The Design and Development of Signal-Processing Algorithms for an Airborne X-band Doppler Weather Radar

Shaun R. Nicholson

Radar Systems and Remote Sensing Laboratory
Department of Electrical Engineering and Computer Science, University of Kansas
2291 Irving Hill Road, Lawrence, Kansas 66045-2969
TEL: 913/864-4835 * FAX: 913/864-7789 * OMNET: KANSAS.U.RSL

RSL Technical Report 10540-1

September 1994

Supported by:

Global Change Fellowship Program
National Aeronautics and Space Administration
Washington DC 20546

Grant No. NGT-30202

Abstract

Improved measurements of precipitation will aid our understanding of the role of latent heating on global circulations. Spaceborne meteorological sensors such as the planned precipitation radar and microwave radiometers on the Tropical Rainfall Measurement Mission (TRMM) provide for the first time a comprehensive means of making these global measurements. Pre-TRMM activities include development of precipitation algorithms using existing satellite data, computer simulations, and measurements from limited aircraft campaigns. Since the TRMM radar will be the fist spaceborne precipitation radar, there is limited experience with such measurements, and only recently have airborne radars become available that can attempt to address the issue of the limitations of a spaceborne radar. There are many questions regarding how much attenuation occurs in various cloud types and the effect of cloud vertical motions on the estimation of precipitation rates. The EDOP program being developed by NASA GSFC will provide data useful for testing both rain-retrieval algorithms and the importance of vertical motions on the rain measurements. The purpose of this report is to describe the design and development of real-time embedded parallel algorithms used by EDOP to extract reflectivity and Doppler products (velocity, spectrum width, and signal-to-noise ratio) as the first step in the aforementioned goals.

Table of Contents

List of Figures

Chapter 1 - Introduction

1.0 Introduction

Global climate models depend upon knowledge of the atmospheric variables on a worldwide scale. Though satellites have greatly improved the inputs to these models, much remains to be done before inputs are adequate for long-range forecasts and study of major global changes. In particular, we can only crudely model the major heat exchange between atmosphere and surface that takes place in tropical rainfall (Simpson, 1988). Current models for global circulation depend largely on sparsely-spaced upper-air soundings and surface measurements at infrequent intervals in time and space. Moreover, lack of most meteorological parameters on model grid scales (200 km) at the surface and aloft over the oceans represents a major gap in knowledge. Clearly this information is a requirement to improve global circulation models.

Tropical rainfall is critical to the distribution of life-giving water throughout our Earth system. Over two-thirds of the worldwide precipitation falls in the Tropics, releasing latent heat and thereby powering the global atmospheric circulation that shapes our world climate. English scientist G. Hadley (18th century) was the first to understand the central role played by tropical rainfall in shaping Earth's climate. In brief, cool low-level moist air flowing toward the equator replaces heated equatorial air as it ascends and moves toward the poles at high altitudes. Stored solar energy is released as latent heat when the water vapor condenses into cloud clusters or Hadley cells reaching hundreds of kilometers in horizontal extent. Of these, perhaps one in ten deepens into a tropical cyclone. The solar energy released as latent heat by raining tropical clouds is the main source of energy for global atmospheric motions. In fact, low-level Hadley circulation carries evaporated water from over one-half the earth's surface into the Inter Tropical Convergence Zone (ITCZ) where it is precipitated by raining tropical systems.

Rainfall in the Tropics additionally plays a fundamental role in the El Niño climate anomalies whose disruptions to worldwide precipitation patterns trigger floods and drought around the globe. It's four- to five-year-average cycle is associated with anomalous warming of equatorial surface waters extending from the Eastern Pacific to the coast of South America. This warming is nearly always accompanied by large-scale shifts in atmospheric pressure that go from the East to the West Pacific and back in an oscillatory manner often referred to as the Southern Oscillation. The El Niño and Southern Oscillation are merely two manifestations of the same climatic disturbance—the ENSO event.

The importance of determining, at least climatologically, the rainfall in the Tropics is the subject of many papers and reports leading to the Tropical Rainfall Measurement Mission (TRMM). Lau et al. state that latent heating by precipitation causes about 40% of the energy flux to and from the atmosphere. This takes place largely in poorly instrumented tropical regions: oceans with little ship traffic and less-developed land areas with few weather stations. The information on this rainfall is so sparse that we do not even have adequate climatological data, much less synoptic data. The purpose of TRMM is to provide a first look at this climatology globally below 35° latitude.

Whereas TRMM will be able to measure rainfall rates on a climatological basis, it cannot provide wind velocities over the same tropical environment. Observations of horizontal winds and convection on a global scale are necessary for advancing our knowledge of large-scale atmospheric circulation and climate dynamics, thereby improving our understanding of the global biogeochemical and hydrologic cycles. Wind measurements are particularly important in the Tropics where there are few observing stations—indeed, it has been suggested (NASA, 1987) that wind profiles are the single most important new

data source because of their potential for dramatically advancing our skill at Numerical Weather Prediction (NWP).

1.1 Earth System Interactions

The atmosphere, oceans, and land biomass systems are closely coupled through the unifying role of global hydrology. Because of this, studies of rainfall are essential to gain deeper understanding of our Earth system.

General Circulation Models (GCM) simulate general atmospheric features to investigate the climate variations that take place on a variety of temporal scales. Existing GCM predictions are extremely sensitive to the assumed vertical distribution of latent heating by equatorial cloud systems; however, the vertical distribution of cloud layers is poorly observed. This casts doubts on our understanding of the role played by clouds in the climate system and their effects on the surface-radiation balance and profile of heat absorption and re-radiation throughout the atmosphere.

The response of cloud systems to environment is an important link in a chain of processes responsible for monsoons, ENSO events, and other climate variations. Numerical models of precipitation systems provide essential insights to the interaction of clouds with each other, their surroundings, and land/ocean surfaces. They are needed to convert rain-rates derived utilizing radiometric temperatures from spaceborne microwave and radiometers into latent-heating profiles. Such models further aid computations of the total energy budget, which consists of radiative and latent heating effects. Though poorly observed, cloud profiles strongly impact the hydrologic and carbon cycle. Clearly, improved observations of clouds are critical for updating global circulation models used in climate prediction.

Ocean-atmosphere interactions likewise render a key role in the Earth system. For example, as horizontal winds stress the sea surface, oceanic circulations develop influencing the transport of heat across the ocean-atmosphere boundary. Fresh water from precipitation further alters sea-surface temperatures as do clouds that shield the ocean surface from solar radiation. The result is a complex web of interaction between the wind and water that acts to regulate our climate.

In equatorial forest regions, tropical rainfall directly influences the terrestrial water and energy balance. Widespread cloudiness shields tropical forests from direct sunlight encouraging high growth rates and a large standing biomass. Most of the water received as rainfall is returned to the atmosphere in ten-day cycles through evaporation and transpiration, leading us to view the tropical forest and atmosphere as a single water and energy regulation system. Forests and soils are also a major source of many atmospheric trace constituents (e.g., carbon monoxide, methane, and the hydroxyl radical) that are taken into the lower atmosphere during convective rainstorms. This exchange provides a direct link between rainfall and the global biogeochemical cycles that regulate life on Earth.

1.2 What now?

Existing spaceborne atmospheric sensors are passive in the visible, infrared (IR), and microwave parts of the spectrum, and the measured radiometric temperatures are vertically integrated quantities. The visible and IR instruments sense upwelling radiation near cloud tops. Passive-microwave instruments provide integrated liquid water and water vapor estimates. Thus, interpretations of these temperatures to retrieve physical quantities are often difficult and frequently necessitate the use of complementary instruments to provide vertical structure because of the inherent lack of passive, vertical-profiling capability. Now we can look forward to development and deployment of

spaceborne radars to provide vertical precipitation structure for meteorology. The first of these will be on the TRMM, scheduled for launch in 1997, and will provide vertical precipitation profiles using the radar reflectivity. Such measurements can then be combined with cloud models to estimate profiles of latent heating.

The basic problem with existing studies is that they have been tested with limited aircraft data. Intensive aircraft campaigns are necessary to validate cloud models and algorithms as well as provide case studies and reference measurements for their spaceborne counterparts. To test them, both radars and radiometers must fly above thunderstorms and precipitation systems in regions where surface measurements of precipitation are available. Whereas surface measurements provide rain rates and total rain amounts directly, aircraft or satellite measurements provide only indirect indications of the rain rate from the backscattered power (or radar reflectivity). These estimates are based on empirical relations and can be ambiguous. For example, they assume that vertical air motions in clouds are negligible, when in fact they can be substantial. Existing rain-retrieval algorithms fail to utilize knowledge of vertical wind motions because they are unavailable. While it has been argued that these errors are minimized in the monthly-averaged data such as will be provided by TRMM, it is desirable to more closely examine the effect of vertical motions on rain-rate estimates using aircraft instrumented with radar and radiometers.

An X-band Doppler radar, the EDOP developed by the National Aeronautics and Space Administration's Goddard Space Flight Center (NASA GSFC), has now begun flying at high altitude on the ER-2 aircraft (Heymsfield et al., 1989; Heymsfield et al., 1993). This system will make possible the testing of algorithms for rain retrieval, the concepts for radar wind measurement, and the validation of existing cloud models.

1.3 Purpose of this paper

The purpose of this Masters project is to document the design and development of real-time embedded parallel-processing algorithms executed by the EDOP Data System. These algorithms reduce raw radar measurements to reflectivity and Doppler products (velocity, spectrum width, and signal-to-noise ratio) resulting an effective compression of the raw data exceeding 400:1.

To provide the reader with necessary background material, Chapter Two introduces EDOP's scientific objectives and compares the role of EDOP to existing airborne radars in meeting these needs.

EDOP is understandably a complicated system and discussion of its signal processing algorithms is incomplete without an overview of the system design. Chapter Three examines the RF and digital hardware in limited detail to give the reader an appreciation of the interaction between various system components during operation and the manner in which they constrain algorithm development.

The design and development of the reflectivity and Doppler signal-processing algorithms are investigated in detail in Chapter Four. This chapter forms the basis for this report and additionally derives estimates of the measurement accuracies.

Chapter Five concludes this report by exploring the remaining tasks needing completion before EDOP fully meets its specified operational state. Two remaining tasks of particular importance include nadir antenna stabilization and development of an automatic gain control to prevent receiver saturation due to the large dynamic range of received weather echoes.

Chapter Two - Background

2.0 Historical Digression

The earliest known treatises on meteorology date back to Aristotle's (384-322 B.C.) publication of *Meteorologica*. Though his pupil, Theophrastus, continued the work of writing about winds and weather signs, progress was excruciatingly slow for the next 2000 years—largely due to a lack of instruments for observing the primary measurables. The early seventeenth century heralded rapid growth in meteorological research due to the invention of the thermometer (Galileo, 1607) and barometer (Evangelista Torricelli, 1643) as well as the discovery of Boyle's Law (1659). Less than forty years later, Edmund Halley (1696) ushered the era of Earth System Science through his attempt to explain general circulation by variable surface heating. Further attempts were made to explain atmospheric motions during the 1700's (Hadley, 1735; Lavoisier, 1783), but it wasn't until John Dalton realized in 1800 the relation between expanding air and atmospheric condensation that the physical basis of modern meteorology was established. Between 1800 and 1815, J. B. Lamarck, with the help of Lavoisier, Laplace, and others, created the first international compilation of weather observations. These synoptic observations affirmed the existence of characteristic patterns of pressure, temperature, winds, and, moreover, empirical rules for their development, movement, and the consequent weather changes. Such knowledge was of immediate benefit to the large number of sailing ship captains who invested in M. F. Maury's 1848 publication of global wind-field maps. The development of rubber balloons (1890's) led to the frequent sampling of the upper atmosphere by France, Germany, and England. Observations of winds aloft could then be made by watching the motion of the balloons, provided they didn't enter the clouds!

The first verified report of a precipitation-radar echo was by a 10 cm system that tracked a shower to a distance of seven miles off the English Coast on 20 February, 1941. Some

very weak echoes were thought to be attributed to storms as early as 1938; however, the detection of hydrometeors was severely limited by the fact that available methods of RF power generation resulted maximum frequencies between 200 and 400 MHz. The subsequent development of the highly secret British magnetron expanded the boundary to 3000 MHz (S-band) and later 10,000 MHz (X-band) inspiring the genesis of radar meteorology.

2.1 Scientific Objectives

The EDOP radar's major justification is to provide a means to study the dynamic and hydrometeor structure of thunderstorms and mesoscale convective systems using a comprehensive instrument package on the ER-2. The aim is to improve the understanding of thunderstorms and larger scale or mesoscale convective systems using present and future satellites (e.g. GOES, TRMM, EOS-MIMR). Specifically, EDOP's scientific objectives are to provide radar data in conjunction with the multi-frequency measurements from other ER-2 complementary instruments to:

- Better understand the structure of deep isolated thunderstorms,
- Determine the structure and dynamics of both the overshooting and anvil portions of individual thunderstorms,
- Define the mesoscale structure and dynamics of mesoscale convective systems (MCS) including squall lines and mesoscale convective complexes,
- Better understand the cloud microphysics in isolated thunderstorms and MCS's,
- Test proposed satellite passive- and active-microwave precipitation-estimation algorithms.

2.1.1 Structure of deep isolated thunderstorms

It is of interest to utilize satellite IR imagery for nowcasting of thunderstorms, precipitation estimation, and studies of thunderstorm anvil structure. Intense midwest thunderstorms frequently exhibit peak vertical-wind motions of 40-50 m/s and reflectivities exceeding 60 dBZ with large hail in the vicinity of the updraft core. Numerous studies have probed the updraft structure utilizing multiple ground-based Doppler radars by integration of the mass-continuity equation which uses horizontal wind measurements to deduce vertical wind speeds. Yet these vertical-wind speeds estimates often have accuracies only to within a few meters per second. Contamination of radial velocities by ground clutter and weak signal-to-noise ratios (SNR) result in inadequate estimates of mass divergence in the boundary layer leading to poor estimates of vertical velocities (**w**). Interpolation of radial velocities from different radars to a common 3-dimensional grid, grid filtering, and estimation of top and bottom boundary conditions required for **w** calculations further degrade vertical velocity accuracy. Moreover, typical radar sampling resolutions limit resolvable scales of motion to greater than 5 km (Carbone, 1985). *A vertically pointing airborne radar can mitigate these problems.*

2.1.2 Anvil structure and dynamics

Satellite infrared observations of severe thunderstorms frequently show a thermal couplet near cloud tops and a "V" shaped region of cold temperatures on the anvil scale (Heymsfield et al, 1983). Explanations of these phenomena are controversial; however, it is generally accepted that cloud air is warmed as overshooting and negatively buoyant updraft air subsides downwind of the cloud top (Heymsfield, 1983; Schlesinger, 1984; Adler and Mack, 1986). An alternative explanation (Heymsfield and Blackmer, 1988), schematically illustrated in Figure 2-1, is based on the generation of lee waves by stratospheric flow over the penetrating cloud tops; however, *no insitu data exist to confirm the various hypotheses.* This inadequate knowledge of vertical-motion structure

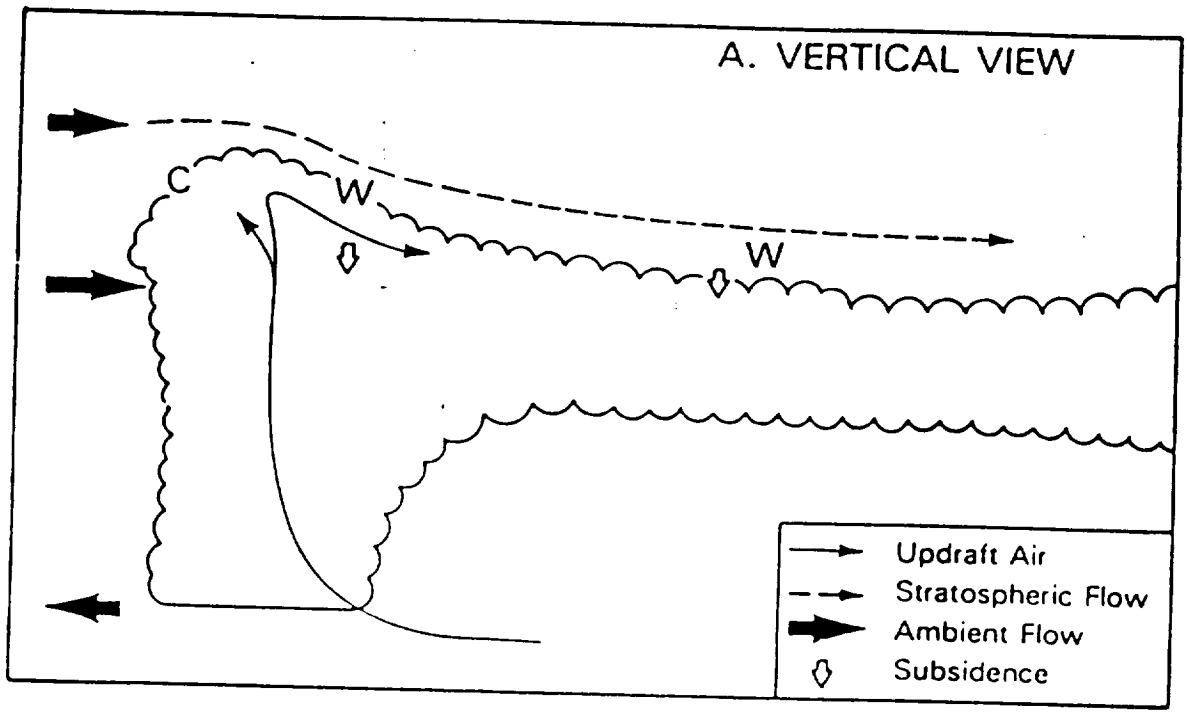


Figure 2-1: Schematic of cloud-top structure for a Midwest severe thunderstorm (Heymsfield, 1979)

and precipitation profiles hampers effective utilization of GOES imagery to deduce the mixing of penetrating updraft air with stratospheric air.

Both airborne and ground-based radars typically determine vertical velocity by integrating horizontal winds using the mass-continuity equation. The limits of integration require knowledge of the top and bottom boundary conditions; however, existing radars regularly fail to see the small ice crystals in the tops of storms, and the resulting ambiguous estimates of the top boundary introduce error into the vertical-velocity estimates. *An ER-2 radar can measure the weak cloud top echoes with greater accuracy because of its close proximity— often the cloud tops are only 3-5 km below the aircraft.*

In addition to the overshooting portion of the storm, the surrounding anvil is of great interest. Anvils produced by deep isolated thunderstorms are frequently extensive—much larger than the core region itself. The horizontal extent, depth, and hydrology of the anvil are related to storm factors such as updraft intensity, longevity, precipitation efficiency, and vertical shear; yet, convection can perturb the mesoscale environment in such a way as to feed on the convection itself. To better understand the complete dynamics of deep convection, it is necessary to obtain improved observations of air motions within the anvil region. Ground-based radars inherently obtain poor samples of the anvil region because of its large horizontal extent (hundreds of kilometers). Furthermore, such radars are rarely situated directly under the thunderstorm for ideal viewing; instead, they must observe distant events. *The mobility of an ER-2 radar makes it ideally suited for anvil-region studies.*

2.1.3 Structure of mesoscale convective systems

Within the past decade, considerable attention has been devoted to probing the two- and three-dimensional structure of mesoscale convective systems (Figure 2-2). These systems, extending a few hundred kilometers, are often composed of both convective and stratiform precipitating regions. Updrafts of 20 m/s or more are common in the convective region whereas the weaker stratiform circulations are driven largely by diabatic heating and cooling processes such as melting of ice or evaporation of rain. While multiple Doppler networks of ground based radars can be used to observe these mesoscale systems, airborne platforms are better suited due to their greater mobility. Conventional techniques require integration of the mass-continuity equation thereby resulting estimates of vertical motion with sometimes larger than desirable uncertainties. Recently, the scanning pattern of the NOAA P-3 radar has been changed to utilize an approach similar to that of ELDORA. This has provided effective vertical incidence data in limited regions flyable by the P-3 (Marks and Houze, 1987). *An ER-2 based radar would have the mobility to provide similar measurements from directly above convective storms.*

2.1.4 Cloud microphysics

The EDOP radar should additionally have the capability to deduce the precipitation phase of hydrometeors by making polarimetric measurements. The linear depolarization ratio (LDR) is conventionally defined as the ratio of received power at two orthogonal polarizations from a linearly-polarized transmitted pulse:

$$\mathrm{LDR} = 10 \log \frac{Z_{HV}}{Z_{HH}}$$

Hailstones tend to show no preference to polarization because they tumble during descent due to their large size and typically exhibit LDR ratios of -10 dB. Raindrops, however,

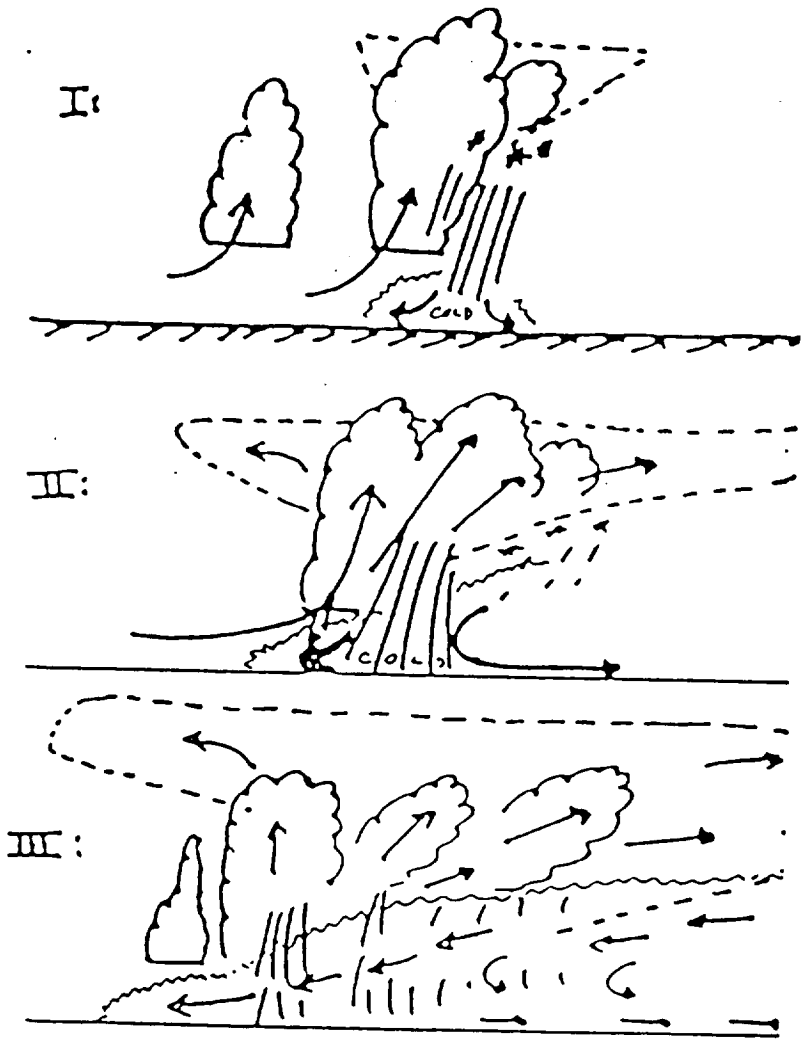


Figure 2-2: Evolving structure of a mesoscale convective system during initiation (I), organization (II), and mature stages (III). From Zipser (1987).

tend to flatten while falling resulting in ratios less than -30 dB. Snow takes on intermediate values. The design of EDOP's antennas and radome provide moderate isolation (~28 dB) between the polarizations; nevertheless, *this should be sufficient to discriminate hail from other hydrometeor types thereby providing additional justification for assuming a particular hydrometeor fallspeed.*

2.1.5 Testing of satellite precipitation algorithms

Ground-based radars offer excellent spatial and temporal coverage over limited regions, but often weather systems are beyond range of these radars (e.g. no data exist over the oceans). These ground-based measurements consequently provide insufficient precipitation estimates for use in global-scale weather prediction, climate, and radiative-transfer models. Spaceborne meteorological sensors such as the planned precipitation radar and microwave radiometers on the Tropical Rainfall Measurement Mission (TRMM) provide for the first time a comprehensive means of making these global measurements, but much needs to be learned about algorithms combining passive-microwave and radar measurements. Necessary pre-TRMM activities include development of precipitation algorithms using existing satellite data, computer simulations, and measurements from limited aircraft campaigns. Since the TRMM radar will be the first spaceborne precipitation radar, there is limited experience with such measurements, and only recently have airborne radars become available that can attempt to address the issue of the limitations of a spaceborne radar. Although the operating frequency of EDOP is different from that of TRMM, the less-attenuated wavelength should be valuable in this "combined" algorithm development and provide necessary ground truth for calibrating spaceborne radars. T*he dual-beam approach used by EDOP will additionally provide a unique opportunity to test the dual-beam rainfall-estimation algorithm proposed by Testud (1987) for use with airborne and spaceborne systems.* This method uses a variational approach to obtain path attenuation and relates it to the rain rate using a power law.

2.2 Existing and proposed airborne radars

There presently exists an abundance of airborne radars which can be grouped into two general categories: precipitation and cloud radars. Precipitation radars are used to measure the structure and evolution of actively precipitating and possibly convective systems whereas cloud radars penetrate and observe the vertical distribution of cirrus, stratocumulus, shallow cumulus, and other radiatively important cloud types. Often these systems are designed for specific applications so it is important to confer the uniqueness of EDOP in relation to existing and proposed systems (Table 2-1) and show its role in advancing our understanding of meteorology.

The NOAA P-3 radar is mounted on the tail of the P-3 aircraft and scans in a helical fashion to efficiently radiate the space around the aircraft. This radar, in operation since 1983, can sample reflectivity fields and provide both single- and dual-Doppler observations from a maximum altitude of about 8 km. In the dual-Doppler mode, the aircraft must fly a checkerboard flight track such that alternating flight tracks are normal to each other. If the observed storms undergo significant changes during the data collection period, the time difference between orthogonal tracks can result in large errors in the calculated wind fields (Hildebrand and Moore, 1992).

The ELectra DOppler RAdar (ELDORA) is an X-band, scanning, dual-beam, stepped-frequency system developed jointly by the National Center for Atmospheric Research (NCAR) and the French Centre de Recherché en Physique de l'Environnement (CRPE). Under an agreement signed in 1990, NCAR supplied the transmitter, receiver, signal processor, data system, and aircraft whereas CRPE provided the antennas (Hildebrand, Testud, 1992)

ELDORA/ASTRAEA flies between 8 and 10 km on the NCAR Electra making radial dual-Doppler velocity and reflectivity measurements of storm structure and kinematics with sufficient accuracy to derive thermodynamic properties of the storm such as buoyancy, heat and water transport.

NASA's Jet Propulsion Laboratory began development of the Airborne Rain Mapping Radar (ARMAR) in 1986. This DC-8 based, dual-wavelength, polarization diverse, pulse-compression system will largely be used to validate TRMM (Im et al., 1992). It's short term goals are to verify the TRMM radar system design, allow development and testing of spaceborne rain-retrieval algorithms, and serve as a reference for the validation of TRMM data after launch in 1997. This dedicated research tool will further serve as a testbed for developing future spaceborne techniques and technologies for spaceborne rain radars.

An airborne rain-scatterometer/radiometer has been developed by the Radio Research Laboratories (CRL) in Japan (Okamoto et al., 1982). This dual-wavelength, X- and K_a-band system has been operated on a variety of aircraft (e.g. Cessna 404, NASA T-39, NASA DC-8, NASA P-3) at altitudes up to 12 km. This system was originally designed to study the bright-band structure of rain but has since been used to examine hydrometeor growth processes. It should be noted that this is a non-Doppler system.

The NASA Goddard Space Flight Center is presently developing a W-band (94-GHz), polarization-diverse, scanning, Doppler radar to be flown with a host of ER-2 based instruments including EDOP, CLS, AMPR, and MIR (Racette, Heymsfield, 1993). The Doppler observations will provide insight into the dynamics of clouds. It will further serve as a useful test-bed to determine the utility of a future spaceborne cloud radar.

The Universities of Massachusetts and Wyoming have collaborated to produce a fully polarimetric, W-band, King-Air based cloud radar (Vali, 1993). This system can be pointed either horizontally or vertically perpendicular to the flight path to determine co- and cross-polarization power and phase and Doppler velocity. Preliminary test flights revealed an unexpected sharp peak in LDR at the melting-layer boundary. It is believed that this is the result of an asymmetry in the melt-water distributions on the ice crystals.

Clearly no one instrument can do it all. *A comprehensive package of passive- and active-microwave sensors such as those on the ER-2 in conjunction with EDOP is ideal for study of these complicated phenomena.*

Table 2-1: Existing and Planned Airborne Meteorological Radars

Instrument	Frequency Band	Operator	Comments
NOAA-P3	X	NOAA	scanning, dual-Doppler
ELDORA/ ASTRAEA	X	NCAR/CRPE (France)	scanning, dual-beam, stepped frequency
ARMAR	K, K_u	JPL	polarization-diverse, pulse-compression, TRMM simulator
CRL	X, K_a	CRL (Japan)	non-Doppler
CRS	W	NASA/GSFC	scanning, polarization-diverse, cloud radar, still in development phase
KING-AIR	W	Universities of Massachusetts/ Wyoming	fully polarimetric

Chapter 3 - System Design

3.0 System Overview

EDOP is an X-band, dual-beam, polarization-diverse weather radar developed for the NASA ER-2 aircraft (Heymsfield et. al., 1989; Heymsfield et. al., 1991). Figure 3-1 illustrates a typical configuration of EDOP and complementary instruments in the ER-2 payload bays. The system flies at a nominal altitude of 20 km (70,000 ft.) to map out time-height sections of hydrometeor reflectivity as well as vertical and along-track wind velocities (Figure 3-2). The nadir antenna measures co-polarization reflectivity, Doppler velocity, and spectrum width. Its velocity measurements are used to provide a direct indication of vertical air motion once the hydrometeor fall speed is removed. A second forward-pointing antenna (30° from nadir) measures cross-polarization reflectivity in addition to the previous parameters and can be used to determine the linear depolarization ratio (LDR) useful for assessing hydrometeor type and providing additional justification for the assumed fall speed.

Real-time reflectivity and Doppler processing of seven (four linear and three logarithmic) channels of data are accomplished by a data system built in-house at Goddard. Two four-channel A/D VME 9U cards sample the analog signals at 4 MHz to provide 37.5 m resolution in range. Three Martin Marietta LUA-200 linear-mapped array signal-processing boards each containing eight AT&T WEDSP32C digital-signal-processors provide peak computational speeds of 600 million floating-point operations per second (MFLOPs) when clocked at 50 MHz. A Radar Interface Card (RIB) establishes direct control of transmitter and receiver functions. Finally, a 68030 VME-based Eltec E-6 Single Board Computer is used as a host.

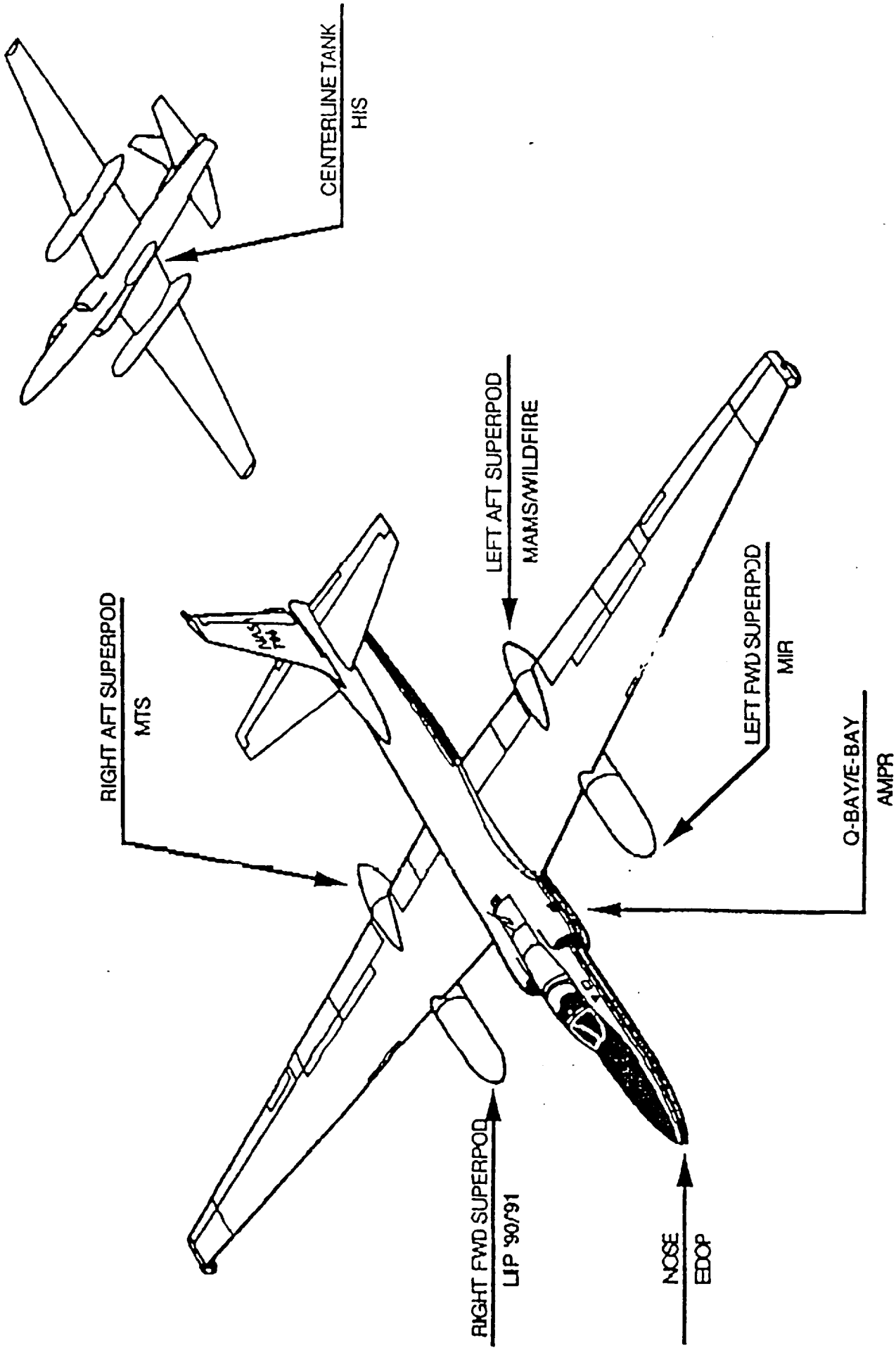


Figure 3-1: Typical mulit-instrument configuration of ER-2.

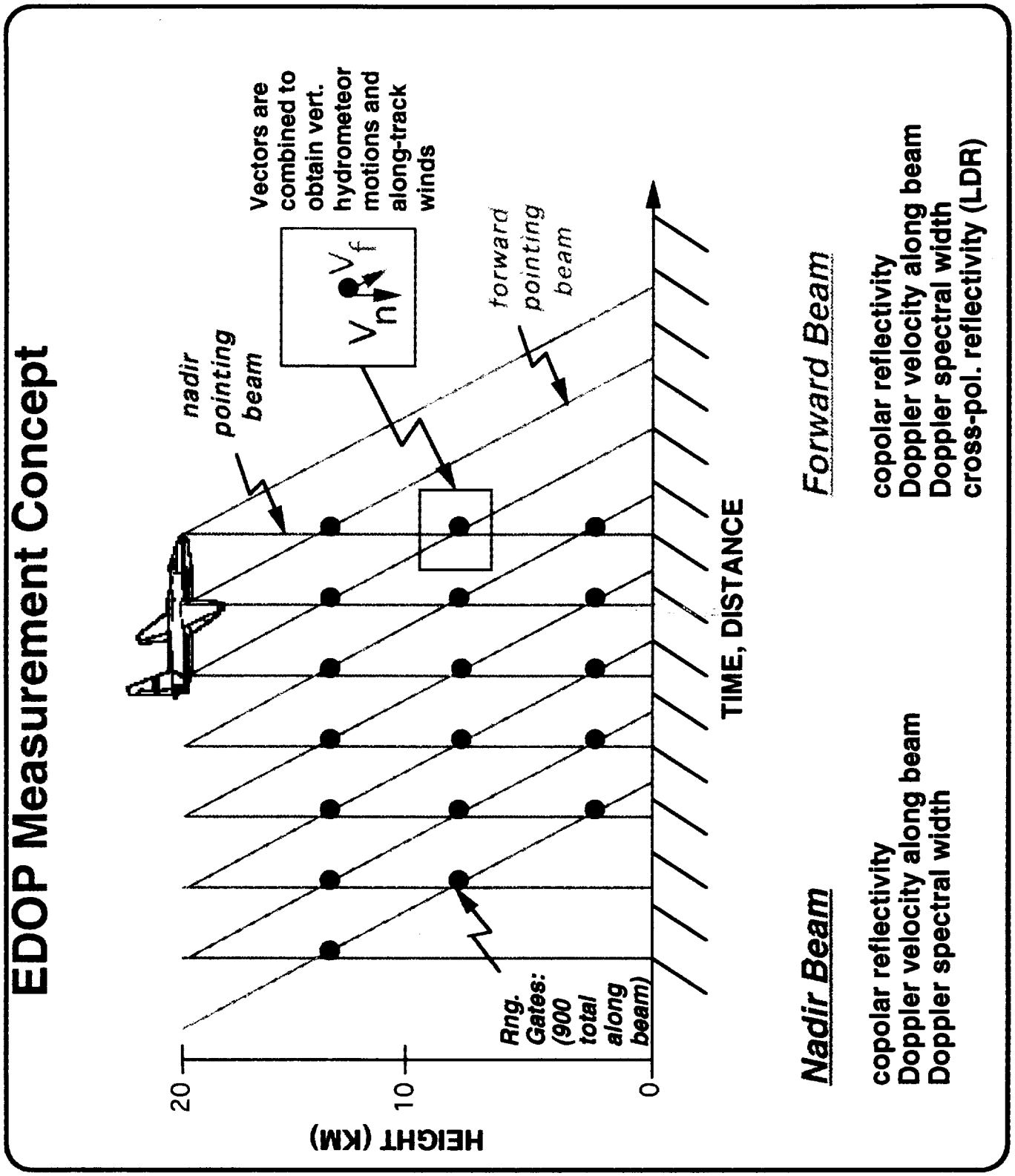


Figure 3-2: EDOP measurement concept (Heymsfield, 1993)

3.1 RF Section

Performance specifications for EDOP and a system block diagram are provided in Table 3-1 and Figure 3-3, respectively. In addition, Figure 3-4 illustrates the layout of system components in the ER-2 nose cone.

Table 3-1: EDOP Specifications

Frequency	9.72 GHz
Peak Power	20 kW
Duty Cycle	1% max
Pulse Length	0.25, 1.00 μs
PRF	2200, 4400 Hz
Antenna Diameter	0.76 m
Antenna Beamwidth	2.9°
First Sidelobe Level	-30 dB
Cross-polarization Level	-38 dB
Receiver Dynamic Range	110 dB
Number of Doppler Channels	2
Number of Log Reflectivity Channels	3
Nadir-Beam: Transmit Polarization Received Polarization	 Horizontal Co-polarized
Forward-Beam: Transmit Polarization Received Polarization	 Vertical Co-pol. and Cross-pol.

A 9.66 GHz coherent phase-locked oscillator is used to generate the transmit carrier frequency of the system. This RF energy is mixed with the 60 MHz local-oscillator and gated by a set of switched PIN diodes before being amplified by a high-gain 20 kW Litton air-cooled Traveling Wave Tube (TWT) amplifier and routed through circulators to dual high-gain (36 dBi) offset-fed parabolic antennas. The existing mode of operation allows power to be radiated simultaneously through both antennas; however, some consideration has been given to a future second-mode where power is radiated in alternate pulses between the two antennas.

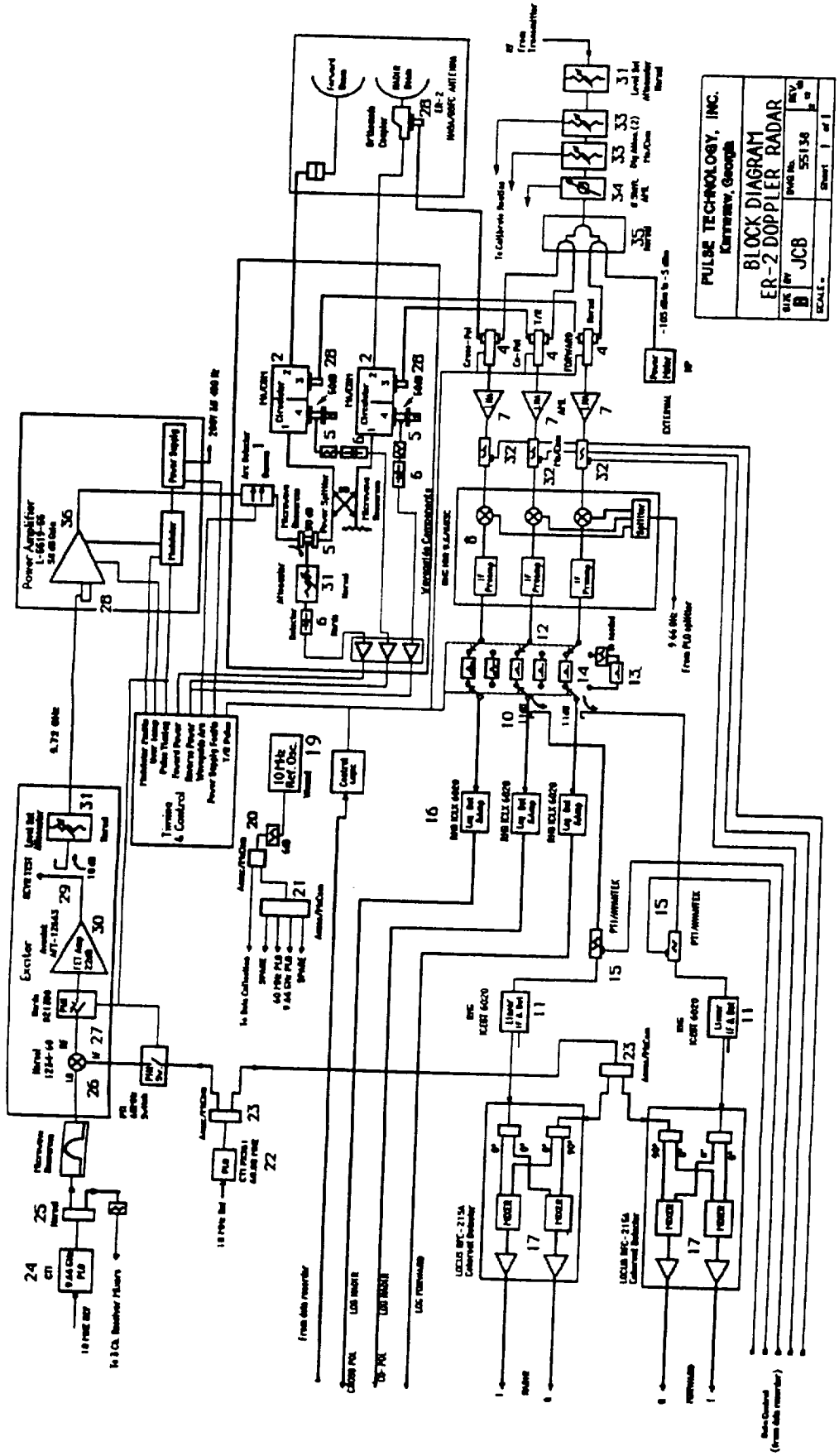


Figure 3-3: Block diagram of RF subsystem

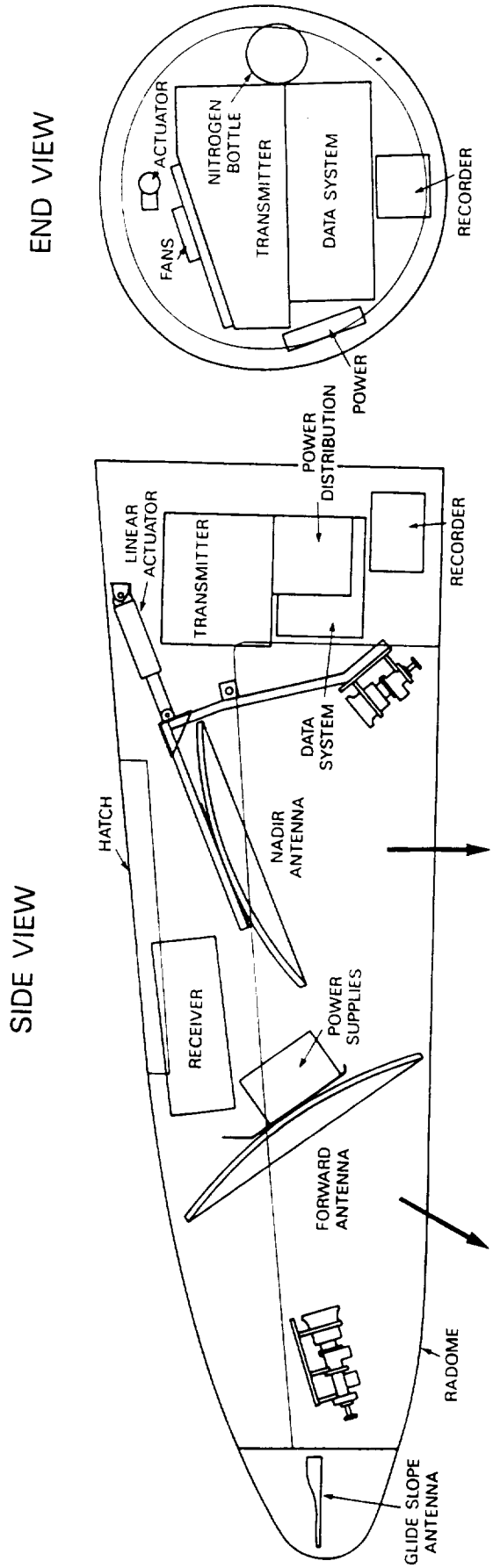


Figure 3-4: Layout of EDOP system in ER-2 nose (Heymsfield, 1993)

A low-noise RF gallium-arsenide pre-amplifier positioned near the antennas limits the receiver noise figure to approximately 1.8 dB. Nadir co-polar and forward co- and cross-polarized reflectivity channels pass through logarithmic detectors at the 60 MHz IF and then to the data system for real-time processing. Meanwhile, linear synchronous detectors beat dual-Doppler RF channels to baseband providing the data system with four (4) I and Q channels for Doppler-velocity extraction. Digitally controlled RF and IF attenuators are utilized to manipulate receiver gain, but an automatic gain-control (AGC) algorithm has not yet been implemented.

3.2 Data System

The major objective of real-time processing is the reduction of the raw data to levels that can be stored using cost-effective technology. It would be preferable to store raw data and perform off-line processing with a variety of algorithms; however, a simple calculation illustrates that this goal is not currently practical: in the highest resolution mode (37.5 m range gate spacing), the A/D converters on each ACQ board acquire 12-bit samples sign-extended to 16 bits at a rate of 4 MHz. The conversion of three logarithmic and four linear channels (seven total) results a raw-data bandwidth of 56 Mbytes/second. At this rate, an eight-hour flight will necessitate storage of 1.6 Terabytes of information! By performing real-time reflectivity and Doppler processing and storing only the products, the data rate can be reduced by a factor of 400 or more (depending on the chosen integration cycles) allowing for convenient data archival on a 5 Gb Exabyte tape. A block diagram is provided in Figure 3-5 to illustrate the flow of information between the data system and other system components. The individual circuit boards that comprise the data system are described in limited detail in the remaining paragraphs of this section.

ER-2 DOPPLER RADAR DATA SYSTEM

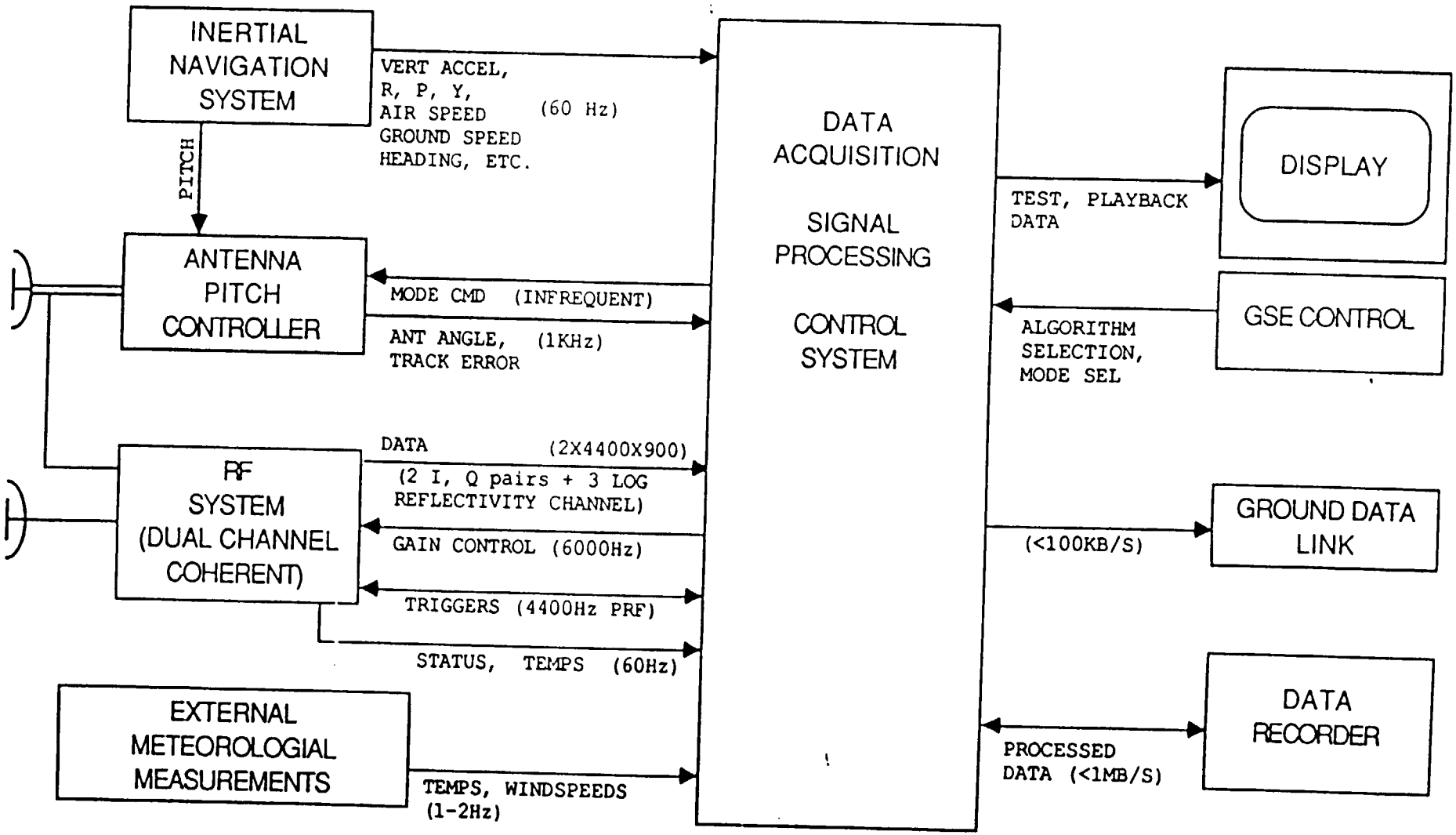


Figure 3-5: Data System block diagram

3.2.1 Radar Interface Board (RIB)

The Radar Interface Board (RIB) is responsible for generating the various system clocks and phase-locking them to the 10 MHz STALO oscillator in the radar transmitter. It is further used to control radar transmitter and receiver operations by supplying a transmit waveform used to modulate the RF carrier, selecting receiver bandwidth, and performing radar calibration operations. Automatic gain control is accomplished by manipulating RIB supplied analog signals to 60 dB RF- and IF-attenuators in the receiver. Acquisition of analog system status information such as temperature and pressure measurements within the transmitter, receiver, and data system, is achieved using a 16-channel 12-bit A/D converter. Furthermore, an octal 8-bit DAC provides the board with analog output for signaling and control. This capability is enhanced by 16 bits of digital I/O which are not presently utilized.

Central to the operation of the RIB is a single programmable AT&T WEDSP32C digital-signal-processor chip which executes embedded software to provide radar control. A typical embedded program appears in Appendix A and was used during the 1993 CAMEX experiments at the NASA Wallops Flight Facility.

This program begins by assigning pointers to the Board Control Register (BCR) and Radar Status Register (RSR). These registers, shown in Tables 3-2 and 3-3, specify the radar operating parameters (e.g. PRF, pulse width, etc.) and indicate the operating state of the system. Immediately after system powerup, the radar enters a warmup mode, heating the TWT to the proper operating temperature before transmitting. During this warmup period the RSR reports an operational status of STANDBY, and the RIB embedded code acquires and records the following status information twice each minute:

- Board Status Register (BSR)
- Radar Status Register (RSR)
- Transmitter Base Temperature (ADCDAC0)
- Receiver Base Temperature (ADCDAC1)
- Transmitter Air Temperature (ADCDAC2)
- Receiver Air Temperature (ADCDAC3)
- Transmitter Local Oscillator Temperature (ADCDAC4)
- Unused Channel (ADCDAC5)
- Transmitter Nitrogen Pressure (ADCDAC6)
- Unused Channel (ADCDAC7)
- RIB Ambient Temperature (ADCDAC8)

The ADCDAC registers are 12-bit sampled analog signals connected to various thermistors and pressure transducers.

After the pilot moves the operation switch to the RADIATE position and warmup is complete, the transmitter attempts to enter a radiate mode. To facilitate a successful mode change the RIB must immediately provide a trigger signal by asserting PRFSTART within the Board Control Register. If the RIB fails to provide this signal, and the transmitter is otherwise working properly, the transmitter will assert RSR[0] to indicate a *first-level fault* (FAULT1). This kind of fault (first-level) will also be generated if sensors indicate the transmitter or receiver are not working properly. For example, the following conditions will all initiate a first-level fault: transmitter or receiver outside of temperature operating range, TWT high-voltage off, transmitter power low , or transmitter VSWR out of range. If no faults are indicated, the transmitter will enter its normal operational state. However, if some failure results in a FAULT1 state, the transmitter will attempt to reset itself and enter the RADIATE state a maximum of three times. If the transition remains unsuccessful after the three attempts, the transmitter will completely shut down, assert RSR[1] to enter a catastrophic failure or *second-level fault* (FAULT2), and remain in this state until power is cycled.

Table 3-2: Board Control Register - BCR[15:0]

Bit	Net Name	Destination	Pin #	Function
0	RSRDEBUGN	PBCTL2 PAL	11	Radar Status Register debug (0)
1	PIOROUTN	PBCTL2 PAL	10	Programmable IO are Outputs (0) & Inputs (1)
2	FREQINP	PLL Clk Driver	6	Freq I/P Select b/w STALO (1) & Local Osc (0)
3	ADCDIFF	INAIS	1	Select Differential (1) or Non-diff (0) on ADCs
4	PRFCTL	PRFCTL PAL	E1	Select PRF in combination with PRF bit - HI: 8.8/1.1 KHz LO: 4.4/2.2 KHz
5	PRFSTART	PRFCTL PAL	E2	Start PRF
6	OPTPULW	PRFCTL PAL	B6	Select Pulse Width in combination with the PULSWID bit - HI: 0.5 µs LO: 1.0/0.25 µs
7	AGCSTART	AGCCTL PAL	2	Start AGC
8	DSPINT0	DSPINT PAL	2	DSP Interrupt Control Bit 0
9	DSPINT1	DSPINT PAL	3	DSP Interrupt Control Bit 1
10	DSPINT2	DSPINT PAL	4	DSP Interrupt Control Bit 2
11	DSPINT3	DSPINT PAL	5	DSP Interrupt Control Bit 3
12	LDPCTL	7-SEG DISPLAY	4	Left Decimal Point Control
13	RDPCTL	7-SEG DISPLAY	10	Right Decimal Point Control
14	DONE	VME CTL2 PAL	11	Signal End of Acquisition
15	NEWDWELL	AGCCTL PAL	7	New Dwell Signal from DSP

Table 3-3: Radar Status Register - RSR[15:0]

Bit	Net Name	Source	Pin #	Function
0	FAULT1	J1	1	Fault #1 - HI: Fault LO: Normal
1	FAULT2	J1	2	Fault #2 - HI: Fault LO: Normal
2	PRES	J1	3	Pressure - HI: Under LO: Normal
3	TROVTEMP	J1	4	Transmitter Over Temperature - HI: Over LO: Normal
4	TRUNTEMP	J1	5	Transmitter Under Temperature - HI: Under LO: Normal
5	RCOVTEMP	J1	6	Receiver Over Temperature - HI: Over LO: Normal
6	RCUNTEMP	J1	7	Receiver Under Temperature - HI: Under LO: Normal
7	TWTAST	J1	8	TWT Status - HI: Hi Voltage Off LO: Hi Voltage On
8	STHEATDEL	J1	9	Status Heater Delay - HI: Complete LO: Delay
9	STDBY_RAD	J1	10	Standby/Radiate - HI: Radiate LO: Standby
10	TXLPOW	J1	15	Transmitter Low Power (-0.2 dB) - HI: Low LO: Normal
11	RSRVD			
12	RSRVD			
13	RSRVD			
14	RSRVD			
15	RSRVD			

During the normal operational state RADIATE, the RIB records status information frequently (~3 second intervals) and monitors RSR[9] for assurance that all is well. An RSR[9] status change from RADIATE to STANDBY might indicate a transmitter or receiver first-level fault due to some random glitch; however, it could also indicate the pilot has turned off the transmitter! To determine which is the case, the RIB waits ten seconds. If the event was a transient fault, the minor glitches will work themselves out during this time and the radar will automatically return to the RADIATE state. If the system remains in STANDBY for more than ten seconds, it is assumed that the pilot has turned off the transmitter or that a catastrophic second-level fault has occurred. Regardless, he RIB deasserts the trigger and signals the E-6 host computer to terminate acquisition by closing files.

The RIB DSP32C processor receives a hardware interrupt each time the transmitter fires a pulse. These regular interrupts form the basis of internal program timing and provide synchronization among all processors. Every five minutes, the transmitter is switched into a calibration mode by asserting CALIB of Radar Control Register Zero (RCR0[3]). Attenuation settings are then read from a table and applied to the calibration attenuators by writing to the Radar Control Register One (RCR1). Presently, those calibration values step from 0 dB to -120 dB in 10 dB increments.

Table 3-4: Radar Control Register 0 - RCR0[15:0]

Bit	Net Name	Dest	Pin #	Function
0	RSRVD			
1	PULSWID	J1	11	Pulse Width - HI: 0.25 µs LO: 1.0/0.5 µs
2	PRF	J1	12	Pulse Repetition Freq - HI: 2.2/1.1 KHz LO: 4.4/8.8 KHz
3	CALIB	J1	13	Calibrate Mode - HI: Off LO: On
4	RECBW	J1	14	Receiver Bandwidth - HI: 2.2 MHz LO: 8 MHz
5	RSRVD			
6	PHASESH0	J2	1	Phase Shifter Bit 0 - HI: 5.6° LO: 0°
7	PHASESH1	J2	2	Phase Shifter Bit 1 - HI: 11.3° LO: 0°
8	PHASESH2	J2	3	Phase Shifter Bit 2 - HI: 22.5° LO: 0°
9	PHASESH3	J2	4	Phase Shifter Bit 3 - HI: 45.0° LO: 0°
10	PHASESH4	J2	5	Phase Shifter Bit 4 - HI: 90.0° LO: 0°
11	PHASESH5	J2	6	Phase Shifter Bit 5 - HI: 180° LO: 0°
12	PHASESHLAT	J2	7	Phase Shifter Latch - HI: Q=D LO: $Q=Q_o$
13	RSRVD			
14	RSRVD			
15	RSRVD			

Table 3-5: Radar Control Register 1 - RCR1[15:0]

Bit	Net Name	Dest	Pin #	Function
0	ATNA0	J2	8	Attenuator A Bit 0 - HI: 1 dB LO: 0 dB
1	ATNA1	J2	9	Attenuator A Bit 1 - HI: 2 dB LO: 0 dB
2	ATNA2	J2	10	Attenuator A Bit 2 - HI: 4 dB LO: 0 dB
3	ATNA3	J2	11	Attenuator A Bit 3 - HI: 8 dB LO: 0 dB
4	ATNA4	J2	12	Attenuator A Bit 4 - HI: 16 dB LO: 0 dB
5	ATNA5	J2	13	Attenuator A Bit 5 - HI: 32 dB LO: 0 dB
6	ATNB0	J2	14	Attenuator B Bit 0 - HI: 1 dB LO: 0 dB
7	ATNB1	J2	15	Attenuator B Bit 1 - HI: 2 dB LO: 0 dB
8	ATNB2	J2	16	Attenuator B Bit 2 - HI: 4 dB LO: 0 dB
9	ATNB3	J2	17	Attenuator B Bit 3 - HI: 8 dB LO: 0 dB
10	ATNB4	J2	18	Attenuator B Bit 4 - HI: 16 dB LO: 0 dB
11	ATNB5	J2	19	Attenuator B Bit 5 - HI: 32 dB LO: 0 dB
12	RSRVD			
13	RSRVD			
14	RSRVD			
15	RSRVD			

Table 3-6: Board Status Register - BSR[15:0]

Bit	Net Name	Source	Pin #	Function
0	VMEQBAFN	VMEQB	6	VME Queue B Almost Full Flag
1	VMEQBAEN	VMEQB	7	VME Queue B Almost Empty Flag
2	DATDQAAFN	DATDQA	6	Data Destination Queue A Almost Full Flag
3	DATDQAAEN	DATDQA	7	Data Desttination Queue A Almost Empty Flag
4	DATDQBAFN	DATDQB	6	Data Desttination Queue B Almost Full Flag
5	DATDQBAEN	DATDQB	7	Data Desttination Queue B Almost Empty Flag
6	DSPINQAFFN	DSPINQA	13	DSPINQA Full Flag
7	DSPINQAEFN	DSPINQA	12	DSPINQA Empty Flag
8	DSPINQBFFN	DSPINQB	13	DSPINQB Full Flag
9	DSPINQBEFN	DSPINQB	12	DSPINQB Empty Flag
10	NGAINQFFN	NGAINQ	13	NGAINQ Full Flag
11	NGAINQEFN	NGAINQ	12	NGAINQ Empty Flag
12	OGAINQFFFN	OGAINQ	13	OGAINQ Full Flag
13	OGAINQEFN	OGAINQ	12	OGAINQ Empty Flag
14	RNRNA	HSXCVRA	14	Receiver Not Ready Bus A
15	RNRNB	HSXCVRB	14	Receiver Not Ready Bus B

3.2.2 Data Acquisition Board (ACQ)

Received echoes are sampled by dual 4 MHz four-channel 9U VME boards featuring 12 bits of resolution per sample. Each board operates under the direction of a DSP32C digital signal processor chip whose job is to compute destination tags for each sample which are then paired with the samples acquired by the A/D's as routing information. The High-Speed Data Bus Controller (HSDBC) uses these tags to route each sample to some desired location, often a processing-zone on the LMAP Unit Array Processor Board (LUA-200), much like the Post Office uses address information on a letter to route it to the desired house.

The software used to scatter EDOP raw data for reflectivity processing is provided in Listing 2. A hardware interrupt notifies the program each time a pulse has been transmitted. After each interrupt, the software queues up destination tags for all 436 range gates to be acquired this pulse. This is accomplished by reading a table of

predetermined addresses. Four processors are required for reflectivity processing; hence, the first 109 tags route samples to Zone 0, the next 109 are routed to Zone 1, and so on. After sufficient data for an entire dwell have been scattered for processing, the global token (see §3.2.3) is passed to the LUA so that its results can be delivered from its output queue.

3.2.3 LMAP Unit Array Processor Board (LUA-200)

The LUA-200 is a highly-parallel signal-processing board containing a Global I/O Interface, Local I/O Interface, VME Interface, Input and Output Data Queues, and eight linearly-mapped computational zones. The reader is referred to Figure 3-6 for a block diagram explaining the relationship between these components.

Raw data arrive at the board for processing via two 40-bit bi-directional high-speed data buses (HSDB) dedicated to the transfer of information between the ACQ and LUA boards. Each bus is comprised of 32 data lines, 4 control lines, 4 lines for destination tags, and runs at ¼ the system clock (10 MHz) to achieve a maximum throughput of 50 Mbyte/sec/bus.

Inter-zone communications is facilitated by hardware-based Input and Output data queues. Each queue is a 40-bit x 512-word FIFO possessing a global and local side. The global side of the queue is connected to the Global I/O Interface which manages the transfer of information between the I/O queues and the HSDB. The Local I/O interface similarly manages information transfer between the I/O queues and processing zones. Data (i.e. a destination tag/data word pair) received from the HSDB by the Global Interface controller are placed in the Input Data Queue of the appropriate bus where they remain until their destined zone requests that they be read from the queue and transferred to the zone's local memory. The data are removed from the queue and flexibly routed to the desired zone

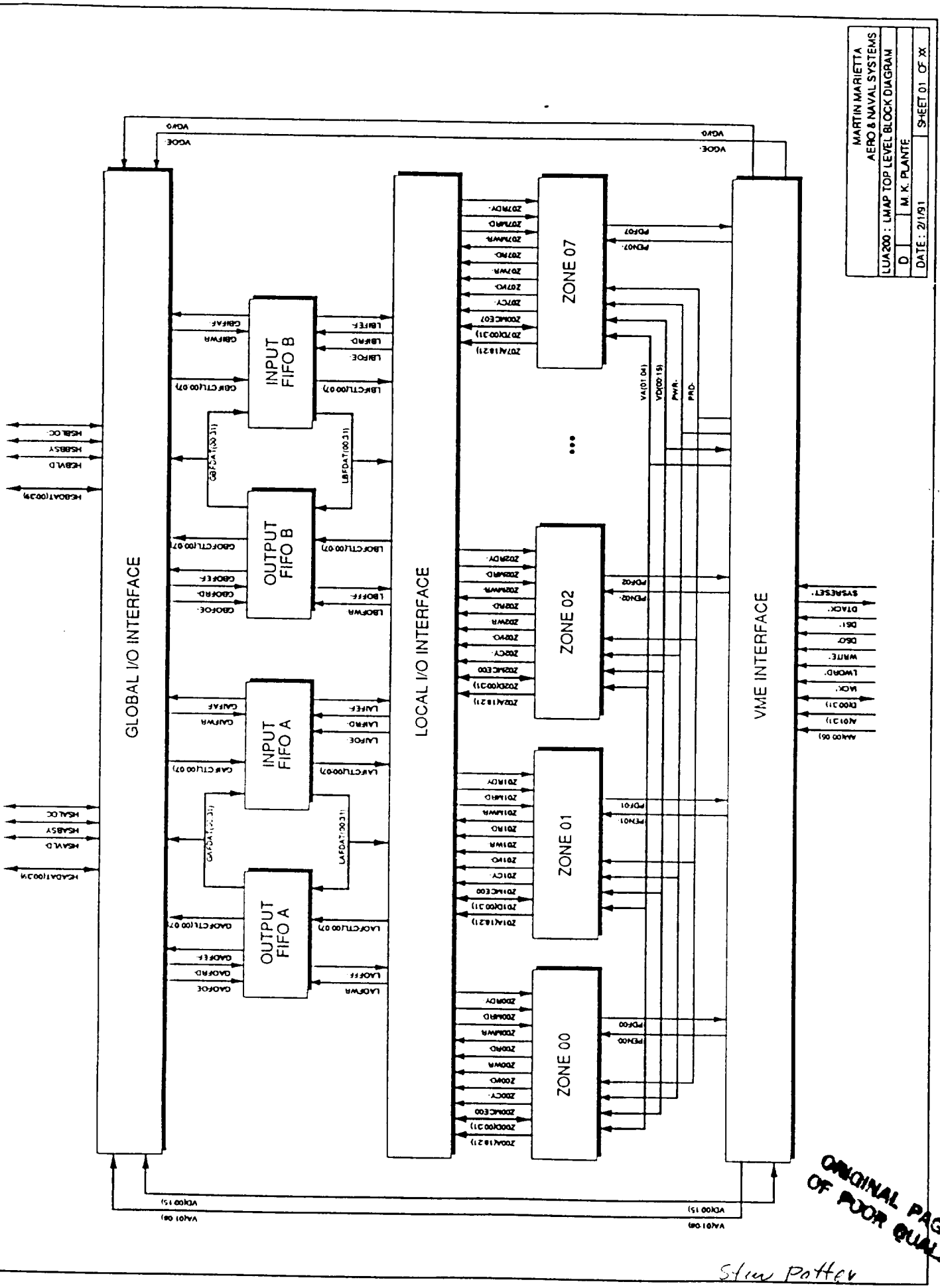


Figure 3-6: Block diagram of LUA-200 signal-processing board

Steve Potter

through a series of cross-point data switches on each board—the Local I/O Interface. Because the queue is based on a FIFO architecture, they are read out in the same order they are received. If a processing zone requests additional data and data destined for that zone are not at the front of the queue, the zone will *block* or wait until such data are available. We say that the input data stream is *flow controlled.*

When a zone is finished processing and desires to transfer information to another location, it does so by writing the data and appropriate destination tag to the Output Data Queue. To control the order is which output data are queued up, only one zone is given permission to write at any given time. This permission or *write arbitration* is governed by the use of a semaphore called the *local token*, and there is one local token for each local bus. Any zone attempting to access the Output Data queue without possessing the write access privileges granted by the local token will be held in a pending-access state until the zone receives the local token from the current token holder.

After data are placed in the Output Data Queue, the Global I/O Interface reads the output destination queue and passes its associated data word to the proper destination. If the data are destined to another zone on the same board, the Global I/O Circuit merely places them back in the Input Data Queue. If they are instead destined for another board, the Global I/O Circuit checks to see if the board is a *global token* holder. The global token provides bus arbitration on the HSDB in a manner similar to the local bus so that only one board in the system has access to the HSDB at any given time. If the board is possesses the global token, the Global I/O interface removes data from the Output queue and passes them to the HSDB Controller for transfer to the Global I/O Controller of the destination board (via the high-speed data bus). If the board does not possess the global token, the data remain in the Output queue until the global token is received by the current token holder.

Three operating modes for the LUA Board, the Master, Slave, and Local Access modes, further govern Global I/O.

Master Mode allows write access to the High-speed Data Bus and is allowed only by the Global Token holder. In this mode, the presence of data in the Output Data Queue initiates a data transfer to the destination specified in the Data Destination Queue. Valid destinations include the Data Input Queue of the same board, Data Input Queue of a different board, or any global bus command (Figure 3-7).

Boards which do not possess the Global Token are considered to be in *Slave* mode and will remain Slaves until the Global Token is relinquished by the Master. A Slave can receive data from its Data Input Queue and additionally write to the Data Output Queue as Local Token issues permit. However, no data will be read from the Output Queue until the board becomes a Master by receiving the Global Token from the current Master.

In some cases it is desired to pass data only between zones of the same board. Because no data are transferred over the HSDB, it does not make sense to restrict data flow with a global token; however, a the Global I/O controller will not read the Output queue of a Slave. To address this problem, a third mode, *Local Access* mode, is available. In Local Access Mode, the board isolates itself from the HSDB allowing data to be passed through the Output Queue back to the Input Queue without ever possessing the Global Token. In this mode, the Global DMA Controller processes data from the Data Output Queue as though the board was a Master—except that destinations to other boards are invalid. Any data destined to different board will be lost. Similarly, any data transferred to a board in Local Access mode over the HSDB will be lost. To enter Local Access Mode, the board must first be the Global Token holder (i.e. Master). After setting HCR[1], the board must

DESTINATION CODE (HEX)	*BOARD NODE* (DEC)	*DSP ZONE* (DEC)	*REGISTER/ FUNCTION*	*ZONE OPERATION*	*VME OPERATION*
'00' : '07'	0 " 0	0 (000) : 7 (007)	DATA " DATA	WR " WR	WR/RD " WR/RD
'08'	0		MASK REG	WR	WR/RD
'09'	0		CTLR REG	WR	WR/RD
0A	0		NODE	WR	WR/RD
0B					
0C	ALL		DATA	BCAST WR	BCAST WR
0D	ALL		MASK REG	BCAST WR	BCAST WR
0E					
0F	0	*	TOKEN	PASS	PASS
10 : 17	1 " 1	0 (008) : 7 (0015)	DATA " DATA	WR " WR	WR/RD " WR/RD
18	1		MASK REG	WR	WR/RD
19	1		CTLR REG	WR	WR/RD
1A	1		NODE	WR	WR/RD
1B					
1C	ALL		DATA	BCAST WR	BCAST WR
1D	ALL		MASK REG	BCAST WR	BCAST WR
1E					
1F	1	*	TOKEN	PASS	PASS
20 : 27	2 " 2	0 (016) : 7 (023)	DATA " DATA	WR " WR	WR/RD " WR/RD
28	2		MASK REG	WR	WR/RD
29	2		CTLR REG	WR	WR/RD
2A	2		NODE	WR	WR/RD
2B					
2C	ALL		DATA	BCAST WR	BCAST WR
2D	ALL		MASK REG	BCAST WR	BCAST WR
2E					
2F	2	*	TOKEN	PASS	PASS

Figure 3-7: Data Destination Codes (courtesy of Martin Marietta)

pass away the Global Token. The board will remain in Local Access Mode until resetting HCR[1] at which time it will become a Slave.

Each of the eight linearly-mapped parallel processing *zones* is centered around an AT&T WEDSP32C digital signal processor housed in a standard 133-pin square pin-grid array (PGA) package. CMOS fabrication technology provides high-speed operation with low power consumption and two execution units increase computational performance. The Data Arithmetic Unit (DAU) utilizes four 40-bit floating-point accumulators to store operation results and data type conversions. A 32-bit floating-point adder and 40-bit floating-point multiplier work in tandem to jointly perform 25 million floating-point multiply-accumulate operations per second (peak) when clocked at the maximum speed of 50 MHz. Integer operations, as well as control, logical, and data move operations are conducted by a second execution unit—the Control Arithmetic Unit (CAU)—at a rate of 12.5 million instructions per second (MIPS). Memory internal to the processor includes1536 32-bit words allocated in three 512-word banks. An external memory of 32K x 32-bit 20-ns zero wait-state static RAM per zone provides ample room for program and data storage. Three 40-MHz LUA-200's are utilized by the EDOP data system yielding an aggregate 480 MFLOPS of computational power for Doppler extraction, reflectivity averaging, and automatic gain control.

3.2.4 Host Computer (E-6)

An E-6 Eltec single-board computer based on the 68030 microprocessor is utilized as a host. This host is responsible for initializing various system registers at power up and loading all DSP32Cs with software stored on a battery-backed non-volatile 8 Mb RAM drive. Status information are additionally stored on this RAM drive by the E-6 which is also responsible for archiving processed data sent to it via the VME bus during operation.

A 1.2 Gb Micropolis hard drive is currently used to store processed results, and a 5 Gb 8505C Exabyte streaming tape is being evaluated for future use.

Chapter 4 - Algorithm Development and Analysis

4.1 Reflectivity Processing

Conventional weather radars relate the received echo power from a precipitating target to its radar reflectivity factor, Z, through a special form of the radar equation applicable to distributed targets. The radar reflectivity factor can then be empirically related to the rain rate. Nevertheless, the problem of accurately assessing Z based on power measurements is not without difficulties. As wind turbulence shuffles the hydrometeors, backscattered microwaves undergo constructive and destructive interference causing the magnitude of received power to vary greatly. To effectively estimate the true mean received power, one must average many power samples to reduce the variance of this *fading* process.

4.1.1 The Need for Reflectivity Averaging

To better illustrate the nature of fading, let the receiver input voltage due to the i^{th} scatterer be expressed in polar form as:

$$V_i e^{j\phi_i}$$

where V_i is its magnitude and ϕ_i is the instantaneous phase. We are normally interested in the amplitude and phase of the complex voltage, V, due to a collection of scatterers and can easily express this as a superposition of received voltages from individual scatterers:

$$V = \sum_{i=1}^{N_t} V_i e^{j\phi_i}$$

Of course, the sum of a complex-number series is itself a complex number, so it is convenient to express the complex voltage, V, in terms of an envelope, V_e, and a phase angle ϕ:

$$V = V_e e^{j\phi}$$

If the number of randomly phased scatterers is sufficiently large, the central limit theorem provides reasonable assumption that the received voltage is a bivariate Gaussian distribution of the form

$$p(V_e, \phi) = \frac{V_e}{2\pi\sigma^2} e^{-V_e^2/2\sigma^2}$$

The individual density functions for V_e and ϕ can then be determined by "integrating out" the opposing variable.

$$p(V_e) = \int_0^{2\pi} p(V_e, \phi)\, d\phi = \frac{V_e}{\sigma^2} e^{-V_e^2/2\sigma^2}$$

$$p(\phi) = \int_0^{\infty} p(V_e, \phi)\, dV_e = \frac{1}{2\pi}$$

It is interesting to observe that $p(V_e)$ is the familiar Rayleigh distribution whose first two moments are given by

$$\overline{V_e} = \sqrt{\frac{\pi}{2}}\sigma$$

$$\overline{V_e^2} = 2\sigma^2$$

The variance can then be determined as the difference between the second moment and the square of the first:

$$\sigma_{V_e}^2 = \overline{V_e^2} - \overline{V_e}^2 = \left(2 - \frac{\pi}{2}\right)\sigma^2 = 0.429\,\sigma^2$$

Assume the power in the envelope voltage is developed across a 1-ohm resistor. The envelope power can be expressed as the square of the envelope voltage.

$$P = V_e^2$$

If we further note that the differential power is related to the differential voltage as

$$dP = 2\,V_e\,dV_e$$

then the Rayleigh distributed envelope voltage can be converted to a distribution for power by transforming the voltage variable to obtain an expression for the distribution of power at the radar receiver input

$$p(P)dP = p(V_e)dV_e = \frac{1}{2\sigma^2}e^{-P/2\sigma^2}$$

Notice that the power is distributed *exponentially* whereas the voltage was Rayleigh distributed! The first two moments of the exponential power distribution are easily seen to be:

$$\overline{P} = 2\sigma^2$$
$$\overline{P^2} = 2\overline{P}^2$$

revealing a classic property of exponential distributions wherein the standard deviation is equal to the mean!

$$\sigma_P^2 = \overline{P^2} - \overline{P}^2 = \overline{P}^2$$
$$\sigma_P = \overline{P}$$

By now it should be evident that the power fluctuations due to fading can be significant and some type of estimator based on multiple observations must be employed to reduce this variance to acceptable levels. Before we can address that issue, however, we must follow the received pulse through the receiver to understand its effects on the fading statistics. Weather targets, especially convective targets, exhibit a range of echo strengths exceeding 80 dB. This rather large variation places tough demands on the receiver design and frequently logarithmic detectors, whose output signal is proportional to the logarithm of the input power, are used to minimize receiver saturation. Let L be defined as input to the log-receiver, expressed in logarithmic form, where P is the instantaneous, exponentially-distributed received power:

$$L = 10\log(P/P_1)$$

The reference value P_1 is chosen to be 1 mW so that the units of power-level L are dBm. The mean power can be similarly written:

$$L_o = 10\log(\overline{P}/P_1)$$

Certainly, the logarithmic transfer function compresses the range of output, but additionally has less intuitive effect of adding a statistical bias to the measurement. To see that this is true, it is again necessary to transform the power variable using

$$L - L_o = 10\log(P/\overline{P})$$

As might be expected, the distribution of log-power is no longer exponential but is instead a slight variation given by (Sauvageot, 19xx)

$$p(L) = m\exp[m(L - L_o) - e^{m(L-L_o)}]$$

with m = ln(10)/10. The first two moments, can be obtained by integrating the density function

$$E\{L\} = \int_{-\infty}^{\infty} L \cdot m\exp[m(L - L_o) - e^{m(L-L_o)}]dL$$

$$E\{L^2\} = \int_{-\infty}^{\infty} L^2 m\exp[m(L - L_o) - e^{m(L-L_o)}]dL$$

These equations were numerically integrated to determine the mean and variance of the detected spectrum. By definition, the true power mean is L_o. However, the expected value of L is given by $E\{L\} = L_o$-2.51, indicating that the estimate is statistically biased. This means that one must add 2.51 dB to the estimate to get the true mean. The variance is nearly constant at 31 dB.

4.1.2 An Averaging Algorithm for the DSP32C

Clearly, the power estimate must be improved if it is to be scientifically useful. A particular type of estimator known as the *maximum-likelihood estimator* was chosen to provide mean-power estimates for the EDOP application. It works as follows: Let $f(x_1,...,x_n|\theta)$ be the joint probability density function of the random variables X_1, X_2, ..., X_n, where θ is the parameter to be estimated—in this case, the mean power. This function is

called the *likelihood* function and represents the likelihood or certainty that the values x_1, x_2, ..., x_n will be observed when θ is the true value of the unknown power mean. It is then reasonable to assume that θ can be estimated by choosing its value such as to maximize the likelihood of observing x_1, x_2, ..., x_n. In other words, the maximum-likelihood estimate is defined to be that value of θ which maximizes the likelihood function.

The maximum-likelihood estimator for the mean of an exponential distribution is given by:

$$\hat{\theta} = \frac{1}{N} \sum_{i=1}^{N} X_i$$

To generate estimates of the mean echo-power received by EDOP, a real-time embedded, parallel algorithm was developed for the EDOP data system to average a predefined number of samples, N, and archive the results. A listing of this software is included in Appendix A.

To implement the algorithm, two constraints were observed to maximize the performance:

1. Keep the pipeline full
2. Minimize use of memory

As previously mentioned, the DSP32C is a pipelined digital signal processor. Pipelined architectures execute their instructions in assembly-line fashion to realize superior performance. Each *stage* of the pipeline completes a small part of the instruction being executed in a fraction of the time needed to complete execution of the entire instruction. Moreover, each stage is connected together to form a pipe so that instructions enter the

pipe at one end, undergo various stages of execution in each pipe segment, and exit other end.

Throughput of the pipeline is the rate at which completely executed instructions exit the pipeline. Because multiple instructions are overlapped in execution, an instruction can exit the pipe with each machine cycle resulting an increase in CPU performance directly proportional to the length of the pipe. However, superior performance is sustained only so long as the pipeline is kept full. If the pipeline is not kept full, some segments will be idle when they could be executing instructions and the throughput will decrease. As this *bubble* of inactivity propagates through the pipe and exits the other end, one or more machine cycles will be wasted while the CPU waits for the next instruction to exit the pipe.

The data arithmetic unit (DAU) of the DSP32C employs a four-stage pipeline to perform 25 million floating-point operations per second. Its instruction set revolves around a multiply-accumulate operation wherein a floating-point multiplier and adder work in parallel to perform computations of the form a = b + c*d. Each DAU multiply-accumulate operation involves three floating-point operands. Two of these are multiplied together and added to the third. The final result is stored in an accumulator as an intermediate result to be used in subsequent operations and can be additionally stored in memory or sent to an I/O unit.

The DAU supports several multiply-accumulate instruction formats. However, for the purpose of discussing the DSP32C pipeline, only one format will be examined:

$Z = aN = aM + Y * X$

This instruction is executed in four stages as follows:

1. XY fetch
2. multiply Y * X
3. accumulate product with aM
4. write result to memory (optional)

If several multiply-accumulate operations are executed back to back, the DSP32C fills the pipeline such that one stage of each instruction is completed every machine cycle. For example, consider the following block of multiply-accumulate instructions:

1. $aN = aM + Y_1 + X_1$
2. $aN = aM + Y_2 + X_2$
3. $aN = aM + Y_3 + X_3$
4. $aN = aM + Y_4 + X_4$
5. $aN = aM + Y_5 + X_5$

where X_1—X_5 and Y_1—Y_5 have been subscripted to aid visualization of the data flow through the pipeline. During each machine cycle, the four instructions will be in various stages of simultaneous execution within the pipeline:

Table 4-1: Pipeline Data Flow

Cycle	Segment 4	Segment 3	Segment 2	Segment 1
1				XY fetch$_1$
2			multiply$_1$	XY fetch$_2$
3		accumulate$_1$	multiply$_2$	XY fetch$_3$
4	write$_1$	accumulate$_2$	multiply$_3$	XY fetch$_4$
5	write$_2$	accumulate$_3$	multiply$_4$	XY fetch$_5$
6	write$_3$	accumulate$_4$	multiply$_5$	
7	write$_4$	accumulate$_5$		
8	write$_5$			

It should be evident from the table that the intermediate accumulated result from the first instruction will not be available for use in subsequent multiplier operations until Cycle 4. Hence there is a *latency* of three instructions before the intermediate results from the first instruction can be used. In other words, if it is desired that the intermediate results of one instruction be used in a subsequent instruction, no less than three *unrelated* DAU instructions must be executed before the intermediate results are available for use. Such latencies are inherent in pipelined architectures and the DSP programmer must design software carefully, keeping the pipeline full to maximize the throughput.

To begin the actual coding of software for this digital-signal processor, a slightly different approach is used than for general purpose CPUs. In this approach, the DAU or floating-point unit is considered to be the central processing unit of the DSP32C while the CAU is viewed as a co-processor to generate addresses and perform occasional logical operations. To fully utilize the processing power of the parallel array of DSP32Cs, consideration must be given to how the processing will be split among the processors while maximizing the pipeline efficiency. For the reflectivity averaging software, it was decided that each processor would perform the same estimation algorithm on a subset of the range gates to be processed.

After each RF pulse is transmitted, three A/D's provide range-gated samples of received power on each of three channels—nadir antenna, forward co-polarized antenna, and forward cross-polarized antenna . These echo power measurements are sent to the data system via both 32-bit HSDB buses, with two 16-bit channels packed into a single 32-bit word per bus. The reflectivity program expects that each word read from HSDB FIFO A contains channel one in the most-significant 16 bits while channel two is stored in the least-significant 16 bits. Similarly, channel three is expected to reside in the most-significant 16 bits of FIFO B; however, channel four is ignored. A typical flight

configuration is to place the Nadir Co-pol. on channel one, the Forward Co-pol on channel two, and the Forward Cross-pol on channel 3. These range-gated measurements are placed into a buffer for accumulation by the following zero-overhead core loop of reflectivity DSP32C-assembler code:

```
do 5, r18
 a1 = float(*r10++)
 a0 = float(*r10++)
 a2 = float(*r11+r15)
 *r6++ = a3 = *r6 + a1
 *r5++ = a3 = *r5 + a0
 *r7++ = a3 = *r7 + a2
```

The first line indicates that the following six (5+1) lines of code are to be executed N times, where N is the value contained within hardware register r18 at the time of execution. In the buffer, addressed by hardware pointers r10 and r11, the received-power samples remain packed together with two channels per word. The next three instructions in the loop extract the 16-bit integer samples from the buffer and convert them to a 32-bit DSP32C proprietary floating-point format, placing each result in a 40-bit accumulator. In this example, samples from channel one are placed in accumulator a1 while samples from channels two and three are placed in accumulators a0 and a2, respectively. The remaining three instructions accumulate the measurements into separate range-gated tables addressed by hardware pointers r5, r6, and r7. At the end of each integration period, the accumulated results are placed in the Output queue along with appropriate destination tags sending the results to the VME bus for subsequent archival by the E-6 host. A *pass global-token* command is then queued to return the global token to the ACQ board so the HSDB Controller can deliver additional samples to the LUA for processing.

4.1.3 Number of Independent Samples

The reflectivity algorithm used by EDOP averages a series of observations, L_i, for one second to reduce the variance to acceptable levels. However, the observations are partially correlated, so to determine how much the variance is reduced through averaging, one cannot simply reduce the variance by the number of averaged samples order. Instead, the variance is reduced by the number of *effectively independent* samples used in forming the average. Traditionally, samples are considered to be independent when the correlation between samples approaches zero.

Several authors (e.g. Marshall and Hitschfeld, 1953; Sauvageot; Walker et. al, 1980) have attempted to develop receiver models for the purpose of establishing the rate of sample decorrelation. For example, Kerr (1951) provides the following relation between the normalized input and output correlation functions of a logarithmic receiver:

$$\rho_{log}(\tau) = \frac{6}{\pi^2} \sum_{m=1}^{\infty} \rho^{2m} m^{-2}$$

Here, $\rho_{log}(\tau)$ is the desired normalized output correlation and $\rho(\tau)$ is the correlation function of the input process. If the input signal is assumed to be Gaussian correlated,

$$\rho(\tau) = \exp(-\tau^2/2\sigma_\tau^2),$$

then the output signal is correlated as

$$\rho_{log}(\tau) = \frac{6}{\pi^2} \sum_{m=1}^{\infty} m^{-2} \exp(-m\tau^2/\sigma_\tau^2)$$

One definition for the number of independent samples is given as a ratio of the variance of a single sample to the variance of the sampled mean (Lee, 1961; Nathanson, 1969). For discrete sampling, (Walker et. al, 1980) give

$$N_i^{-1} = \frac{1}{N_q} + \frac{2}{N_q^2} \sum_{K_q=1}^{N_q-1} (N_q - K_q) \rho_q (K_q l_q)$$

where N_q is the number of samples spaced l_q apart where q represents time. Thus to determine the number of independent samples, N_i, one merely substitutes the appropriate correlation function.

These efforts are of somewhat questionable utility because it is so difficult to specify the meteorological processes that are viewed by the airborne receiver. Highly turbulent conditions will certainly cause rapid decorrelation between samples and the ensuing large number of independent samples will result excellent mean-power estimates for even short integration periods. On the other hand, stratiform conditions with weak air motions might push decorrelation times on the order of seconds so that observed sample decorrelation results primarily from aircraft displacement. In view of the consequence of incorrectly specifying the observable meteorology, it is best to be conservative.

For airborne radars, it is generally accepted that pulses are decorrelated in the time it takes the aircraft to move one beamwidth. The EDOP radar flies with 0.76 m antennas on an ER-2 platform possessing an airspeed of approximately 200 m/s; therefore, pulses are expected to decorrelate in ~4 ms due to aircraft motion, providing a minimum of 250 independent samples during a 1-second integration period. Again, certain meteorological conditions (e.g. severe convective regions of a storm) can decorrelate the pulses at a faster rate. It seems the best way to determine independent samples obtained on a particular

flight is to examine the taken measurements. If the conservative decorrelation estimate of 4 ms is used, the expected post-integration (1 second) variance is

$$\sigma_{avg}^2 = \frac{\sigma_L^2}{250} = 31\,\mathrm{dB} - 10\log(250) \approx 7.02\,\mathrm{dB}$$

4.2 Doppler Processing

The EDOP radar is primarily intended to study deep, well-developed, precipitating systems where both horizontal- and vertical-wind velocities can exceed 40 m/s. The desired accuracy of velocity estimates is on the order of 0.5-1.0 m/s. However, errors arising from signal-processing operations and aircraft platform instabilities can induce significant errors in the vertical velocity that are sometimes comparable to the meteorology of interest (Heymsfield, 1989). The principle sources of velocity measurement error for the EDOP system result from uncertainties in the auto-covariance estimator and antenna pointing errors as discussed in §4.2.3 and §5.2.

4.2.1 Autocovariance Processing

The autocovariance $C_{xx}(t_1,t_2)$ partially describes the time-domain structure of a random process X(t) and is defined:

$$C_{XX}(t_1,t_2) \equiv R_{XX}(t_1,t_2) - \mu_X^*(t_1)\mu_X(t_2)$$

where the autocorrelation R_{XX} is given by the expected value of the product of the process at two times, t_1 and t_2:

$$R_{XX}(t_1,t_2) \equiv E\{X^*(t_1)X(t_2)\}$$

Loosely speaking, a process is called stationary if its distribution functions or certain expected values are invariant with a shift of the time axis [Shanmugan, 1988]. If the autocorrelation function depends only on the time difference τ but is not explicitly a function of times t_1 and t_2, and the mean is constant, then the process is said to be *wide-sense stationary*.

$$R_{XX}(\tau) \equiv E\{X^*(t)X(t+\tau)\},$$
$$E\{X(t)\} = \mu_x = \text{constant},$$

The autocorrelation function of a stationary random process tells us something about how rapidly we can expect the random signal X(t) to vary as a function of time. If the autocorrelation function rapidly decays to zero, we can expect the signal X(t) to vary rapidly with time. If the autocorrelation decays slowly to zero, then X(t) will be a slowly changing process. Likewise, a process X(t) with periodic components will exhibit a periodic autocorrelation. Thus, we may correctly conclude that the autocorrelation function provides some information describing the underlying spectral content of the random process. The exact relation is given by the Wiener-Khinchine relationship which states that the frequency-domain power distribution or power-spectral density of a random process X(t) is given by the Fourier transform of its autocorrelation function:

$$S_{XX}(f) = F\{R_{XX}(\tau)\} = \int_{-\infty}^{\infty} R_{XX}(\tau) e^{-j2\pi ft} d\tau$$

For any given range gate, the receiver produces a series of noise-corrupted complex voltage samples, $V(kT_s)$, separated by the interpulse period, T_s. Each sample can be written as the sum of signal and noise:

$$V(kT_s) = V_k + n_k, \qquad k = 0,1,\ldots,M-1$$

The signal term can be alternatively expressed as a Doppler shifted weather-echo signal whose spectrum is centered about zero frequency:

$$V(k\,T_s) = s_k\, e^{j\omega_d k T_s} + n_k$$

The autocorrelation function of the process is therefore

$$R_{vv}(m\,T_s) \equiv E\{V^*(k\,T_s)X[(k+m)\,T_s]\} = S\rho(m\,T_s)\,e^{j\omega_d m T_s} + N_p\,\delta_m$$

where N_p is defined to be the mean noise power, ρ is the normalized correlation function, and δ_m is 1 for m = 0 and zero elsewhere. The difficulty in evaluating this autocorrelation function centers around determining the normalized correlation function. An easier alternative is to determine the autocorrelation directly from the power spectral density by inverting the Wiener-Khinchine equation above:

$$R_{xx}(\tau) = F^{-1}\{S_{xx}(f)\} = \int_{-\infty}^{\infty} S_{xx}(f)\,e^{j2\pi ft}\,df$$

For the purpose of determining statistical estimators, it is both convenient and reasonable to assume that the power spectrum has a Gaussian shape:

$$S(\upsilon) = \frac{S}{\sqrt{2\pi}\,\sigma_v}\exp\left(-\frac{(\upsilon-\overline{\upsilon})^2}{2\sigma_v^2}\right) + \frac{2N_p T_s}{\lambda}$$

Substitution yields

$$R_{vv}(mT_s) = S\exp\left(-8\left(\frac{\pi\sigma_v mT_s}{\lambda}\right)^2\right)e^{j4\pi\bar{v}mT_s/\lambda} + N_p\delta_m \quad .$$

This autocorrelation function can then be compared with the one above to identify the normalized correlation function as:

$$\rho(mT_s) = \exp\left(-8\left(\frac{\pi\sigma_v mT_s}{\lambda}\right)^2\right)$$

At this point, it should start to become evident that if the complex autocorrelation function of the random process is known, the mean velocity can be obtained from its argument!

$$\arg[R_{vv}(mT_s)] = \frac{4\pi\bar{v}mT_s}{\lambda}, \qquad m \neq 0$$

The simplest case is to solve for the mean velocity using a single time lag (m = 1). The resulting velocity estimate is then given by:

$$\bar{v} = (\lambda/4\pi T_s)\arg[R_{vv}(T_s)]$$

This well-known autocovariance or pulse-pair estimator is applied to each range-gate to provide estimates of the Doppler velocity. This algorithm was first described by Rummler (1968) and has since become the primary Doppler velocity estimator used by the meteorological community. A spectral-width estimator can also be derived as a function of the complex autocorrelation function and is based on the ratio of lag-1 and lag-2 estimates (Srivastava et al., 1979):

$$\sigma^2 = \frac{\lambda^2}{24\pi^2 T_s^2}\ln\left|\frac{R(T_s)}{R(2T_s)}\right|$$

The new variance estimator is of comparable quality to its more popular lag-1/lag-0 counterpart (Benham et al., 1972; Sirmans and Bumgarner, 1975) but does not explicitly require an estimate of the signal-to-noise ratio (SNR):

$$\sigma_{old}^2 = \frac{\lambda^2}{8\pi^2 T_s^2}\left[1 - \frac{|R(T_s)|}{R(0)}\left(1 + SNR^{-1}\right)\right]$$

Interestingly, the new variance estimator of Srivastava can be combined with the old one to provide an estimate of the SNR itself based on all three lags. The SNR can then be used as a statement of confidence for the reliability of these estimates:

$$SNR = \frac{|R(T_s)|^{4/3}}{\left[R(0)\left|R(2T_s)^{1/3}\right| - |R(T_s)|^{4/3}\right]}$$

4.2.2 DSP32C Implementation

Doppler processing of the four linear channels represents the bulk of computations to be performed by the EDOP data system and centers around the computation of the complex autocorrelation function as described in §4.2.1:

$$V_p = \frac{\lambda}{2}\frac{1}{2\pi T_s}\arg\left(\frac{1}{n}\sum_i Z_i Z_{i+1}^*\right), \qquad i = 0,1,\ldots N-1$$

Z_i is the received complex echo-voltage of the i^{th} pulse, λ is the transmitted wavelength, N is the number of pulses per dwell, and T_s is the interpulse period.

The autocorrelation is essentially a series of complex multiply-accumulate operations and is ideally handled by the DSP32C floating-point DSP. Consider the lag-1 autocorrelation for a dwell of N pulses:

$$R(T_s) \propto \sum_{p=0}^{N-1} Z_p^* Z_{p+1}, \qquad p = 0,1,\ldots,N-1$$

Both sides of the equation can be expressed as real and imaginary components:

$$\begin{aligned} I(T_s) + jQ(T_s) &= \sum_{p=0}^{N-1} \left[I_p + jQ_p\right]^* \left[I_{p+1} + jQ_{p+1}\right] \\ &= \sum_{p=0}^{N-1} \left[I_p - jQ_p\right]\left[I_{p+1} + jQ_{p+1}\right] \\ &= \sum_{p=0}^{N-1} I_p I_{p+1} - jI_{p+1}Q_p + jI_pQ_{p+1} + Q_pQ_{p+1} \end{aligned}$$

Collecting terms,

$$I(T_s) = \sum_{p=0}^{N-1} I_p I_{p+1} + Q_p Q_{p+1}$$

$$Q(T_s) = \sum_{p=0}^{N-1} I_p Q_{p+1} - I_{p+1} Q_p$$

The lag-0 and lag-2 autocorrelation products are computed in a similar manner and are summarized below:

$$I(0) = \sum_{p=0}^{N-1} I_p^2 + Q_p^2$$

$$Q(0) = 0$$

$$I(T_s) = \sum_{p=0}^{N-1} I_p I_{p+1} + Q_p Q_{p+1}$$

$$Q(T_s) = \sum_{p=0}^{N-1} I_p Q_{p+1} - I_{p+1} Q_p$$

$$I(2T_s) = \sum_{p=0}^{N-1} I_p I_{p+2} + Q_p Q_{p+2}$$

$$Q(2T_s) = \sum_{p=0}^{N-1} I_p Q_{p+2} - I_{p+2} Q_p$$

To implement the algorithm in DSP32C assembler, it is again necessary to design the code so that the pipelined architecture of the CPU is optimally utilized. Shown below is the fourteen line core of EDOP's pulse-pair code (the complete listing is contained within Appendix B) which is used to accumulate autocorrelation products as described above. In flight, pulses will be correlated for a relatively long period of time, say, one-half second. At EDOP's maximum PRF of 4400 Hz, 2200 pulses will be correlated for each range gate. This loop correlates the 2200 pulses in small blocks to provide optimum utilization of the pipeline, accumulating the I and Q intermodulation products needed for the first three autocorrelation lags:

```
do 12,r16
    a0 = *r6++              /* IpPrev */
    a1 = *r9++              /* QpPrev */
    nop                     /* latency... */
    a2 = *r1 + a0 * a0      /* a2 = I0 + Ip ^ 2 */
    *r1 = a2 = a2 + a1 * a1 /* I0 = Ip ^ 2 + Qp ^ 2 */
    a2 = *r2 + a0 * *r7     /* a2 = I1 + IpIp+1 */
    *r2 = a2 = a2 + a1 * *r10 /* I1 = a2 + QpQp+1 */
    a2 = *r4 - a0 * *r10++  /* a2 = Q1 - IpQp+1 */
    *r4 = a2 = a2 + a1 * *r7++ /* Q1 = a2 + QpIp+1 */
    a2 = *r3 + a0 * *r8     /* a2 = I2 + IpIp+2 */
    *r3 = a2 = a2 + a1 * *r11 /* I2 = a2 + QpQp+2 */
    a2 = *r5 - a0 * *r11++  /* a2 = Q2 - IpQp+2 */
```

```
*r5 = a2 = a2 + a1 * *r8++        /* Q2 = a2 + QpIp+2 */
```

Hardware pointers r1—r5 point to range-gated tables where the accumulated intermodulation products are kept. Often the total number of pulses being processed is too large to simultaneously store them in memory. Therefore, a subset of these pulses is pulled into memory, processed, and then discarded so that an additional subset may be accessed. To ensure continuity of processing between the subsets, the last three pulses from each subset must be retained in memory as each new subset is created. These *retained* pulses are addressed by pointers r6—r11.

To determine the load (i.e. number of range gates and subset size) placed on each processor, simulations of the completed algorithm were run to calculate the number of instruction cycles needed to execute the algorithm for various load configurations.

4.2.3 Pulse-pair Velocity Estimate Uncertainties

An expression for the variance of the pulse-pair velocity estimate is given by Zrnic (1977) for correlated but spaced pairs

$$\mathrm{var}(\hat{\upsilon}) = \lambda^2[32\pi^2 T_s^2 \rho^2(T_s)]^{-1}\{M^{-2}[1-\rho^2(T_s)] \quad X$$
$$X \sum_{m=-(M-1)}^{M-1} \rho^2(mT)(M-|m|) + N^2/MS^2 + (2N/MS)[1+\rho(2T_s)(1/M-1)\delta_{T-T_s,0}]\}$$

where M is the number of sample pairs, and T is the spacing between pairs. If a Gaussian spectrum is assumed, this expression can be approximated by

$$\mathrm{var}(\hat{\upsilon}) \approx \lambda^2[32\pi^2 T_s^2 \rho^2(T_s)]^{-1}\{[1-\rho^2(T_s)]T_s/2\sigma_{vn}T\sqrt{\pi} + N^2/S^2 + 2(N/S)[1+\rho(2T_s)\delta_{T-T_s,0}]\}$$

For large signal-to-noise ratios, narrow spectrum widths, and contiguous pairs, this expression further reduces to

$$\mathrm{var}(\hat{v}) \approx \sigma_v \lambda / (8 M T_s \sqrt{\pi})$$

To illustrate the quality of this estimator, consider the following example based on a 3 cm radar. In a convective storm, a typical spectrum width of 5 m/s would result in a velocity estimate uncertainty of 0.047 m/s when integrating 1000 pulses at a PRF of 4400 Hz.

These equations are derived using perturbation analysis. For very narrow spectrum widths or low signal-to-noise ratios, a large number of pulses must be integrated to obtain valid results. Specifically, the following two conditions are necessary to ensure validity of the estimated variances:

$$2\pi M \sigma_{vn} >> 1,$$
$$\rho^2(T_s) M >> (N/S+1)^2$$

The first condition expresses the requirement for a large number of samples whereas the second condition ensures that the $\arg[R(T_s)]$ is small compared with 2π.

Chapter 5 - Conclusions

5.0 Status of Software Development

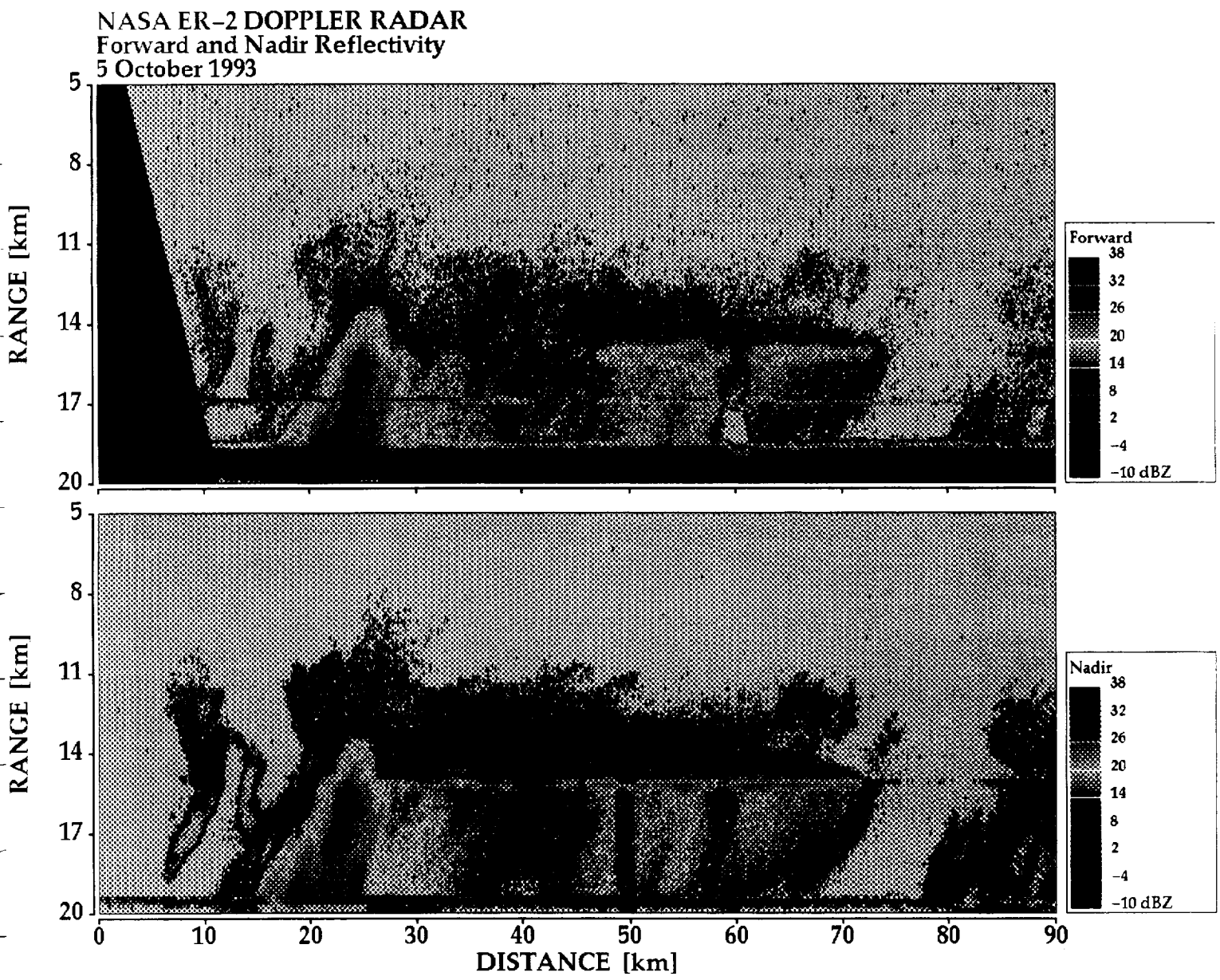
The EDOP radar made it's first successful flight during the CAMEX experiments at the Wallops Flight Facility in September 1993. In this flight, the real-time reflectivity algorithm was tested, and the inital results are encouraging (Figure 5-1). Though the real-time Doppler algorithm was not tested in flight, it has been thoroughly simulated in the lab and appears to be properly implemented. Additional flights are planned at Wallops for the summer of 1994 in which the Doppler algorithms will be tested for the first time in the operational mode.

Additional work must be completed before EDOP is fully meets the operational design goals specified in the original proposal. Perhaps the most important remaining task in completion of an automatic gain control (AGC) algorithm to increase the effective dynamic range of the system. This will be particularly important for Doppler processing because the linear coherent detectors are easily saturated.

5.1 Automatic Gain Control

One solution to this problem involves the implementation of hardware to perform sensitivity-time control and software to provide global gain control to compensate for gradual variations in the mean reflective echo.

Sensitivity time control (STC) is a method used to avoid saturation of the receiver by strong returns at short ranges without compromising receiver sensitivity at longer ranges (Figure 5-2). After each pulse is transmitted, the receiver gain, which was initially greatly reduced, is increased with time to match the decrease in amplitude with range due to spreading. Over a range of 30 km, this amounts to an increase in receiver sensitivity by roughly 70 dB. Hence, maximum sensitivity is provided to enable detection of the



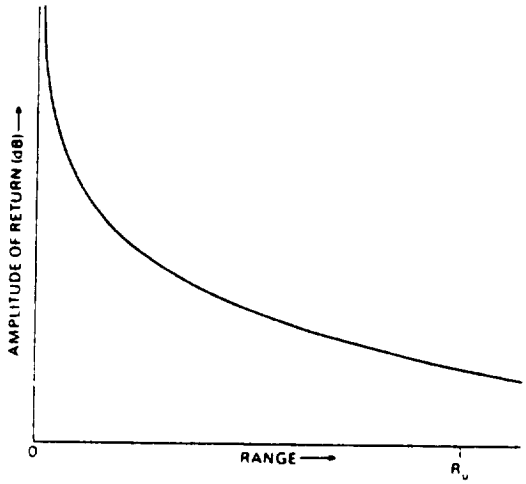
Convective clouds with a trailing stratiform region observed by the ER-2 Doppler Radar (EDOP) on 5 October 1993 during the Convection and Atmospheric Moisture Experiment (CAMEX). The top panel shows reflectivity data (dBZ) from the forward looking antenna (30° off nadir) and the bottom panel shows the corresponding reflectivity data from the nadir viewing antenna. The forward image has been transformed to the coordinate system of the nadir looking antenna.

The surface return is near 20km range and the "bright band" is near 15km. Each x-axis division is approximately 9km along the flight path. The thin line of enhanced reflectivity seen at 17km range on the forward image is an artifact that will be removed in future flights by modification of the radar hardware.

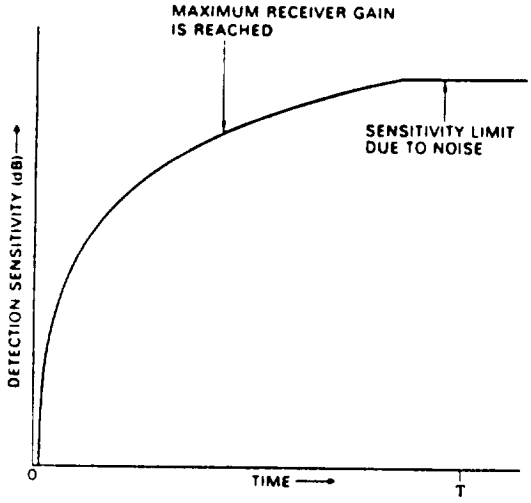
Figure 5-1: Convective clouds observed during CAMEX (courtesy Jeff Caylor

SENSITIVITY TIME CONTROL

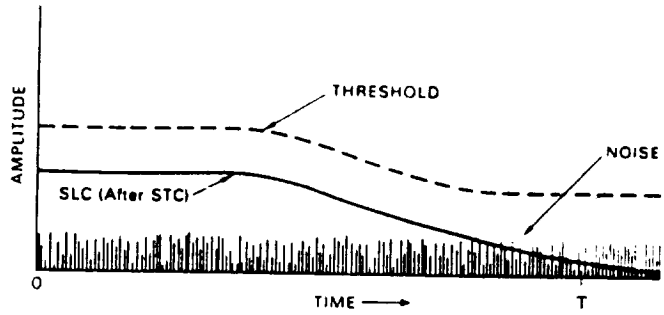
At low PRFs, saturation of the receiving system by strong return from short ranges is commonly avoided without loss of detection sensitivity at greater ranges through a feature called sensitivity time control, STC.



After each pulse has been transmitted, the system gain, which initially is greatly reduced, is increased with time to match the decrease in amplitude of the radar return with range. Maximum gain is usually reached well before the end of the interpulse period.



Thereafter, the increase in sensitivity is continued by lowering the detection threshold until the noise limit is reached—i.e., to the point where the threshold is just far enough above the mean noise level to limit the false-alarm probability to an acceptable value.



Thus, maximum sensitivity is provided at long ranges, where it is needed to detect the weak echoes of distant targets, while the strong return from short ranges is prevented from saturating the system.

STC may be applied at various points in a system. From whatever point it is applied, it helps prevent saturation in all following stages.

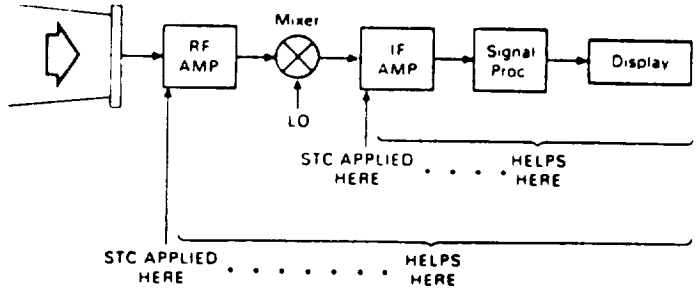


Figure 5-2: STC concept (Stimson, 1983)

weakest echoes near the surface without saturating the receiver by strong echoes at higher altitudes.

If necessary, the dynamic range of the system can be further increased by implementing digital automatic gain control (DAGC). This feature monitors the output of the A/D converters to build a continuously updated profile of the average receiver output. On the basis of the profile, embedded software produces a gain-control signal that is applied to a set of IF-attenuators ahead of the IF linear-detectors. A Kalman filter might be used to track the dc-level of the linear-detector output and center it within the dynamic range of the A/D converters.

Additional resolution in range is possible by increasing the A/D sampling rate to 4 MHz from 2 MHz and narrowing the pulse width to 0.25 μs. These changes would allow an improved resolution of 37.5 meters or could be used to increase the number of effectively independent samples through range averaging.

Additional test flights must be taken to thoroughly calibrate the radar system. These flights can be easily conducted up and down the coast of California over the ocean.

5.2 Antenna Stabilization

The Doppler velocity measured by an airborne radar is given by the dot product of the aircraft-velocity vector with the antenna-pointing vector. This product may then be related to the three-dimensional air-motion vector by transforming the aircraft body-fixed coordinate references to earth-fixed coordinates. Figure 5-3 shows the relation between the two coordinate systems. Heymsfield (1989) develops this transformation by first expressing the aircraft body-fixed coordinates as a linear transformation of the earth-fixed coordinates. It should be understood by the reader that such a transformation neglects the

2nd-order effects that result from the transformation between rotating and non-rotating reference frames. This section will instead assume that the aircraft undergoes rotation before the measurements are made.

To describe the orientation of the aircraft relative to an earth-fixed coordinate system, it is sufficient to specify a three-dimensional transformation matrix that relates the two systems. The transformation is carried out by three *consecutive* rotations about the yaw-, pitch-, and roll-axis, respectively. Emphasis is placed on the term consecutive because the order in which they are carried must be consistent; these rotations do not obey a commutative property as demonstrated through Figure 5-4.

Let the earth-fixed coordinate system be defined as the vector (x_e, y_e, z_e) and the aircraft body-fixed coordinate system be defined as the vector (x_a, y_a, z_a).

Then it is possible to specify a rotation matrix **A** such that:

$$\begin{pmatrix} x_a \\ y_a \\ z_a \end{pmatrix} = \mathbf{A} \begin{pmatrix} x_e \\ y_e \\ z_e \end{pmatrix}$$

To transform the earth-fixed coordinates to the aircraft, the following rotations are applied in order:

1) Rotate coordinate system (x1, y1, z1) about the z_1 axis over an angle ψ as demonstrated in Figure 5-5. This angle is referred to as the heading or yaw angle. Note that earth-fixed system has been renamed from (x_e, y_e, z_e) to (x_1, y_1, z_1) for the purpose of this rotation.

$$x_2 = x_1 \cos\psi + y_1 \sin\psi$$
$$y_2 = y_1 \cos\psi - x_1 \sin\psi$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos\psi & \sin\psi \\ -\sin\psi & \cos\psi \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

2) Rotate coordinate system (x_2, y_2, z_2) about the y_2 axis over an angle θ as demonstrated in Figure 5-5. This angle is referred to as the pitch angle.

$$x_3 = x_2 \cos\theta - z_2 \sin\theta$$
$$z_3 = z_2 \cos\theta + x_2 \sin\theta$$

$$\begin{bmatrix} x_3 \\ z_3 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_2 \\ z_2 \end{bmatrix}$$

3) Rotate coordinate system (x_3, y_3, z_3) about the x_3 axis over an angle ϕ as demonstrated in Figure 5-5. This angle is referred to as the bank or roll angle.

$$y = y_3 \cos\phi + z_3 \sin\phi$$
$$z = -y_3 \sin\phi + z_3 \cos\phi$$

$$\begin{bmatrix} y \\ z \end{bmatrix} = \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} y_3 \\ z_3 \end{bmatrix}$$

Therefore, the combined operation on the three-dimensional earth-fixed system appears as

$$\begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \end{bmatrix}$$

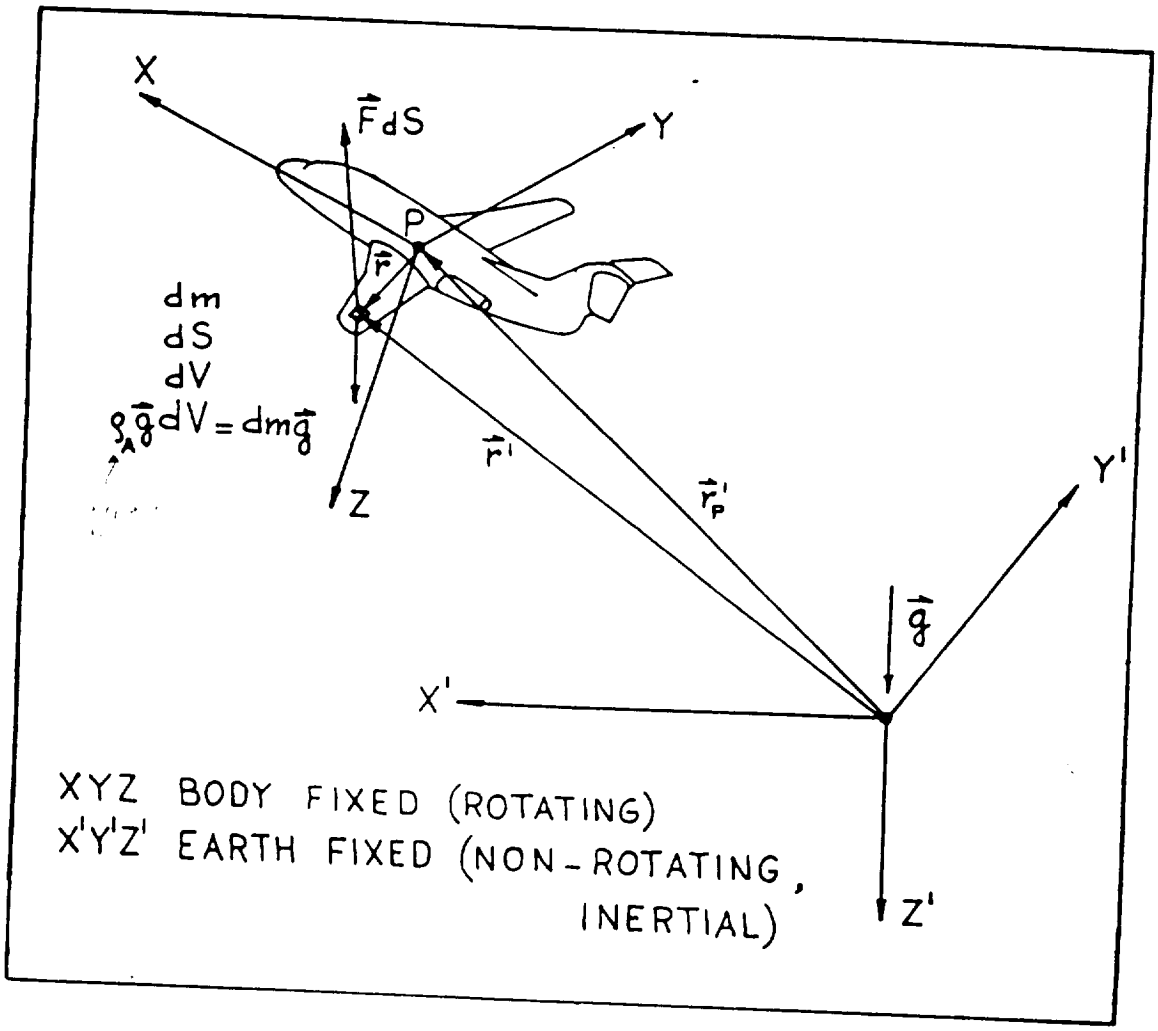


Figure 5-3: Relationship between rotating and inertial coordinate systems (Roskam, 1979)

For meteorological purposes, it is convenient to define the earth-fixed coordinate system such that x_e points east, y_e points north, and z_e point up. Likewise it is convenient to define the aircraft body-fixed system such that x_a points out the nose of the aircraft, y_a points out the right wing, and z_a points downward from the aircraft belly as depicted in Figure 5-3. Furthermore, the aircraft inertial navigation system reports the heading angle relative to North. The result of this is that, for zero angles of heading, pitch, and roll, the aircraft will be heading north. Therefore, the rotation matrices must be further modified to correctly transform the earth-fixed system to the body-fixed system.

To do this, it is necessary to rotate the earth-fixed system by -90° in addition to the heading angle, ψ. This is accomplished by replacing ψ in the above matrices by $-\psi$-90°. Next, one must pitch the earth-fixed system by 180° in addition to the pitch of the aircraft so that the z-axes of both systems will be correctly aligned. This is accomplished by replacing θ in the above matrices by θ-180°. These matrix modifications effectively align the two coordinate systems in the default state (no heading, pitch, or roll angles).

$$\begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} -\cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & -\cos\theta \end{bmatrix} \begin{bmatrix} -\sin\psi & -\cos\psi & 0 \\ \cos\psi & -\sin\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \end{bmatrix}$$

The rotations about each axis are then combined to give the rotation matrix, **A**:

$$\mathbf{A} = \begin{bmatrix} \cos\theta\sin\psi & \cos\theta\cos\psi & \sin\theta \\ \sin\theta\sin\phi\sin\psi + \cos\phi\cos\psi & \sin\theta\sin\phi\cos\psi - \sin\psi\cos\phi & -\sin\phi\cos\theta \\ \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & -\cos\phi\cos\theta \end{bmatrix}$$

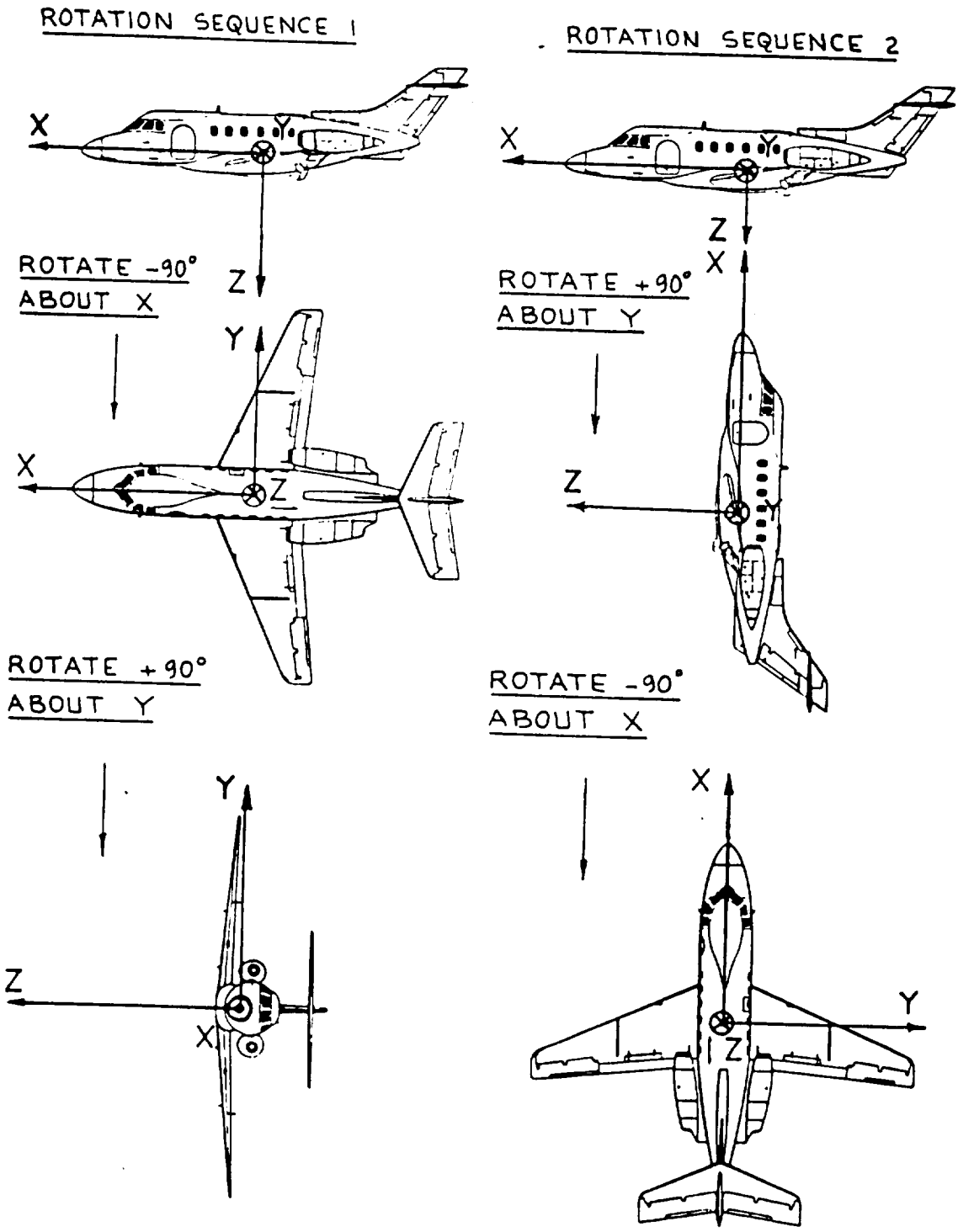


Figure 5-4: Demonstration that vector rotations are not commutative (Roskam, 1979)

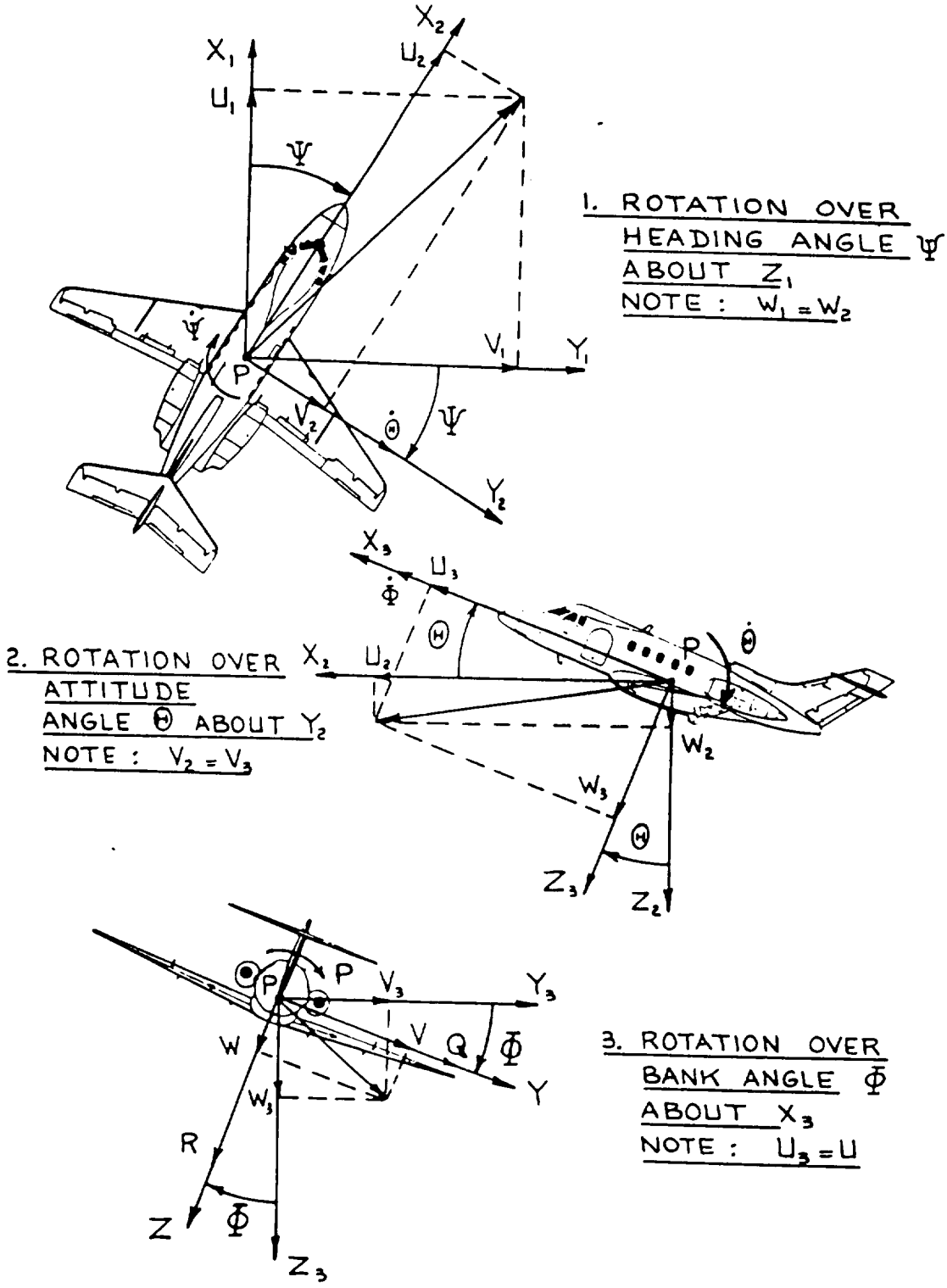


Figure 5-5: Rotations used to derive rotation matrix (Roskam, 1979)

The aircraft-motion vector can be expressed relative to the earth frame of reference by

$$\mathbf{V}_{ac} = GS\sin T\,\mathbf{x}_e + GS\cos T\,\mathbf{y}_e + w_p\mathbf{z}_e$$

while the precipitation motion is described by

$$\mathbf{V}_p = u_a\,\mathbf{x}_e + v_a\,\mathbf{y}_e + (w_a + v_t)\mathbf{z}_e$$

where

T	aircraft track angle
GS	aircraft ground speed
u_a, v_a	horizontal winds at observation altitude
w_a	air vertical velocity at observation altitude
w_p	aircraft vertical velocity
v_t	hydrometeor fallspeed at observation altitude

Here, it is assumed that the hydrometeors follow the three-dimensional air motion and fall at their terminal velocities (v_t<0). The radial velocity between the radar and precipitation is then given by

$$\mathbf{V}_r = (\mathbf{V}_p - \mathbf{V}_{ac})\cdot\mathbf{e}_r$$

For the case of a nadir-pointing antenna, $\mathbf{e}_r = \mathbf{z}_a$, reducing the Doppler velocity to

$$\begin{aligned} v_r = {} & -w_a\cos P\cos R - GS(\sin P\cos D + \cos P\sin R\sin D) \quad + \\ & + w_p\cos P\cos R - v_t\cos P\cos R + (u_a\sin H + v_a\cos H)\sin P + (u_a\cos H - v_a\sin H)\cos P\sin R \end{aligned}$$

Clearly, these terms result from the various vertical and horizontal air and aircraft motions. The effect of each term on the measured Doppler velocity is best illustrated by labeling and considering each term separately

Vertical Air Motion:	$-w_a \cos P \cos R$
Horizontal Aircraft Motion:	$-GS(\sin P \cos D + \cos P \sin R \sin D)$
Vertical Aircraft Motion:	$+w_p \cos P \cos R$
Fallspeed:	$-v_t \cos P \cos R$
Horizontal Air Motion:	$+(u_a \sin H + v_a \cos H)\sin P + (u_a \cos H - v_a \sin H)\cos P \sin R$

These variables can be determined both directly and indirectly from the information provided by the aircraft Inertial Navigation System. Consider first, a case whereby the antenna is fixed rigidly to the aircraft frame. It is desired to extract w_a from the Doppler velocity. Though the Doppler velocity will be contaminated by terms related to the horizontal and vertical aircraft motion, these terms can be accounted for using information from the INS. The hydrometeor fallspeed can additionally be removed by estimating the fallspeed from an empirical relation of radar reflectivity to fallspeed. However, the motion of the horizontal winds is unknown for any observed altitude below the aircraft. Therefore, there is an error in the estimated w_a which can be significant for strong horizontal winds.

If the antenna is instead stabilized at nadir, these errors can be removed. In this case,

$P \approx 0,$
$R \approx 0$

and the measured Doppler velocity reduces to

$$v_r = -(w_a + v_t) + w_p$$

Again, the hydrometeor fallspeed can be removed using an empirical relation. If the aircraft vertical speed, w_p, can be obtained from the inertial navigation system, the mean precipitation vertical velocity can be extracted from the measured Doppler velocity.

The inertial navigation system on the U-2 provides angle measurements that are accurate to within 15 seconds of arc or ¼°. Neglecting the uncertainties in the radial velocity estimate as described in §4.2.3, the uncertain angle measurements affect the measured Doppler velocity in the following way:

Table 5-1: Typical Flight Parameters

Variable	Typical Value	Uncertainty
P	1°	0.06°
R	1°	0.06°
H	5°	0.1°
T	10°	0.2°
D	5°	0.3°
GS	200°	2.0 ms^{-1}
w_a	0 ms^{-1}	0.5 ms^{-1}
u_a	0 ms^{-1}	
v_a	20 ms^{-1}	
w_a	0 ms^{-1}	

Table 5-2: Motion Contributions to Doppler Velocity

Doppler Velocity Term	Calculated Contribution	Calculated Uncertainty
Vertical Air Motion	5.0 ms^{-1}	0 ms^{-1}
Horizontal Aircraft Motion	2.28 ms^{-1}	0.12 ms^{-1}
Vertical Aircraft Motion	0	0.5 ms^{-1}
Fallspeed		
Horizontal Air Motion 1	0.03 ms^{-1}	0.0 ms^{-1}
Horizontal Air Motion 2	0.35 ms^{-1}	0.0 ms^{-1}
Total	7.66 ms^{-1}	0.62 ms^{-1}

Here, the fallspeed contribution to Doppler velocity and its uncertainty have been neglected. If the aircraft-motion contributions are subtracted from the measured Doppler velocity, the resulting vertical air-motion estimate is 5.38 ms^{-1}. It should be evident that the horizontal air-motion term is the significant contributor to this bias. As stated previously, because the horizontal winds at the observation altitude are almost never known, the only accurate way to remove this bias is through stabilization of the nadir antenna.

References

Atlas, David, J. Eckerman, R. Meneghini, and R. K. Moore, 1982, "The Outlook for Precipitation Measurements from Space," *Canadian Meteorological and Oceanographic Society,* **20:1**, 50-61.

Bretherton, Francis P., 1985, "Earth System Science and Remote Sensing," *Proceedings of the IEEE*, **73:6**, 1118-1127.

Bridges, J. E., and J. R. Feldman, 1966, "A Radar Attenuation-Reflectivity Technique for the Remote Measurement of Drop-Size Distributions of Rain," *Journal of Applied Meteorology*, **5**, 349-357.

Doviak, R. J., D. Sirmans, D. Zrnic, and G. B. Walker, 1976, "Resolution of Pulse-Doppler Radar Range and Velocity Ambiguities in Severe Storms," Preprints, *17th International Radar Meteorology Conference*, American Meteorological Society, Seattle, WA, 15-22.

Doviak, R. J., D. Sirmans, D. Zrnic, and G. B. Walker, 1978, "Considerations for Pulse-Doppler Radar Observations of Severe Thunderstorms," *Journal of Applied Meteorology*, **17:2**, 189-205.

Doviak, R. J., and D. S. Zrnic, 1993, "Doppler radar and Weather Observations," Academic Press, New York, 562pp.

Heymsfield, Gerry M., 1989, "Accuracy of Vertical Air Motions from Nadir-Viewing Doppler Airborne Radars," *Journal of Atmospheric Sciences*, **6:6**.

Heymsfield, Gerry M., C. Parsons, L. T. Dod, and L. Miller, 1989, "Planned ER-2 Doppler radar (EDOP) for studying convective storms and mesoscale phenomena," Preprints, *24th Conference on Radar Meteorology*, Tallahassee, American Meteorological Society, 581-584.

Heymsfield, Gerry M., W. Boncyk, S. Bidwell, D. Vandemark, S. Ameen, S. Nicholson, and L. Miller, 1993, "Status of the NASA/EDOP Airborne Radar System," Preprints, *26th International Conference on Radar Meteorology*, Norman, American Meteorological Society, 374-375.

Heymsfield, Gerry M., 1989??, "ER-2 Doppler Radar (EDOP) for Studying Convective Storms and Mesoscale Phenomena," Proposal submitted to NASA Headquarters, NASA Goddard Space Flight Center.

Hildebrand, Peter H., and Jacques Testud, 1992, "ELDORA/ASTRAEA Capabilities for TOGA COARE," *Specialty Meeting on Airborne Radars and Lidars*, Toulouse, France.

Hildebrand, Peter H., and R. K. Moore, 1992, "Meteorological Radar Observations from Mobile Platforms," pp. 287-314 in Atlas, D. (ed.), Radar in Meteorology, Boston: American Meteorological Society.

Holton, James, R., 1993, "An Introduction to Dynamic Meteorology," *Academic Press*, Chapter 2.

Im, Eastwood, S. Durden, and F. Li, 1992, "NASA/JPL Airborne Rain Mapping Radar Development Status and 2/92 Rain Observations in Los Angeles," *Preprint, Proceedings of the International Workshop on the Processing and Utilization of the Rainfall Data Measured in Space*, Tokyo, Japan.

Jameson, A. R., and D. B. Johnson, 1990, "Cloud Microphysics and Radar," pp. 323-338 in Atlas, D. (ed.), Radar in Meteorology, Boston: American Meteorological Society.

Jorgensen, David P., and Robert Meneghini, 1990, "Airborne/Spaceborned Radar: Panel Report," pp. 321-322 in Atlas, D. (ed.), Radar in Meteorology, Boston: American Meteorological Society.

Lan, Chuan-Tau, and Jan Roskam, 1980, "Airplane Aerodynamics and Performance," *Roskam Aviation and Engineering Corporation*, Chapter 8, Ottawa, KS.

Lease, Steven A., 1990, "Mapping Signal Processing Algorithms to the LMAP Parallel Processor," *Tech Memo*, Martin Marietta, Baltimore, MD.

Marshall, J. S., and Walter Hitschfeld, 1953, "Interpretations of the fluctuating echo from randomly distributed scatterers," *Canadian Journal of Physics*, **31**, 962-994.

NASA, 1987, "Laser Atmospheric Wind Sounder (LAWS) Instrument Panel Report," *Earth Observing System Instrument Panel Report, IIg*.

NASA, 1990, "The Earth Observing System (EOS): A Mission to Planet Earth," EOS Program Office, NASA Headquarters, Washington D.C.

NASA, "Tropical Rainfall Measuring Mission (TRMM)," Office for Interdisciplinary Earth Studies University Corporation for Atmospheric Research, Boulder CO.

Okamoto, K., S. Yoshikado, H. Masuko, T. Ojima, N. Fugono, K. Nakamura, J. awaka, and H. Inomata, 1982, "Airborne Microwave Rain-scatterometer/radiometer," *International Journal of Remote Sensing*, **3:3**, 277-294.

Pazmany, A. L., J. Galloway, I. Popstafniga, R. E. McIntosh, R. Kelly, and G. Vali, 1993, "A Three Millimeter Airborne Radar for High Resolution Polarimetric Cloud Measurements," Preprints, 26th International Conference on Radar Meteorology, Norman, OK, 376-380.

"A Proposed Spaceborne Cloud Radar System," 1993, prepared under the auspices of the *UK GEWEX Forum*, A contribution to The World Climate Research Programme's Global Energy and Water Cycle Experiment (GEWEX).

Pulse Technology, 1989, "Technical Proposal (for the EDOP radar)," Solicitation # RFP5-27183/204, NASA Goddard Space Flight Center, Greenbelt, MD.

Racette, Paul, G. M. Heymsfield, R. Meneghini, and L. Miller, 1993, "Design of a 94 GHz Airborne Cloud Radar System," *Preprints, 26th International Conference on Radar Meteorology*, Norman, Oklahoma, American Meteorological Society.

Rogers, R. R., 1970, "The Effect of Variable Target Reflectivity on Weather Radar Measurements," *Quarterly Journal of Royal Meteorology*, **97**, 154-167.

Roskam, Jan, 1979, "Airplane Flight Dynamics and Automatic Flight Controls," *Roskam Aviation and Engineering Corporation*, Chapter 2, Ottawa, KS.

Rummler, W. D., 1968, "Introduction of a New Estimator for Velocity Parameters," *Tech Memo, MM-68-4121-5*, Bell Telephone Labs, Whippany, NJ.

Sauvegeot,

Schols, J. L., and J. A. Weinman, 1992, "Retrieval of Hydrometeor Distributions over the Ocean from Airborne Single-frequency Radar and Multi-Frequency Radiometric Measurements," *11th International Conference on Clouds and Precipitation*, Montreal, Canada.

Simpson, Joanne (ed), 1988, "Report of the Science Steering Group for a Tropical Rainfall Measuring Mission (TRMM)," Earth Science and Applications Division, Office of Space Science and Applications, NASA Goddard Space Flight Center, Greenbelt, MD.

Simpson, J., R. F. Adler, and G. North, 1988, "A Proposed Tropical Rainfall Measuring Mission (TRMM) satelllite," *Bulletin of the American Meteorological Society*, **69**, 278-295.

Smith, Paul L., 1985, "Fundamentals of Weather Radar," South Dakota School of Mines and Technology.

Smith, Paul L., 1986, "On the Sensitivity of Weather Radars," *Journal of Atmospheric and Oceanic Technology*, **3**, 704-713.

Srivastava, R. C., A. R. Jameson, and P. H. Hildebrand, 1979, "Time-domain Computation of Mean and Variance of Doppler Spectra," *Journal of Applied Meteorology*, **18**, 189-194.

Stimson, George W., 1983, "Introduction to Airborne Radar," *Hughes Aircraft Corporation*, Chapter 25, El Segundo, CA.

Ulaby, Fawaz T., Richard K. Moore, and Adrian K. Fung, 1981, "Microwave Remote Sensing - Active and Passive, Volume I," *Artech House*, Chapter 5.

Ulaby, Fawaz T., Richard K. Moore, and Adrian K. Fung, 1981, "Microwave Remote Sensing - Active and Passive, Volume II," *Artech House*, Chapter 7.

Walker, Gene B., P. S. Ray, D. Zrnic, and R. Doviak, 1980, "Time, Angle, and Range Averaging of Radar Echoes from Distributed Targets," *Journal of Applied Meteorology*, **19:3**, 315-323.

Zrnic, D. S., 1975, "Moments of estimated input power for finite sample averages of radar receiverr outputs," *IEEE Transactions on Aerospace and Electronic Systems*, **AES-11**, 109-113.

Zrnic, D. S., 1975, "Signal-to-noise ratio at the output of nonlinear devices," *IEEE Transactions on Information Theory*, **IT-21**, 662-663.

Zrnic, D. S., 1975, "Simulation of Weatherlike Doppler Spectra and Signals," *Journal of Applied Meteorology*, **14**, 619-620.

Appendix A - RIB Code

```
/*
 * File   :    ribcwlox.s
 *
 * Purpose : This program acquires temperature information
 *           each pulse and sends it to the VME queues.
 *           After every 1000 pulses, the transmitter is
 *           switched to calibrate mode.
 *
 * Calls  :
 *
 * Author :  Shaun R. Nicholson
 *           Center For Research, Inc.
 *           University of Kansas
 *           (913) 864-4835
 *
 * Registers: r1  => 0x610000      ; Board Status
 *            r2  => 0x630000      ; Radar Status
 *            r3  => counter       ; calibration counter
 *            r4  => misc          ; Status register addr's
 *            r5  => Attenuate     ; Calibration Attenuator table
 *            r6  => misc
 *            r7  =>
 *            r8  =>
 *            r9  =>
 *            r10 => BCR           ; Board Control Register
 *            r11 => 0x264000      ; VME A output queue
 *            r12 => Num_Pulses    ; # pulses/dwell - 2
 *            r13 => Dwell_cnt      ; Current Dwell #
 *            r14 => One           ; 1.0
 *            r15 => misc          ; Pulse #
 *            r16 => timer         ; for standby status
 *            r17 => timer         ; for standby status
 *            r18 => misc
 *            r19 => misc          ; Attenuator table offset
 *            r20 => misc          ; misc status register
 *            r21 =>
 *            r22 => Service       ; Interrupt Service Routine
 *
 *
 *
 * Revision
 * History :
 *
 *    (0.x = Beta)
 *    Revision:     By:         Date:        Description:
```

```
 *    0.0          S.R. Nicholson  07-28-93      Original.
 *
 */

#include "d:\shaun\dsp32c\bin\zone_addr.h"
#include "d:\shaun\dsp32c\bin\rib_addr.h"

.global main,Start,Serv_chk,Wait_rad,wait_loop,Wait_int,Service
.global done,Dwell_cnt,One,temp,Num_Pulses,Cal_flag,Attenuate

.rsect ".text"
main:
/*
 * (1) Disable Interrupt 1 (Pulse transmitted)
 * (2) Disable DMA transfers
 * (3) Set external memory partition A to 0 wait states
 * (4) Set external memory partition B to 2+ wait states
 */
          r1 = 0x000d        /* No interrupts till trigger starts */
          nop
          pcw = r1           /* interrupts and memory wait states */
          ioc = 0x0          /* DMA */
          r22e = Service     /* Interrupt service routine */
          goto Start
          nop

.rsect ".R0"
/* Initialize pointers */

Start:     r1e = BSR              /* Board status register */
          r2e = RSR               /* Radar status register */
          r3e = 660001            /* #pulses + 1 BETWEEN calibrate cycles */
          r11e = 0x264000         /* VME queue B */
          r12e = Num_Pulses
          r13e = Dwell_cnt        /* Dwell # */
          r14e = One              /* Start at Dwell #1 */
          r15 = *r12              /* # pulses per dwell - 2 */
          r16 = 32018             /* standby status timer */
          r17 = 937               /* timer */

/*
 *
 * Now, wait for the pilot to flip the operation switch from
 * STANDBY to RADIATE.  While in STANDBY mode, status will be
 * recorded approximately once every 30 sec.
```

```
 *
 */
Wait_rad:     r18 = *r2              /* read RSR */
              nop
              r18 = *r2              /* must read twice */
              nop
              r6 = r18
              r6 = r6 & 0x0200       /* check STDBY_RAD bit */
              if(ne) pcgoto Begin_trig   /* bit lo - standby */
              nop
              if(r16-->=0) pcgoto Wait_rad      /* millisecond timer */
              nop
              if(r17-->=0) pcgoto Wait_rad      /* time for status? */
              r16 = 32018            /* regardless, reload inner timer */
              call Status (r8)       /* collect status */
              nop
              r17 = 937              /* reload outer timer */
              goto Wait_rad
              nop

/*
 * The pilot has flipped the switch to the RADIATE position.
 * The trigger will now be turned on...
 *
 */

Begin_trig:   r10e = BCR             /* Board Control register */
              r6 = 0x00a3            /* the radiate signal */
              *r10 = r6              /* has been captured - */
              r6 = 0x80a3            /* start trigger now */
              *r10 = r6
              r6 = 0x00a3
              *r10 = r6              /* trigger started */
              r18 = 0x800d           /* Now start interrupts! */
              pcw = r18              /* interrupts and memory wait states */
/*
 * Once the trigger is provided, the A/D's will begin dumping samples
 * into their queues.  The E-6 will in turn start dumping results
 * to the hard drive for storage.  At this point, the program simply
 * waits for interrupts and checks the fault registers for an
 * indication of trouble or transmitter turnoff.  An interrupt will
 * occur with the transmission of each pulse - at the end of each
 * dwell, sensor and status information are sent to the VME queue B
 * for storage on physical media.
 */
```

```
Wait_int:  r18 = *r2                  /* read RSR */
           nop
           r18 = *r2                  /* must read twice */
           nop
           r6 = r18
           r6 = r6 & 0x0200           /* check STDBY_RAD bit */
           if(ne) pcgoto Wait_int     /* still transmitting... */
           nop

           call Timer (r8)            /* wait a bit */
           nop

           r18 = 0x000d               /* stop interrupts */
           pcw = r18

           r18 = *r2                  /* read RSR */
           nop
           r18 = *r2                  /* must read twice */
           nop
           r6 = r18
           r6 = r6 & 0x0200           /* check STDBY_RAD bit */

           if(eq) pcgoto terminate    /* transmitting */
           nop
           r18 = 0x800d               /* restart interrupts */
           pcw = r18

           goto Wait_int
           nop

/* If FAULT2 (hard fault) was asserted, the transmitter has failed during
 * both the main start and attempted restart.  In this case, (or if
 * the transmitter was turned off by the pilot), the GO/STOP bit should
 * be set to one thereby indicating that the E-6 should close its
 * data files and terminate.
 */
terminate:  r18 = 0x4003              /* Signal E-6 to terminate */
           *r10 = r18

NextLife:   pcgoto NextLife
           nop

/* SUBROUTINE: STATUS
 *
```

```
* Description: This subroutine is used to acquire status information
* (temperature and pressure) about the radar transmitter and receiver.
* These data are collected periodicly and sent to the hard drive via
* VME bus B.
*/

Status:     r14e = temp
            r20 = *r1               /* Board Status register */
            nop
            *r14 = r20
            a0 = (*r11=*r14) + a0

            r20 = *r2               /* Radar Status register */
            nop
            r20 = *r2               /* must read twice */
            nop
            *r14 = r20
            a0 = (*r11=*r14) + a0

            r4e = ADCDAC0
            r20 = *r4               /* temp 1 */
            nop
            r20 = *r4               /* must read twice */
            nop
            *r14 = r20
            a0 = (*r11=*r14) + a0

            r4e = ADCDAC1
            r20 = *r4               /* temp 2 */
            nop
            r20 = *r4               /* must read twice */
            nop
            *r14 = r20
            a0 = (*r11=*r14) + a0

            r4e = ADCDAC2
            r20 = *r4               /* temp 3 */
            nop
            r20 = *r4               /* must read twice */
            nop
            *r14 = r20
            a0 = (*r11=*r14) + a0

            r4e = ADCDAC3
            r20 = *r4               /* temp 4 */
```

```
nop
r20 = *r4                    /* must read twice */
nop
*r14 = r20
a0 = (*r11=*r14) + a0

r4e = ADCDAC4
r20 = *r4                    /* temp 5 */
nop
r20 = *r4                    /* must read twice */
nop
*r14 = r20
a0 = (*r11=*r14) + a0

r4e = ADCDAC5
r20 = *r4                    /* temp 6 */
nop
r20 = *r4                    /* must read twice */
nop
*r14 = r20
a0 = (*r11=*r14) + a0

r4e = ADCDAC6
r20 = *r4                    /* pressure 1 */
nop
r20 = *r4                    /* must read twice */
nop
*r14 = r20
a0 = (*r11=*r14) + a0

r4e = ADCDAC7
r20 = *r4                    /* pressure 2 */
nop
r20 = *r4                    /* must read twice */
nop
*r14 = r20
a0 = (*r11=*r14) + a0

r4e = ADCDAC8
r20 = *r4                    /* rib temp */
nop
r20 = *r4                    /* must read twice */
nop
*r14 = r20
a0 = (*r11=*r14) + a0
```

```
            r4e = One
            a0 = (*r11=*r13) + a0          /* Dwell # */

            return (r8)
            nop


/* SUBROUTINE:  TIMER
 *
 * Description: This subroutine provides a 10 second time delay.
 *
 */

Timer:      r6e = 10000000          /* about 10 seconds */
                                    /* (1 microsecond/loop) */
wait_loop:  r6e = r6 - 1            /* tick tock, tick tock... */
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            if(ne) pcgoto wait_loop /* done? */
            nop
            return (r8)
            nop


/*
 * This is the interrupt service routine.  When an interrupt occurs,
 * the pulse # counter is decremented.  Once it goes to zero (ie.
 * at the end of each dwell), status information is transferred to
 * the VME queue for storage and the counter reset.
 */

Service:    r20 = 0x000d            /* Mask interrupts */
            nop
            r4e = Cal_chk
            pcw = r20

            if(r15-->=0) goto r4    /* no status info this time */
            nop
```

```
/* Found last pulse this dwell - record status information */
Serv_chk:     r14e = temp
              r20 = *r1                    /* Board Status register */
              nop
              *r14 = r20
              a0 = (*r11=*r14) + a0

              r20 = *r2                    /* Radar Status register */
              nop
              r20 = *r2                    /* must read twice */
              nop
              *r14 = r20
              a0 = (*r11=*r14) + a0

              r4e = ADCDAC0
              r20 = *r4                    /* temp 1 */
              nop
              r20 = *r4                    /* must read twice */
              nop
              *r14 = r20
              a0 = (*r11=*r14) + a0

              r4e = ADCDAC1
              r20 = *r4                    /* temp 2 */
              nop
              r20 = *r4                    /* must read twice */
              nop
              *r14 = r20
              a0 = (*r11=*r14) + a0

              r4e = ADCDAC2
              r20 = *r4                    /* temp 3 */
              nop
              r20 = *r4                    /* must read twice */
              nop
              *r14 = r20
              a0 = (*r11=*r14) + a0

              r4e = ADCDAC3
              r20 = *r4                    /* temp 4 */
              nop
              r20 = *r4                    /* must read twice */
              nop
              *r14 = r20
              a0 = (*r11=*r14) + a0
```

```
r4e = ADCDAC4
r20 = *r4                    /* temp 5 */
nop
r20 = *r4                    /* must read twice */
nop
*r14 = r20
a0 = (*r11=*r14) + a0

r4e = ADCDAC5
r20 = *r4                    /* temp 6 */
nop
r20 = *r4                    /* must read twice */
nop
*r14 = r20
a0 = (*r11=*r14) + a0

r4e = ADCDAC6
r20 = *r4                    /* pressure 1 */
nop
r20 = *r4                    /* must read twice */
nop
*r14 = r20
a0 = (*r11=*r14) + a0

r4e = ADCDAC7
r20 = *r4                    /* pressure 2 */
nop
r20 = *r4                    /* must read twice */
nop
*r14 = r20
a0 = (*r11=*r14) + a0

r4e = ADCDAC8
r20 = *r4                    /* rib temp */
nop
r20 = *r4                    /* must read twice */
nop
*r14 = r20
a0 = (*r11=*r14) + a0

a0 = (*r11=*r13) + a0        /* Dwell # */
r14e = *r13
nop
r14e = r14 + 1               /* Increment Dwell # */
```

```
            *r13 = r14e
            r15 = *r12                  /* reset pulse counter */

Cal_chk:    r4e = Cal_flag              /* are we now calibrating? */
            r20 = *r4
            nop
            if(ne) pcgoto Calibrate     /* yup! */
            nop

            r3e = r3 - 1                /* time to begin calibration? */
            if(ne) pcgoto done          /* not yet! */
            nop

Cal_Start:  r4e = RCR0                  /* Radar Control reg. */
            r20 = 0x16                  /* 16=2200 Hz, 12=4400 Hz */
            *r4 = r20                   /* turn on calibrate mode */
            r19 = 0                     /* offset into table */
            r4e = Cal_flag
            *r4 = r20                   /* set flag */

Calibrate:  r5e = Attenuate             /* Table base address */
            r4e = RCR1                  /* Radar Control reg. */
            r5e = r5 + r19              /* add offset */
            r20 = *r5                   /* get desired setting */
            nop
            if(eq) pcgoto nomore        /* end of table? */
            nop
            *r4 = r20                   /* nope, set attenuators! */
            r19 = r19 + 2               /* point to next entry */

/* finished with current calibration dwell */
cal_done:   goto done
            nop

/* finished with all calibration dwells, return to normal processing */
nomore:     r4e = Cal_flag
            *r4 = r20                   /* clear calibrate flag */
            r4e = RCR0                  /* Radar Control reg. */
            r20 = 0x1e                  /* end calibrate mode */
            *r4 = r20                   /* 1e=2200 Hz, 1a=4400 Hz */
            r3e = 660001                /* #pulses + 1 BETWEEN cal cycles */


done:       if(ireq1_lo) pcgoto done    /* wait till irq returns */
```

```
            nop                     /* high before exiting */
            r20 = 0x800d            /* restore interrupts */
            pcw = r20
            r13e = Dwell_cnt        /* restore pointer */
            r14e = One              /* restore pointer */
            ireturn
            nop

.rsect ".text"
.align 4
Dwell_cnt:    fltbits 0x0           /* current Dwell # */
One:          float 1.0
temp:         fltbits 0x0
Num_Pulses:   int 6598              /* # xmit pulses/status - 2 */
Cal_flag:     int 0x0               /* Calibration dwell flag */
Junk_variable:  int 0xface

.align 4
Attenuate:    220*int 0x0fc0
              220*int 0x0fc1
              220*int 0x0fc2
              220*int 0x0fc4
              220*int 0x0fc8
              220*int 0x0fd0
              220*int 0x0fe0
              220*int 0x0fc0
              int 0x0000            /* end of table */
```

Appendix B - ACQ Code

```
/*
 * File   :    acqreal.s
 * Purpose :   This is the data scattering program to be flown
 *             during the Wallops Island experiments during
 *             September 1993.  For each transmitted pulse,
 *             436 tags are supplied to route the raw data to
 *             LUA Board 1 for reflectivity processing.  It
 *             is intended that the system operate at a PRF of
 *             2200 Hz, 0.25 usec pulse width, and 1 MHz sampling.
 *
 * Author :    S.R. Nicholson
 * Date   :    8/10/93
 */

#include "d:\shaun\dsp32c\bin\acq_addr.h"

.rsect ".text"
.align 4
main:
/*
 * (1) Enable interrupt one from dataqaffn
 * (2) Disable DMA transfers
 * (2) Set external memory partition A to 0 wait states
 * (3) Set external memory partition B to 2+ wait states
 */

                ioc = 0x0               /* DMA */
                goto Start
                nop

.rsect ".R0"
.align 4
Start:          r1e = ADDQAB            /* VME output queue's */
                r3e = Range_Gates       /* # range_gates */
                r5e = ADCTL0            /* to control A/D's */
                r6 = 0x000d             /* mask Int_1 */
                r7 = 0x800d             /* enable Int_1 */
                r11e = Int_flag
                r13 = 0                 /* A/D control bit */
                                        /* 0 = ARM, 1 = Stop */
                r14 = 0x0               /* Always! */
                r16 = 998               /* #good - 2 before token pass */
                r21 = 1                 /* Always! */
                r22e = Intsvc           /* load address of qempty svc routine */
                pcw = r7
```


C.2

```
Send_Good:  call Write_Tags (r8)
            nop

Monitor:    r20 = *r11                  /* wait for interrupt */
            nop
            if(eq) pcgoto Monitor
            nop
            *r11 = r14                  /* clear interrupt flag */

            if(r16-->=0) pcgoto Send_Good /* time to pass token? */
            nop

            r16 = 998                   /* reload pulse counter */
            r8e = First_Time            /* don't pass the first time */
            r20 = *r8
            nop
            if(eq) pcgoto Send_Good
            *r8 = r21                   /* set flag regardless */

Pass_Token: r2e = Pass_Cmd
            a0 = (*r1=*r2) + a0         /* pass token to LUA */

            goto Send_Good
            nop

/* subroutine - write valid tags for 1 pulse */
Write_Tags:
            r18 = *r3                   /* #ranges */
            r2e = Keep_Tag
Write_loop: do 0,r18
            a0 = (*r1=*r2++) + a0       /* Tag 1 */
            nop
            *r11 = r14                  /* clear Int_flag */
            return (r8)
            nop

/* Service routine for external interrupt one:  this interrupt is
 * generated each time the pre-trig goes low.  During the interrupt,
 * the Trigger Source bit in ADCTL0 is pulled low, thereby arming
 * the A/D's to begin conversions once the pre-trig returns high.
 * The DSP then burst writes destination tags for each range gate
 * to be acquired, increments a dwell counter, and reloads a
 * range-gate count down counter that is used in interrupt 6.
```

```
 */
Intsvc:     pcw = r6                    /* mask Interrupt 1 */
            *r5 = r13                   /* reload A/D control bit */
            *r11 = r21                  /* set Int_flag */
Exit_1:     if(ireq1_lo) pcgoto Exit_1
            nop
            pcw = r7                    /* re-enable interrupt 1 */
            ireturn



.rsect ".text"
.align 4
Range_Gates: int 217                    /* # ranges - 1 */
Int_flag:   int 0x0                     /* set high in Int. 1 */
First_Time: int 0x0                     /* first dwell flag */

.rsect ".R1"
/*  Format   : Data_B : Data_A : Dest_B : Dest_A: */
.align 4
Pass_Cmd:   fltbits 0x00001f1f
Keep_Tag:   10*fltbits 0x00000010
            10*fltbits 0x00000011
            10*fltbits 0x00000012
            10*fltbits 0x00000013
            200*fltbits 0x00000000
```

Appendix C - Reflectivity Code

```
/*
 * File   :    refzone0.s
 *
 * Purpose :   This program is used for speed testing of the
 *             reflectivity algorithm.  It is nearly identical
 *             to reflect.s.
 * Calls  :
 *
 * Author :    Shaun R. Nicholson
 *             Center For Research, Inc.
 *             University of Kansas
 *             (913) 864-4835
 *
 * Registers:  r1  => QDATA            ; DSP Queue A
 *             r2  => QDATB            ; DSP Queue B
 *             r3  => DESTA            ; Bus A Destination Register
 *             r4  => DESTB            ; Bus B Destination Register
 *             r5  => NadirXMean       ; Nadir X-Pol Echo Power Means
 *             r6  => NadirCoMean      ; Nadir Co-Pol Echo Power Means
 *             r7  => FwdCoMean        ; Forward Co-Pol Echo Power Means
 *             r8  => Range_Gates      ; Addr containing # range gates
 *             r9  =>
 *             r10 => Nadir_Hold       ; Input Buffer for Nadir Channel
 *             r11 => Fwd_Hold         ; Input Buffer for Forward Channel
 *             r12 => misc
 *             r13 => misc
 *             r14 => misc
 *             r15 => misc             ; Used to output results
 *             r16 => misc             ; Used for results dest. tags
 *             r17 => misc
 *             r18 => Range count      ; # range gates
 *             r19 =>
 *             r20 => Pulse count      ; # pulses/dwell
 *             r21 =>
 *             r22 =>
 *
 *
 *
 * Revision
 * History :
 *
 *    (0.x = Beta)
 *    Revision:      By:              Date:       Description:
 *    0.0            S.R. Nicholson   07-24-92    Original.
 *
```

```
 */

/* #include "d:\shaun\dsp32c\bin\zone_addr.h" */


.global main,Start,Next_Dwell,Next_Block,Next_Pulse,Compress,Output
.global NadirXMean,NadirCoMean,FwdCoMean
.global Count2,Num_Block,Num_Pulses,Range_Gates,Comp_Factor
.global Nadir_Hold,Fwd_Hold,PassToken
.global QDATA,QDATB,DESTA,DESTB

.rsect ".text"
main:
/*
 * (1) Disable interrupts
 * (2) Disable DMA transfers
 * (2) Set external memory partition A to 0 wait states
 * (3) Set external memory partition B to 2+ wait states
 */
        r1 = 0x000d
        pcw = r1          /* interrupts and memory wait states */
        ioc = 0x0         /* DMA */
        goto Start
        nop

.rsect ".R0"
/* Initialize pointers */
Start:      r1e = QDATA       /* Bus A DSP Queue */
            r2e = QDATB       /* Bus B DSP Queue */
            r3e = DESTA       /* Bus A Destination Register */
            r4e = DESTB       /* Bus B Destination Register */
            r8e = Range_Gates /* Addr containing # range gates */
Next_Dwell:  r14e = Dwell_cnt /* Update Dwell count */
            r15 = *r14
            nop
            r15 = r15 + 1     /* increment count */
            *r14 = r15        /* and save... */
            r14e = Range_cnt  /* next, zero out */
            r15 = 0           /* the range count */
            *r14 = r15
            r5e = NadirXMean
            r6e = NadirCoMean
            r7e = FwdCoMean
/*
 * Clear arrays used for holding partial sums and resultant means
```

```
 * for each range gate.
 */
                r13e = Zero
                r18 = *r8
                nop
                do 2,r18                        /* N range gates */
                  a0 = (*r5++=*r13) + a0        /* Clear NadirXMean array */
                  a0 = (*r6++=*r13) + a0        /* Clear NadirCoMean array */
                  a0 = (*r7++=*r13) + a0        /* Clear FwdCoMean array */

/*
 * Now, read a pulse of reflectivity data from the input FIFOs and
 * store in separate buffers. The data for an entire dwell will be
 * read pulse by pulse.
 *
 * This program expects that each word read from FIFO A contains
 * a single range gate from the Nadir Cross-polarization receiver
 * in the upper 16 bits, and a single gate from the Nadir Co-polarization
 * receiver in the lower 16 bits. Each word read from FIFO B is
 * expected to contain a gate from the Forward Co-polarization
 * receiver in the lower 16 bits.
 *
 * For each range gate, the pulses in a given dwell are summed
 * and multiplied by a scaling factor (Comp_Factor). They are then
 * stored in the mean-reflectivity table. Data for
 * a dwell are simultaneously summed and scaled, taking full
 * advantage of the DSP32Cs multiply-accumulate architecture.
 */

                r12e = Num_Pulses               /* # pulses/dwell */
                r20 = *r12
                r5e = NadirXMean
                r6e = NadirCoMean
                r7e = FwdCoMean
                r15 = 4                         /* offset */

Next_Pulse:     r18 = *r8                       /* get # range gates */
                r10e = Nadir_Hold
                r11e = Fwd_Hold
                do 1,r18                        /* read pulse from input FIFOs */
                   a0 = (*r10++=*r1) + a0       /* Read one range from Nadir */
                   a0 = (*r11++=*r2) + a0       /* and one from Forward ant. */

                r18 = *r8                       /* get # range gates */
                r10e = Nadir_Hold
```

```
                r11e = Fwd_Hold
                do 5,r18
                    a1 = float(*r10++)          /* Nadir Co */
                    a0 = float(*r10++)          /* Nadir X */
                    a2 = float(*r11++r15)       /* Forward Co */
                    *r6++ = a3 = *r6 + a1       /* measurements for all */
                    *r5++ = a3 = *r5 + a0       /* accumulate echo power */
                    *r7++ = a3 = *r7 + a2       /* three channels */
                r5e = NadirXMean                /* reset pointers */
                r6e = NadirCoMean               /* for next pulse */

                if(r20-->=0) pcgoto Next_Pulse  /* more pulses this dwell? */
                r7e = FwdCoMean                 /* (this one is executed too!) */

Compress:       r18 = *r8                       /* get # range gates */
                r12e = Comp_Factor              /* Scaling factor */
                do 2,r18
                    *r5++ = a0 = *r5 * *r12
                    *r6++ = a0 = *r6 * *r12
                    *r7++ = a0 = *r7 * *r12


/*
 * Output results:
 *
 * Results are output as follows: Nadir polarisations are output
 * via Bus A with the cross-polarisation in the upper 16 bits and
 * the co-polarisation in the lower 16 bits.  Forward Co-polarisation
 * is sent in the lower 16 bits of Bus B.  Bits 16-25 (Bus B) contain
 * range gate # while bits 26-31 contain the Dwell # (modulo 64)
 */

Output:         r5e = NadirXMean
                r6e = NadirCoMean
                r7e = FwdCoMean
                r18 = *r8                       /* get # range gates */
                r12e = pack                     /* pack X & Co polarisations */
                r13e = Range_cnt                /* tag current range */
                r16 = 0x44                      /* send results to Board 4, zone 4 */
                *r3 = r16                       /* Bus A destination */
                *r4 = r16                       /* Bus B destination */
                do 26,r18
                    *r12++ = a0 = int(*r6++)    /* get Nadir Co */
                    *r12-- = a0 = int(*r5++)    /* get Nadir X */
                    r14e = Dwell_cnt
```

```
                    r15 = *r14                  /* Get current dwell # */
                    nop
                    a0 = (*r1=*r12) + a0        /* send Nadir polarisations */
                    r15 = r15 & 0x3f            /* truncate to 6 bits */
                    r15 = r15>>>1               /* shift right 1 bit */
                    r15 = r15>>>1               /* shift right 2 bits */
                    r15 = r15>>>1               /* shift right 3 bits */
                    r15 = r15>>>1               /* shift right 4 bits */
                    r15 = r15>>>1               /* shift right 5 bits */
                    r15 = r15>>>1               /* shift right 6 bits */
                    r15 = r15>>>1               /* shift right 7 bits */
                    r14 = *r13                  /* get previous range # */
                    nop
                    r14 = r14 + 1               /* update it */
                    *r13 = r14                  /* and save it */
                    r14 = r14 & 0x3ff           /* truncate to 10 bits */
                    r15 = r15 | r14             /* and pack it in place */
                    *r12++ = a0 = int(*r7++)    /* Forward Co in lower 16 */
                   nop
                    *r12-- = r15                /* store in upper 16 bits */
                    nop                         /* latency... */
                    nop                         /* latency... */
                    nop                         /* latency... */
                    a0 = (*r2=*r12) + a0        /* ship it! */
                    nop                         /* latency */
/*
 * Pass the token
 */
PassToken:    r13 = 0x1f                        /* Pass token cmd */
              *r3 = r13                         /* Bus A */
              *r4 = r13                         /* Bus B */
              r13 = 0x11                        /* pass to node 1, zone 1 */
              *r1 = r13
              *r2 = r13
              goto Next_Dwell
              nop


.rsect ".text"
.align 4
NadirXMean:         436*float 0.0               /* Mean Reflectivity table */
NadirCoMean:        436*float 0.0               /* # range gates */
FwdCoMean:          436*float 0.0

Nadir_Hold:         436*fltbits 0x0             /* Holding area for packed reflectivity */
```

```
                                                /* measurements from nadir antenna */
                                                /* # range gates */
Fwd_Hold:       436*fltbits 0x0                 /* Holding area for packed reflectivity */
                                                /* measurements from forward antenna */
                                                /* # range gates */

.rsect ".R1"
.align 4
Zero:           float 0.0
Num_Pulses:     int 1998        /* # pulses/dwell - 2 */
Range_Gates:    int 108         /* Number of range gates - 1 */

.align 4
pack:           float 0.0       /* holding spot for result packing */
holdpack:       int 0           /* holds extra result between range packing */
PackFlag:       int 0           /* >0 => SNR from last range waiting to be
                                 * packed.  It is sitting in holdpack.
                                 * =0 => previous range completely packed */
Fubar:          int 0xface      /* For test purposes only */
stopflag:       int 0xface      /* For test purposes only */
.align 4
addrflag:       float 0.0       /* for test purposes only */

Comp_Factor:    float 0.0005    /* 1/(# pulses/dwell) */
.align 4
Range_cnt:      int 0           /* for test purposes */
Dwell_cnt:      int 0           /* for test purposes */
QDATA:          float 0.0
QDATB:          float 0.0
DESTA:          float 0.0
DESTB:          float 0.0
```

Appendix D - Autocovariance Code

```
/*
 * File   :          ppzone0.s
 *
 * Purpose :         This program implements the pulse-pair algorithm.
 *                   It is intended to be run as zone1 on an LUA200
 *                   designated as Board ID = 1. Results are packed.
 * Calls  :
 *
 * Author :          Shaun R. Nicholson
 *                   Center For Research, Inc.
 *                   University of Kansas
 *                   (913) 864-4835
 *
 * Revision
 * History :
 *
 *    (0.x = Beta)
 *    Revision:    By:              Date:       Description:
 *    0.0          S.R. Nicholson   11-15-91    Original.
 *    0.1          S.R. Nicholson   01-09-92    Vp calculations were
 *                                              not done correctly.
 *    0.2          S.R. Nicholson   03-10-92    Token passing was not
 *                                              done correctly.
 *                                              Removed FIFO allocation.
 *    0.3          S.R. Nicholson   03-31-92    Replace if() goto <addr>
 *                                              with if() goto rxx for
 *                                              24-bit addressing.
 *    0.4          S.R. Nicholson   04-07-92    ver. 0.3 was corrupt
 *                                              Replace if() goto rxx
 *                                              with if() pcgoto <addr>
 *                                              to use offset addressing.
 *    0.5          S.R. Nicholson   06-22-92    Reduce #pulses/range gates
 *                                              for token pass debugging.
 *    0.6          S.R. Nicholson   07-01-92    Stop after one dwell
 *
 *    0.7          S.R. Nicholson   07-02-92    Count1 defined wrong.
 *
 *    0.8          S.R. Nicholson   07-12-92    Remove code to save
 *                                              current DSP output
 *                                              queue. This was used
 *                                              only for simulation.
 *
 *
 *
 *
 */
```

```
.global main,Dwell,Block,Range,IQstart,RStart,Vpstart,Out
.global _atan,_div,_loge,_xtoy

.global QDATA,Rout,DESTA,IpPrev,Ip,QpPrev,Qp,NumPulse,RangeGates,Vpcoeff
.global VarCoeff,OneThird,FourThirds,Zero,temp1,temp2,temp3,temp4,Count1
.global Count2,Count3,Count4,Block,IQPack,I0,I1,I2,Q1,Q2,R0,R1,R2,Vp,Var,SNR
.global Routaddr,PartSum,theta,pack,holdpack,PackFlag,xtra,noxtra
.global Fubar

#include "c:\dosapps\dsp32c\bin\zone_addr.h"

.rsect ".text"
main:
/*
 * (1) Disable interrupts
 * (2) Disable DMA transfers
 * (2) Set external memory partition A to 0 wait states
 * (3) Set external memory partition B to 2+ wait states
 */
        r1 = 0x000d
        pcw = r1            /* interrupts and memory wait states */
        ioc = 0x0           /* DMA */
        goto Dwell
        nop
/*
 * At the beginning of each dwell, memory registers used to accumulate
 * results need to be cleared...
 */
.rsect ".R2"
Dwell:  r2e = Dwell_cnt     /* Update dwell count */
        r14 = *r2
        nop
        r14 = r14 + 1       /* increment count */
        *r2 = r14           /* and save... */
        r2e = Range_cnt     /* next, zero out */
        r14 = 0             /* the range count */
        *r2 = r14

        r2e = PartSum
        r14e = Count4
        r1 = *r14
        r3e = Zero
        do 0,r1
```

```
            a0 = (*r2++=*r3) + a0
           r14e = NumBlock        /* # blocks per dwell - 2 */
           r18 = *r14

/*
 * Read 32 bit data from the input FIFO and store in external memory. The
 * data are packed I and Q (12 bits each sign extended to 16 bits) with I
 * in the most significant 16 bits. These data will be transferred directly
 * to memory and unpacked later. This algorithm will process a variable
 * number of range gates. To allow this, the SBC must load the required number
 * of iterations to be done into the 16 bit Zone memory variable Count1.
 * Count1 = #range gates * #pulses each range - 2
 */
Block:     r1e = QDATA
           r2e = IQPack
           r14e = Count1
           r3 = *r14                /* read data from input FIFO */
loop:        a0 = (*r2++=*r1) + a0
           if(r3-->=0) pcgoto loop
           nop
           r1e = RangeGates       /* #range gates to process */
           r17 = *r1
           r1e = Routaddr
           r2e = QDATA
           *r1 = r2e              /* Save address of Output FIFO */
           r13e = IQPack          /* Base address for packed I & Q */
           r12e = PartSum         /* Partial results of complex math */
/*
 * At this point, a block of packed I & Q data has been read from the input
 * FIFO and stored in external RAM. Next, they will be separated into
 * I and Q components, converted to the special DSP32C floating point format,
 * and stored in on-chip RAM. In-phase data will be stored in internal RAM
 * bank 0 while Quadrature data will be stored in internal RAM bank 1. Note
 * that only sufficient data for one range gate is expanded at a time. These
 * are then processed and intermediate results stored. They cycle repeats for
 * the next range gate. This is due to memory limitations.
 */

/*
 * Separate I and Q data, and convert to float...
 */
Range:     r1e = Ip
           r2e = Qp
           r3e = r13              /* Get base addr of data for range X */
           r13e = r13 + 4         /* Point to base for next range */
```

```
            r14e = Count3       /* get pulse-to-pulse distance */
            r15 = *r14
            r14e = Count2       /* # pulses each (partial) range */
            r5 = *r14
            nop
            do 1,r5             /* enough data for 1 partial range */
              *r1++ = a0 = float(*r3++)
              *r2++ = a0 = float(*r3++r15)
/*
 * Define Pointers...
 */
            r1e = r12           /* Points to IpPrev stored in PartSum
                                 * for the current range */
            r6e = IpPrev
            r7e = IpPrev+4
            r8e = IpPrev+8
            r9e = QpPrev
            r10e = QpPrev+4
            r11e = QpPrev+8
            a0 = (*r6=*r1++) + a0     /* Restore IpPrev */
            a0 = (*r7=*r1++) + a0     /* Restore IpPrev+1 */
            a0 = (*r9=*r1++) + a0     /* Restore QpPrev */
            a0 = (*r10=*r1++) + a0    /* Restore QpPrev+1 */
                                      /* Now, r1 points to Previous I0 */
            r2e = r12+20              /* Previous I1 */
            r3e = r12+24              /* Previous I2 */
            r4e = r12+28              /* Previous Q1 */
            r5e = r12+32              /* Previous Q2 */
/*********************************************************************
 *
 * Accumulate pulse pairs...
 *
 *     I0 += Ip^2 + Qp^2
 *     I1 += IpIp+1 + QpQp+1
 *     Q1 += -IpQp+1 + QpIp+1
 *     I2 += IpIp+2 + QpQp+2
 *     Q2 += -IpQp+2 + QpIp+2
 *
 *********************************************************************/
IQstart:    r14e = Count2                 /* How many pulses? */
            r16 = *r14
            nop
            do 12,r16
              a0 = *r6++                  /* IpPrev */
              a1 = *r9++                  /* QpPrev */
```

```
                nop                             /* latency... */
                a2 = *r1 + a0 * a0              /* a2 = I0 + Ip^2 */
                *r1 = a2 = a2 + a1 * a1         /* I0 = Ip^2 + Qp^2 */
                a2 = *r2 + a0 * *r7             /* a2 = I1 + IpIp+1 */
                *r2 = a2 = a2 + a1 * *r10       /* I1 = a2 + QpQp+1 */
                a2 = *r4 - a0 * *r10++          /* a2 = Q1 - IpQp+1 */
                *r4 = a2 = a2 + a1 * *r7++      /* Q1 = a2 + QpIp+1 */
                a2 = *r3 + a0 * *r8             /* a2 = I2 + IpIp+2 */
                *r3 = a2 = a2 + a1 * *r11       /* I2 = a2 + QpQp+2 */
                a2 = *r5 - a0 * *r11++          /* a2 = Q2 - IpQp+2 */
                *r5 = a2 = a2 + a1 * *r8++      /* Q2 = a2 + QpIp+2 */
/*
 * Save last two I & Q pulses - they will be needed for
 * the next data block!
 */
                a0 = (*r12++=*r6) + a0          /* First, Ip */
                a0 = (*r12++=*r7) + a0          /* then Ip+1 */
                a0 = (*r12++=*r9) + a0          /* Qp */
                a0 = (*r12++=*r10) + a0         /* and Qp+1 */
                nop
                r12e = r12 + 20                 /* skip past copies of I0-Q2
                                                 * stored in PartSum */
/*
 * More range gates to process this Block ?
 */
                if(r17-->=0) pcgoto Range       /* Yup! */
                nop
                r12e = PartSum                  /* Nope, reset partial sum */
                                                /* pointer for the next block */

/*
 * Is there another block of data to process this Dwell ?
 */
                if(r18-->=0) pcgoto Block       /* Yup! */
                nop
/*
 * If not, then the complex summation is finished and it is
 * time to compute the various parameters of interest.
 */
                r14e = RangeGates
                r17 = *r14
RStart:           r1e = I0
                r12e = r12 + 16                 /* point to I0 for current range */
                do 0,4                          /* copy I & Q sums to named memory
                                                 * variables for further
```

```
                                        * processing */
          a0 = (*r1++=*r12++) + a0
/*********************************************************************/
/* R(0) = I0/N                                                       */
/*********************************************************************/
chk2:     nop
          nop
          nop
          r1e=I0
          r2e=R0
          a0=(*r2=*r1) + a0

/*********************************************************************/
/* |R(1)| = sqrt( I1^2 + Q1^2 )/N                                    */
/*********************************************************************/
          r1e = I1
          r2e = Q1
          r3e = temp1
          a0 = *r1 * *r1
          *r3 = a0 = a0 + *r2 * *r2
          call   _sqr (r14)
          nop
.align 4
          int24  temp3
          int24  temp1,R1
.align 4

/*********************************************************************/
/* |R(2)| = sqrt( I2^2 + Q2^2 )/N                                    */
/*********************************************************************/
          r1e = I2
          r2e = Q2
          r3e = temp1
          a0 = *r1 * *r1
          *r3 = a0 = a0 + *r2 * *r2
          call   _sqr (r14)
          nop
.align 4
          int24  temp3
          int24  temp1,R2
.align 4

/*********************************************************************/
/*  Vp = VpCoeff * arg[R(1)]                                         */
/*********************************************************************/
```

```
Vpstart:   r1e=I1
           r2e=temp1
           a0=-*r1
           *r2++=a0=ifalt(*r1)    /* temp1=abs(I1) */
           r1e=Q1
           a0=-*r1
           *r2=a0=ifalt(*r1)      /* temp2=abs(Q1) */
           call   _div (r14)      /* temp3=Q/I */
           nop
           int24  temp2,temp1,temp3
.align 4
           call   _atan (r14)     /* find principal angle */
           nop
           int24  temp1
           int24  temp3,theta
.align 4
/* find arg(Z) where Z=I1+jQ1 */
           r1e=I1
           a0=*r1                 /* set flag if I1 is negative */
           r2e=Q1
           a1=*r2                 /* set flag if Q1 is positive */
           r3e=theta
           r4e=pi
           if(alt) pcgoto lefthalf    /* I1 was negative! */
           nop
           if(age) pcgoto quaddone    /* Q1 positive - quadrant I */
           nop
quad4:     *r3=a0=-*r3            /* negate theta */
           goto quaddone
           nop
lefthalf:    if(age) pcgoto quad2     /* Q1 positive - quadrant II */
           nop
quad3:       *r3=a0=*r3-*r4         /* arg = theta - 180 */
           goto quaddone
           nop
quad2:       *r3=a0=*r4-*r3         /* arg = 180 - theta */
           nop
           nop
quaddone:  r1e = VpCoeff
           r2e = Vp
           *r2 = a0 = *r3 * *r1       /* Vp = VpCoeff * theta */
/*****************************************************************/
/* Variance = VarCoeff * lnlR1/R2l                               */
/*****************************************************************/
           call   _div (r14)
```

```
            nop
            int24   R1,R2,temp1
.align 4
            call    _loge (r14)
            nop
            int24   temp3
            int24   temp1,temp2
.align 4
            r1e = VarCoeff
            r2e = Var
            *r2 = a0 = a0 * *r1
/*************************************************************************/
/* SNR = [|R1|^(4/3)] / [R0 * |R2|^(1/3) - |R1|^(4/3)]                   */
/*************************************************************************/
            call    _xtoy (r14)         /* R2^(1/3) */
            nop
            int24   temp3
            int24   R2,OneThird,temp1
.align 4
            r8e = R0
            r9e = temp1                 /* *r9 = R2^(1/3) */
            r10e = temp2                /* *r10 = R1^(4/3) */
            *r9 = a0 =  *r8 * a0        /* *r9 = R0 * R2^(1/3) */
            call    _xtoy (r14)
            nop
            int24   temp3
            int24   R1,FourThirds,temp2
.align 4

            *r9 = a0 = *r9 - a0         /* R0 * R2^(1/3) - R1^(4/3) */
            call    _div (r14)
            nop
            int24   temp2,temp1,SNR
.align 4
            goto    Out
            nop
/*
 * Dump results to the output FIFO...
 */
.rsect ".R1"
Out:        r1e = DESTA
            r2e = QDATA
            r5 = 0x44                   /* send results to board 4, zone 4 */
            *r1 = r5
            r5e = Vp                    /* mean doppler velocity */
```

```
                r7e = pack                  /* pack Vp and variance */

                *r7++ = a0 = int(*r5++)     /* pack Vp */
                *r7-- = a0 = int(*r5++)     /* with variance...*/
                r8e = Dwell_cnt
                r6 = *r8                    /* Get current dwell # */
                nop
                a0 = (*r2=*r7) + a0         /* send packed result on Bus A */
                r6 = r6 & 0x3f              /* truncate to 6 bits */
                r6 = r6>>>1                 /* shift right 1 bit */
                r6 = r6>>>1                 /* shift right 2 bits */
                r6 = r6>>>1                 /* shift right 3 bits */
                r6 = r6>>>1                 /* shift right 4 bits */
                r6 = r6>>>1                 /* shift right 5 bits */
                r6 = r6>>>1                 /* shift right 6 bits */
                r6 = r6>>>1                 /* shift right 7 bits */
                r8e = Range_cnt
                r1 = *r8                    /* get previous range # */
                nop
                r1 = r1 + 1                 /* update it */
                *r8 = r1                    /* and save it */
                r1 = r1 & 0x3ff             /* truncate to 10 bits */
                r6 = r6 | r1                /* and pack it into place */
                *r7++ = a0 = int(*r5)       /* pack SNR */
                nop
                *r7-- = r6                  /* with marker */
                nop
                nop
                nop
                a0 = (*r2=*r7) + a0         /* send SNR and marker on Bus A */
                nop
                nop
                nop
                r1e = RStart
                if(r17-->=0) goto r1        /* next range */
                nop
                goto PassToken              /* pass token to next process... */
                nop

/*
 * Pass the token A back to zone 0
 */
PassToken:      r1e = DESTA
                r2e = QDATA
                r3 = 0x1f                   /* pass token... */
```

```
            *r1 = r3
            r3 = 0x11                   /* ...brd 1, zone 1 */
            *r2 = r3
            goto Dwell                  /* and prepare for next dwell */
            nop
.rsect ".R2"
/*************************************************************/
/* Routine source file: atan.asm                             */
/* DSP32/DSP32C Application Software Library. Version 2.0    */
/*************************************************************/

/*
     _atan(A,B,C)
     float *A,*B,*C;
  scratch register short r4,r5 (for DSP32 only);
  scratch pointer register short r1,r2,r3,r14;
  scratch register float a0,a1,a2,a3;
*/
#include <dspregs.h>            /* Translates DSP32C keywords to DSP32 *
                                 *  keywords. See Section 2.3.1     */

_atan: r1e=*r14++               /* pointer to local variable */
     r2e=*r14++                 /* pointer to x */
     r3e=_atanB
     *r1++=a3=*r2 + *r3         /* x+1 */
#if DSP32C
     a2 = seed(a3)
     *r1=a0=*r2 - *r3++         /* x-1 */
     r2e = *r14                 /* pointer to out */
     a0=*r3++ -a3 * a2          /* a0=1.4074-x*y */
     a1=*r3++ * a2              /* a1=.81*x */
     a2=*r3++ -a3 * a2          /* a2=2.27424-x*y */
#else
     *r1--=a0=*r2 - *r3++       /* x-1 */
     r2e = *r14                 /* pointer to out */
        nop
     r4=*r1++                   /* load x+1=y into cau */
     r5=*r1++
     r4=r4^0xffff               /* invert all bits except sign bit */
     r5=r5^0x7fff
     *r2++=r4                   /* 1st estimate of 1/y,
                                   or x, written to memory */
     *r2--=r5

     a0=*r3++ -a3 * *r2         /* a0=1.4074-x*y */
```

```
        a1=*r3++ * *r2          /* a1=.81*x */
        a2=*r3++ -a3 * *r2      /* a2=2.27424-x*y */   -
    #endif
        a0=*r3++ +a0 * a0       /* a0=1.71742-2.8148*x*y+x^2*y^2 */
        a1=a0*a1                /* a1=1.14*x-.81*x^2*y */
        a2=a2*a1                /* a2=1.842*x-.81*x^2*y */
        nop
        a0=a2+a0*a1             /* a0=3.8x-5.41x^2*y+3.42x^3* ...
                                   y^2-.81x^4*y^3          */
        nop
        nop
        a1=*r3 + a3*a0          /* a1=-1 + y* x(new) */
        a0 = a0 * *r1           /* scale by "(x-1)" */
        nop
        nop
        *r2 = a2 = a0-a1*a0     /* third iteration, ... */
        nop                     /*   a2=(x-1)/(x+1)    */
        nop
        a0=a2*a2                /* y**2 */
        r1e=_atanC
        nop
        a0=a0**r2               /* y**3 */

        a1 = *r1++ + a0 * *r1++  /* c7 + c9*(y^2) */
        a2 = *r1++ + a0 * *r1++  /* c3 + c5*(y^2) */
        a0 = a0 * *r2           /* y^4 */
        a1 = a1 * a0            /* c7*(y^3) + c9*(y^5) */
        a2 = *r1++ + a2 * a0    /* pi/4 + c3*(y^3) + c5*(y^5) */

        a2 = a2 + *r1++ * *r2   /* pi/4+c1*y+ ...
                                   c3*(y^3)+c5*(y^5) */
        *r2 = a0 = a2 + a1 * a0 /* pi/4+c1*y+c3*(y^3)+c5*(y^5)
                                   ... +c7*(y^7)+c9*(y^9)  */
        nop
        goto r14+4
        nop

    _atanB: float 1.0
        float 1.4074074347, 0.81, 2.27424702, -.263374728, -1.0
    _atanC: float .0208351, -.085133    /* c9, c7 */
        float .180141, -.3302995        /* c5, c3 */
        float .785398163, .999866       /* pi/4, c1 */
    /*****************************************************************/
    /*                                                               */
    /*  _div(A,B,C)                                                  */
```

```
/*                                                                    */
/*  float *A,*B,*C;                                                   */
/*  scratch register short r5,r6 (DSP32 only);                        */
/*  scratch pointer register short r1,r2,r3,r4,r14;                   */
/*  scratch register float a0,a1,a2,a3;                               */
/*                                                                    */
/********************************************************************/

_div:  r4e = *r14++    /* numerator pointer */
       r1e = *r14++    /* denominator pointer */
       r2e = *r14      /* point to inverse location */
       a3=*r1          /* load number to be inverted into a3 */

#if DSP32C
       a1 = seed(*r1)
       r3e=_divA
          nop
       a0 = *r3++ + a3 * a1     /* a0=-1.6014-x*y */
       a1 = a1 * *r3++          /* a1=.8237*x */
#else
       r3e=_divA
       r6=*r1++                 /* load number into cau */
       r5=*r1--
       r6=r6^0xffff             /* invert all bits except sign bit */
       r5=r5^0x7fff
       *r2++=r6                 /* 1st estimate written to memory */
       *r2--=r5
       a0 = *r3++ + a3 * *r2    /* a0=-1.6014-x*y */
       a1 = *r2 * *r3++         /* a1=.8237*x */
#endif
       nop
       a0 = *r3++ + a0 * a0     /* a0=3.4166-3.2023*x*
                                   y+x^2*y^2 */
       a2 = a1 * a3             /* a2=.8237*x*y */
       nop
       a0 = a0 * a1             /* a0=2.814-2.638x^2*
                                   y+.8237x^3*y^2 */
       a1 = *r3 + a0 * a2       /* a1=1.814*x*y-2.638*x^2*
                                   y^2+.8237*x^3*y^3 */
       nop
       a0 = a0 * *r4            /* scale by numerator */
       nop
       nop
       *r2 = a0 = a0 - a1 * a0  /* last iteration
                                   written to memory */
```

```
      nop
      goto r14 + 4
      nop

_divA: float -1.60138661 , 0.82371354 , 0.85221935 , -1.0
.align 4
/*************************************************************/
/* Routine source file: loge.asm                             */
/* DSP32/DSP32C Application Software Library. Version 2.0    */
/*************************************************************/

/*
      _loge(A,B,C)
      float *A, *B, *C;
  scratch register short r5,r6;
  scratch pointer register short r1,r2,r3,r4,r14;
  scratch register float a0,a1,a2,a3;
*/
#include <dspregs.h>          /* Translates DSP32C keywords to DSP32 *
                               *  keywords. See Section 2.3.1     */

_loge:  r2e=*r14++            /* pointer to a copy of x */
      r1e=*r14++              /* pointer to operand, x */
      r4e=_logeC+36           /* pointer to c8 */
      *r2=a0=*r1              /* load x into copy */
      nop
      r3e=r2 + 4              /* pointer to scratch pad for exponent */

   /* extract mantissa, M, and divide by 2 */
      r5=-1 + 128             /* -1 plus bias */
      *r2=r5l                 /* replace exponent of operand with "-1" */
      a3=*r2                  /* move M/2 into a3 */

   /* store exponent, E, as a floating point number */
      r6l=*r1                 /* get exponent (byte) */
      r1e=*r14                /* pointer to output */
      r6=r6+1-128             /* remove bias and add 1 */
      *r3=r6                  /* store as an integer, i.e., (E+1) */
      *r3=a0=float(*r3)       /* convert to float and store */

/* compute log(M/2) in base 2  */
  a0 = a3 * a3                /* (M/2)^2 */
  a1 = *r4-- + a3 * *r4--     /* c7 + c8*M/2 */
  a2 = *r4-- + a3 * *r4--     /* c4 + c5*M/2 */
  a0 = a0 * a3                /* (M/2)^3 */
```

```
   a1 = *r4-- + a1 * a3          /* c6+c7*(M/2)+c8*((M/2)^2) */
   a3 = *r4-- + a3 * *r4--       /* c1 + c2*M/2 */
   a2 = *r4-- + a2 * *r2         /* c3+(M/2)*(c4+c5*(M/2)) */
   a1 = a1 * a0                  /* c6*((M/2)^3)+...+c8*((M/2)^5) */
   a3 = *r4-- + a3 * *r2         /* c0 + c1*(M/2) + c2*((M/2)^2) */
   a0 = *r3 * *r4                /* c9 * (E+1) */
   a2 = a3 + a2 * a0             /* c0+c1*(M/2)+...+c5*((M/2)^5) */
   a1 = a2 + a1 * a0             /* c0+c1*(M/2)+...+c8*((M/2)^8) */

    /* add c9*(E +1) and store  */
      *r1 = a0 = a1 + a0
      nop
      goto r14+4
      nop

_logeC: float 0.6931471806  /* c9 */
      float -3.067466148    /* c0 */
      float 51.49518505     /* c3 */
      float 11.30516183     /* c1 */
      float -27.7466647     /* c2 */
      float -33.20167437    /* c6 */
      float -66.69583126    /* c4 */
      float 58.53503341     /* c5 */
      float 10.98927014     /* c7 */
      float -1.613006222    /* c8 */
.align 4
/*****************************************************************/
/*                                                               */
/*  _sqr(A,B,C)                                                  */
/*                                                               */
/*  float *A,*B,*C;                                              */
/*  scratch register short r4,r5;                                */
/*  scratch pointer register short r1,r2,r3,r14;                 */
/*  scratch register float a0,a1,a2;                             */
/*                                                               */
/*****************************************************************/
#include <dspregs.h>          /* Translates DSP32C keywords to DSP32 *
                               *  keywords.  See Section 2.3.1     */

_sqr:
      r2e = *r14++            /* address of local variables */
      r1e = _sqrC
      *r2++ = a0 = *r1++      /* stores exponent adjustment */
      *r2-- = a0 = *r1        /* stores exponent adjustment */
```

```
        r1e = *r14++            /* address of input value */
        r3e = _sqrB
        r4l = *r1               /* load exponent into cau */
        nop
        r5 = -r4                /* 2's complement inverts exponent */
        *r2 = r5l
        a2 = *r1 * *r2          /* reset exponent of operand to zero */
        r4 = r4 >>1
        if (cc) pcgoto _sqrA    /* skip sqrt(2) adjustment */
        a1 = a2 * a2            /* x^2 */
        r2e = r2 + 4            /* set pointer for sqrt(2) adjustment */
_sqrA:  a0 = a2 * *r3++
        a0 = a0 + *r3++
        a0 = a0 + a1 * *r3++
        a1 = a2 * *r3++
        a1 = a1 + *r3++
        a0 = a1 + a1 * a0       /* end of polynomial approximation */
        a2 = a2 * *r3++         /* start of iteration approximation */
        r4 = r4 + 64            /* final exponent = (opexp)/2 + 64 */
        a0 = a2 * a0
        a1 = a0 * a0
        *r2 = r4l
        a0 = a0 * *r2
        a1 = *r3 - a1 * a2
        r1e = *r14              /* address of output location */
        nop
        *r1 = a0 = a0 * a1      /* stores result */
        nop
        goto r14+4
        nop

_sqrB:  float -0.38289166, 1.18787772, 0.049693281, -1.92155474
        float 2.0667909, 0.5, 1.5000001

_sqrC: float 1.0, 1.4142136
.align 4
/*************************************************************/
/* Routine source file: xtoy.asm                             */
/* DSP32/DSP32C Application Software Library. Version 2.0    */
/*************************************************************/

/*
 To use this routine with DSP32C, bit 4 of DAUC must be set to 0.

        _xtoy(A,B,C,D)
```

```
    float *A, *B, *C, *D;
  scratch register short r5, r6;
  scratch pointer register short r1,r2,r3,,r4,r7,r14;
  scratch register float a0,a1,a2,a3;
*/
#include <dspregs.h>          /* Translates DSP32C keywords to DSP32 *
                               *  keywords.  See Section 2.3.1     */

_xtoy:                        /* computes log(x) [base 2] */
  r2e=*r14++                  /* pointer to a copy of x */
  r1e=*r14++                  /* pointer to operand, x */
  r4e=_xtoyC+32               /* pointer to c8 */
  *r2=a0=*r1                  /* load x into copy */
  r7e=*r14++                  /* pointer to y */
  r3e=r2 + 4                  /* pointer to scratch pad for exponent */

/* extract mantissa, M, and divide by 2 */
  r5=-1 + 128                 /* -1 plus bias */
  *r2=r5l                     /* replace exponent of operand with "-1" */
  a3=*r2                      /* move M/2 into a3 */
/* store exponent, E, as a floating point number */
  r6l=*r1                     /* get exponent (byte) */
  nop
  r6=r6+1-128                 /* remove bias and add 1 */
  *r3=r6                      /* store as an integer */
  *r3=a0=float(*r3)           /* convert to float and store */

/* compute log(M/2) in base 2  */
  a0 = a3 * a3                /* (M/2)^2 */
  a1 = *r4-- + a3 * *r4--     /* c7 + c8*M/2 */
  a2 = *r4-- + a3 * *r4--     /* c4 + c5*M/2 */
  a0 = a0 * a3                /* (M/2)^3 */
  a1 = *r4-- + a1 * a3        /* c6 + c7*(M/2) + c8*((M/2)^2) */
  a3 = *r4-- + a3 * *r4--     /* c1 + c2*M/2 */
  a2 = *r4-- + a2 * *r2       /* c3 + (M/2)*(c4 + c5*(M/2)) */
  a1 = a1 * a0                /* c6*((M/2)^3)+c7*((M/2)^4)+
                                 c8*((M/2)^5) */
  a3 = *r4 + a3 * *r2         /* c0 + c1*(M/2) + c2*((M/2)^2) */
  a2 = a3 + a2 *a0            /* c0 + c1*(M/2) + ... +
                                 c5*((M/2)^5) */
  a1 = a2 + a1 * a0           /* c0 + c1*(M/2) + ... +
                                 c8*((M/2)^8) */
  a0 = a1 + *r3               /*  add (E+1) ==> a0=log(x) [base 2] */

/* computes 2^{y*log(x)} [base 2] */
```

```
    r2e = *r14++            /* r2 points to output location */
    r3e = _xtoyD            /* r3 points to 0.5 and then ci's */
    a0 = a0 * *r7           /* y*log(x) [base 2] */

    a3 = a0 - *r3++         /* x - 0.5 */
    *r2 = a3 = int(a3)      /* stores i, the "integer" part */

    a3 = float(a3)          /* converts i to float format */
    a3 = -a3 + a0           /* computes f, the "fractional part */

    nop
    r6l = *r2               /* stores i in r6l */

  /* computes 2^f below */
    a0 = a3 * a3            /* f^2 */
    a1 = *r3++ + a3 * *r3++ /* c2+c3(f) */
    a2 = *r3++ + a3 * *r3++ /* c4+c5(f) */
    a0 = a0 * a0            /* f^4 */
    a3 = *r3 + a3 * *r3++   /* c0+c1(f) */
    a1 = a3 + a1 * a0       /* c0+c1(f)+c2(f^2)+c3(f^3) */
    *r2 = a2 = a1 + a2 * a0 /* c0+c1(f)+...+c5(f^5) */

    r6 = r6 + 128           /* adjusts exponent for bias */
    nop                     /* waits for DAU to   */
    nop                     /* write the mantissa */
    *r2 = r6l               /* merge in exponent */
    a0 = *r2
    return (r14)
    nop

  _xtoyC: float -4.4254182      /* c0 */
        float 74.29184809343    /* c3 */
        float 16.3099009156     /* c1 */
        float -40.02997556826   /* c2 */
        float -47.89989096077   /* c6 */
        float -96.221745        /* c4 */
        float 84.44820241827    /* c5 */
        float 15.8541655496     /* c7 */
        float -2.32707607725    /* c8 */

  _xtoyD:  float 0.5
        float 0.0558263180623292    /* c3 */
        float 0.240153617040129     /* c2 */
        float 0.0018775766770276    /* c5 */
        float 0.0089893400833312    /* c4 */
```

```
        float 0.6931530732007278    /* c1 */
        float 0.9999999702          /* c0 */


.rsect ".R0"
.align 4
IpPrev:     2*float 42.0        /* Contains Ip from previous block */
Ip:         200*float           /* Contains Ip for current block */
                                /* #pulses each range  */
            float 0.0,0.0       /* Overshoot zone.  DON'T REMOVE! */
.rsect ".R1"
.align 4
QpPrev:     2*float 69.0        /* Contains Qp from previous block */
Qp:         200*float           /* Contains Qp for current block */
                                /* #pulses each range */
            float 0.0,0.0       /* Overshoot zone.  DON'T REMOVE! */

VpCoeff:    float 5.2521131     /* Coefficient needed to get mean Doppler */
                                /* VpCoeff = (lambda/2)*[PRF/(2*pi)] */
VarCoeff:   float 18.389794     /* Ditto for variance of the time series */
                                /* VarCoeff = lambda^2*PRF^2/[24*pi^2] */
OneThird:   float 1.0/3.0
FourThirds: float 4.0/3.0
pi:         float 3.1415926535898
Zero:       float 0.0
temp1:      float 0.0           /* scratch registers... */
temp2:      float 0.0
temp3:      float 0.0
temp4:      float 0.0
Routaddr:   int24 0             /* scratch address register */
.align 4
Count1:     int 598             /* #range gates * #pulses each range - 2 */
                                /* Note: #pulses means ONE BLOCK ONLY!! */
Count2:     int 199             /* #pulses each range - 1 */
                                /* again, #pulses ONE BLOCK ONLY! */
Count3:     int 10              /* Distance between consequtive pulses in
                                 * packed I & Q array for a given range
                                 * distance = #range gates * 4 - 2 */
Count4:     int 26              /* (#range gates * 9) - 1 */
NumBlock:   int 8               /* # blocks of data - 2 that comprise a Dwell */
RangeGates: int 1               /* Number of range gates - 2 */
.align 4
NumPulse:   float 1998.0        /* Number of Pulses each range - 2
                                 * INCLUDES ALL BLOCKS (IE. TOTAL
                                 * # PULSES - 2 FOR ALL BLOCKS!!!) */
```

```
pack:       float 0.0          /* holding spot for result packing */
holdpack:   int 0              /* holds extra result between range packing */
PackFlag:   int 0              /* >0 => SNR from last range waiting to be
                                * packed.  It is sitting in holdpack.
                                * =0 => previous range completely packed */
Fubar:      int 0xaaaa         /* For test purposes only */
stopflag:   int 1              /* For test purposes only */
addrflag:   float 0.0          /* for test purposes only */

.rsect ".R2"
.align 4
I0:         float 0.0          /* IMPORTANT!!  The variables I0-Q2 */
I1:         float 0.0          /* *MUST* remain in this order - the */
I2:         float 0.0          /* program assumes they are blocked as */
Q1:         float 0.0          /* shown when loading them... */
Q2:         float 0.0
theta:      float 0.0          /* argument of lag 1 autocorrelation */
R0:         float 0.0          /* magnitude of lag 0 autocorrelation */
R1:         float 0.0          /* magnitude of lag 1 autocorrelation */
R2:         float 0.0          /* magnitude of lag 2 autocorrelation */
Vp:         float 0.0          /* Doppler velocity */
Var:        float 0.0          /* Variance */
SNR:        float 0.0          /* Signal to Noise Ratio */


.rsect ".text"
.align 4
IQPack:     1200*int 0xffff    /* buffer holding packed I & Q */
                               /* 2*#range gates*#pulses each range */
.align 4
PartSum:    27*float 23.0      /* Partial I & Q sums */
                               /* should be 9*#range gates */
.align 4
Range_cnt:  int 0              /* for test purposes */
Dwell_cnt:  int 0              /* for test purposes */
```