Operation of a Radar Altimeter Over the Greenland Ice Sheet

A Thesis Presented

by

Matthew D. Grund

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

Master of Science in Electrical and Computer Engineering

May 1996

Department of Electrical and Computer Engineering

Operation of a Radar Altimeter Over the Greenland Ice Sheet

A Thesis Presented

by

Matthew D. Grund

Approved as to style and content by:

Calvin T. Swift, Chair

Robert E. McIntosh, Member

Patrick A. Kelly, Member

Daniel H. Schaubert,Department Head,
Electrical & Computer Engineering

To My Parents

Acknowledgements

I would like to thank professors Calvin T. Swift and Robert E. MacIntosh for providing a research environment with plentiful resources and talented, helpful graduate students. In their direction of the Microwave Remote Sensing Laboratory, Swift and MacIntosh work with uncommon dedication to ensure that the MIRSL remains a leader in the microwave remote sensing field.

Special thanks should be given to several graduate students who were particularly helpful during my thesis research. Ellen Ferraro provided invaluable guidance while working with the AAFE Altimeter. Other MIRSL students, especially Greg Sadowy, Delwyn Moller, Gordon Woodington and Rudy Pawul deserve thanks for their friendship, advice, and assistance.

Thanks also to scientists and support staff from NASA Wallops Flight Facility, who proved knowlegeable, helpful and friendly during radar installation and Greenland mission flights. Scientists of note include Bill Krabill, Earl Frederick, and Bob Swift from NASA Observational Sciences Branch. They did all they could to make my time in Greenland as trouble free as possible, despite my inexperience.

ABSTRACT

OPERATION OF A RADAR ALTIMETER OVER THE GREENLAND ICE SHEET

MAY 1996

MATTHEW D. GRUND

B.S.E.E., UNIVERSITY OF LOWELL, LOWELL, MASSACHUSETTS

M.S.E.C.E., UNIVERSITY OF MASSACHUSETTS, AMHERST, MASSACHUSETTS

Directed by: Professor Calvin T. Swift

This thesis presents documentation for the Advanced Application Flight Experiment (AAFE) pulse compression radar altimeter and its role in the NASA Multisensor Airborne Altimetry Experiment over Greenland in 1993. The AAFE Altimeter is a Ku-band microwave radar which has demonstrated 14 centimeter range precision in operation over arctic ice. Recent repairs and improvements were required to make the Greenland missions possible. Transmitter, receiver and software modifications, as well as the integration of a GPS reciever are thoroughly documented. Procedures for installation, and operation of the radar are described. Finally, suggestions are made for further system improvements.

TABLE OF CONTENTS

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

1.1 Motivation

There has been a recent interest in the effects of human combustion of fossil fuels on the environment. Some theorize that increased amounts of carbon dioxide and other trace gases are causing the earth's atmosphere to retain more energy, and upsetting the earth's temperature and climate. [1] One good indicator of the earth's stored thermal energy is the volume of glacial ice present. The volume of existing ice sheets is known to fluctuate, over tens of thousands of years, with changes in climate, [2] but it is not known whether ice sheets are growing or shrinking today.

The relationship between glacial ice sheets and global climate is not simple or well known. Ice sheet mass balance can be used as an indicator of long term climatic changes, but is also considered to have short term effects on other environmental parameters. Some important oceanographic variables affected include changes in sea level, temperature, and salinity. [3] The ice sheets are also thought to influence temperature and circulation patterns in the atmosphere. It is clear that measurement of glacial ice sheet dynamics is critical to our understanding of the global climate.

A direct measurement of the volume of glaciers on earth is not practical at this time, due to their great size, inaccessibility, and harsh surface conditions. Spaceborne measurement of glaciers has been used to provide coarse resolution data, but because of antenna footprint considerations, resolution is less than optimal. However, airborne remote sensing offers an efficient means of measuring ice surface elevation with high spatial resolution, particularly since Global Position Satellite

(GPS) systems can be used to determine the position of the aircraft platform within centimeters. This thesis will focus on one airborne sensor, a Ku-Band microwave pulse compression radar, which is currently being maintained, upgraded and operated by the Microwave Remote Sensing Laboratory at the University of Massachusetts.

1.2 Instrument History

The Advanced Application Flight Experiment Breadboard Pulse Compression Radar Altimeter (AAFE Altimeter) was developed by Hughes Aircraft Company, in 1975. Before that time, the SKYLAB S-193 altimeter had demonstrated the feasibility of using radar altimetry to measure the geoid of the earth from space. The S-193 had a pulse width of only 10 ns, which resulted in a range resolution of only 1.5 meters [4], which was too coarse for the geophysical community, which desires resolution on the order of tens of centimeters. In order to advance the state of the art, the AAFE Altimeter was designed to produce an effective pulse width of 2.7 nanoseconds giving a 41 centimeter raw range resolution [5].

After several years of oceanographic experiments conducted from NASA Wallops Flight Facility, the AAFE altimeter fell into disuse, and was transferred to the University of Massachusetts in 1990. The system had several shortcomings which precluded use in mapping glacial ice. Its signal processing system had only 24 range bins, which allowed the radar to image only a 9.8 meter range window. This made tracking over rougher terrain difficult. The radar control and data acquisition system was obsolete, and included technology such as a teletype for programming, and a punched tape reader for storing and loading programs. Operating the radar with such primitive control and data acquisition hardware, and manual tracking required great operator skill, making long flight missions arduous and impractical.

The Microwave Remote Sensing Laboratory proceeded to refurbish the instrument, replacing the signal processing and data acquisition systems with a modern computer for control, and a state-of-the-art data acquisition system, capable of sample rates up to 1.0 gigasamples per second, which could store entire return waveforms for post processing. [6] These changes allowed the instrument to sample and store 88 meter range intervals, and track the terrain using software algorithms, allowing various terrain characteristics to be observed. In its improved state, the instrument can readily be used to map glacial ice topography.

CHAPTER 2

INSTRUMENT DESCRIPTION

The AAFE Altimeter Radar system occupies two half-height nineteen inch racks, as shown in figure 2.1. The radar has six major subsystems which are installed in the rack, along with the antenna subsystem, which is installed in the aircraft for deployments. Radar subsystems mounted in the rack are Host Computer System, Data Acquisition System, RF Control Subsystem, Transmitter Subsystem, Receiver Subsystem, and GPS Receiver. They are cabled together in the rack, and communicate with each other as shown in figure 2.2. The function and configuration of these subsystems are described here.

2.1 Host Computer System

The host computer system for the AAFE Altimeter is responsible for providing the operator's user interface, initializing and controlling all subsystems, recording the altimetry data on hard disk, later processing the data, and creating tape archives.

The computer currently used is an Hewlett-Packard 9000/382 Workstation, which is a Motorolla 68040 based 32 bit architecture machine. The computer uses Fast, Wide SCSI II peripheral bus to control its 200MB internal hard disk, as well as its 1GB external hard disk, and 2GB 4mm digital tape backup (DAT Format). It also uses two high speed RS-232 serial ports, two IEEE-488 (GPIB) interfaces (one for control of several subsystems, and one for high speed data transfer), and a high speed parallel port (HP GPIO Board).

The altimeter host computer is capable of running several operating systems. Currently, HP-Basic OS is used for diagnostics and data acquisition, and HP-UX

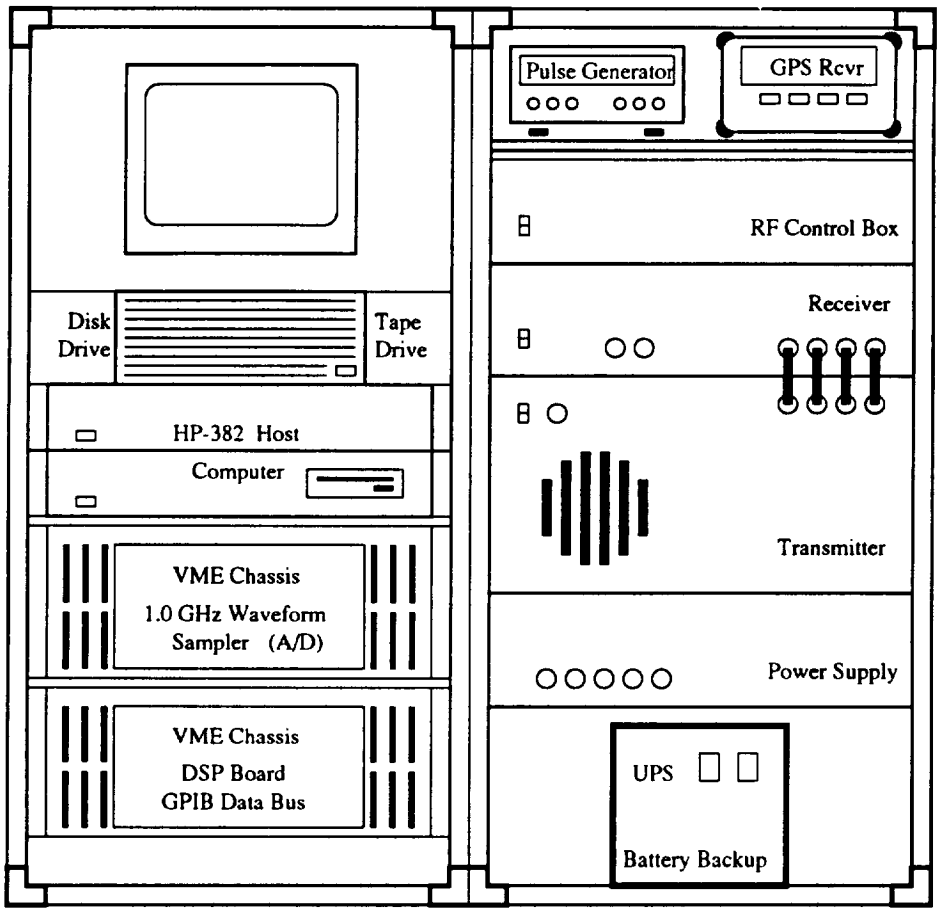

Figure 2.1 AAFE Altimeter Radar System Rack Mount Configuration

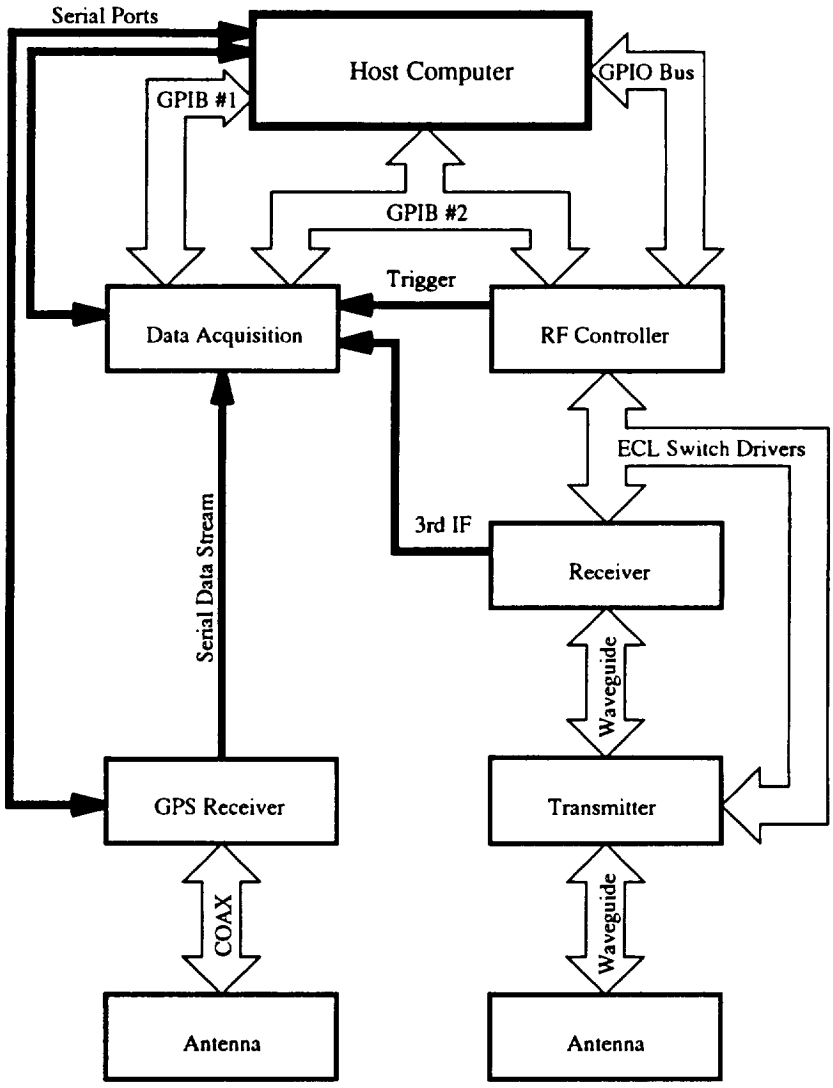

Figure 2.2 AAFE Altimeter Radar System Block Diagram

(unix) is used to process and backup the data. HP Basic is a single-threaded premptive multitasking operating system which is ideal for real time data acuisition [7]. The AAFE radar operating software is written in the HP-Basic programming language, which simplifies high speed data transfers, flexible interrupt driven instrument and peripheral control, and graphical data plotting. HP-UX is a true multitasking operating system which provides an X-Windows user interface that facilitates data processing. Networking capabilities are also provided under HP-UX, allowing data to be exported to other more powerful workstations at the Microwave Remote Sensing Laboratory, for faster analysis.

2.2 Data Acquisition System

The AAFE Altimeter uses a VME based data acquisition system, produced by Analytek. The system resides in two standard size 6U 5 card VME chasses. One chassis contains the sampling section: a 2000T timing module, 2000P controller (a Motorolla 68020 based card), 2004SC high speed sampler (1GHz maximum burst sample rate) and a 2000HA high speed digital bus module. Typically, the controller is configured by the host computer, over the GPIB bus (see figure 2.2), which then calibrates and configures the other modules. The usual mode of operation requires that the timing module be configured to trigger sampling at some integer dividend of it's 1GHz clock (usually $1.0GHz/6$ or $167MHz$), when it receives a trigger from the RF control system. When triggered by the timing module, the 2004SC performs sampling and A/D conversion, making the resulting 512 word block available to the processor where it is averaged with other data blocks, and transferred to the 2000HA high speed digital bus module for transmission to the second VME chassis.

The second VME chassis contains an Analytek 2000VIX high speed bus interface, National Instruments VME GPIB interface and Impact Technologies VIPER 8704/30 digital signal processing (DSP) board. The VIPER board, acting as VME

controller for this chassis, consists of an Motorolla 68030 acting as host for four ZORAN ZR34161 Vector Signal Processors (VSPs). These VSPs have several important features, namely hardware (single instruction) FFT, Complex number (vector) addressing, and block floating point arithmetic instruction set [8].

The distributed programming environment determined by the VIPER DSP board architecture makes software development on the VIPER quite awkward. Programs running on 68030 and VSPs must be cross-assembled using assemblers running on an Intel 386 personal computer (PC) and uploaded to the AAFE host computer, which then dowloads them to the VIPER board. The VIPER board runs a primitive monitor program, booted from on board ROM, which allows the user to download assembly language code over the serial port, and make transfers to VSP memory. An HP Basic program called AAFE START manages this downloading, as shown in figure 2.3, and is the usual starting software for all altimeter operations. This program initializes all peripherals, downloads VIPER code and calls AAFE RUN, the HP Basic altimeter data logging and display program.

The VIPER controller, running an assembly language program downloaded from the host computer over serial port, manages data import from the sampler, data processing on the VSPs, and data export, via GPIB block transfer, to the host computer for recording and display, as shown in figure 2.4. After initializing all resources, it directs data flow from the 2000VIX via VME data bus to the VIPER 68030 host RAM. It then distributes it to the four VSPs for distributed fast Fourier Transform (Butterfly FFT algorithm) [9], re-assembles the blocks, discards negative frequencies, and uploads it to 68030 RAM. A GPIB block transfer is initiated by means of a direct memory access (DMA) to the National Instruments GPIB board, which ports the data block to the the altimeter host computer, over data dedicated GPIB bus 1 (see figure 2.2).

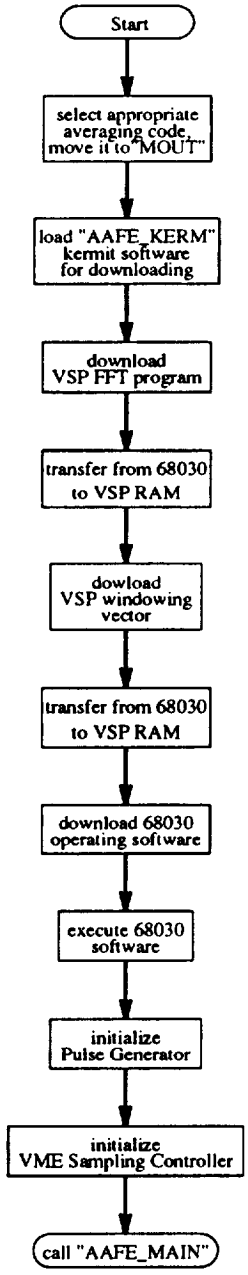

Figure 2.3 VIPER program downloading scheme: AAFE START flowchart

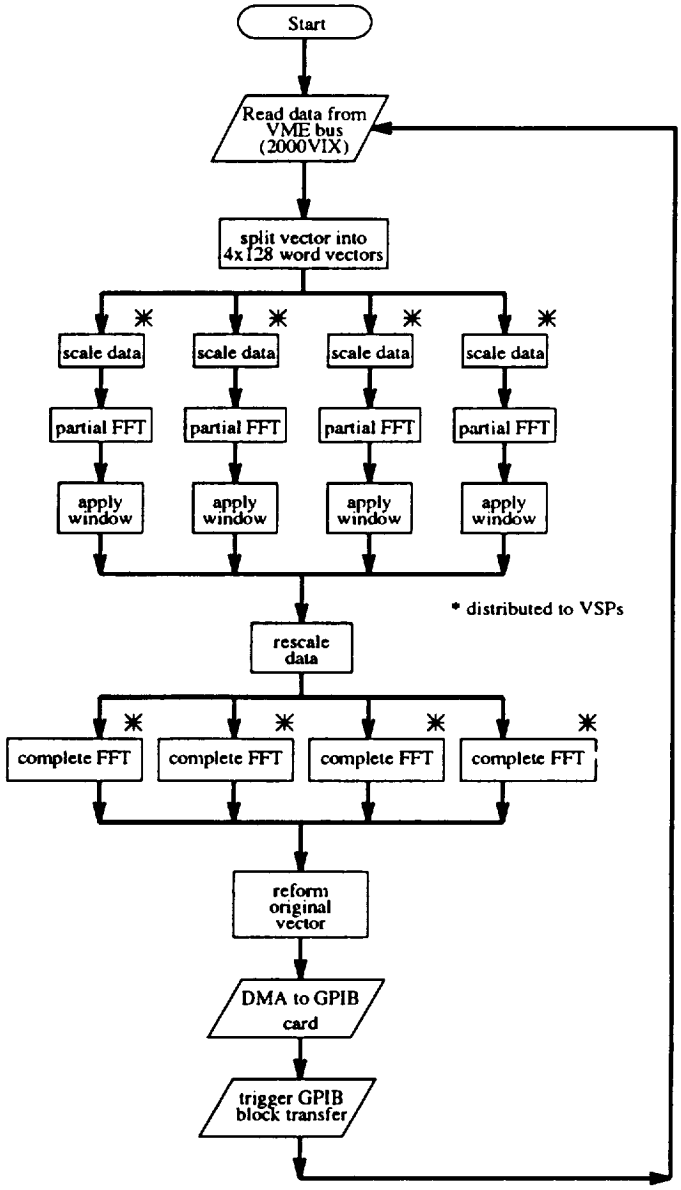

Figure 2.4 VIPER Data Processing Flowchart (including tasks distributed to 4 VSPs)

Currently, the VIPER program is the limiting factor determining the maximum data throughput of the system, restricting the radar to a pulse repetition frequency of 1250 Hz, if no averaging is performed. The structure of the program is a simple loop, with no multitasking. If features need to be added to this program, however, multitasking must be considered, in order to preserve the performance of the altimeter. One simple multitasking model that is often used, is double buffering. The 68030 VIPER controller can be fetching new data, while waiting for the previous data to be fourier transformed by the VSPs. In the programs present state, the 68030 is idle, when the VSPs are active, and the VSPs are idle when the 68030 is active.

2.3 RF Control Subsystem

The AAFE Altimeter RF Control Subsystem is a set of digital (TTL) timing circuits for exactly controlling the triggering of RF diode switches within the transmitter and receiver subsections. The system also uses a Stanford Research Systems Programmable Pulse Generator, for implementing variable delays. The timing circuitry, which resides with the control box, is controlled by the host computer system by means of the HP General Purpose Input / Output (GPIO) interface, while the SRS Pulse Generator gets its instructions over the GPIB bus. Both the GPIB bus and GPIO Interface are fully programmable using HP Basic, which allows for both hardware and software interrupting for preemptive multitasking applications using these peripheral boards [10].

The SRS Pulse Generator is used to generate an exact time delay between transmitted pulse, and dechirp pulse. This time delay is varied with a tracking loop, to keep the relatively small range resolution window of the full deramping chirp receiver centered on the ground return. During acquisition mode, when the altimeter searches for the ground, it is this time delay that is swept, as signal to

noise is estimated,. This pulse generator is also responsible for generating the pulse repetition frequency (PRF).

The control box generates several pulses, of various lengths, and sends them to transmitter, receiver, and data acquisition system, based on the two pulses it receives from the SRS Pulse Generator. It uses five programmable pulse generator chips, to control transmit pulse pulsewidth, chirp pulsewidth and delay, and de-chirp pulsewidth and delay. It also contains four latches which feed Emitter Coupled Logic (ECL) switch drivers, for controlling RF diode switches. These switches are used to re-route signals within the transmitter, and receiver, to enable closed loop transmission, open loop transmission, refered to as "radiate mode", and two calibration modes, one which measures the antenna length, and the other that measures internal delays("test target mode") .

2.4 Transmitter Subsystem

The AAFE Transmitter subsystem is the analog electonics package responsible for creating the chirp pulse waveform, amplifying it, and feeding it to the antenna system. It also provides several oscillator signals to the receiver subsystem for waveform decoding. The transmitter has two oscillators, which are combined with a series of mixers, amplifiers, and switches (see figure 2.5) to create a chirped pulse waveform, sweeping in frequency, from 13.720 GHz up to 14.080 GHz during the 3.0 μ s pulse. The transmitter also creates a replica of the chirp pulse, and feeds it to the full-deramping chirp receiver, which mixes it with the ground-reflected chirp, to create a range image, in the frequency domain.

Several different frequencies are created within the transmitter, which originate from a 108.0 MHz crystal oscillator. The 108MHz tone is fed to a frequency multiplier, which is specially designed to generate the fifth harmonic, and results, after filtering, in a 540MHz tone. The 108MHz signal is also fed to a frequency

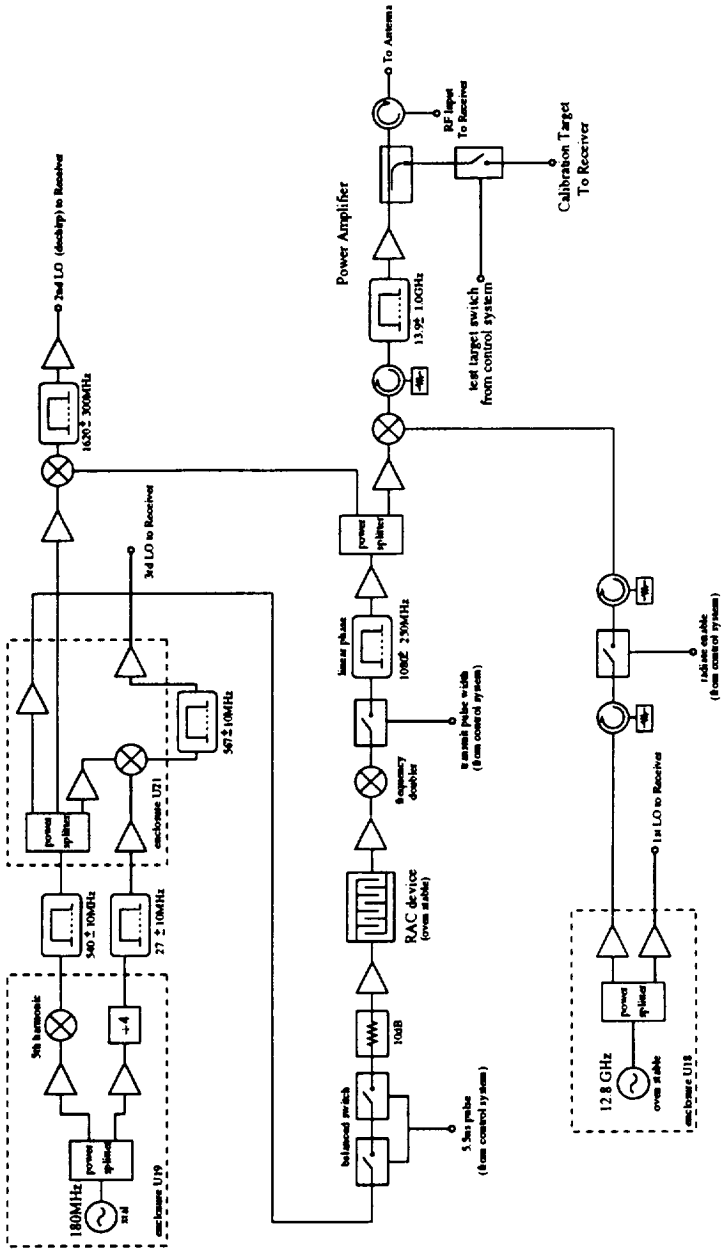

Figure 2.5 AAFE Transmitter Subsystem block diagram

divider, which generates a signal at one quarter of its input frequency, or 27MHz. These tones are mixed, and amplified to supply the 567MHz 3rd local oscillator to the receiver.

The transmitter also uses the 540MHz tone to create a chirp waveform by means of a temperature stabilized reflective array compressor (RAC). This device takes advantage of the acoustic properties of an etched crystal structure to induce frequency varying time delays. Acoustic transducers are used to inject and extract energy from the crystal. A spectrally rich signal, generated by modulating the 540MHz tone with a narrow (5.5ns) pulse, is fed to the input transducer. The acoustic energy propagates through an array of narrow bandwidth reflective gratings, etched in the crystal, which are arranged such that higher frequencies are directed through a longer propagation path than lower frequencies, effectively delaying frequencies in a linear fashion. Energy which was originally coincident in time, is now time delayed, according to its frequency, creating a long chirp from a short pulse having the same frequency content. The 5.5ns 540MHz pulse becomes a 3.0 μ s pulse which chirps from 450MHz to 630MHz.

The 540MHz chirp waveform is then used to create the Ku-Band chirp, by first feeeding it to a frequency doubler, which doubles its center frequency and bandwidth, and then to a linear phase filter, to clean up unwanted harmonics in the broadband signal. Next, the signal, which is now a 3.0μ sec chirp centered at 1080MHz, is amplified, and mixed with a 12.8 GHz tone, generated by a phase locked, temperature stabilized oscillator. This is the final output waveform which is amplified to 2 watts (peak) and applied to the antenna.

The filtered, amplified 1080MHz pulsed chirp waveform is also used to create the 1620MHz second receiver local oscillator. The chirp 1080MHz waveform is mixed with the 540 MHz tone, which was created, as described above, from the 108MHz

crystal oscillator. Unwanted harmonics are again filtered with a broadband linear phase filter, and the signal is amplified and patched to the receiver.

The transmitter contains several diode switches, which enable different modes of the system. The radiate enable switch passes the 12.8 GHz local oscillator to the output stage, making transmission possible. The pulse width switch in the 1080 MHz chirp generator path allows the 3μ sec pulse to be truncated to a shorter pulse, allowing the radar to operate at lower altitudes, trading resolution for shorter minimum range. Another RF diode switch is used to couple some of the transmit pulse energy to the receiver, where it can be used as a test waveform for the receiver.

By using two oscillators, several frequency shifters, mixers and RF switches, the AAFE transmitter susbsystem creates a pulsed FM chirp waveform, which is sampled by the receiver, and amplified for transmission through the antenna. It also provides several intermediate signals to the receiver for use as local oscillators, to be used in demodulating the received signal. All of the signals generated by the transmitter are controlled from the RF control system by means of TTL and ECL controlled RF diode switches.

2.5 Receiver Subsystem

The AAFE receiver subsystem uses local oscillator signals genrated by the transmitter, to demodulate and amplify the return waveform, apply a full deramping dechirp algorithm, and amplify the final intermediate frequency (IF) signal for sampling by the data acquisition system. See figure 2.6 .

The receiver front end consists of a short waveguide run from the transmitter circulator, which is fed to a mixer. The input signal is demodulated down to the first intermediate frequency (IF), where there are several stages of amplification and filtering amounting to 90dB of gain, with a bandwidth of 500MHz.

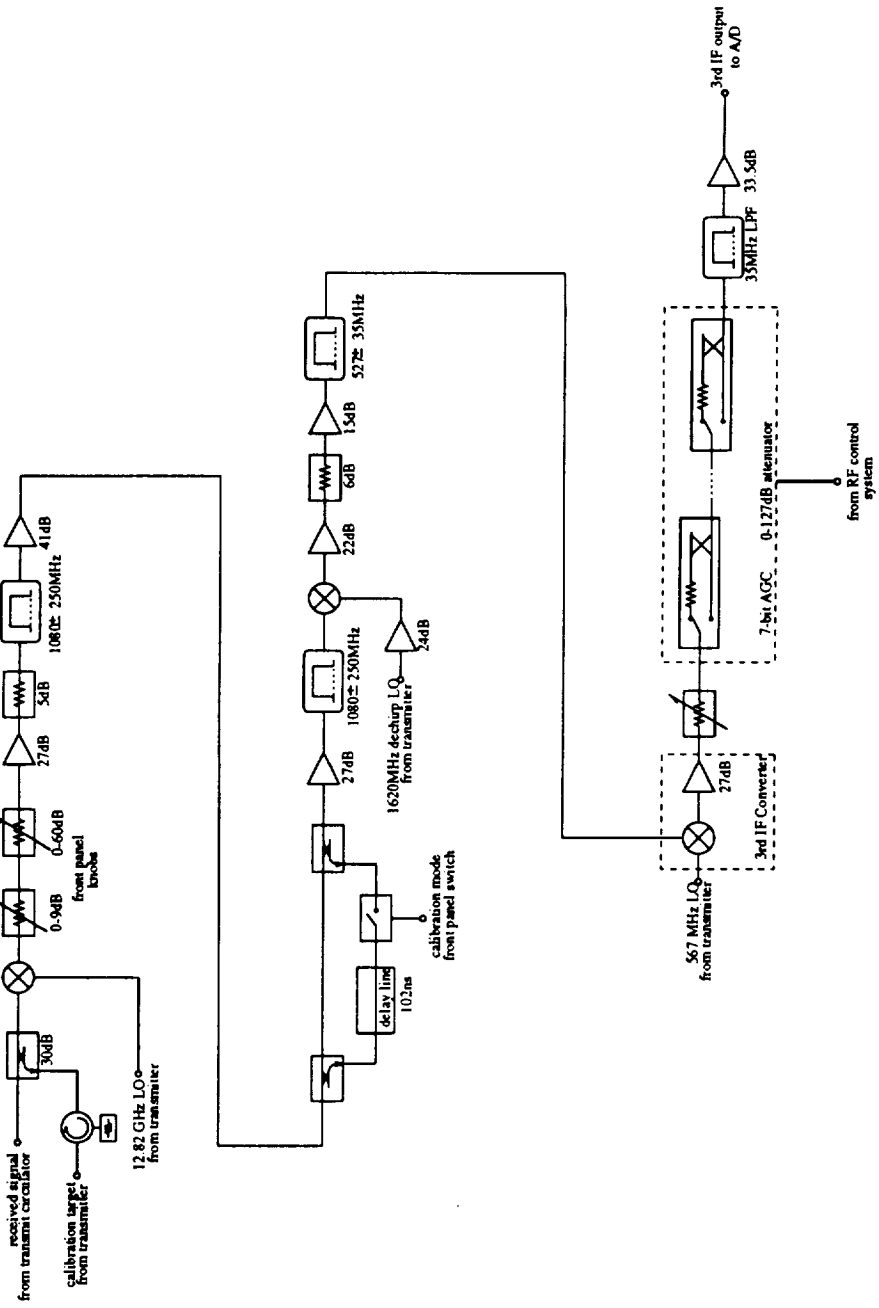

Figure 2.6 AAFE Receiver Subsystem block diagram

The noise figure for a cascade of components whose gain and noise figure are known is

$$F_{cascade} = F_1 + \frac{F_2 - 1}{G_1} + \frac{F_3 - 1}{G_1 G_2} + ... + \frac{F_N - 1}{G_1 G_2 ... G_{N-1}} \tag{2.1}$$

where N is the number of components in the component chain. A good approximation of a receivers noise figure can be made by considering just the components at the beginning of the receiver chain. As can be seen in equation 2.1, the noise figures of elements towards the end of a cascade are insignificant, if they come after an element with large gain. Considering components up to the 41 dB amplifier, the noise figure of the receiver is 12.5 dB, which is dominated by the front end losses, especially the conversion loss of the front end mixer of 8 dB, added with the insertion loss of the waveguide to coax adapter of 1.5 dB . It is important to note, here, that although the receiver has what is considered to be a bad noise figure, receiver sensitivity is not critical for the radar's performance as an airborne altimeter. Typical platform motion is slow enough to allow many pulses to be averaged, without significantly degrading resolution.

The next important feature in the receiver chain is the switchable delay line, before the second (de-chirp) LO. The delay line is included in such a way that it can add a delayed version of the incoming signal back on to the signal by using a 3 dB coupler. This allows time delay measurements made by the full deramping de-chirp algorithm to be calibrated with a single test target pulse.

Next, the received signal is passed through the second mixer, to be combined with the 1620 MHz chirped local oscillator pulse, performing a full deramping de-chirp, which results in a pulsed signal, which has time delay (range) information encoded as frequency shifts. This signal can later be converted to an amplitude versus range plot, commonly known as an A-Scope display, by means of a fast Fourier transform.

After being de-chirped, the received signal is then sent to another amplifier and filter chain, and then to the final mixer. The signal is combined with the 567 MHz local oscillator, resulting in the 3rd IF signal, which is amplified, filtered, and sent through an automatic gain control (AGC) component, to avoid saturation of A/D input buffer amplifier. The AAFE receiver uses a 7-bit switched attenuator capable of applying 0 to 127 dB of attenuation, for AGC, which is controlled with a software algorithm running on the HP382 host computer which detects saturation or low signal level, and adjusts the attenuation accordingly.

2.6 GPS Receiver Subsystem

The AAFE Altimeter has a Global Positioning System (GPS) receiver to provide altimeter position as well as an accurate time reference for time aligning the altimeter data with other airborne sensors, such as inertial navigation sensors, and laser altimeters, that often participate in airborne experiments with the AAFE Altimeter. The receiver itself is an Ashtech XII differential GPS receiver capable of P-code reception, which is connected to an L-Band patch antenna, mounted on the aircraft, and also connected to the AAFE host computer and VME data acquisition system.

The GPS receiver is controlled over one of its serial ports, using the HP 382 system controller. The controller configures the receiver outputs, to provide a ASCII time and position string, and 1 pulse per second time reference to the VME data acquisition system, as shown in figure 2.2. The second GPS receiver serial port is connected to the VIPER 68030 signal processing controller serial port, where the ASCII time and position string is read, compressed, and inserted into the header section of the AAFE data stream. The data blocks are then logged to hard disk by the HP 382 controller, via GPIB bus.

The ASCII string contains time information with 0.5 second resolution. Better time resolution is achieved by using the 1 pulse per second time reference. This

pulse is fed to a custom VME board which uses edge detection to phase lock to the 1Hz pulse, and provides 16 microsecond resolution time reference which is also logged to the header section of AAFE data blocks. Although the custom VME board was designed to give a very high resolution time measurement, the 1MHz crystal oscillator which provides this resolution has been observed to drift with temperature. As a result, the 50 ppm drift of the oscillator with temperature changes, results in a 50 μsec clock accuracy.

CHAPTER 3

RADAR ALTIMETRY PRINCIPLES

Altimetry is one of the most elementary radar applications. It usually consists of a nadir looking radar transceiver, as shown in figure 3.1, configured to transmit pulses and measure the round-trip time of flight, as the radiated microwave energy propagates down to the surface, is reflected, and travels back to be received. The range R is determined from the equation

$$R = \frac{cT}{2} \tag{3.1}$$

where T is the time of flight measured, and c is the speed of light [11]. Furthermore, the range resolution ΔR of such a radar as determined by the effective pulse width of the transmit pulse τ is:

$$\Delta R = \frac{c\tau}{2} \tag{3.2}$$

To construct an altimeter with range resolution on the order of tens of centimeters requires a pulse width of less than a nanosecond. The receiver bandwidth B required to receive such a pulse is

$$B = \frac{1}{\tau} \tag{3.3}$$

Typical altimeter bandwidths range from several hundred megahertz to a few gigahertz. The greatest disadvantage in using broadband receivers is the increased receiver noise associated with the increased bandwidth. The noise power received P_n is given by

$$P_n = kTB \tag{3.4}$$

where k is Boltzmann's constant and T is the antenna temperature. Radars with broad-band receivers require some combination of lower receiver noise and higher

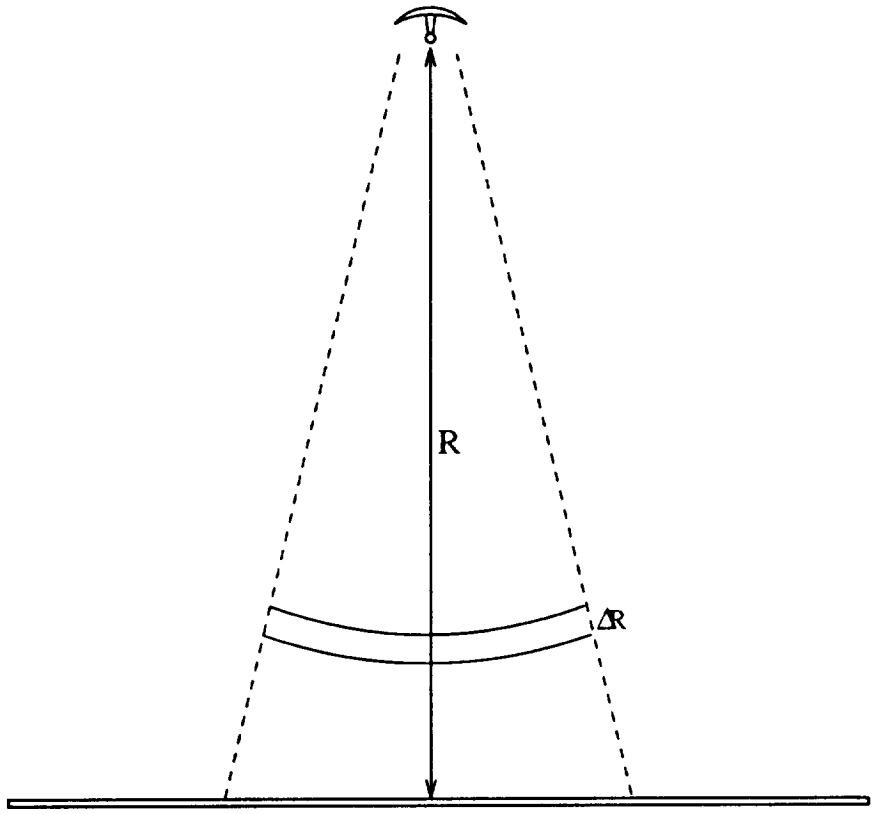

Figure 3.1 Radar Altimetry Measurement Geometry

transmit power to obtain a reasonable signal to noise ratio (SNR). Often, in radars deployed on airborne and spaceborne platforms where instrument weight is a design factor, designers will use pulse compression and its associated "compression gain" to increase transmitted power while avoiding heavy tube amplifiers which would otherwise be required.

One way to increase the energy transmitted in a pulsed radar is to lengthen the pulse in time. This method results in an increased duty cycle, which increases the average power P_{ave} independent of transmitter peak power, and pulse repetition frequency. This is seen in

$$P_{ave} = \frac{P_t \tau}{T_p} \tag{3.5}$$

where P_t is the peak transmitter power, and$\frac{\tau}{T_p}$ is the duty cycle, τ being the pulse width, and T_p being the pulse repetition period. There appears to be an engineering compromise here, between greater average power, using longer pulses and greater range resolution, using shorter pulses. This limitation is overcome, however, by lengthening the pulse, to increase the duty cycle, and modulating the signal within the pulse envelope, to increase it's bandwidth. This compromise yields both increased average power, and increased range resolution.

There are many methods of modulating the pulsed signal, but one of the most popular is linear frequency modulation, commonly referred to as "chirp" radar [12]. A chirp pulse is shown in figure 3.2. This function can be represented as

$$x(t) = \mathrm{Re}\left(\mathrm{rect}\left(\frac{t}{\tau}\right) e^{2\pi j(f_0 t + \frac{kt^2}{2})}\right) \tag{3.6}$$

where $\mathrm{rect}\left(\frac{t}{\tau}\right)$ is the envelope function for the pulse, f_0 is the center frequency of the chirp, and k is the time derivative of the chirp, such that the instantaneous frequency f_i is

$$f_i = f_0 + kt \tag{3.7}$$

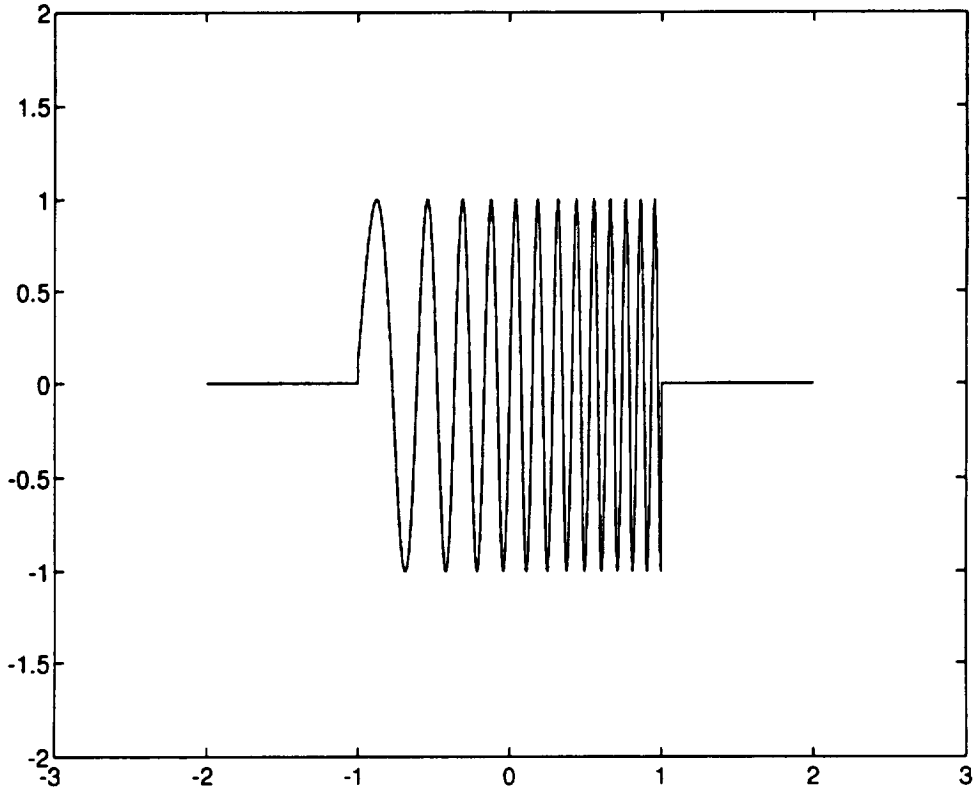

Figure 3.2 Linear FM Pulse "Chirp" Signal

Thus, the frequency sweeps linearly across the pulse, from $f_0 - k\left(\frac{\tau}{2}\right)$ to $f_0 + k\left(\frac{\tau}{2}\right)$. The linear chirp waveform is popular because it is easily created using analog hardware.

Often, a linear phase delay device, known as a dispersive delay line, is used, to convert a short pulse to a chirp. These devices are characterized by a figure of merit known as the compression ratio. The compression ratio Γ_c is determined from the actual and effective pulse widths using

$$\Gamma_c = \frac{\tau}{\tau_{eff}} \tag{3.8}$$

where the effective pulse width τ_{eff} is determined from the chirped bandwidth of the pulse B using

$$\tau_{eff} = \frac{1}{B} \tag{3.9}$$

and the bandwidth B is the chirped bandwidth, defined from above as

$$B = (f_0 - k\left(\frac{\tau}{2}\right)) - (f_0 + k\left(\frac{\tau}{2}\right)) = k\tau \tag{3.10}$$

The compression ratio Γ_c is then related to τ and chirp rate k as

$$\Gamma_c = k\tau^2 \tag{3.11}$$

or to the bandwidth B and pulse width τ as

$$\Gamma_c = B\tau \tag{3.12}$$

This compression ratio can also be interpreted with regards to its equivalent average transmitter power gain. That is, if the peak transmitter power were increased, instead of the duty cycle, the compression ratio could be called the "compression gain". In which case,

$$G_{compression} = 10 \log \Gamma_c = 10 \log(B\tau) \tag{3.13}$$

and the effective output power P_{eff} is this gain applied to the actual output power

$$P_{eff} = P_{out} + \Gamma_c dB \tag{3.14}$$

Compression gains for altimeters are typically between 25 and 35 dB. The AAFE Altimeter has a transmitted pulse width $\tau = 3.0\mu sec$ and a bandwidth $B = 360MHz$, which give a compression gain $\Gamma_c = 30.3dB$. With a peak output power of 2 watts, and the aforementioned bandwidth, the altimeter has an effective pulse width $\tau_{eff} = 2.7nsec$ and an effective output power $P_{eff} = 33.3dBW = 2.2$ kilowatts.

3.1 Target Return Waveform Models

The AAFE Altimeter uses pulse compression to achieve an effective pulse width of $2.7nsec$. This results in a raw range resolution, from equation 3.2, of 41 centimeters. This does not, however, indicate the altitude resolution that can be achieved by the instrument. Finer altitude resolution can be achieved if features within the return waveform can be identified. Waveform tracking in this manner relies on good surface reflection models to predict waveform shape.

3.1.1 Point Target Model

The AAFE receiver, as described in chapter 2, consists of two demodulation stages, and a full deramping dechirp stage. When using a full deramping chirp receiver, a time delay in the received waveform results in a frequency shift in receiver output. This implies that a power spectrum needs to be computed, to obtain traditional A-scope information (received signal magnitude versus range). See figure 3.3. The AAFE altimeter applies a Fast Fourier Transform (FFT) to the sampled receiver output, to create an A-scope waveform.

The simplest waveform model is generated by applying the receiver processing algorithm directly to the transmitted waveform, with no target interaction. This

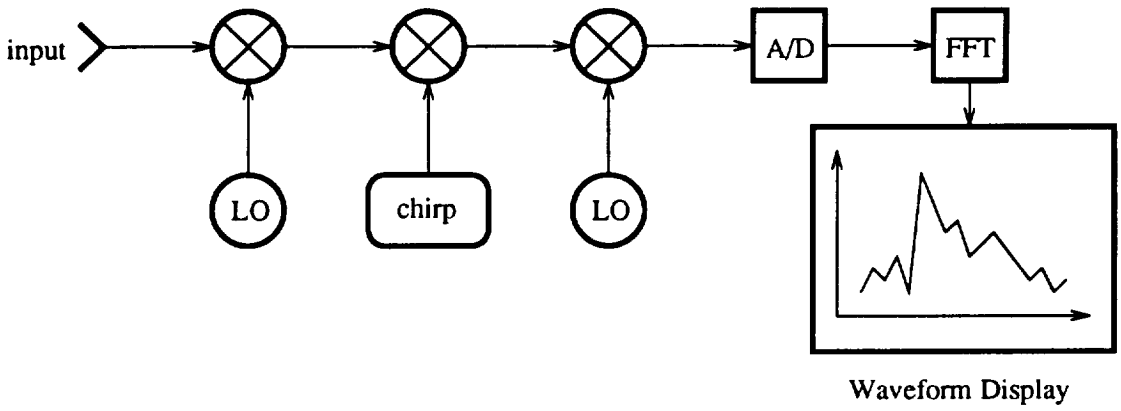

Figure 3.3 AAFE Chirp Receiver Signal Processing Algorithm

is often referred to as the point target response of the receiver. The point target response of the AAFE Altimeter is a CW pulse at the video frequency of 27Mhz whose pulsewidth is that of the transmit waveform, approximately 3μsec. See figure 3.4a. The VME based signal processing sub-system then performs a windowed FFT on the data, after sampling at 167 MHz, and returns this data to the controlling computer for display. See figure 3.4b.

The expected return from a distributed target such as glacial ice is quite different from a point target, due to interesting backscatter characteristics of the irradiated surface. Several factors, such as range difference from center to edge of antenna footprint, surface roughness, and change in surface water content (conductivity) contribute to target return magnitude and delay variation [6].

At this time, there are several models which are used for return waveform estimation. There is extreme variation in the dielectric properties of the ice sheet, which depend on its density and water content, making a single comprehensive mathematic model of the surface impractical. In extremely wet snow conditions where penetration depths as small as $2cm$ have been observed, a rough surface scattering model is used. In dry snow conditions, sub-surface scatterers are irradiated, due to depths of penetration as large as $5m$, and a volume scattering model must be applied [6].

3.1.2 Surface Scattering Model

The response of a rough surface can be expressed as the impulse response of a flat, smooth surface, convolved with the surface height distribution of the rough (randomly distributed) surface. This is then convolved with the system point target response, to model the return from a rough surface [6]. Taking the special case of a nadir looking altimeter, which assumes small incidence angles, a gaussian point target response approximation, an antenna pattern approximation :

$$G(\theta) = G_0 e^{-4ln2\frac{\theta^2}{\theta_{BW}^2}} \tag{3.15}$$

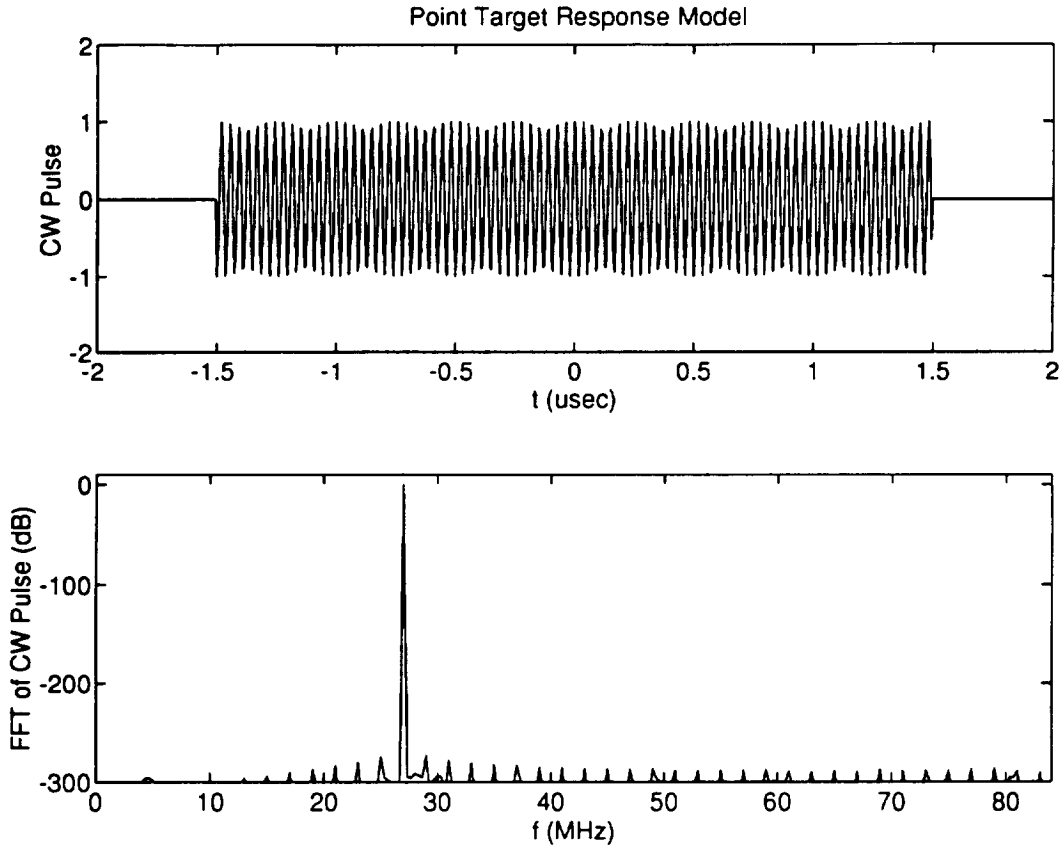

Figure 3.4 AAFE Point Target Response Model a) time domain, b) frequency domain, corresponding to range information (A-Scope Display)

and a backscatter coefficient:

$$\sigma^0(\theta) = \frac{\Gamma(0^o)}{s^2} e^{-\frac{\theta^2}{s^2}} \tag{3.16}$$

the rough surface response is:

$$P_{rs}(\tau) = \frac{C_0}{H^3 s^2} e^{(\frac{t_p}{t_s})^2} \; e^{-(\frac{2\tau}{t_s})} erfc(\frac{t_p}{t_s} - \frac{\tau}{t_p}) \tag{3.17}$$

where:

$$C_0 = \frac{P_t \lambda G_0^2 \Gamma(0^o)}{32\pi^2}, \tag{3.18}$$

$$t_s = \frac{2H}{c(\frac{8ln2}{\theta_{BW}^2} + \frac{1}{s^2})}, \tag{3.19}$$

$$t_p = \sqrt{2}\sqrt{(\frac{2\sigma_h}{c})^2 + \sigma_p^2}, \tag{3.20}$$

$$\tau = t - \frac{2H}{c}, \tag{3.21}$$

and c is the propagation speed, H is the altitude, θ_{BW} is the 3db beamwidth of the antenna, σ_p is the standard deviation associted with the gaussian shaped point target response approximation, s is the RMS slope of the surface, and σ_h is the standard deviation of the height of the surface.

Typically, surface model parameters H,s, and σ_h are extracted from measured waveforms in the wet-snow region, by fitting waveforms using least-mean squared error method (LMSE) [6]. This model tends to break down when surface water content is low, as when measuring percolation and dry snow zone surfaces. This break-down indicated by abnormally high values for s, because s is sensitive to the waveform trailing edge, where water content variations are thought to be indicated most clearly. See figure 3.5.

3.1.3 Volume Scattering Model

When glacier surface water content reduced, the decreased conductivity results in an increase in depth of penetration. When penetration depth increases

significantly, as is the case in the dry snow region of the Greenland ice sheet, a volume scattering model is applied. Because snow particle sizes ($d \leq 0.5$mm) are significantly less than the wavelength of the radiation ($\lambda = 2.16$cm) the scettering is known as Raleigh volume scattering.

The Raleigh volume scattering model for returned power is

$$P_{rv}(\tau) = \frac{C_2}{\beta c - 2\alpha c_s}(e^{-2\alpha c_s \tau} - e^{-\beta c \tau}) \tag{3.22}$$

where

$$C_2 = \frac{\sqrt{2} P_t \lambda^2 T^2 G_0^2 \eta_v}{32\pi^2 H^3}, \tag{3.23}$$

$$\beta = \frac{8ln(2)}{H\theta_B^2}, \tag{3.24}$$

$$\tau = t - \frac{2H}{c}, \tag{3.25}$$

$$c_s = \frac{c}{\sqrt{\epsilon_s}}, \tag{3.26}$$

and c is the propagation speed, H is the altitude,θ_{BW} is the 3db beamwidth of the antenna, P_t is the transmitted power, and ϵ_s is the the dielectric constant of snow at $f_0 = 13.9$GHz.

Examples of both the surface and volume scattering models using typical AAFE parameters are shown in figure 3.5. In the extreme cases of completely dry snow for the volume scattering model, and very wet snow for the surface scattering model, these models could be used individually. However, an incoherent sum of the two power models is used, to form a more general model, which is applicable across the spectrum of varying snow water contents.

3.2 Tracking Algorithms

Altitude resolution is dependent on the choice of tracking algorithm. If some a priori knowledge of the scattering properties of the surface are applied to the data,

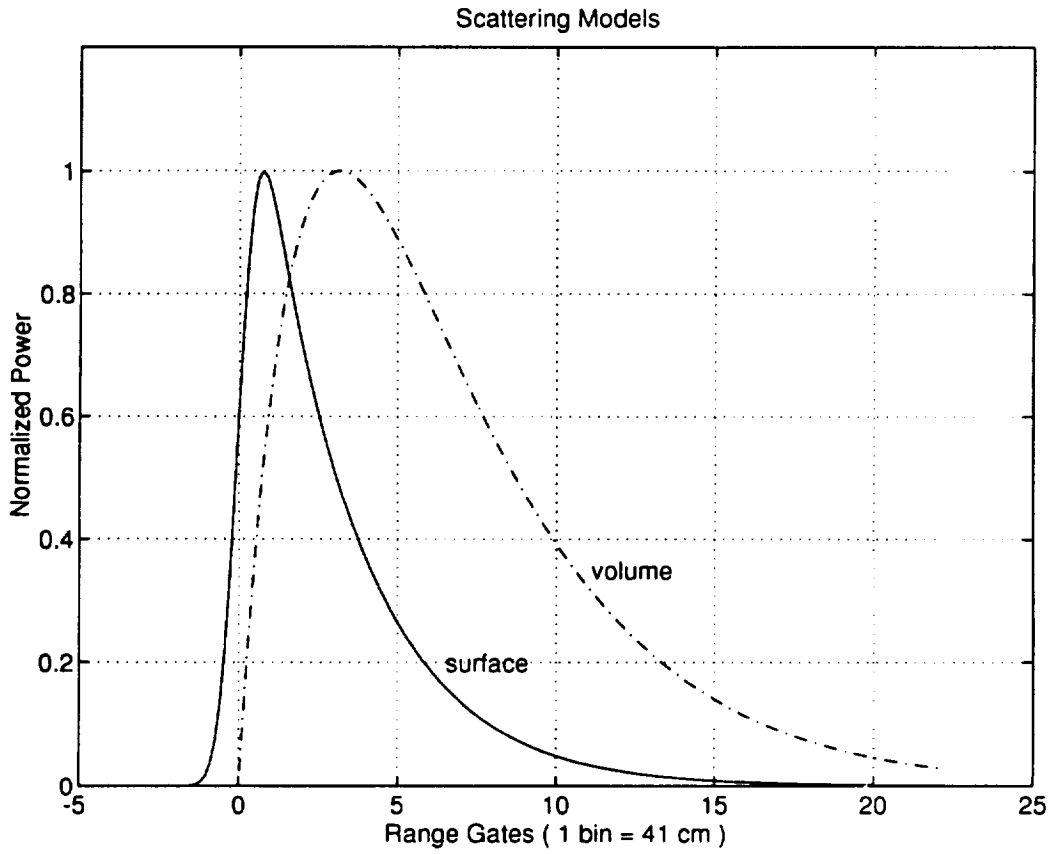

Figure 3.5 AAFE Scattering Models: Gaussian rough surface model, and Raleigh volume scattering model scaled to range gates (A-Scope Display).

the resolution can be further improved. This technique, known as retracking, is often applied to data during post-processing.

There are numerous tracking algorithms which have been used in altimetry. Some of the simpler and computationally faster methods have been integrated into the AAFE data acquisition and control software, which tracks terrain during flight, varying receiver delay, so that the return waveform remains centered in the acquisition window. Some of the more sophisticated and computationally intensive algorithms are used to retrack the AAFE data, after the flight mission is completed.

3.2.1 Maximum Value Tracker

One of the simplest tracking algorithms to implement is the maximum value tracker. This method simply finds the range gate with the largest detected signal and defines the mean range to the surface as the distance to it. This algorithm is computationally simple, but does not give a good altitude estimate, and does not enhance the raw range resolution of the radar. This algorithm is useful for in-flight tracking, during conditions of high signal to noise ratio, because of its computation speed.

3.2.2 OCOG Tracker

A more complicated tracking algorithm, which is sometimes used to track AAFE Altimeter data is the Offset Center of Gravity (OCOG) tracker [13]. This algorithm uses a simplified integration to determine the range to the surface, defined as the center of area of the pulse.

First the range gate normalized pulsewidth W is determined, using

$$W = \frac{(\sum_n p_n)^2}{\sum_n p_n^2} \tag{3.27}$$

where p_n is proportional to the power in range gate n. Next, the center of area of the pulse is calculated, using

$$\frac{(\sum_n np_n)^2}{\sum_n p_n^2} - \frac{W}{2} \tag{3.28}$$

which gives a more accurate estimate of the range to the surface than the maximum value tracker. Both of the algorithms are included in the AAFE mission program, and can be used to initially track the surface. The surface range estimate obtained from the tracking algorithm is used to keep the surface return signal within the data acquisition window, by means of software feedback.

3.2.3 Weighted Split-Gate Tracker

Another algorithm which is available in the AAFE control software is the weighted split-gate tracker. This tracking algorithm picks the center bin p_c in the waveform, such that

$$\rho = \frac{\sum_{k=0}^{c} p_k}{\sum_{k=c+1}^{n} p_k} \tag{3.29}$$

is kept constant, from pulse to pulse, where n is the number of range gates, and p_k is the power in each bin. The ratio ρ is typically 1:3 [13], but can be selected, at startup time, when operating the AAFE Altimeter. In figure3.6 a comparison of several tracking algorithms is shown, using a typical AAFE return model. WSG center bins are shown for $\rho = 1:3$ and $\rho = 1:10$.

It is assumed, when applying any of these tracking methods, that the surface return signal is present within the range window. This is not generally the case. Often, when operating conditions are extreme, or when the radar has just been activated, the range window delay is set incorrectly, and no signal is present within the range window. For this reason, an signal to noise estimate is made by computing the ratio of maximum to minimum power measurement across the window. If this ratio does not exceed $2dB$, the radar is programmed to search over a specified range, for the surface return. This search, called acquisition mode, begins at the pre-defined minimum range, making signal to noise estimates while incrementing the time delay

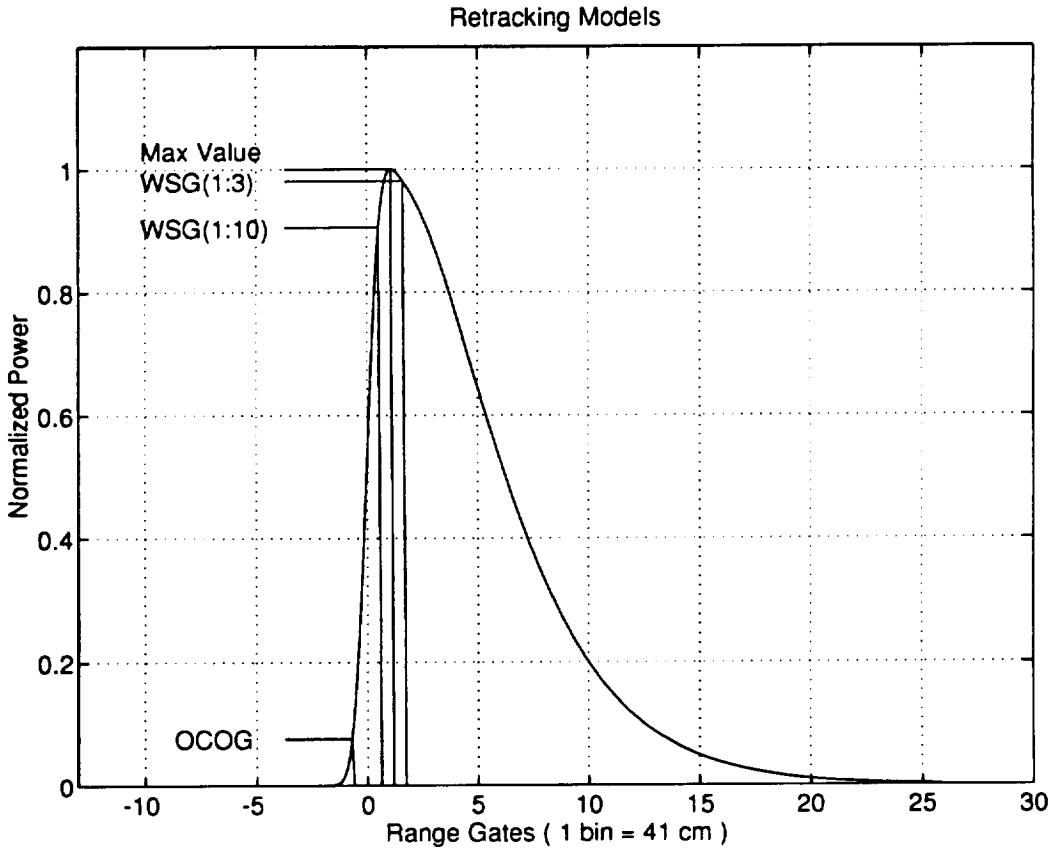

Figure 3.6 AAFE Retracking Algorithms: Several popular retracking algorithms, applied to a model AAFE return waveform.

of the range window. After the maximum range has been measured, the time delay with the highest signal to noise estimate is assumed to contain the surface return, and tracking mode begins at that setting.

CHAPTER 4

PRINCIPLES OF GPS OPERATION

The Global Positioning System (GPS) was developed by the US Department of Defense as a world wide navigation and synchronization tool. GPS capabilities are a direct result of the invention of portable atomic clocks, which represents an increase in timing accuracy of more than three orders of magnitude over anything previously available [14]. Throughout GPS development, significant research effort has been directed towards precision survey capabilities of the system. A GPS survey research timeline is shown in table 4.1. The development of kinematic differential GPS receivers with demonstrated 10cm accuracy [15], has enabled the use of airborne experimental platforms as GPS survey instruments. The following chapter will present some principles of the Global Positioning System, and discuss the operation of the AAFE Altimeter GPS receiver.

4.1 GPS System Theory

The Global Positioning System is a constellation of 18 satellites which transmit very precise time and location information, as well as pseudo-random digital sequence codes for high precision time resolution. GPS receivers can be configured to decode any or all of the encoded satellite data, depending on the receivers intended use. By carefully choosing a receiver strategy, time, geoid referenced position (latitude, longitude, and altitude), and velocity can be determined with great accuracy.

The most important element of the Global Positioning System is the GPS satellite. Each satellite is a beacon which transmits its on-board time and ephemeris

Table 4.1 Significant GPS survey research milestones [14].

Date	Achievement	Researcher	Accuracy
1982	First prototype GPS Receiver	MIT	1-2 ppm
1983	Eifel project: 30 station network	Eifel group	1-2 ppm
1984	Stanford Linear Accelerator(SLAC)	Stanford	1 mm
1985	Centimeter accuracy,roving antenna	NOAA (Remondi)	1 cm
1986	10cm aircraft positioning verified	NASA (Krabill)	10 cm
1987	High precision orbit determinations	Lichten & Border	0.02 ppm

(orbit position) data. At the heart of each satellite is a set of atomic clocks based on a cesium and rubidium quantum energy level transitions which emit energy at 9.1926316770 GHz and 6.834682613 GHz respectively. This clock configuration results in a stability $\frac{Deltaf}{f} = 10^{-12}$ which is equivalent to a 1 second inaccuracy every 30,000 years [14]. The two atomic clocks are mixed and divided to generate a 10.23 MHz stable local oscillator. The satellite transmits a navigation message packet, modulated with a C/A (coarse acquisition) and P (precision) pseudo-random sequence, on two L-band frequencies: $L_1 = 1575.42$ MHz and $L_2 = 1227.60$ MHz. Each satellite is identified by a unique C/A code, and a unique time shift of the 37 week long P code, and is capable of remote clock offset correction, from an earth based control station.

The GPS satellite network provides clock offset and satellite ephemeris data at 50bps, and several carrier modulation sequences to an unlimited number of receivers, which need not transmit any queries. A GPS receiver can be configured to extract any or all of the above information, using either a C/A code, or C/A and P codes, to phase lock to the transmitted carrier. C/A codes are typically used in stationary survey stations, where averaging provides the required accuracy, while the higher precision P codes are used for navigation of moving platforms, where averaging is impractical. Additional information can be gained from phase shift observations in the carriers, by using P code to reconstruct the carrier phase, and comparing it to the receiver LO. Carrier phase can be measured to $\frac{\lambda}{100}$ which yields a theoretical spatial accuracy of 2.5 millimeters.

The receiver determines the position of it's antenna, by using correlation sequences (codes) to determine the exact time of arrival, for a satellite's transmission. The time of transmission is encoded in the signal, along with the ephemeris of the transmitting satellite. The propagation time is used to calculate psuedo-range, which contains errors. Significant sources of error include satellite and receiver clock

inaccuracies, sattelite ephemeris errors, antenna multipath, and the unknown phase shift of the ionosphere. In general these errors can be removed by taking multiple measurements, in which the errors can be reduced to common-mode errors, and subtracted. The most accurate position determinations are made when multiple receivers observe multiple satellites, over multiple measurement epochs, on both frequencies. This technique is known as a triple difference measurement [14], or differential GPS.

4.2 GPS Receiver Operation

The Ashtech XII GPS receiver used in the AAFE altimeter system is capable of both C/A and P code operation, on both L_1 and L_2 . It can optionally log data to its internal memory, for later downloading and post processing, but this option is not currently implemented. For the maximum positioning accuracy, P-codes are used to reconstruct carrier phase, but this is more computationally intensive than is feasible in real time, within the receiver. In the current implementation, the GPS time and position string is logged to the data acquisition system, without P-code information. The NASA/NAVSTAR GPS group, which operates several GPS receivers in with carrier phase decoding, provided more accurate GPS data (after post-processing), for the 1993 Greenland mission. If the AAFE altimeter were to be operated as a stand-alone instrument, this post processing would need to be implemented, using the data logging feature of the Ashtech receiver.

Chapter 5

System Improvements

The radar system was improved in order to prepare the AAFE Altimeter for the 1993 Greenland glacial survey experiment. Ailing RF components were discovered and replaced, and new functionality was added to the control and data acquisition software. Both hardware and software modifications will be described.

5.1 Hardware Improvements

In any airborne hardware installation, reliability is an important design criterion. Several recent transmitter and receiver upgrades were made, not necessarily to improve altimeter performance, but, in most cases, to replace worn and aged hardware. There were also hardware additions, to integrate the GPS receiver into the data acquisition system.

Before the mission, the AAFE receiver was functioning quite poorly. Calibration returns had unusually high noise levels, and this signal was detected only intermittently. First, a general diagnostic was performed on the receiver, testing each active component with a network analyzer. This turned up a bad 24dB IF amplifier in the chirp LO stage (see figure 2.6), which was replaced with an off-the-shelf Miteq 1GHz amplifier. The amplifier which failed was originally custom built by Hughes, but, the new amplifier had better gain, and noise figure, and was available immediately, reflecting advances in microwave electronic devices in the 18 year life of the original amplifier.

Further down the receiver chain, the switching attenuator unit was found to be failing. This was another one-of-a-kind part, originally fabricated by Hughes

in the early '70s, which had simply worn out. Once again, technology had caught up with the altimeter, and a replacement part was available from several vendors, immediately. A 127 dB (7 bit digital) switched attenuator was installed to replace the worn part, and provide AGC capabilities.

Repairs were also made to the altimeter transmitter hardware. The waveform generator, responsible for providing several receiver LO signals as well as the transmitted waveform, seemed to be malfunctioning. Local oscillator signals measured with a spectrum analyser seemed to have weak amplitudes, and formerly filtered harmonics ± 27 MHz away from the 567 MHz LO were present in the receiver LO, effectively creating false returns in the range image. Additional filters were added at the output of the 567 MHz generator knocking the harmonics down to reasonable levels. This was a temporary solution. As this intermittent malfunction becomes more frequent, better diagnostics can be performed.

The AAFE altimeter used an external travelling wave tube (TWT) amplifier as the final power stage, providing 2 Watts peak transmitted power. The TWT package was heavy, required a large power supply, and was unreliable: a backup TWT amplifier was required for all flights. At the time of construction, tube amplifiers were the only technology capable of generating 2W in the Ku band. Modern transistor technology has advanced, to the point where solid state amplifiers are capable of generating the necessary RF power, much more reliably, and requiring a smaller power supply. A 2W Ku-band power amplifier was purchased, along with the appropriate DC power supply, and installed in the transmitter, as shown in figure 2.5.

The most significant change in the altimeter hardware was the addition of the Ashtech GPS receiver. Integrating the GPS data from the receiver into the altimeter data stream required the construction of a custom VME interface board. This board synchronizes its 1MHz digital clock with the GPS pulse per second clock output,

and provides 16μ sec time resolution clock across the VME bus to the 68030/VSP (Viper) board.

The board was constructed using a Logic Design Group VMEbus 5100D Prototype Circuit Card [16]. The card provides TTL logic interfaces to the VME address, control, and data buses, and provides breadboard space and power supply connections for the user circuitry. The circuit was implemented using standard TTL chips, with wire-wrapped interconnections. The final circuit design is shown in figure 5.1. Output is the 16 most significant bits of a 20-bit counter which is clocked with a 1.0000 MHz crystal oscillator. The circuit also blinks a LED on the prototype board front panel, to verify that the GPS pulse per second signal is being detected.

In addition to these receiver, transmitter, and data acquisition system modifications there were many other minor hardware modifications. Many broken cables were replaced, and repairs were made to waveguide components. New computer hardware, including a 1.0 GB hard disk and 4.0 GB 4mm digital tape drive for data storage were installed.

5.2 Software Improvements

In addition to the hardware changes, there were quite a few software modifications that were made. New HP-BASIC software was written to interface to the Ashtek GPS receiver, and altimeter operating software was improved both at the HP-BASIC level, and the Viper 68030 assembly language level, to integrate the GPS data into the altimeter data stream.

The first new piece of software is a simple GPS receiver interface program, written in HP-BASIC, to run on the HP 9000/382 system controller. This program uses one of the HP serial ports to configure the GPS receiver (which can also be done manually from the receiver front panel) and synchronize the system controller

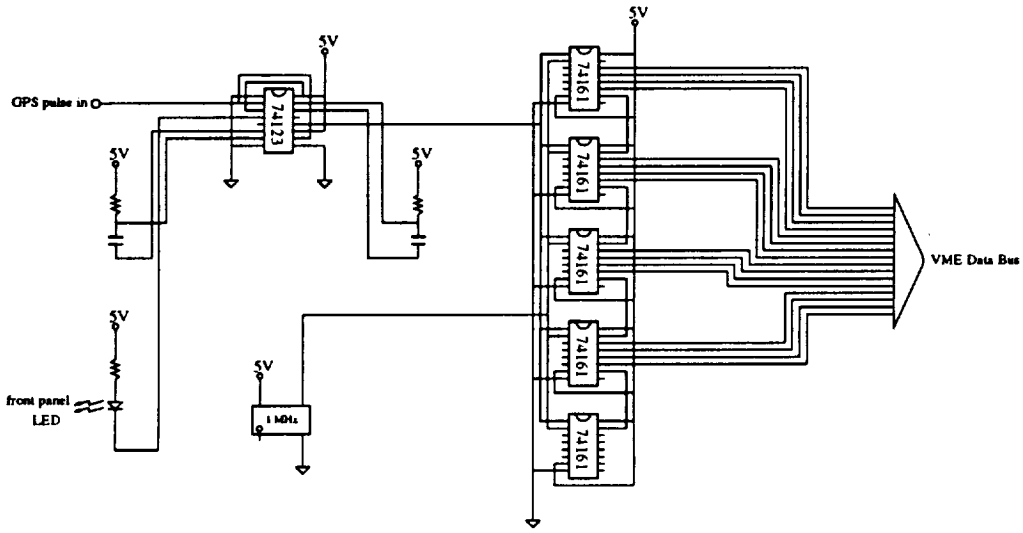

Figure 5.1 Prototype VMEbus GPS clock interface circuit

clock with GPS time. This feature is necessary to provide a backup timestamp for altimeter data. Each data record is stamped with GPS time and position data, in the VME data acquisition system, and then, when it is passed to the HP system controller, it is stamped again, with HP time and altimeter configuration information, such as de-chirp delay time, AGC setting, number of pulses being averaged. With an HP timestamp, GPS recording can be interrupted without invalidating the altimeter data.

Modifications to the HP-BASIC control program were made to display GPS data in the real-time display. This required re-defining several bytes of the AAFE data block header, to include the data, and re-writing the display routine, to include more text data in the window. The new header block format is shown in table 5.1.

More significant changes were made to the 68030 assembly language program. It has the responsibility of reading the GPS string as it arrives on the Viper card serial port. This task is not quite as simple as it first seems. Data are sent by the GPS receiver asynchronously, such that the viper program must be able to read and process the string, compressing it, and placing it in the data memory buffer, while also handling its traditional task, controlling data flow to and from the signal processors and GPIB card.

An interrupt driver algorithm was used, with normal data flow processing continuing as it did in the previous implementation. A software interrupt was enabled which is triggered by the arrival of new data at the SCN68681 DUART serial port chip. The interrupt service routine (ISR) would read the data into the GPS data buffer, reset the serial port, and exit, allowing data processing to continue. Although the 68030 is idle approximately 50% of radar operating time, there are time critical routines interfacing with the VSP control and memory buses, which require the ISR to be small and fast. For this reason, serial port communication is configured for the fastest baud rate: 38.4 K bps, and the routine is a simple block transfer to an

Table 5.1 New AAFE Altimeter data block header

Word	Format	Contents
1-4	packed string	"AAFE...#"
5-6	32 bit int	frame number
7	byte	GPIB byte count
8	byte	GPIB status
11-13	16 bit ints	VME prototype card data
14	16 bit int	waveform peak value
15	16 bit int	number of averages
20-24	16 bit ints	HP clock (d,h,m,s,msec
30	16 bit int	Radar mode
31	byte	AGC control byte (MSB unused)
32	byte	next AGC control byte
35-69	64 bit ints	receive delay data
70	16 bit int	new AGC value
71-74	16 bit ints	delay last modified (h,m,s,msec)
75-78	16 bit ints	AGC last modified (h,m,s,msec)
79-82	16 bit ints	GPS time (h,m,s,msec)
83-94	packed string	24 char position string (lat,long,altitude)
101-356	16 bit ints	normalized A-Scope waveform image

intermediate buffer. The main processing loop was also modified to complete the data transfer by copying the most recently received GPS data from the intermediate buffer to each block it transfers.

By being careful to write a small, fast interrupt service routine, time critical data transfer routines are only preempted briefly. The software modifications were tested using a PRF of 750 Hz, with no pulse averaging, which was the maximum data processing capability of the system, before modification. Results showed that data flow was not significantly reduced by the added processing.

Significant time was spent diagnosing, repairing, and upgrading the AAFE altimeter. After these hardware and software repairs and improvements were made, the radar was ready to be deployed for the 1993 Greenland glacial ice survey.

CHAPTER 6

1993 GREENLAND MISSION DESCRIPTION

The AAFE Altimeter was deployed along with two laser altimeters, a UHF glacial ice profiler and an array of differential P-code GPS receivers on a NASA airborne platform, to survey the Greenland glacial ice cap, during the summer of 1993. NASA's N426NA P-3 Orion was based in Kangerlussuaq, Greenland from 6/21/93 to 7/10/93 .

6.1 Experiment Procedure

The 1993 Greenland Mission began with pre-staging and aircraft installation at NASA Wallops Flight Facility, Wallops Island, Virginia. The radar was installed on board, placing the rack in the forward laboratory bay. See figure 6.1. Waveguide was run from the transciever isolation circulator (see figure 2.5) to the horn antenna, which was installed in the P-3 bomb bay. A pressure window was installed in the waveguide near the cabin floor. All mechanical modifications to the airframe were performed by flightline mechanics, with safety inspections performed by NASA Quality Control department. After the waveguide was installed, guide length measurements were made, and a VSWR measurement was made using a network analyzer, and terminating the waveguide with a matched load. After satisfactory waveguide impedance was measured, the antenna was connected.

Next, a series of flight tests were performed. Two test flights were made, on 6/03/93 and 6/14/93. The aircraft was flown from Wallops Flight Center, and surveys of Chincoteague Bay were made. Data was also taken over the runway, as

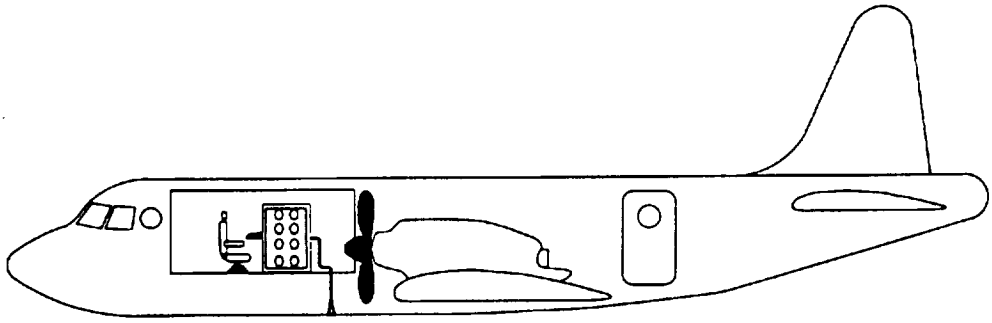

Figure 6.1 AAFE Altimeter Installation on NASA P-3

a calibration. At this point, the radar appeared to be working satisfactorily, with a signal to noise ratio (SNR) of approximately 9 dB. The transit flight to Greenland was made after some final aircraft repairs.

Glacier survey missions began on 6/23/93 and proceeded whenever weather permitted, until 7/9/93. Flights were conducted only when clear visibility from the aircraft to the ice was expected, as the laser altimeters could not tolerate snow rain or fog. A typical survey flight began with a ground-based calibration while the aircraft was sitting on the runway, followed by a calibration pass over the runway, after takeoff. Mission flight lines consisted of a short transit, an outbound survey line, and a short traverse to another survey line for the inbound trip. Flights lasted between six and eight hours, including transits, and ground calibration and data backup typically took another hour or two.

Several minor electronics problems were encountered during the missions. Most were due to mechanical problems associated with severe vibrations during landings. Data was lost on one occasion, because the serial port connector on the Viper board failed. As a solution, both RJ-11 (phone jack) style connectors were replaced with 15-pin D subminiature connectors, with screw locks. As the missions progressed, data quality began to degrade. Low SNR was observed, and, during several calibrations, waveform artifacts attributed to harmonic leakage in the waveform generator were observed. This problem was thought to have been fixed earlier, with the addition of some filters (see figure 2.5) but, it was clear that some active component was gradually failing. Luckily, the component did not completely fail, before all missions were completed.

The field experience also uncovered several shortcomings in the operation of the AAFE Altimeter. It was often difficult to quantitatively evaluate the performance of the radar. The operator would determine radar health, in an overall sense, by observing graphical data on the real-time display: if the data appeared reasonable,

the radar was assumed to be operating normally. If the waveform plot had an unexpected shape, little could be inferred about radar performance, because of the complicated nature of the hardware, and variability of target terrain. A diagnostic self-check routine would be a useful addition to radar control software. With the addition of several RF couplers and RF detector diodes, and a simple A/D board, signal levels within the RF subsytem could be checked before and during missions, in an automated fashion.

Despite several problems with aging microwave electronics hardware, the AAFE altimeter performed adequately, in a microwave radar survey of the southern portion of the Greenland ice sheet. Further repairs and hardware replacements, particular in the waveform generating components of the transmitter are recommended, however, before more missions can be performed reliably.

CHAPTER 7

CONCLUSION

The relationship between glacial ice sheets and global climate is not simple or well known. Ice sheet mass balance can be used as an indicator of long term climatic changes, but is also considered to have short term effects on other environmental parameters. Glacial ice is thought to influence temperature and circulation patterns in the atmosphere. It is clear that measurement of glacial ice sheet dynamics is critical to our understanding of the global climate.

Airborne remote sensing offers an efficient means of measuring ice surface elevation, particularly since Global Position Satellite systems can be used to determine the position of the aircraft platform within centimeters. This thesis documents the preparation, deployment and operation of one such airborne sensor, a Ku-Band microwave pulse compression radar, which is currently being maintained, upgraded and operated by the Microwave Remote Sensing Laboratory at the University of Massachusetts.

Documentation for the Advanced Application Flight Experiment (AAFE) pulse compression radar altimeter and its role in the NASA Multisensor Airborne Altimetry Experiment over Greenland in 1993. A description of radar hardware, software, and theory of operation is provided, including documentation of recent changes to the radar system. It is intended that this thesis will be a useful guide for future caretakers of the AAFE Altimeter.

APPENDIX A

HP-BASIC SOFTWARE

The following programs are HP-BASIC programs used for data aquisition during the Greenland Mission. The programs were run as described in chapter 2. First GPS_READ is run, to synch the computer's clock to the GPS clock. Then AAFE_START is invoked, to download programs to the VIPER data acquisition system. The last task AAFE_START performs is execution of AAFE_MAIN, the actual control program for the radar. Should AAFE_MAIN crash, AAFE_RUN can be called to restart the system quickly, assuming that code does not have to be downloaded to the VIPER modules again.

A.1 GPS_READ

```
10    CLEAR SCREEN
20    COM /Gps/ Instring$[96],INTEGER Address,REAL Gps_time,INTEGER Sync
30    ! DATACOMM IS AT 20
40    !
50    Address=20
60    Sync=0
70    !
80    Instring$="---n/a--- ? ----n/a--- ?"
90    Gps_time=-1.0
100   ! INITIALIZE SERIAL PORT FOR GPS READER
110   !
120   CALL Init_serial
130   !
140   !
150   !  SETUP INTERRUPT SERVICE ROUTINE CALL (GPS READER)
160   !
170   PRINT "CREATING INTERRUPT VECTOR."
180   ON INTR Address CALL Read_gps
190   PRINT "CREATED."
```

```
200   !
210   !
220   !  SET INTERRUPT MASK FOR DATA DETECTED
230   !
240   PRINT "ESTABLISHING INTERRUPT MASK."
250   ENABLE INTR Address;1
260   PRINT "DONE."
270   !
280   !
290   ON KEY 1 LABEL "Sync  to   GPS" CALL Sync_enable
300   ON KEY 2 LABEL "  Stop" CALL Disconnect
310   ON KEY 3 LABEL " " GOTO Begin
320   ON KEY 4 LABEL " " GOTO Begin
330   !
340   !
350 Init:  Icount=1
360        PRINT TABXY(33,12);"GPS Interface"
370 Begin: PRINT TABXY(31,14);"GPS Time: ";Gps_time;"          "
380        PRINT TABXY(31,15);"HP Time: ";TIME$(TIMEDATE)
390        PRINT TABXY(31,17);"Lat:  ";Instring$[1,9];" ";Instring$[11,11];"  "
400        PRINT TABXY(31,18);"Lon: ";Instring$[13,22];" ";Instring$[24,24];"  "
410        D$="**"
420        PRINT TABXY(17,20);"
430        PRINT TABXY(17+(Icount MOD 40),20);D$
440        Icount=Icount+1
450   GOTO Begin
460   !
470   END
480   !
490   !
500   SUB Init_serial
510   COM /Gps/ Instring$[96],INTEGER Address,REAL Gps_time,INTEGER Sync
520   !
530   !
540   !--- RESET BOARD
550   !
560   CONTROL Address,0;1
570   !
580   !--- SET MODE TO ASYNCHRONOUS
590   CONTROL Address,3;1
600   !
610   !--- RESET BOARD AGAIN TO ENTER NEW MODE
620   CONTROL Address,0;1
630   !
640   !--- SET BAUD RATE TO 9600
650   CONTROL Address,21;14
```

```
660   !
670   !--- TURN OFF HARDWARE HANDSHAKING
680   CONTROL Address,23;0
690   !
700   !--- SET CHARACTER LENGTH TO 8 BITS
710   CONTROL Address,34;3
720   !
730   !--- SET STOP BITS=1
740   CONTROL Address,35;0
750   !
760   ! --- SET PARITY 0=NONE
770   CONTROL Address,36;0
780   !
790   ! --- CONNECT TO LINE
800   CONTROL Address,12;1
810   !
820   ! --- SET # OF TERMINATORS TO 1
830   CONTROL Address,28;1
840   !
850   ! --- SET TERMINATOR TO <CR>
860   CONTROL Address,29;13
870   !
880   PRINT "Serial Port Initialized."
890   !
900   SUBEND
910   !
920   !
930   SUB Read_gps
940   COM /Gps/ Instring$[96],INTEGER Address,REAL Gps_time,INTEGER Sync
950   !
960   !  PUT THE GPS IN STRING$
970   !
980   PRINT TABXY(30,23);"Attempting Read."
990   !ENTER Address USING "#,7X,K";Gps_time
1000  ENTER Address USING "#,7X,K,24A";Gps_time,Instring$
1010  !
1020  ! SYNC HP TIME TO GPS TIME
1030  !
1040  IF Sync<>1 THEN GOTO Reset
1050  !
1060  Sync=0
1070  Secs=Gps_time
1080  Hours=Secs DIV 10000
1090  Secs=Secs-(Hours*10000)
1100  Mins=Secs DIV 100
1110  Secs=Secs-(Mins*100)
```

```
1120  Totalsecs=Secs+(Mins*60.0)+(Hours*3600.0)
1130  SET TIME Totalsecs
1140  DISP "SETTING HP TIME: ";Hours;":";Mins;":";Secs
1150 Reset: !--- RESET BOARD AGAIN TO CLEAR BUFFER
1160  CONTROL Address,0;1
1170  CONTROL Address,21;14
1180  CONTROL Address,23;0
1190  CONTROL Address,34;3
1200  CONTROL Address,35;0
1210  CONTROL Address,36;0
1220  CONTROL Address,12;1
1230  CONTROL Address,28;1
1240  CONTROL Address,29;13
1250  PRINT TABXY(30,23);"                "
1260  ENABLE INTR Address;1
1270  SUBEND
1280  !
1290  !
1300  SUB Sync_enable
1310  !
1320  COM /Gps/ Instring$[96],INTEGER Address,REAL Gps_time,INTEGER Sync
1330  !
1340  Sync=1
1350  !
1360  SUBEND
1370  SUB Disconnect
1380  ! DISCONNECT CLEANLY FROM GPS INTERFACE....STOP PROGRAM
1390  !
1400  COM /Gps/ Instring$[96],INTEGER Address,REAL Gps_time,INTEGER Sync
1410  !
1420  CONTROL Address,12;0
1430  !
1440  STOP
1450  !
1460  SUBEND
```

A.2 AAFE_START and AAFE_MAIN

The programs AAFE_START and AAFE_MAIN were orignially to be included here but after viewing the listings in their entirety, it was decided that they would be omitted for the sake of brevity. The listings are several hundred pages long, and can be found in the MIRSL internal document "AAFE listings" should they be required.

APPENDIX B

MOTOROLLA 68030 ASSEMBLY SOFTWARE

The following program is a Motorolla 68030 program used for data aquisition during the Greenland Mission. The program is assembled on a PC, uploaded to the HP 382 radar contol computer, and subsequently, downloaded to the VIPER embedded DSP system.

B.1 A68K64T.8_2

```
* A68K64T.ASM  ver 8.2   (does not include ISR for GPS)
*  This is the source code for the AAFE altimeter host processor (a Motorola
*  68030) residing on the Impact Technologies VIPER VSP card.
*  SPECIAL NOTE:  With this version, we do the sample averageing in the VIPER!
*  This change was necessary to properly calculate an average power spectrum.
*  The data are first converted to a power spectrum on a frame by frame basis,
*  then averaged for "no_avgd" cycles (nominally 128). TIMING is now CRITICAL!
*  See the _FRESH_DATA subroutine for more detailed information.
*      The GPIB transfer routine has been moved to the end of the main loop.
*  Previously, it was run in parallel with the FFT phase 1 operation.  The
*  get timing data routines will now be run in parallel with the FFT phase 1.
*      The GPIB initialization has been changed to send 356 words of ASCII
*  "HI".  This change is to please the HP-9000/300 computer.
*      This version allows timing of the FFT process via the VIPER ISR
* CTRL-6 line.  CTRL-6 is LOW for transfer of data from the Analytek and HIGH
* for FFT calculations, waiting for new data, and GPIB transfer.
* >> NOTE: Version 4.2 fixes a subtle bug in the main timing loop...without
*   this fix, the loop averages ONE MORE than the correct number of samples!
* >>>>> Version 5.2 implements 68030 instruction and data cacheing, using the
* necessary CACR commands to start the caches within the initialization
* routine.  This results in a significant improvement in process loop
* execution speed.  8/26/91 wcb
* >>>> Version 6.2 changes the number of sampled points in the AGC max value
* computation algorithm from 32 to 64.  This will improve AGC performance,
* but it may reduce process loop speed somewhat.
* >>>> Version 7.2 was found to be ineffective, since timing problems also
* present in the HP prevented total synchronization of the two machines.  So,
* 7.2 IS NOT VALID FOR AAFE!!
* >>>> Version 8.2 implements changes that allow mods in the Analytek Hard-
```

```
* ware to perform scaling of input data directly.  This will greatly speed up
* the data transfer, and will increase the loop execution speed, allowing for
* higher system PRFs. (Pulse Rep Rates with software scaling were limited to
* 750 Hz or less).  These changes are primarily located in the _FRESH_DATA
* routine.  USE THIS VERSION ONLY WITH versions of WORKS.VSP 8.0 or later!!
*
* This version supports the BLOCK FLT PT mode FFT calculations for the VIPER!
* >>ATTENTION!<< Registers A6, A7, D5, D6 and D7 are RESERVED.  DO NOT MODIFY!
*******  TABLE  OF EQUATES        ******
*
        ORG             $10000          * 68K Code Segment Start Add.
h_stack         EQU     $6000           * 68K Code Stack Address.
isr             EQU     $F8F50000       *Viper INPUT STATUS REGISTER
*
cfg_reg         EQU     $FEFD0000       * 68K Configuration Register Add.
std             EQU     $00182701       * Standard CFG REG setup word
DMA_mod         EQU     $00102301       * Mod to allow DMA via VMEBus A24:D16
Dcach_mod       EQU     $00186701       * Mod allows Data Cache activation.
* NOTE: Above values changed to allow enabling of instruction and data caches.
* REMEMBER: While DCach_mod is active, ALL OTHER VMEBus USERS ARE LOCKED OUT!
*   8/27/91  wcb.
*
in_buf_base_1   EQU     $F8F3C000       *Input Buffers (for Analytek
in_buf_base_2   EQU     $F8F3D000       *Hardware) Base Addresses.
in_buf_base_3   EQU     $F8F3E000
in_buf_base_4   EQU     $F8F3F000
in_68k_work     EQU     $00015000       *Input data buffer for 68K peak avg.
*
vsp_bottom      EQU     $F8F20000       *Lowest VSP data address.
vsp_num_sam     EQU     $F8F20040       *VSP Data Seg location of num_samples.
*
*
sum_vec_base    EQU     $F8F20100       *Sum Vector Start Location.
v_shift_addr    EQU     $F8F20008       *Address of VIPER bias shift word.
*
sam_scl         EQU     $6666           *Default scale for 64 samples.
no_avgd         EQU     $0040           *# of Samples to be Averaged (64).
v_bias_shift    EQU     -8190           *Bias shift: TWO bit shift (6dB gain).
*>>>>>>>NOTE: Change the above values for 128, 256, 512 samples!
*
v_fetch_1       EQU     $0000           *Start addresses for VSP modules.
v_fetch_2       EQU     $0040           *(Fetch Module.)
v_fetch_3       EQU     $0080
v_fetch_4       EQU     $00C0
v_fftb_1        EQU     $0100           *(FFT Phase 2.)
v_fftb_2        EQU     $0140
```

```
v_fftb_3        EQU     $0180
v_fftb_4        EQU     $01C0
v_clear_1       EQU     $0200           *(Clear VIPER output spectrum.)
v_clear_2       EQU     $0220
v_clear_3       EQU     $0240
v_clear_4       EQU     $0260
*
v_1_start       EQU     $F8F4060C       *Viper Program Counter Set Registers.
v_2_start       EQU     $F8F40E0C
v_3_start       EQU     $F8F4160C
v_4_start       EQU     $F8F41E0C
*
v_spectrum      EQU     $F8F22000       *Viper Output Spectrum Base Address.
out_spectrum    EQU     $00014000       *68K Output Spectrum Base Address.
VME_out         EQU     $00114000       *VMEBus Output Spectrum Base Address.
*
v_scale         EQU     $F8F20020       *Scale Factor external mem location.
old_max_1       EQU     $F8F40608       *VSP 1 Old Max Scale Register addr.
old_max_2       EQU     $F8F40E08       *VSP 2 Old Max Scale Register addr.
old_max_3       EQU     $F8F41608       *VSP 3 Old Max Scale Register addr.
old_max_4       EQU     $F8F41E08       *VSP 4 Old Max Scale Register addr.
v_scl_ram1      EQU     $F8F40620       *VSP 1 Scale RAM Address.
v_scl_ram2      EQU     $F8F40E20       *VSP 2 Scale RAM Address.
v_scl_ram3      EQU     $F8F41620       *VSP 3 Scale RAM Address.
v_scl_ram4      EQU     $F8F41E20       *VSP 4 Scale RAM Address.
*
* Output header data.
AAFE            EQU     $41414645       * ASCII hex representation of "AAFE"
dots            EQU     $2E2E2E23       * ASCII         hex for "...#"
*
* >>>> TIME CODE I/F Addresses (on VME-5100D VMEBus I/F proto card).
tim_ctr         EQU     $FFFFFF84       * GPS 1pps time reference counter.
tim_wda         EQU     $FFFFFF88       * DataMetrics time word A.
tim_wdb         EQU     $FFFFFF90       * DataMetrics time word B.
*
* >>>> 68030 CACHE CONTROL BITS <<<<
*
idis            EQU     $00             * Instruction Cache DISABLE.
iena            EQU     $01             *     "         "   ENABLE.
ifrz            EQU     $02             *     "         "   FREEZE.
iclr            EQU     $08             *     "         "   CLEAR.
ibst            EQU     $10             *     "         "   BURST ENABLE.
ddis            EQU     $000            * Data Cache DISABLE.
dena            EQU     $100            *   "    "   ENABLE.
dfrz            EQU     $200            *   "    "   FREEZE.
dclr            EQU     $800            *   "    "   CLEAR.
```

```
dbst            EQU     $1000           *   "    "   BURST ENABLE.
dwrt            EQU     $2000           *   "    "   WRITE ALLOCATE.
*
* >>>> GPIB-1014 unique equates! <<<<
*
GPIB_Add        EQU     $06          * 1014 GPIB address:6.
BASE            EQU     $FFFF6000    * A16 SPACE: VME BASE ADDRESS FOR 1014.
*
* The following defines the memory map of a single
* channel on the Hitachi 68450.
*
*                  DMA REGISTER GROUP
*
MAR     EQU     $0C                     * memory address register
MTCR    EQU     $0A                     * memory transfer counter
MFCR    EQU     $29                     * memory function codes
*
DAR     EQU     $14                     * device address register
DFCR    EQU     $31                     * device function codes
*
BAR     EQU     $1C                     * base address register
BTCR    EQU     $1A                     * base transfer counter
BFCR    EQU     $39                     * base function codes
*
CSR     EQU     $00                     * channel status register
CER     EQU     $01                     * channel error register
DCR     EQU     $04                     * device control register
OCR     EQU     $05                     * operation control register
SCR     EQU     $06                     * sequence control register
CCRR    EQU     $07                     * channel control register
CPR     EQU     $2D                     * channel priority register
*
GCR     EQU     $FF                     * general control register
*                  INTERRUPT VECTOR REGISTERS
NIVR    EQU     $25                     * normal interrupt vector
EIVR    EQU     $27                     * error interrupt vector
*                  CONFIGURATION REGISTER GROUP
CFG1    EQU     BASE+$101               * config reg 1
CFG2    EQU     BASE+$105               * config reg 2
*                  Channel offset for accessing channel 0-3
CH0     EQU     0
CH1     EQU     $40
CH2     EQU     $80
CH3     EQU     $C0
*
CH0CSR  EQU     (CH0+BASE+CSR)
```

```
CH0CER  EQU     (CH0+BASE+CER)
CH0DCR  EQU     (CH0+BASE+DCR)
CH0OCR  EQU     (CH0+BASE+OCR)
CH0SCR  EQU     (CH0+BASE+SCR)
CH0CCR  EQU     (CH0+BASE+CCRR)
CH0CPR  EQU     (CH0+BASE+CPR)
*
CH0MTCR EQU     (CH0+BASE+MTCR)
CH0MFCR EQU     (CH0+BASE+MFCR)
CH0DFCR EQU     (CH0+BASE+DFCR)
CH0BFCR EQU     (CH0+BASE+BFCR)
CH0BTCR EQU     (CH0+BASE+BTCR)
CH0BAR  EQU     (CH0+BASE+BAR)          * address of cc array
CH0DAR  EQU     (CH0+BASE+DAR)          *
CH0MAR  EQU     (CH0+BASE+MAR)          *
*
CH1CSR  EQU     (CH1+BASE+CSR)
CH1CER  EQU     (CH1+BASE+CER)
CH1DCR  EQU     (CH1+BASE+DCR)
CH1OCR  EQU     (CH1+BASE+OCR)
CH1SCR  EQU     (CH1+BASE+SCR)
CH1CCR  EQU     (CH1+BASE+CCRR)
CH1CPR  EQU     (CH1+BASE+CPR)
*
CH1MTCR EQU     (CH1+BASE+MTCR)
CH1MFCR EQU     (CH1+BASE+MFCR)
CH1DFCR EQU     (CH1+BASE+DFCR)
CH1BFCR EQU     (CH1+BASE+BFCR)
CH1BTCR EQU     (CH1+BASE+BTCR)
CH1BAR  EQU     (CH1+BASE+BAR)          * address of cc array
CH1DAR  EQU     (CH1+BASE+DAR)          *
CH1MAR  EQU     (CH1+BASE+MAR)          *
*                  7210TLC registers
DIR     EQU     BASE+$111
CDOR    EQU     BASE+$111               * COMMAND/byte out register
ISR1    EQU     BASE+$113               * interrupt STATUS register 1
IMR1    EQU     BASE+$113               * interrupt mask register 1
ISR2    EQU     BASE+$115               * interrupt STATUS register 2
IMR2    EQU     BASE+$115               * interrupt mask register 2
SPSR    EQU     BASE+$117               * serial poll STATUS register
SPMR    EQU     BASE+$117               * serial poll mode register
ADSR    EQU     BASE+$119               * address STATUS register
ADMR    EQU BASE+$119                   * address mode register
CPTR    EQU BASE+$11B                   * COMMAND PASS THRU REgister
AUXMR   EQU BASE+$11B                   * auxiliary mode register
ADR0    EQU BASE+$11D                   * address register 01
```

```
ADR      EQU BASE+$11D                  * address register 01
ADR1     EQU BASE+$11F                  * ADDRESS REGISTER 1
EOSR     EQU BASE+$11F                  * end of string register
*
ADMR_TRMODE          EQU $30            * TRANS/REC BITS
ADMR_ADMODE          EQU $01            * NORMAL DUAL ADDRESING
*                  68450 DMAC register definitions
*                  Configuration Register 1(cfg1)bits
CFG1_OUT   EQU $00                        * direction memory to GPIB
CFG1_IN    EQU $01                        * direction GPIB to memory
CFG1_DBM   EQU $02                        * disarm Bus Monitor mode
CFG1_ECC   EQU $04                        * arm automatic carry cycle feature
CFG1_BRG0  EQU $00                        * select bus requestgrant line 1
CFG1_BRG1  EQU $08                        * select bus requestgrant line 1
CFG1_BRG2  EQU $10                        * select bus requestgrant line 2
CFG1_BRG3  EQU $18                        * select bus requestgrant line 3
*
*                  Configuration Register 2(cfg2)bits
CFG2_SC    EQU $01                        * set system controller (SC)bit
CFG2_LMR   EQU $02                        * set local master reset bit
CFG2_SPAC  EQU $04                        * set supervisor only access
CFG2_SFL   EQU $08                        * clear SYSFAIL line
*
*                  Control masks for hidden registers (auxmr)
HR_ICR           EQU $20                * HIDDEN REG VIA AUXMR
HR_PPR           EQU $60                *         "
HR_AUXRA         EQU $80                *         "
HR_AUXRB         EQU $A0                *         "
HR_AUXRE         EQU $C0                *         "
*
PPR_U            EQU $10                * UNCONFIGURE PPR
PPR_S            EQU $80                * STATUS BIT POLARITY
*
*                  7210 AUXiliary Commands
AUX_PON          EQU $00                * Immediate Execute pon
AUX_SEOI         EQU $06                * Send EOI
AUX_DSC          EQU $14                * DISABLE SYSTEM CONTROL
AUX_RST          EQU $02                * TLC RESET
*
*
******  END TABLE OF EQUATES  *******
**
*******  >>> MAIN ROUTINE <<< *******
**
_MAIN
                 MOVEA.L         #h_stack, A7   * USE AS STACK ADDRESS.
```

```
                BSR             _INIT           * First run initialization.
*
* Now change the system configuration to allow for instruction cacheing.
        MOVE.L          #std, cfg_reg
* Turn ON the Instruction Cache at this point.
        MOVE.L      #iena+iclr+ibst, D0      * Enable instruction cache with
        MOVEC       D0, CACR                 * Burst Mode fetch.
*
* Then acquire Analytek Data, up to # of points to be averaged.
MN_FETCH        BSR             _FRESH_DATA     * Find Fresh Data Location.
*
**
* Start the VSPs Fetching Fresh Data.
**
                BSR             _TAKE_VBUS
                MOVE.W  #v_fetch_1, v_1_start   * Tell 'em where to go!
                MOVE.W  #v_fetch_2, v_2_start
                MOVE.W  #v_fetch_3, v_3_start
                MOVE.W  #v_fetch_4, v_4_start
                MOVE.W          #$00C0, isr             *(Release VSP Bus.)
*
* This code computes a running sum of the largest input Analytek data point
* magnitude, which is later averaged, then written to the output buffer to
* be sent to the HP.  This info is needed by Ellen's tracker algorithm.
                CLR.L           D4              * Holds peak value.
                MOVE.L          #63,  D1        * Set up loop counter.
                MOVE.L  #in_68k_work, A2        * 68K input work addresses.
MAX_PEAK        CLR.L           D3              * Holds value to be tested.
                MOVE.W          (A2)+, D3       * Bring a point in.
                SUB.W           #$07FF, D3      * Bias Shift (now signed!)
                BPL             POS_VALUE       * Positive numbers jump thru.
                NOT.W           D3              * Take Abs value of D3...
                ADDQ.W          #1, D3          * ...etc.
POS_VALUE       CMP.W           D3, D4
                BGT             NO_CHANGE
                MOVE.W          D3, D4
NO_CHANGE       DBMI            D1, MAX_PEAK    * Repeat for  32 points.
                ADD.L           D4, D5          * >>>D5 IS RESERVED!!  Holds
*                                                 Sum of all peak values.
*
                BSR     _WAIT_10us
* Poll to verify Fetch task complete.
MN_POLL_FETCH   MOVE.W          isr, D0         * Move ISR into Data Reg.
                BFCLR           D0,{0:28}       * Clear Bits not tested.
                CMPI.L          #$000F, D0      * Are all interrupts set?
                BNE             MN_POLL_FETCH   * If not, try again...
```

```
**
*  DETERMINE THE MAX SCL FACTOR.
**
                BSR             _TAKE_VBUS      * First take the Viper bus.
                CLR.L           D0              * Clear working Registers.
                CLR.L           D1
                CLR.L           D2
                CLR.L           D3
                CLR.L           D4
*
                MOVE.W          v_scale, D0     * Bring in VSP Scale Regs.
                MOVE.W          v_scale+2, D1
                MOVE.W          v_scale+4, D2
                MOVE.W          v_scale+6, D3
*
                MOVE.W          D0, D4          * Find Max Scale nibble.
                CMP.B           D4, D1
                BLS             TRY_D2
                MOVE.W          D1, D4
TRY_D2          CMP.B           D4, D2
                BLS             TRY_D3
                MOVE.W          D2, D4
TRY_D3          CMP.B           D4, D3
                BLS             GOT_IT
                MOVE.W          D3, D4          * D4 holds Max nibble.
GOT_IT          MOVE.W          D4, old_max_1   * Finally, write this word
                MOVE.W          D4, old_max_2   * to the Old Max Scale add-
                MOVE.W          D4, old_max_3   * resses of all 4 VSP chips.
                MOVE.W          D4, old_max_4
* Now align all the individual scale nibbles into a Scale Vector that all
* the VSPs will use for the final pass.  Notice D0 is already properly
* aligned, so we'll build the scale vector into D0.
                BFINS           D1, D0{24:4}    * D1 after D0,
                BFINS           D2, D0{20:4}    * D2 after D1,
                BFINS           D3, D0{16:4}    * D3 after D2.
*
                MOVE.W          D0, v_scl_ram1  * Stuff into VSP Scale RAMs.
                MOVE.W          D0, v_scl_ram2  * Stuff into VSP Scale RAMs.
                MOVE.W          D0, v_scl_ram3  * Stuff into VSP Scale RAMs.
                MOVE.W          D0, v_scl_ram4  * Stuff into VSP Scale RAMs.
*
                MOVE.W          #$00C0, isr             *(Release VSP Bus.)
*
                BSR     _WAIT_10us
**
* Start the FFT Phase 2 Operation.
```

```
**
                BSR             _TAKE_VBUS
                MOVE.W  #v_fftb_1, v_1_start    * Tell 'em where to go!
                MOVE.W  #v_fftb_2, v_2_start
                MOVE.W  #v_fftb_3, v_3_start
                MOVE.W  #v_fftb_4, v_4_start
                MOVE.W          #$00C0, isr     *(Release VSP Bus.)
*
                BSR     _WAIT_10us
*
* Poll to verify FFT Phase 2 task complete.
MN_POLL_FFTB    MOVE.W          isr, D0         * Move ISR into Data Reg.
                BFCLR           D0,{0:28}       * Clear Bits not tested.
                CMPI.L          #$000F, D0      * Are all interrupts set?
                BNE             MN_POLL_FFTB    * If not, try again...
*
* That completes this sample cycle, continue for the number of averaged
* samples, then reset the sample counter and move on...
                DBMI            D7, MN_FETCH    * Loop until done.
                MOVE.L        #no_avgd-1, D7    * Reset samples counter.
*
* Read most current time data, and stuff into output frame.
                MOVE.W  tim_ctr, out_spectrum-180
                MOVE.W  tim_wda, out_spectrum-178
                MOVE.W  tim_wdb, out_spectrum-176
*
* Now let's finish up the average peak Analytek input value calculation.
                DIVU.W          #no_avgd, D5    * Complete avg of peak value.
                BFCLR           D5{0:8}
                MOVE.W  D5, out_spectrum-174    * Quotient moved to output.
                CLR.L           D5              * Ready for the next round!
*
* The following code transfers first 256 words of spectral data from Viper
* address space to 68K address space pending output via next GPIB cycle.
*
MN_XFR          BSR             _TAKE_VBUS
                MOVEA.L    #v_spectrum, A0 * Start point in Viper for xfr
                MOVEA.L    #out_spectrum, A1    * in 68K for xfr.
                MOVE.L          #255, D2        * Words to be moved, less 1.
XFR_LOOP        MOVE.W          (A0)+, (A1)+    * Move the data.
                DBMI            D2, XFR_LOOP    * Do until D2 < 0.
*
                MOVE.W          #$00C0, isr             *(Release VSP Bus.)
* Now Clear the Output Spectrum in VSP Space.
                BSR             _TAKE_VBUS
                MOVE.W  #v_clear_1, v_1_start    * Tell 'em where to go!
```

```
                MOVE.W  #v_clear_2, v_2_start
                MOVE.W  #v_clear_3, v_3_start
                MOVE.W  #v_clear_4, v_4_start
                MOVE.W          #$00C0, isr     *(Release VSP Bus.)
*
                BSR     _WAIT_10us
*
* Poll to verify VSP Clearing task complete.
MN_POLL_CLR     MOVE.W          isr, D0         * Move ISR into Data Reg.
                BFCLR           D0,{0:28}       * Clear Bits not tested.
                CMPI.L          #$000F, D0      * Are all interrupts set?
                BNE             MN_POLL_CLR     * If not, try again...
*
* Place the current frame counter value in the output frame, and then
* increment the counter.
* ATTENTION: This makes D6 a >>RESERVED<< register!
                MOVE.L          D6, out_spectrum-192
                ADDI.L          #1, D6
*
**
*  GPIB-1024 Handling routine.  This version supports DMA GPIB transfers.
*  7/10/91  wcb.
**
* After setting up as a GPIB talker,
* the first thing we do is test to see if we have been granted GPIB Active
* Talker status.  If not, we will branch around the TALK_TEST loop this
* time around, and not begin output until we have Talker status.  NOTE: the
* frame counter will NOT increment if we do not ship out a frame.  This
* will be needed to sync with other radar system data.  When GPIB transfer
* is complete, we will drop out of active talker status, to free up the bus.
_GPIB   MOVE.B      #CFG1_BRG3+CFG1_DBM, CFG1 * use bus request/ grant line 3
*                                             disarm bus monitor
*                                             ROR=1 NO RELEASE ON REQUEST
        BSR         _WAIT_100us
        MOVE.B      CPTR, D1                * clear regs by reading
        MOVE.B      ISR1, D1
        MOVE.B      ISR2, D1
        MOVE.B      #GPIB_Add, ADR          * set GPIB address; 6
        MOVE.B      #$00, SPMR              * Clearing serial poll mode reg
        MOVE.B      #$31, ADMR              * set TRM1 and TRM0, addr mode 1
        BSR         _WAIT_100us
        MOVE.B      #HR_ICR+15, AUXMR       * set internal counter reg to 8
*                                             book says to do this
        BSR         _WAIT_100us
*                   WRITE TO HIDDEN REG PPR WITH U=UNCONFIGURE.
        MOVE.B      #HR_PPR+PPR_U, AUXMR      * parallel poll unconfigure
```

```
        BSR         _WAIT_100us
*                   WRITE TO HIDDEN REG AUXRA
*                   EOS AT 7 BITS
*                   NO XMIT END WITH EOS
*                   NO SET END ON RECEIPT OF EOS
*                   NORMAL HANDSHAKE  ( NO RFD HOLDOFF ON END OR DATA)
        MOVE.B      #HR_AUXRA+0, AUXMR
        BSR         _WAIT_100us
*                   ISS,INV,TRI,SPEOI, AND CPT ENABLE SET TO 0
        MOVE.B      #HR_AUXRB+0, AUXMR
        BSR         _WAIT_100us
*                   DHDC, DHDT SET TO 0 (HOLDOFF ON DCAS & DTAS)
        MOVE.B      #HR_AUXRE+0, AUXMR
        BSR         _WAIT_100us
*
*  Request Service Interrupt, to let the HP know we have data for it.
*
PEND_TST    MOVE.B          SPSR, D0      * Get pending bit
            BTST.L          #6, D0        * If not zero,
            BNE             PEND_TST      * wait until it is.
*
** Now we will toggle the state of ISR_CTRL6, which is available as an
** external monitor point, so we can observe the timing of the S-Poll response.
*           MOVE.W      #$0080, isr     * This sets ISR_CTRL6 low, but keeps
**                                        the VIPER bus released.
            MOVE.B          #$40, SPMR      * Generate SRQ.
*
PEND_TST2   BSR              _WAIT_3us
            MOVE.B          SPSR, D0        * Confirm SRQ ACK
            BTST.L          #6, D0          * If not zero, wait
            BNE             PEND_TST2       * until it is.
*
            MOVE.W      #$00C0, isr     * Returns ISR_CTRL6 to high state.
*
SEND_DATA       BSR             _WAIT_100us
*
                MOVE.B          #$B0, ADMR      * fORCE TO tALK oNLY.
                MOVE.B          #$06, ADR       * RE-ENABLE TALK/LISTEN
                BSR             _WAIT_100us
                CLR.L           D0
TALK_TEST       MOVE.B          ADSR, D1
                BTST.L          #1, D1          * Are we Active Talker?
                BEQ             TALK_TEST       * If not, hang in there.
*

** Now we will toggle the state of ISR_CTRL6, which is available as an
```

```
** external monitor point, so we can observe the timing of the GPIB transfer.
*           MOVE.W      #$0080, isr     * This sets ISR_CTRL6 low, but keeps
**                                          the VIPER bus released.
*
* Next change the 68K configuration to allow A24:D16 VMEBus access to the
* local DRAM.  This will also turn ON User IND LED (Yellow) on the VIPER.
                MOVE.L          #DMA_mod, cfg_reg
*
* Then we set up configuration registers for Channel 0 fly-by DMA transfer
* as detailed in the GPIB-1024 manual, Chapter 5.
                MOVE.B          #$1A, CFG1      * Set Configuration Reg. 1.
                BSR             _WAIT_3us       * Wait 3 microseconds.
                MOVE.B          #$10, CH0CCR    * Abort stops undesired proc.
                MOVE.B          #$10, CH1CCR    * Abort stops undesired proc.
                BSR             _WAIT_3us       * Wait 3 microseconds.
                MOVE.B          #$FF, CH0CSR    * Reset Chip.
                MOVE.B          #$FF, CH1CSR    * Reset Chip.
                BSR             _WAIT_3us       * Wait 3 microseconds.
                MOVE.B          #$E0, CH0DCR    * Config Device.
                MOVE.B          #$05, BASE+GCR  * DMAC hold: 20usec.
                BSR             _WAIT_3us       * Wait 3 microseconds.
                MOVE.B          #$02, CH0OCR    * Memory to GPIB direction.
                MOVE.B          #$04, CH0SCR    * Address Cntr counts UP.
                MOVE.B          #$06, CH0MFCR   * AM code allows A24 R/W.
                MOVE.B          #$00, CH0BFCR   * Zero unused register.
                MOVE.B          #$00, CH0DFCR   * Zero unused register.
                BSR             _WAIT_3us       * Wait 3 microseconds.
                MOVE.L  #VME_out-200, CH0MAR    * Start Address.
                MOVE.L          #$00, CH0BAR    * Zero unused register.
                MOVE.W          #712, CH0MTCR   * No. of BYTES transferred.
                BSR             _WAIT_3us       * Wait 3 microseconds.
                MOVE.B          #$E0, CH1DCR    * Allows status monitor.
                BSR             _WAIT_3us       * Wait 3 microseconds.
                MOVE.B          #$80, CH0CCR    * Gentlemen, Start your DMAC!
                MOVE.B          #$04, IMR1      * Set IMR1
                MOVE.B          #$20, IMR2      * Set IMR2
* That's it!  The DMA transfer should now be executing.  We next poll to
* determine when it finishes.
*
* !!!! ERROR DIAGNOSTIC!  The following code checks for a 1014 config error.
                BSR             _WAIT_1ms
                MOVE.B          CH0CSR, D4      * test for the error.
                BTST.L          #4, D4
                BEQ             END_GPIB        * if no error, leave.
                MOVEA.L #out_spectrum-186, A3   * Record the error in output
                MOVE.B          CH0CER, (A3)    * data frame.
```

```
* !!!! End of ERROR DIAGNOSTIC!
*
* First poll to detect a transition in the Ch1 CSR PCT:
END_GPIB        BSR             _WAIT_10us      * Time to finish.
                MOVE.B          CH1CSR, D0      * Check Ch. Stat. Reg., PCT
*                                                 bit, to verify end of DMA.
                BTST.L          #1, D0
                BEQ             END_GPIB
* Read ISR2 and ISR1.
                MOVE.B          ISR2, D2
                MOVE.B          ISR1, D1
* Clear IMR2 and IMR1.
                MOVE.B          #$00, IMR2
                MOVE.B          #$00, IMR1
* Reset CFG1 to unassert PCL of Channel 1.
                MOVE.B          #$1A, CFG1
* Test for GPIB error; if true, wait for completion of DMA transfer.
                MOVE.W  #712, out_spectrum-188 * Last GOOD BYTE count
                BTST.L          #2, D1          * ISR1 is in D1.
                BEQ             DMA_COMPLETE    * If no Error, Finish up.
                MOVE.B          CH0MTCR, D3     * If there is a GPIB error,
DMA_WAIT        BSR             _WAIT_10us      * make sure DMA done before
                MOVE.B          D3, D4          * aborting the transfer.
                MOVE.B          CH0MTCR, D3
                CMP.B           D3, D4
                BNE             DMA_WAIT
*
* !!!! ERROR DIAGNOSTIC!  If there has been a GPIB error, the last byte
*   count is written to the GPIB Last True Byte location in the output block.
                MOVEA.L #out_spectrum-188, A3
                MOVE.W          D3, (A3)
* !!!! End of ERROR DIAGNOSTIC!
*
DMA_COMPLETE    MOVE.B          #$10, CH0CCR    * Stop DMA by writing an
                BSR             _WAIT_3us
                MOVE.B          #$00, CH0CSR    * abort to Ch 0.  Also
                MOVE.B          #$00, CH1CSR    * clear Ch 1 CSR.
                MOVE.B          #$66, ADR       * Clear Talk/Listen Enable.
                MOVE.B          #$30, ADMR      * Clear Active Talk Status.
SKIP_OUT        MOVE.B          #AUX_RST, AUXMR * RESET the TLC!!
                BSR             _WAIT_500us
                MOVE.B          #AUX_PON, AUXMR * IMMEDIATE EXECUTE!!!
*
                MOVE.L          #std, cfg_reg   * Return 68K to std config.
*                                                 and turn off IND LED.
*
```

```
            MOVE.W      #$00C0, isr      * Returns ISR_CTRL6 to high state.
* End GPIB DMA transfer routine.
*
                MOVE.B          #$00, SPMR      * Clear Serial Poll Reg.
*
                JMP             MN_FETCH        * Complete Main Loop
*
******* END MAIN CODE MODULE *******
*
******* SUBROUTINE: WAIT 10us ********
* This routine just waits ten microseconds or so...
*  Set up decrement counter.
_WAIT_10us      MOVE.L          #14, D1
MARK_A10        MOVE.L          D0, D0          * Kills time.
                DBMI            D1, MARK_A10
                RTS
*******  END SUBROUTINE: WAIT 10us *******
*
******* SUBROUTINE: WAIT 3us ********
* This routine just waits three microseconds or so...
*  Set up decrement counter.
_WAIT_3us       MOVE.L          #0, D1
MARK_A3         MOVE.L          D0, D0          * Kills time.
                DBMI            D1, MARK_A3
                RTS
*******  END SUBROUTINE: WAIT 3us *******
*
******* SUBROUTINE: WAIT 100us ********
* This routine just waits 100 microseconds or so...
*  Set up decrement counter.
_WAIT_100us     MOVE.L          #155, D1
MARK_A100       MOVE.L          D0, D0          * Kills time.
                DBMI            D1, MARK_A100
                RTS
*******  END SUBROUTINE: WAIT 100us *******
*
******* SUBROUTINE: WAIT 500us ********
* This routine just waits 500 microseconds or so...
*  Set up decrement counter.
_WAIT_500us     MOVE.L          #779, D1
MARK_A500       MOVE.L          D0, D0          * Kills time.
                DBMI            D1, MARK_A500
                RTS
*******  END SUBROUTINE: WAIT 500us *******
*
******* SUBROUTINE: WAIT 1ms ********
```

```
* This routine just waits one millisecond or so...
*  Set up decrement counter.
_WAIT_1ms       MOVE.L          #1560, D1
MARK_A1         MOVE.L          D0, D0          * Kills time.
                DBMI            D1, MARK_A1
                RTS
*******  END SUBROUTINE: WAIT 1ms *******
*
******* SUBROUTINE: WAIT 5ms ********
* This routine just waits five millisecond or so...
*  Set up decrement counter.
_WAIT_5ms       MOVE.L          #7810, D1
MARK_A5         MOVE.L          D0, D0          * Kills time.
                DBMI            D1, MARK_A5
                RTS
*******  END SUBROUTINE: WAIT 5ms *******
**
*********  SUBROUTINE: TAKEVBUS  *******
*
* This routine requests control of the VIPER VSP bus, and waits until it is
* sure control has been granted before proceeding.
_TAKE_VBUS
                MOVE.W          #$0040, isr     * Request bus control.
WAITVBUS        MOVE.L          D0, D0          * Kill a little time.
                MOVE.W          isr, D0
                BTST.L          #5, D0          * Test for ack from VIPER -
                BEQ.S           WAITVBUS        * If no ack yet, wait more.
                RTS
*******  END SUBROUTINE: TAKEVBUS  *******
*
*
******* SUBROUTINE: FIND FRESH DATA  *******
*  This subroutine waits until the Analytek has provided new data, selects
*  peak values for AGC processing, and returns to the FFT phase 1.
****
*
_FRESH_DATA     BSR             _WAIT_3us       *Give Analytek Time....
                BSR             _TAKE_VBUS      *Take VSP Bus.
                CLR.L           D2
                MOVE.W  in_buf_base_1, D2       *Get first WORD of data
                MOVE.W          #$00C0, isr     *(Release VSP Bus.)
                CLR.L           D0
                MOVE.W          #$FFFF, D0
*
*  New Data will NOT be $FFFF.
                CMP.W           D2, D0          *Is the data new?
```

```
                BEQ             _FRESH_DATA     *If not, try again.
*
                BSR             _TAKE_VBUS      * Take VSP Bus.
* Toggle state of ISR_CTRL_6 while maintaining control of VSP Bus.
                MOVE.W          #$0000, isr
*
* Now enable and clear the Data Cache.
                MOVE.L #Dcach_mod, cfg_reg      * Allow Data Cacheing.
* NOTE: The above locks out VMEBus to all other users.  RESET when through!!
                MOVEC           CACR, D3        * Read current cache settings.
                BFCLR           D3{0:24}        * Clear current D-cache cmds.
                ORI.L  #dena+dclr+dbst, D3      * Preserve I_Cache settings.
                MOVEC           D3, CACR        * Enable, Clear, Burst-mode.
*
* Now move 64 equi-spaced points into the 68K work space.
                MOVE.L  #in_buf_base_1, A1
                MOVE.L  #in_68k_work, A2
                MOVE.L          #63, D1         * Set loop counter.
MAX_MOVE        MOVE.W          (A1), D2        * Input data word.
                ADDA.L          #16, A1         * Num Bytes to step address.
                MOVE.W          D2, (A2)+       * Transfer data to 68K space.
                DBMI            D1, MAX_MOVE    * Repeat until through.
* Now disable the Data Cache.
                MOVEC           CACR, D3        * Read current cache cmds.
                BFCLR           D3{0:24}        * Clear current D-cache cmds.
                ORI.L           #ddis, D3       * Preserve I_Cache settings.
                MOVEC           D3, CACR        * Disable the D_Cache.
                MOVE.L          #std, cfg_reg   * Unlock VMEBus.
*
** Deactivate this data set.
                MOVE.W  #$FFFF, in_buf_base_1   * It's dead, Jim!
*
                MOVE.W          #$00C0, isr     *(Release VSP Bus, and
*                                               restore ISR_CTRL_6.)
                RTS
*******  END SUBROUTINE: FIND FRESH DATA  *******
*
*
*******  SUBROUTINE: INITIALIZATION MODULE  *******
*
_INIT           BSR             _TAKE_VBUS              *Take ctrl of VSP Bus
*  Clear the Analytek Input Buffers.
                MOVEA.L         #in_buf_base_1, A0      *Put buffer adds in
                MOVEA.L         #in_buf_base_2, A1      *Address regs 0-3...
                MOVEA.L         #in_buf_base_3, A2
                MOVEA.L         #in_buf_base_4, A3
```

```
                MOVEA.L         A0, A6                  *Inp Buffer Pointer!
                MOVE.L       #1023, D1                  *Set up for 1024 words
* >>>> NOTE: A6 is a RESERVED register!
CLR_ANAL_LP
                CLR.W           (A0)+                   *Then clear values in
                CLR.W           (A1)+                   *those mem locations.
                CLR.W           (A2)+
                CLR.W           (A3)+
                DBMI            D1,CLR_ANAL_LP
* Initialize this input buffer by declaring it DEAD!
                MOVE.W  #$FFFF, in_buf_base_1     * Mark input location dead,
*                                                 until overwritten by Analytek
*
* Clear the Output Spectrum in both Viper and 68K RAM space.
                MOVEA.L         #v_spectrum, A0
                MOVEA.L         #out_spectrum-200, A1  *68K Addr. are BYTES
* Note: 100 words in advance of out spectrum are cleared for sync, time, INS,
*       housekeeping, etc.
                MOVE.L          #355, D1
CLR_OUT_LP      CLR.W           (A0)+           * This clears Viper space.
                CLR.W           (A1)+           * This clears 68K space.
                DBMI            D1, CLR_OUT_LP
* Place the "AAFE...#" sync word at the beginning of the output frame.
                MOVE.L          #AAFE, out_spectrum-200
                MOVE.L          #dots, out_spectrum-196
*  Clear the VSP Sum Vector Storage Locations.
                MOVEA.L         #sum_vec_base, A0
                MOVE.L          #511, D1
CLR_SUM_LP      CLR.W           (A0)+
                DBMI            D1, CLR_SUM_LP
*
* Move # Samples scale factor to VSP data space.
                MOVE.W          #sam_scl, vsp_num_sam
* Move the Analytek bias shift value to VSP data space (NOTE: depends on amt
*     of gain in the Fresh Data Routine (0, 3, 6, 9 dB).
                MOVE.W          #v_bias_shift, v_shift_addr
*
*  Set up data reg D7 as number of averaged samples. >>NOTE:
*  D7 is now a RESERVED register.
                MOVE.L        #no_avgd-1, D7
*  Release VSP Bus.
                MOVE.W          #$00C0, isr
*
* Place the number of samples (returns) averaged per frame in output block.
                MOVE.W          #no_avgd, out_spectrum-172
* Initialize the GPIB-1014 interface board.
```

```
*
IBONL
        MOVE.B      #CFG2_LMR+CFG2_SFL, CFG2    * Local master reset
        BSR         _WAIT_500us
*
        MOVE.B      #CFG2_SFL, CFG2         * clear LMR...
*                                            NOT SYSTEM CONTROLLER
*                                            SUP=0 SO SUPO AND USER ACCESS
        BSR         _WAIT_500us
*
        MOVE.B      #CFG1_BRG3+CFG1_DBM, CFG1 * use bus request/ grant line 3
*                                               disarm bus monitor
*                                               ROR=1 NO RELEASE ON REQUEST
        BSR         _WAIT_500us
        MOVE.B      CPTR, D1                * clear regs by reading
        MOVE.B      ISR1, D1
        MOVE.B      ISR2, D1
*
        MOVE.B      #0, IMR1                * disable all interrups
        MOVE.B      #0, IMR2                *   "
        MOVE.B      #0, SPMR                * CLEAR SERIAL POLL MODE REG
*
        MOVE.B      #GPIB_Add, ADR          * set GPIB address; 6
        BSR         _WAIT_500us
        MOVE.B      #$E0, ADR               * disable secondary addressing
        BSR         _WAIT_500us
        MOVE.B      #$31, ADMR              *  SET TRM1 AND TRM0
        BSR         _WAIT_500us
*                                             AND ADDRESS MODE 1
        MOVE.B      #0, EOSR                * CLEAR THE END OF STRING REG
        BSR         _WAIT_500us
*
        MOVE.B      #HR_ICR+15, AUXMR       * set internal counter reg to 8
*                                             book says to do this
        BSR         _WAIT_500us
*               WRITE TO HIDDEN REG PPR WITH U=UNCONFIGURE.
        MOVE.B      #HR_PPR+PPR_U, AUXMR      * parallel poll unconfigure
        BSR         _WAIT_500us
*               WRITE TO HIDDEN REG AUXRA
*               EOS AT 7 BITS
*               NO XMIT END WITH EOS
*               NO SET END ON RECEIPT OF EOS
*               NORMAL HANDSHAKE  ( NO RFD HOLDOFF ON END OR DATA)
        MOVE.B      #HR_AUXRA+0, AUXMR
        BSR         _WAIT_500us
*               ISS,INV,TRI,SPEOI, AND CPT ENABLE SET TO 0
```

```
        MOVE.B      #HR_AUXRB+0, AUXMR
        BSR         _WAIT_500us
*                   DHDC, DHDT SET TO 0 (HOLDOFF ON DCAS & DTAS)
        MOVE.B      #HR_AUXRE+0, AUXMR
        BSR         _WAIT_500us
*
        MOVE.B      #AUX_DSC, AUXMR          * TURN OFF SYSTEM CONTROL
        BSR         _WAIT_500us
*
        MOVE.B      #AUX_PON, AUXMR          * DO A PON
*
* The following lines of code test the integrity of the GPIB (i.e. is the
* cable connected, is the controller online, etc) by attempting to write two
* characters, "HI", onto the Bus.  We will sit here waiting to write until
* the GPIB controller addresses us successfully.  This is necessary
* to avoid potential Bus lockup problems during DMA transfers occurring
* later on in the program.
SYNC_LP BSR             _WAIT_5ms       * Wait for 1014 to stabilize.
        MOVE.B          ISR2, D2
        MOVE.B          ADSR, D3
        BTST.L          #1, D3
        BEQ             SYNC_LP         * If not active talker, try again.
*
DO_LP   BSR             _WAIT_5ms       * OK to send a character?
        MOVE.B          ISR1, D1
        BTST.L          #1, D1
        BEQ             DO_LP           * If not, wait until it is OK.
*
*  The HP computer works better if it always receives GPIB transfers of
*   the same block size.  Therefore, this code will send 356 words
*   (=2x356 bytes) of ASCII "HI".
*   7/17/91, awp
*
SEND_H  MOVE.B          #$48, CDOR      * Send an "H".
*
I_NEXT  BSR             _WAIT_10us
        MOVE.B          ISR1, D1        * Test to see if DOut is active.
        MOVE.B          ISR2, D2
        MOVE.B          ADSR, D3
        BTST.L          #1, D1
        BEQ             I_NEXT          * If not active, wait until it is.
        MOVE.B          #$49, CDOR      * Send an "I"
* Set up and loop for the next 354 words.
        MOVE.L          #353, D0        * # Words to send less one.
HI_LOOP BSR             _WAIT_10us
H_NEXT  MOVE.B          ISR1, D1        * Test to see if DOut is active.
```

```
        MOVE.B          ISR2, D2
        MOVE.B          ADSR, D3
        BTST.L          #1, D1
        BEQ             HI_LOOP         * If not active, wait until it is.
        MOVE.B          #$48, CDOR      * Send an "H"
*
I_NEXT2 BSR             _WAIT_10us
        MOVE.B          ISR1, D1        * Test to see if DOut is active.
        MOVE.B          ISR2, D2
        MOVE.B          ADSR, D3
        BTST.L          #1, D1
        BEQ             I_NEXT2         * If not active, wait until it is.
        MOVE.B          #$49, CDOR      * Send an "I"
        DBMI            D0, HI_LOOP     * Send the next word
* For the final "HI" send EOI on last character
H_NEXT2 BSR             _WAIT_10us
        MOVE.B          ISR1, D1        * Test to see if DOut is active.
        MOVE.B          ISR2, D2
        MOVE.B          ADSR, D3
        BTST.L          #1, D1
        BEQ             H_NEXT2         * If not active, wait until it is.
*
        MOVE.B          #$48, CDOR      * Send an "H"
I_NEXT3 BSR             _WAIT_10us
        MOVE.B          ISR1, D1        * Test to see if DOut is active.
        MOVE.B          ISR2, D2
        MOVE.B          ADSR, D3
        BTST.L          #1, D1
        BEQ             I_NEXT3         * If not active, wait until it is.
*
        MOVE.B          #$49, CDOR      * This one will work, so just
        BSR             _WAIT_5ms       * punch out when thru.
        MOVE.B          #$66, ADR       * Clear Talk/Listen Enable.
        MOVE.B          #$30, ADMR      * Clear active talk status.
        MOVE.B          #AUX_RST, AUXMR * RESET the TLC!!
        BSR             _WAIT_500us
        MOVE.B          #AUX_PON, AUXMR * IMMEDIATE EXECUTE!!!
*
* Wait 5msec.
        BSR         _WAIT_5ms
*
* We'll set the "Last True Byte Written" counter to 712.  This value
* can only be changed in the event of a GPIB error.
        MOVE.W          #712, out_spectrum-188
*
* An error on the 1014 card can also terminate transfer early.  The value of
```

```
* the GPIB error register is written into the output data block if this
* occurs.  The following line makes sure the default output value is zero.
        MOVE.W          #0, out_spectrum-186
*
*
* Initialize D6, the outout frame counter, to 1.
        MOVE.L          #$01, D6
*
* Clear all working data registers.
        CLR.L       D0
        CLR.L       D1
        CLR.L       D2
        CLR.L       D3
        CLR.L       D4
        CLR.L       D5
*
        RTS
*
*******  END SUBROUTINE: INITIALIZATION  *******
*
```

References

[1] Climate-Research-Board, . Carbon dioxide and climate: A scientific asessment. Technical report, National Academy of Sciences, Washington, DC, 1979.

[2] Denton, G.H. and Hughes, T.J. *The Last Great Ice Sheets.* John Wiley and Sons, Inc., 1981.

[3] Bindschadler, R. A., Zwally, H. J., Major, J. A., and Brenner, A. C. Surface topography of the greenland ice sheet from satellite radar altimetry. Technical Report SP-503, NASA, 1989.

[4] McGoogan, J. T., Miller, L. S., and Brown, G. S. The S-193 radar altimeter experiment. *Proceedings of the IEEE*, 62(6):793–803, 1974.

[5] Hughes-Aircraft, Inc. *Final Report of the Advanced Application Flight Experiment Breadboard Pulse Compression Radar Altimeter Program*, 1976. NASA Contractor Report CR-141411.

[6] Ferraro, E. J. *Analysis of Airborne Radar Altimetry Measurements of the Greenland Ice Sheet.* PhD thesis, University of Massachusetts, 1993.

[7] Hewlett-Packard, . *HP BASIC 6.2 Programming Guide.* Hewlett-Packard Company, Sunnyvale, CA, 1991.

[8] Impact, . *Viper 8704/30 User's Manual.* Impact Technologies, Inc., Santa Clara, CA, 1987.

[9] Impact, . *161ASM User's Manual.* Impact Technologies, Inc., Santa Clara, CA, 1988.

[10] Hewlett-Packard, . *HP BASIC 6.2 Interface Reference.* Hewlett-Packard Company, Sunnyvale, CA, 1991.

[11] Skolnik, Merrill I. *Introduction to Radar Systems.* McGraw Hill Inc., 1980.

[12] Klauder, J. R., Price, A. C., Darlington, S., and Albersheim, W. J. The theory and design of chirp radars. *The Bell System Technical Journal*, 39(4):745–807, 1960.

[13] Wingham, D. J., Rapley, C. G., and Griffiths, H. New techniques in satellite altimeter tracking systems. *Proceedings of IGARSS 1986 Symposium*, pages 1139–1344, 1986.

[14] Leick, Alfred. *GPS Satellite Surveying.* John Wiley & Sons, 1990.

[15] Krabill, W. B. and Martin, C. F. Aircraft positioning using global positioning system carrier phase data. *Journal of Navigation*, 3(1), 1987.

[16] Boyette, Ken. *VMEbus Prototype Circuit Card User's Manual.* Logical Design Group, Raleigh, NC, 1984.