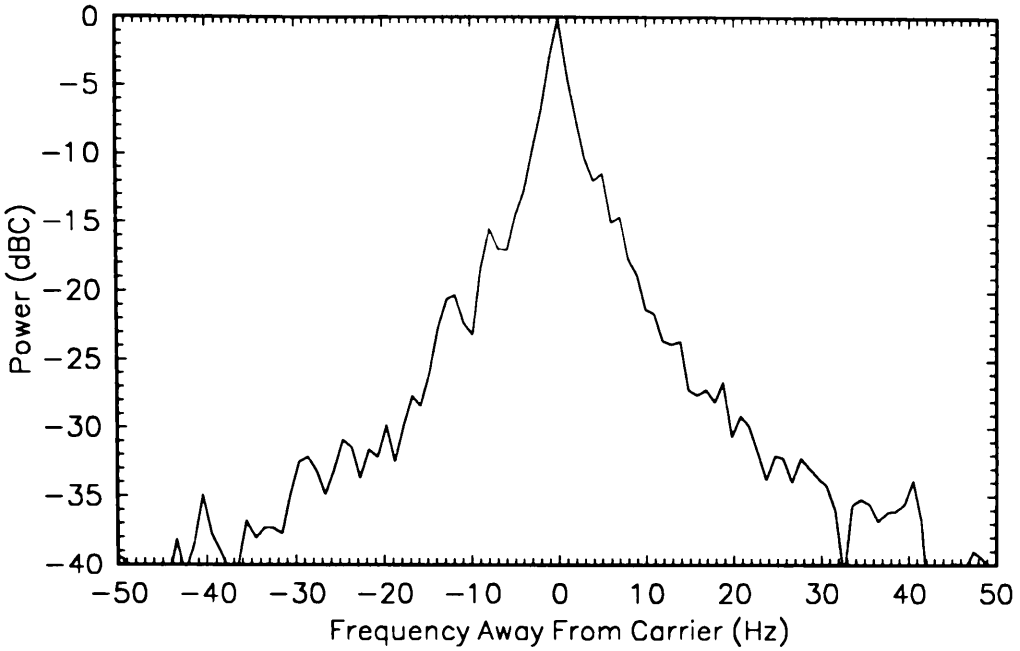


Figure G. Geometric mean of 10 signal power measurement mode spectra of the ACTS 27.5 GHz carrier.

Vita

William Richard Sylvester Jr. was born on March 14, 1967 in West Chester, Pennsylvania. William graduated Cum Laude from Tower Hill High School before receiving his Bachelor of Science degree in electrical engineering from Virginia Polytechnic Institute and State University in May, 1990. At the completion of his Master of Science degree, William will be employed by IBM Corporation in Manassas, VA.