A Balloon Born Instrument for Measuring Three Dimensional Charge Movement in Thunderstorms

by

John D. Battles

Submitted in Partial Fulfillment

of the Requirements for the Degree of

Master of Science in Physics

with a Specialization in Instrumentation

New Mexico Institute of Mining and Technology

Socorro, New Mexico

August, 2005

ABSTRACT

ACKNOWLEDGMENT

This thesis was typeset with LaTeX[1] by the author.

[1] LaTeX document preparation system was developed by Leslie Lamport as a special version of Donald Knuth's TeX program for computer typesetting. TeX is a trademark of the American Mathematical Society. The LaTeX macro package for the New Mexico Institute of Mining and Technology thesis format was adapted from Gerald Arnold's modification of the LaTeX macro package for The University of Texas at Austin by Khe-Sing The.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

This thesis is accepted on behalf of the faculty of the Institute by the following committee:

Richard R. Sonnenfeld,Advisor

John D. Battles Date

CHAPTER 1

Instrument History, Theory, and Goals

1.1 Thunderstorm Electric Field Measurement History

Cover history of ground and balloon based electric field sensors.

1.2 Basic Thunderstorm Charge Distribution Models

Talk about basic charge mechanisms and simplified storm models.

[Krehbiel et al., 1979]

1.3 E-sonde Theory of Operation

Describe how the slow electric field antenna works. Derive Mathematical equations for E-field to Voltage.

1.4 Science Goals of the E-sonde Project

Describe what is to be measured and how it will enhance knowledge of storms.

CHAPTER 2

System Hardware

2.1 Overview

2.1.1 Complete E-sonde System

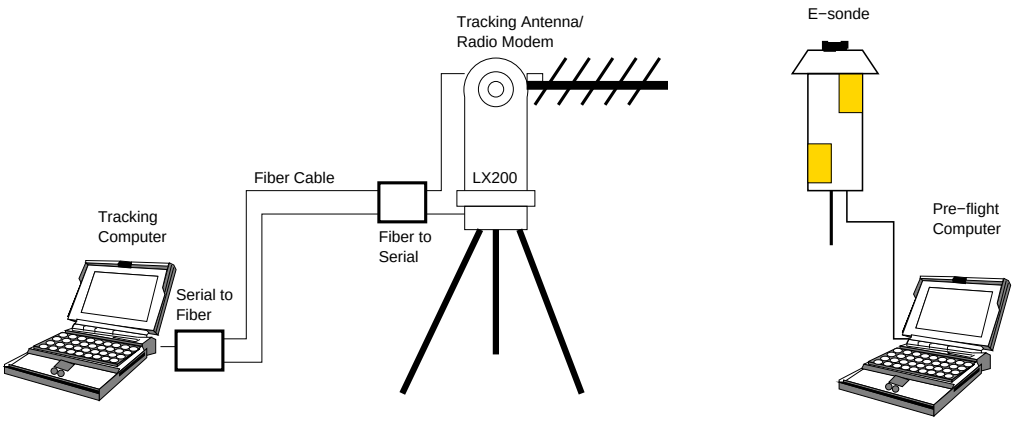


Figure 2.1: E-sonde system diagram.

The E-sonde system is composed of the balloon train, tracking computer, tracking antenna, and preflight computer. The preflight computer is notebook PC running a communication program called Minicom under the Linux OS. It is connected to the DB-9 serial port at the bottom of the E-sonde. See Figure 2.6 on page 11. When the E-sonde is first turned on all the boot-up information can be seen and controlled from the terminal program which is running in VT100 emulator mode. In addition to the OS boot sequence the CMOS setup can be accessed and changed if necessary. Once the system boots

the preflight operator can login and perform the preflight operations described in section 3.7 on page 74.

The tracking computer is used to control the tracking antenna and record and display telemetry information from the E-sonde. A real-time map is displayed showing the E-sonde's location. Figure 3.6 on page 73 shows a picture of the tracking computers screen during a flight. The tracking computer is also used during pre-launch to verify all operational parameters are normal.

The tracking antenna shown in Figure 2.2 on page 4 is used to receive the telemetry signal of the E-sonde. The antenna is a 900 MHz yagi antenna mounted on a Meade LX-200 azimuth and elevation telescope mount. Attached to the antenna is a Maxstream 900 MHz radio modem and a counterweight. The antenna system is connected to the tracking computer via a pair of fiber optic transceivers which convert fiber to RS-232 serial data. The up-link from the computer to the telescope is 4800 baud 8N1 and the down link from the Maxstream modem to the tracking computer is 19.2K baud 8N1.

Below the antenna is a white box which houses a 12V Optima Bluetop deep-cycle marine battery to power the telescope and the Maxstream modem. It also houses the fiber to RS-232 transceiver and its 9V battery supply. Two fiber cables 30 meters long run through 1 in. PVC pipe connected the two transceivers together.

The balloon train consist of several pieces show in figure 2.3 on page 5. At the top of the train providing the lift is a polyethylene balloon filled with helium. The balloon was filled to have a lifting capacity of 7.3 kg kilograms.

Attached to the balloon is a cut down package. The cut down package

Figure 2.2: Picture of the E-sonde tracking antenna.

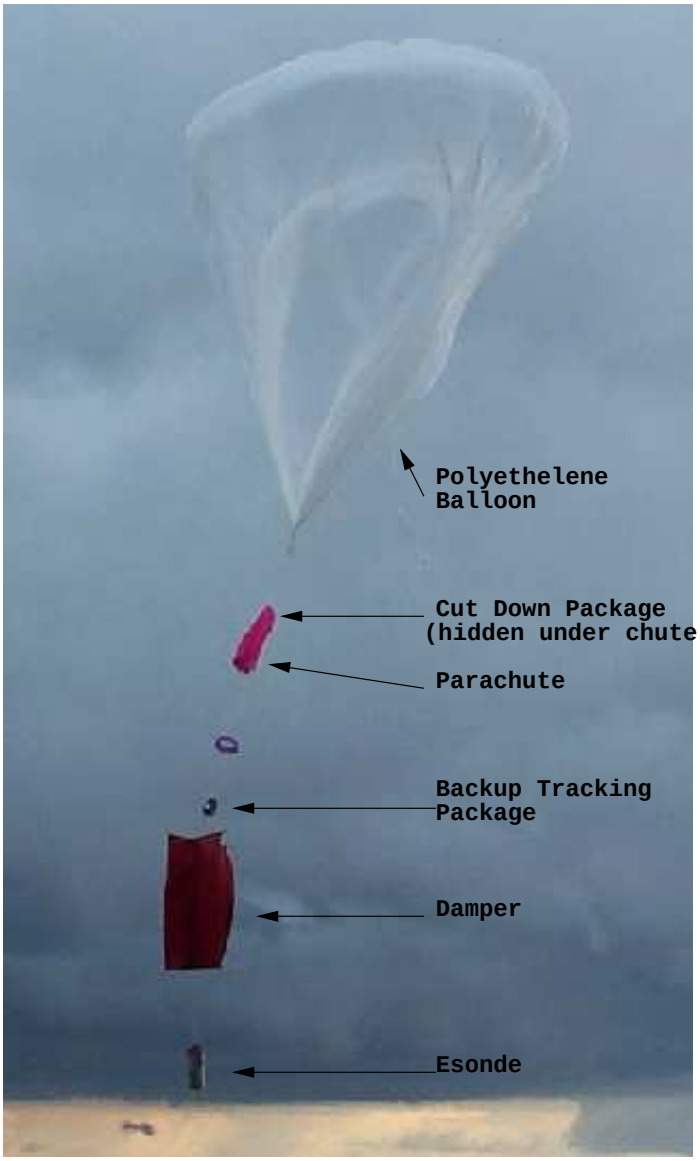


Figure 2.3: Picture of the entire balloon train in flight.

has a timer and an pressure based altitude sensor. If either the timer expires or a predetermined altitude is reached the package heats up a wire which melts a mono-filament string connecting the parachute and lower parts of the train to the balloon. This is to prevent the E-sonde package from landing too far away from the launch site. The timer was set to 43 minutes and the altimeter to a pressure of fixme mbar.

The parachute was constructed of nylon and used to slow the descent velocity of the E-sonde to 10 m/s before hitting the ground. Below the parachute was a small plastic ring. The purpose of the ring was to keep the parachute's cords from tangling and to make sure the chute filled with air.

After the parachute was a backup tracking package. It consisted of a GPS, battery, 70 cm ham radio and micro-controller. The package transmitted GPS coordinates using 1200 baud AFSK modulation in the APRS format on a frequency of fixme. The purpose of this package was to enable location of the E-sonde if the E-sonde stopped transmitting its telemetry position information during the flight.

The damper sits between the backup tracking package and the E-sonde. It's purpose is to slow down the movements of the E-sonde. The E-sonde's magnetometer is susceptible to errors from tilt so the damper was used to stop the swaying of the E-sonde package under the balloon. The goal was to keep the swaying of the E-sonde to less than 10^o from vertical.

At the bottom of the train is the E-sonde which records electric field changes from its four electrodes. The E-sonde stores all the data to the compact flash card but transmits its position and other telemetry data. The E-sonde

must be recovered in order to retrieve the electric field data data.

2.2 Sonde Layout External

2.2.1 Sonde Package

The electric field change sonde, referred to as the E-sonde for short, is show in figure 2.4 on page 9. The E-sonde package is an aluminum cylinder 0.34 m tall by 0.14 m in diameter. The aluminum cylinder is composed of two shell halves fastened onto the main support frame with screws. The weight of the E-sonde is 2.73 kg. On the outside of the sonde are four brass electrodes having a length of 0.110 m and a width of 0.090 m. The top of the sonde is covered with a plastic hat upon which the GPS is mounted. Protruding from the top of the E-sonde are four mount points for connecting to the balloon train. The bottom of the E-sonde has the user interface panel and an antenna. During flight aluminum tape is used to seal the edges of the aluminum halves and a plastic sheath is place over the cylinder to help seal out water. On the bottom of the E-sonde are stand-offs so that it can rest in the upright position if the antenna is removed.

Figure 2.5 on page 10 shows the E-sonde with the plastic top hat removed and one of the shells detached. This diagram shows how the internal of the E-sonde is composed of a series of circular plates holding various components are stacked on top of each other. The plates are separated by aluminum spacers which are held in place by four lengths of all-thread running the length of the sonde. The connectors where the shells attach can be seen. Also, on the top deck, a red electrode connector can be seen.

Figure 2.4: Picture of the E-field sonde package.

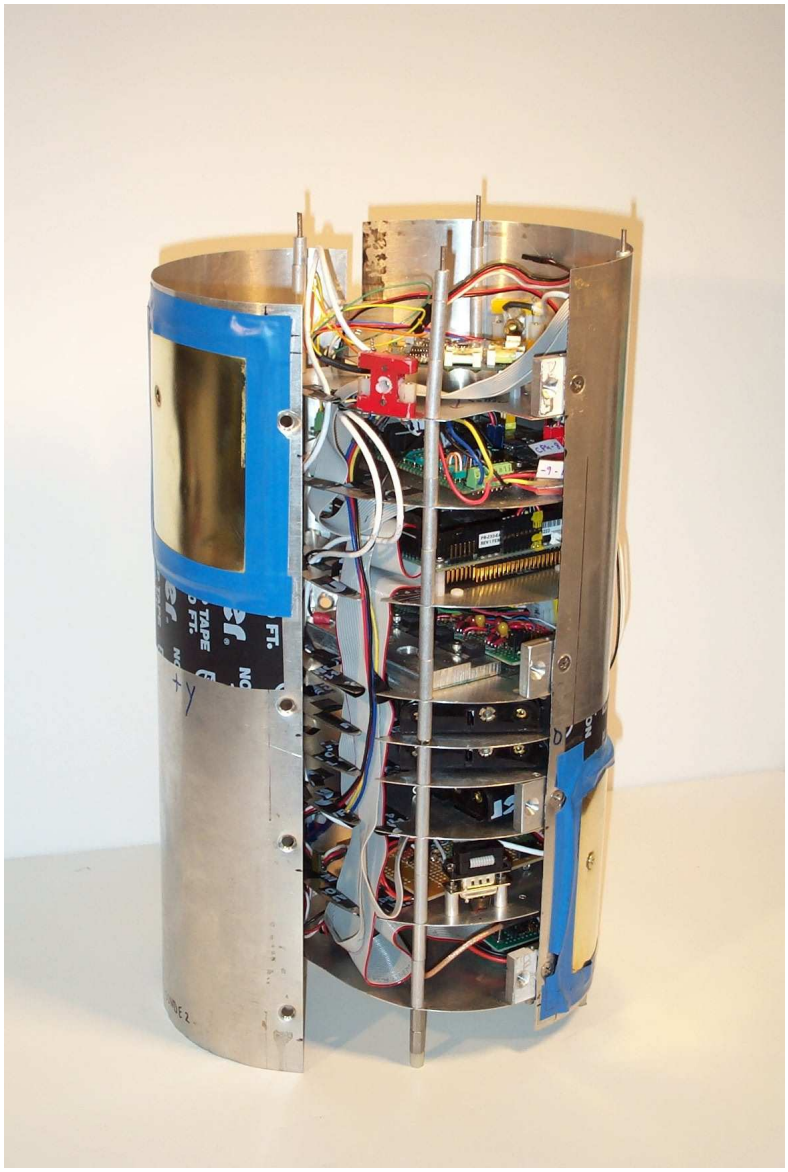


Figure 2.5: Picture of sonde outer shell open showing internal plates.

2.2.2 Sonde User Interface

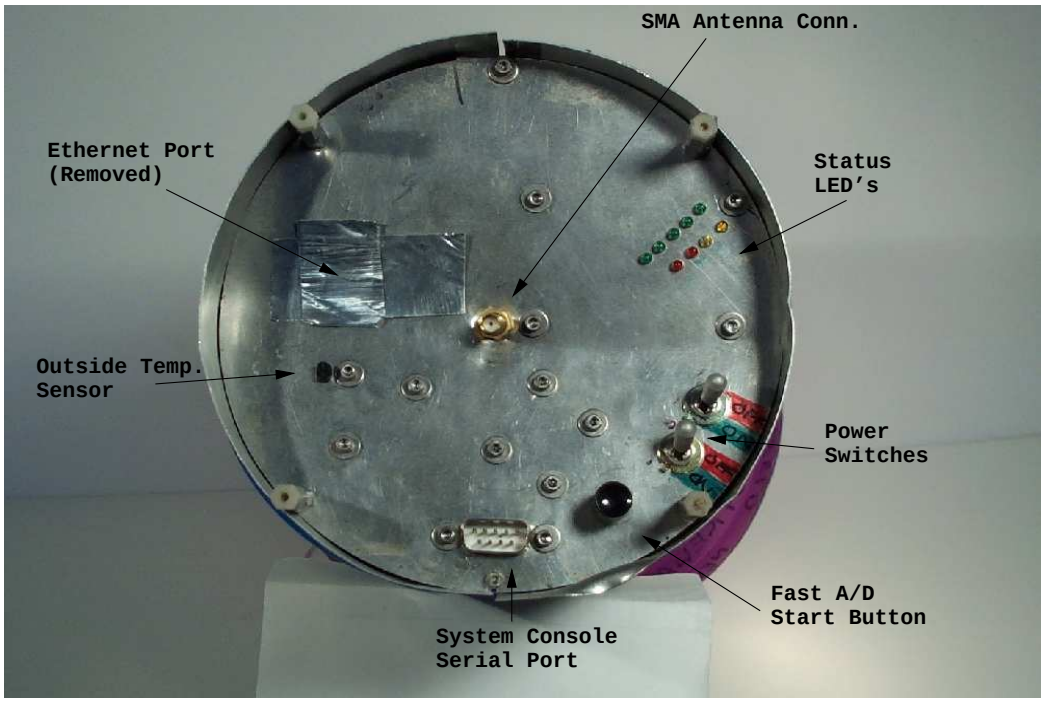


Figure 2.6: Picture of bottom plate of sonde with user interface.

Figure 2.6 on page 11 shows the bottom of the E-sonde where the user can control and monitor the E-sonde status. There are two power switches used to turn the E-sonde on and off. One of the switches controls the three of the 5V regulators and the 9V battery. The second switch turns on the -9V supply and -5V regulator. The two switches were used keep the negative voltage from causing problems with the 5V regulator grounds.

The fast A/D start button is pressed once just before launch to begin the fast data acquisition. The purpose of this button is so the E-sonde does not have to collect the fast data while the sitting on the ground during preflight. When the button is pressed the fast A/D indicator LED will come on.

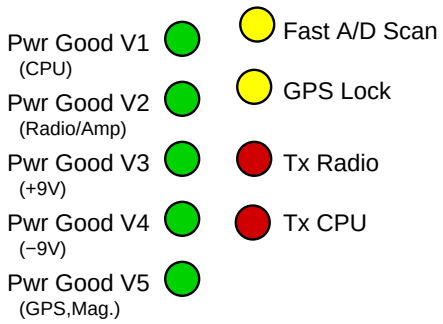


Figure 2.7: Status LED panel diagram.

Figure 2.7 on page 12 shows the nine indicator LEDs. These indicator LEDs are used to see what the status of the E-sonde is during preflight. There are five green LED's that show that the either the battery voltage or regulator voltages are at or above a safe level programmed into the software. The selection is chosen by a set of jumpers on the power regulator board.

There are two yellow LEDs. One lights up to show a fast A/D scan is occurring. Since these scan are back to back as much as possible it may appear to be lit most of the time with a slight flicker. The second yellow LED indicates that the GPS has acquired a position lock and knows where it is.

The two red LEDs are used to indicate transmission of data. The first one indicates that the radio receiver is powered and it blinks as data is received. The second one is used to show that the balloon_tx program is sending data to the modem. Between the two LED's the user can validate the program is trying to send data and the radio modem is receiving it.

The system console serial port connector is a DB9 male which is used by the preflight computer to see the system boot console. This allows monitoring of the CPU as the system boots. The user can also log-on to the

system and start the balloon_tx program as well as check the NTP daemon to make sure the GPS has locked the time.

The system originally had an Ethernet port which was removed due to damage to its connecting wire. This port (when installed) can be used to transfer the data files off of the E-sonde. This is useful if the fast data acquisition mode is used while on the ground to test the electric field sensors.

2.3 Sonde Layout Internal

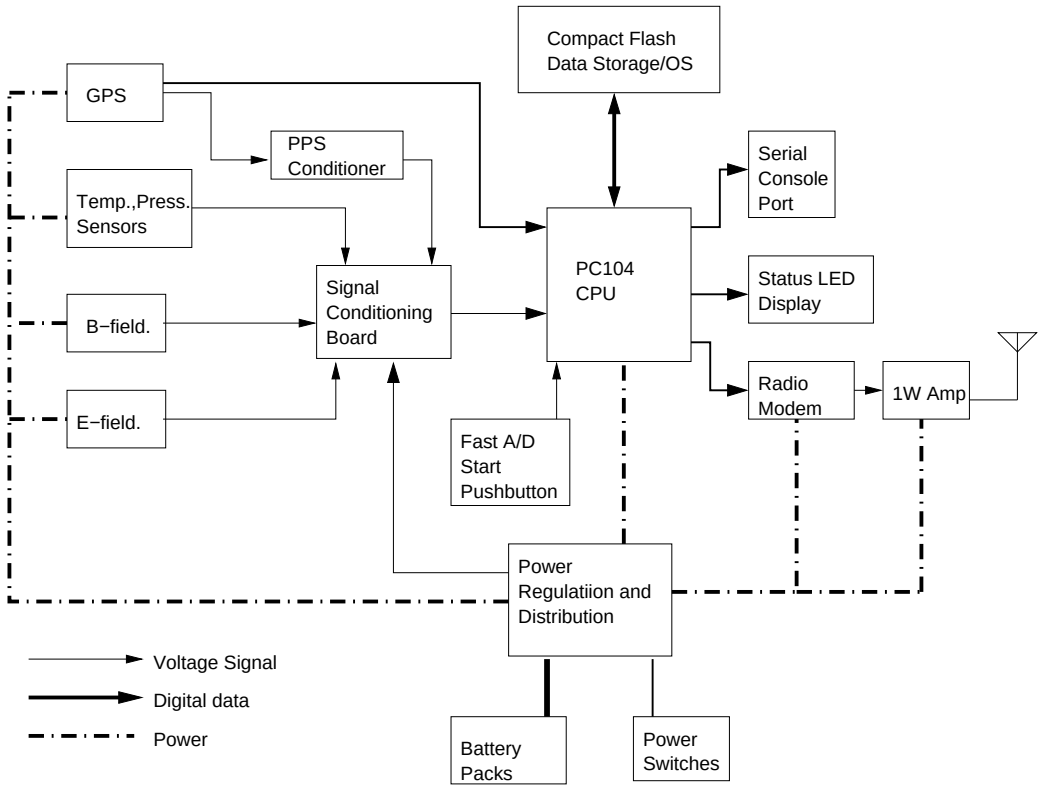


Figure 2.8: E-Sonde system block diagram.

The interior of the sonde is composed of nine plates to which the major components are mounted to the top or the bottom. Figure 2.9 on page 15 shows the location of the major components. Four threaded rods pass through the plates and 0.25" aluminum spacers with lengths from 0.50" to 1.00" determine the spacing between plates.

Figure 2.10 on page 16 shows the threaded rods which hold the plates together. Plate 4 is secured between two 0.25" nuts. This center plate is used to maintain the reference position for aligning the other plates and spacers. The

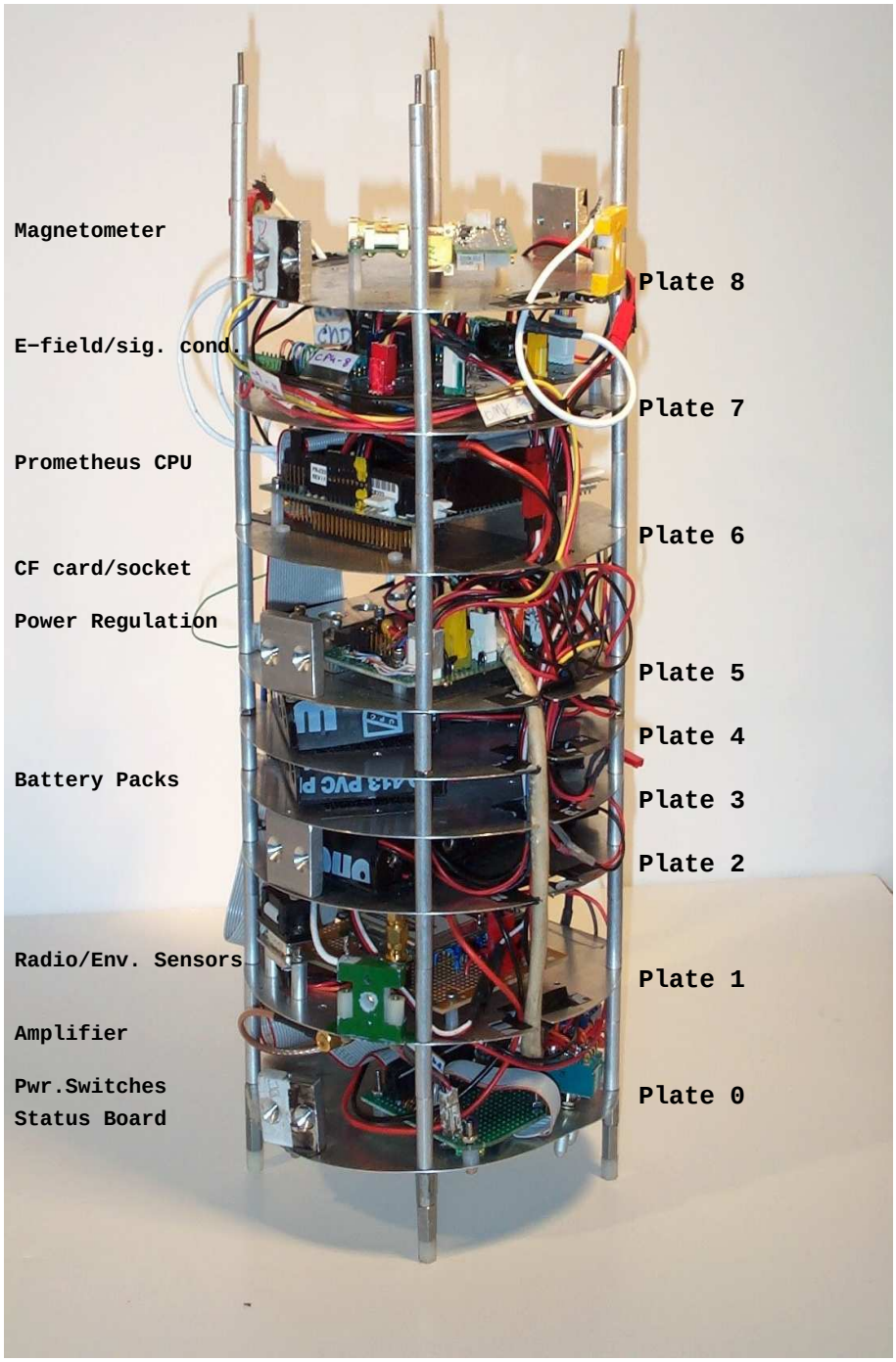


Figure 2.9: E-Sonde with shells removed showing major component locations.

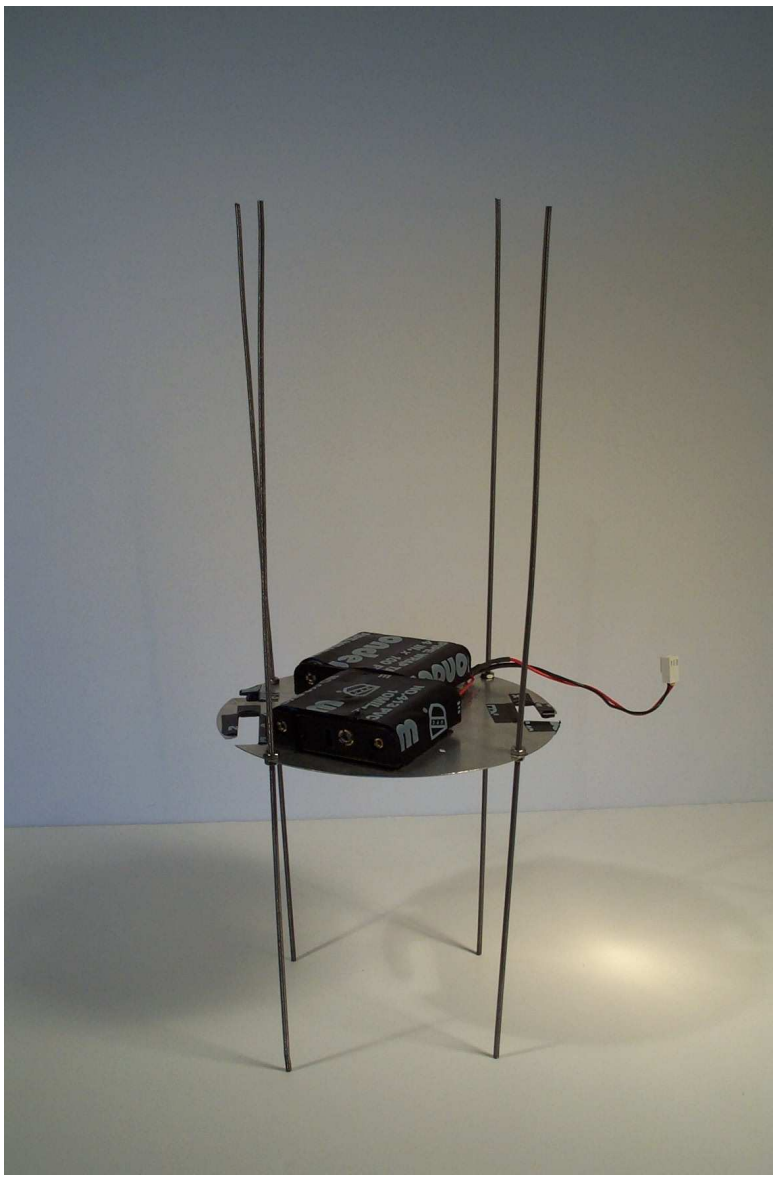


Figure 2.10: Picture of threaded rods that plates are mounted to.

Plates above this one must be removed sequentially off the top of the sonde from 8 to 5. The plates below this one must be removed sequentially from 0 to 3 from the bottom of the sonde.

Due to this construction replacing the batteries is time consuming. To replace the batteries one or two plates are removed from each end. This allows the remaining plates to be moved father apart from each other. The batteries can then be removed and replaced.

2.4 E-field Sensors

2.4.1 Sensor Layout

In order to be able to calculate a three dimensional electric field vector an array of four electrodes was chosen. There are two sets of electrodes aligned on the x-axis and two aligned on the y-axis. The pairs of sensors would be vertically spaced apart on the z-axis along the length of the cylinder. Each pair of electrodes face 180 degrees apart. Figure 2.11 on page 19 shows the positioning of the electrodes on the E-sonde's shell halves. In G on page 108 details of deriving the electric field vector from the sensors are given.

2.4.2 Electrode Design

Figure 2.12 on page 20 shows a close-up picture of the electrode. The electrode is 9 cm by 11 cm with an area of $9.9\text{x}10^{-3}$ m^2. A layer of electrical tape is place on the bottom side of the electrode and on the shell half to prevent it from making contact with the metal shell. Electrical tape is then place around the outside edges of of the electrode to secure it to the shell half. The electrode is made of brass and is 0.28 mm thick. A brass screw is used to electrically connect the electrode to a connector inside the E-sonde on plates 1 and 8. The connector positions can be seen in figure 2.9 on page 15. Figure 2.22 on page 35 shows a close up detail of the electrode connector. In the figure a brass nut can be seen where the screw enters. The connector has a wire which then runs to the signal conditioning board. Each of the connectors has been painted a unique color a for reference purposes.

Figure 2.11: Sonde outer shells showing location of electrodes.

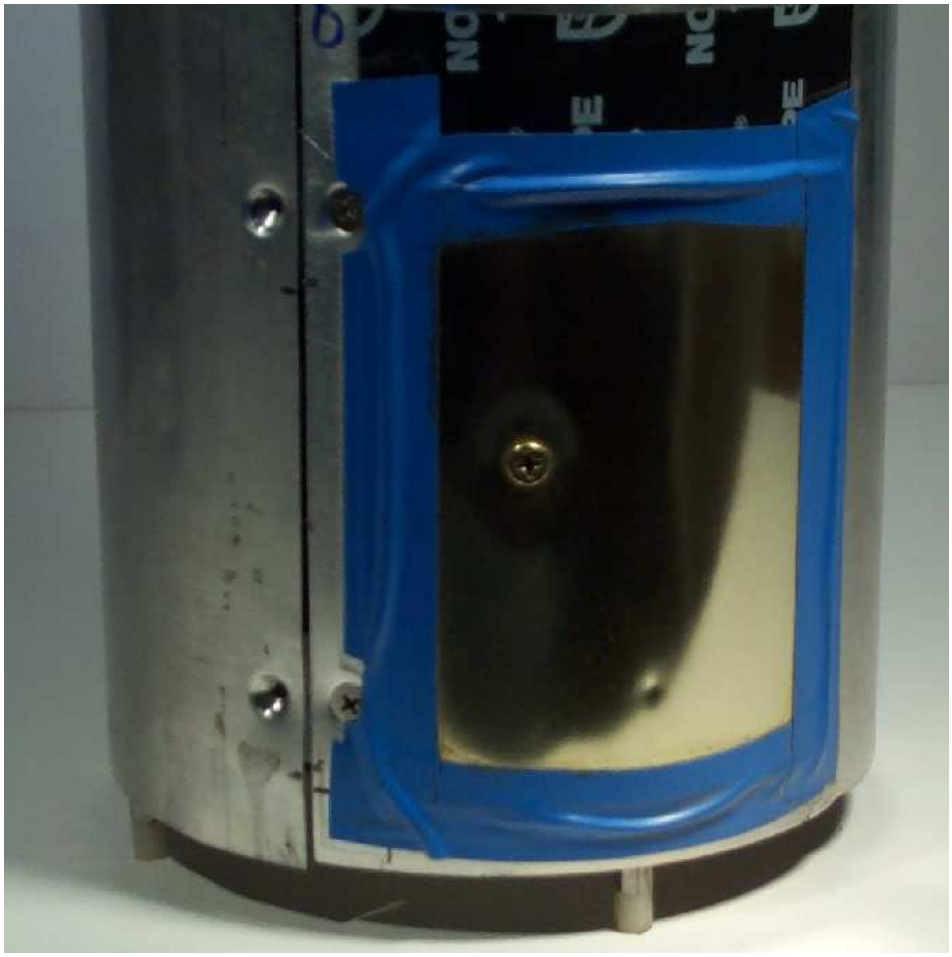


Figure 2.12: Close-up picture of electric field sensor plate.

2.5 E-field Sensor Board

2.5.1 Charge Amplifier Basics

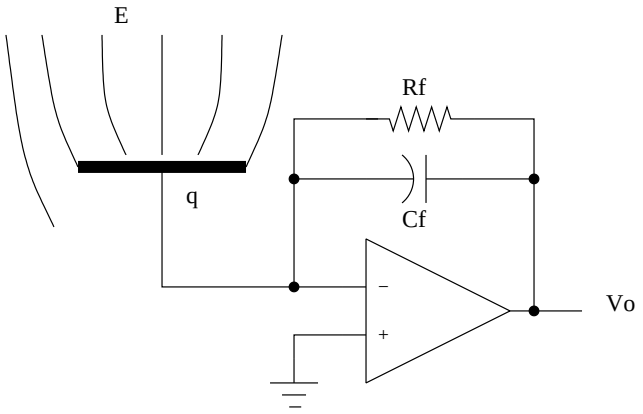


Figure 2.13: Schematic of basic charge amplifier.

The function of the charge amplifier is to convert a charge induced on an electrode into a voltage. It is also used to convert a change in charge into a change in voltage. Figure 2.13 on page 21 shows a schematic of a basic charge amplifier circuit. When a charge is induced on the electrode by an electric field the voltage at the negative terminal changes from zero. Since the positive terminal is grounded the negative terminal is effectively ground or to be more accurate a virtual ground. The op-amp compensates by changing the voltage output until the change is voltage across the capacitor induces a current which effectively neutralizes the current from the induced change in charge. Therefore, the output voltage is proportional to the change in charge on the plate which represents the change in e-field.Equation 2.1 shows the basic relationship between the electric field magnitude and the output voltage.

$$E \propto -\frac{V_0}{C_F} \tag{2.1}$$

The purpose of C_F is to discharge the capacitor. If not discharged the V_o will rise until reaching the maximum output voltage the amplifier can produce from successive changes in charge in one direction. The decay constant for the output voltage is $\tau = C_f R_f$. This implies that the events to be observed must take place in a time frame smaller then τ.

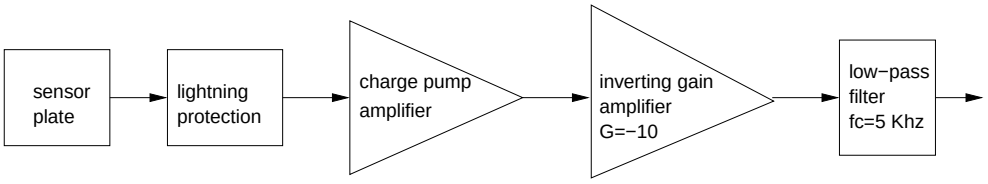


Figure 2.14: Block diagram of a single e-field sensor circuit.

Figure 2.14 on page 22 shows a simplified block diagram of the schematic in figure 2.15 on page 24. Equation 2.2 shows the relationship between the output voltage and the electric field change. A is the area of the electrode. G is the gain of the system. C is the value of feedback capacitor C_F. ΔE is the change in electric field. ϵ_0 is the permitivity of free space. The design is set so that a 1 V output represents a field change of 24.8 kV/m.

$$\Delta V = -\epsilon_0 \Delta E A \frac{G}{C} \tag{2.2}$$

The charge-amp in figure 2.15 is using a T-network feedback resistor configuration. Equation 2.3 shows how to calculate the R_{eq} value for the network. Since R_1 and R_2 are 1 M ohm, and R_3 is 21.5 K ohm, the resultant R_{eq} is 48.5 M ohm. C_1 is 22 nF giving a τ=1.0673 s.

$$R_{eq} = R_1 + R_2 + \frac{R_1 R_2}{R_3} \tag{2.3}$$

In the block diagram of Figure 2.14 a few additions other than the charge-amp are added. The first is a block for lightning protection. Details of the lightning protection circuit can be seen in the schematic in figure 2.15. They consist primarily of a neon discharge bulb and a zener diode. The circuits are designed to prevent large voltages from damaging the circuits further down stream. There are also two 10 MHz low-pass filters to prevent the RF signal from the radio transmitter from entering the charge-amp circuit. During the initial flights the protection circuitry along with the two 10 MHz low-pass filters were not installed on the e-field board due to time constraints.

After the charge-amp is a -10 gain stage and 5 kHz low pass filter. The gain stage inverts the signal so that a positive output voltage represents a positive change in field and the gain is used to increase the scale of the voltage range. The low pass filter center frequency is set by the Nyquist criteria for the A/D converter. Since the sample rate is 10 kHz the low pass is centered at 5 kHz to prevent aliasing in the data.

Figure 2.16 on page 25 and figure 2.17 on page 26 show the PCB (printed circuit board) for the e-field sensor board. The board is 10.2 cm by 5.1 cm in size and is mounted on plate 7 of the E-sonde in figure 2.9 on page 15. The electrode connectors are color coded to make wiring to the proper electrode connector more error proof. After populating the circuit boards there were a few initial problems discovered. There was a feedback oscillation problem with the op-amps, but more bypass capacitors eliminated the problem. Another problem was the size of the pads for the op-amp IC were too small requiring some tricky soldering and produced some reliability problems during

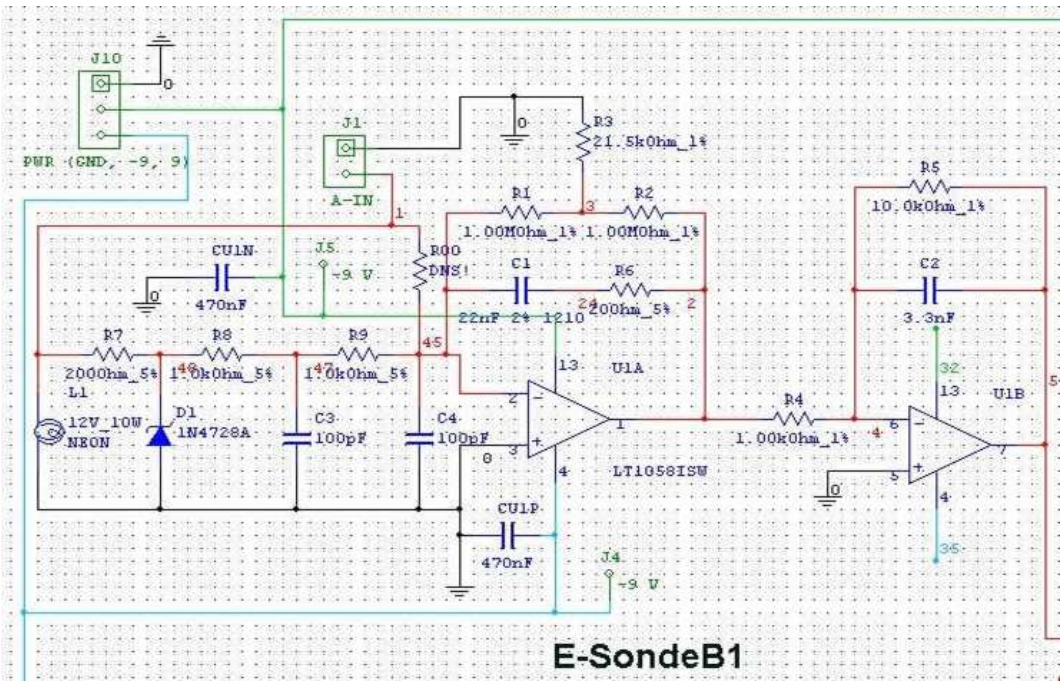


Figure 2.15: Schematic of a single electric field sensor circuit.

board checkout. After the bad solder joints were re-soldered the boards worked properly. Some of the early boards had problems with a single channel have large offsets. Or the channel would stay at the rail voltage if signal output went close to the rail. The cause was never determined, but the board used in the second and third flights board #3 never showed either problems.

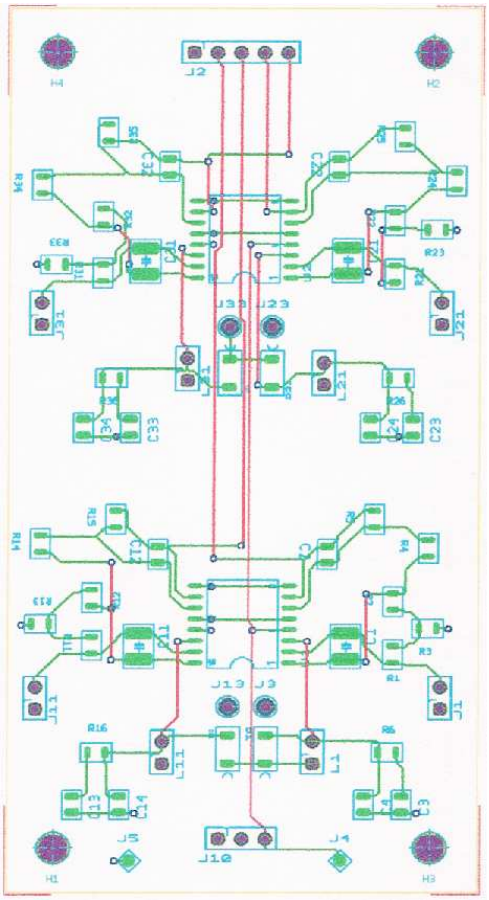


Figure 2.16: PCB mask for e-field sensor board.

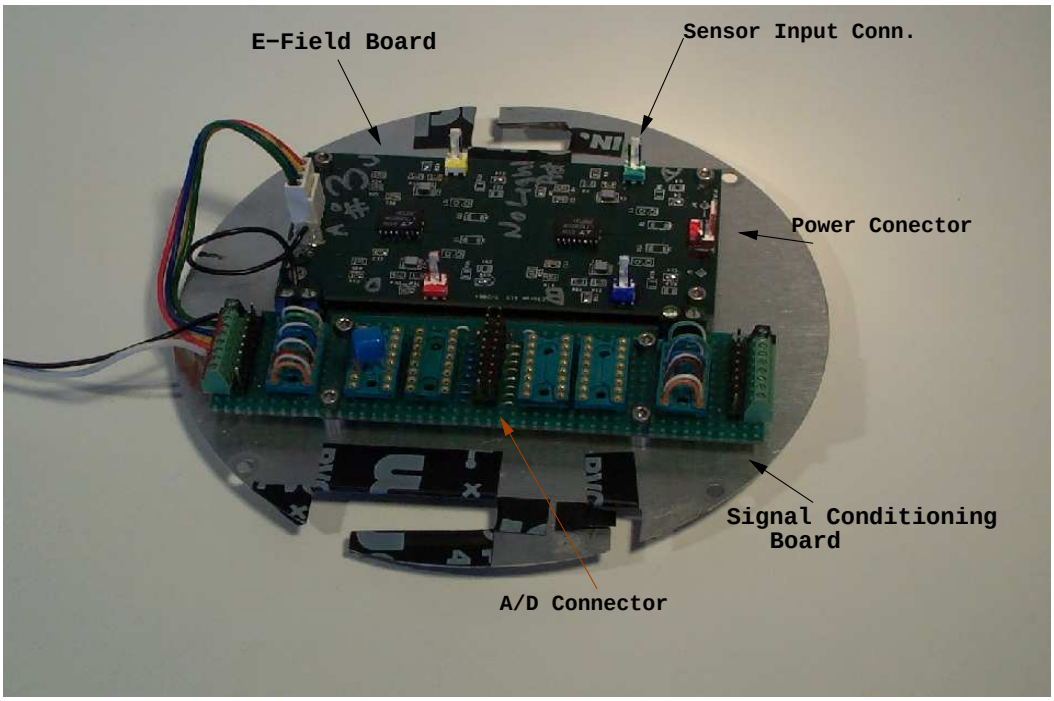


Figure 2.17: E-field and signal conditioning boards mounted to plate.

2.6 A/D

The data acquisition unit (DAQ) for the E-sonde was built onto the Prometheus PC/104 CPU board. The converter had a multiplexed 16 channel input block that passes through a programmable gain amplifier. Figure 2.18 on page 27 shows a block diagram of the DAQ system. The analog to digital converter (ADC) has a 16 bit resolution and was programmed for ±10 V. This gave a minimum voltage resolution of 305 uV. The maximum sample rate for the system was 100 Ksps. The DAQ has a 48 sample buffer before being transferred to the CPU with an overflow indicator.

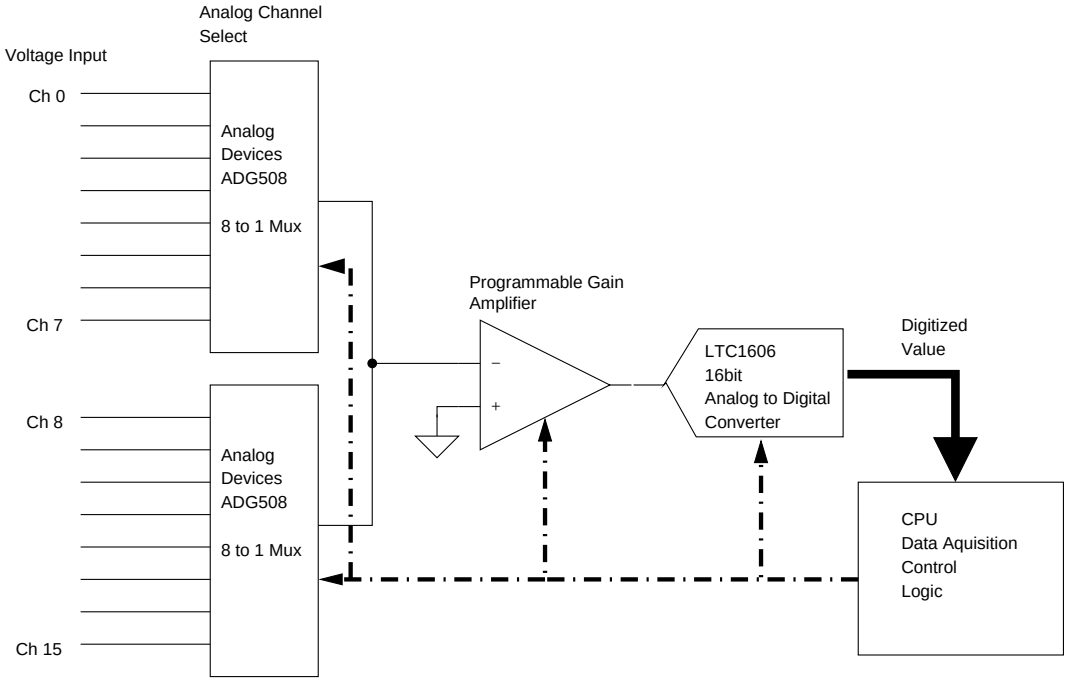


Figure 2.18: Block diagram of data acquisition system.

During testing of the system, it was determined that a sample rate of 10 Ksps over eight channels for two seconds was the maximum sustainable

Table 2.1: Channel allocation and sample rate for the A/D converter.

A/D Channel	**Input Signal**	**Sampling**
0	E-field 1	fast,slow
1	E-field 2	fast,slow
2	E-field 3	fast,slow
3	E-field 4	fast,slow
4	PPS	fast,slow
5	Mag. x-axis	fast,slow
6	Mag. y-axis	fast,slow
7	Max. z-axis	fast,slow
8	V batt./reg. 1 (CPU)	slow
9	V batt./reg. 2 (GPS,Mag.,Temp.Press.)	slow
10	V batt./reg. 3 (E-field Board +V)	slow
11	V batt. 4 (E-field Board -V)	slow
12	V batt./reg 5 (Radio/Amplifier)	slow
13	Atm. pressure sensor	slow
14	Internal temp. sensor	slow
15	External temp. sensor	slow

Note: Fast is 10 Ksps and slow is programmable but typically every 17 s.

data rate for the system without sustaining overflow errors. The system also had a relatively high input impedance of 300 Ohms. Since some of the input channels had high impedance sources such as the temperature sensors and magnetometer there was some voltage carried over from the previously scanned channel. In particular, the x-axis from the magnetometer saw the PPS signal pulses. The internal temperature sensors saw the voltage from the pressure sensor. The x-axis input was fixed by a capacitor places in parallel, but the internal temperature sensor corruption was not fixed during the three flights. Table 2.1 on page 28 shows the allocation of the channels and there respective sample rate.

2.7 A/D Signal Conditioning Board

The signal condition board is used to pre-condition the sensor signals before arriving at the DAQ input. The board is located on plate 7 with the e-field board. The signal conditioning board has three machined DIP sockets for placing resistors, capacitors, or shunts in. Figure 2.17 on page 26 shows a picture of the board. Figure 2.19 on page 29 shows a schematic of the board and possible configuration options.

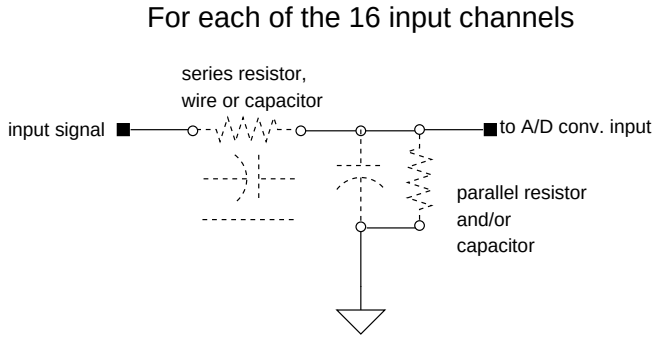


Figure 2.19: Schematic of A/D signal conditioning circuit.

The board can be used to implement high-pass or low-pass first order filters as well as in-line and parallel resistors and capacitors. This allows for a wide range of signal conditioning possibilities for dealing with noise and other problems of the input signals without have to solder components down. During the balloon flights a parallel capacitor of 1 uF was added to the x-axis of the magnetometer so the PPS signal would not bleed over into its value.

2.8 CPU and Storage

In order to facilitate rapid prototyping and flexible software design a microcomputer platform that could run existing operating systems was chosen. This system allowed for development of the E-sonde system software on a PC a which could then be transferred to the E-sonde. The E-sonde CPU system needed to run the Linux OS, fit within a 15 cm cylinder, run off a single voltage supply, and be able to run for several hours off of batteries. In order to meet these requirements a PC/104 platform was determined to be optimal.

The Prometheus PC/104 platform from Diamond Systems was chosen because it had a built in A/D converter and digital I/O with support for Linux. It had an IDE interface which could be connected to a compact flash (CF) card. It had four serial ports of which a minimum of three were needed for the design. It also had a stand alone development platform which would allow for early design of the system software. Table 2.2 on page 31 shows the specifications for the Prometheus board. Figure 2.20 on page 32 shows a picture of the CPU board mounted to the plate.

The CPU board was placed on plate 6 just above the voltage regulator to minimize the lead lengths between the voltage regulator and the CPU board. Since the Prometheus CPU is a digital system with a high current draw it can produce RF noise and voltage transients. By minimizing the length of its power leads the RF noise from the power leads should be minimized as well. The CPU board was given its own voltage regulator so it would cause problems with the more sensitive analog systems such as the GPS and sensors.

The E-sonde uses a compact flash card for both data storage and to

Table 2.2: PC/104 CPU Specifications.
Diamond Systems Corp. Prometheus PC/104 integrated system

Processor	486-DX2 100 MHz
Memory	32 MB SDRAM 50 MHz
BIOS	Boot from CF, serial boot monitor
Serial Ports	4 each with 115 Kbps
USB Ports	2 Ver 1.1
IDE	44 pin connector, 2 devices
Ethernet	100 BaseT full duplex
Other Ports	PS/2 keyboard & mouse,speaker,LED's
Clock	Real-time battery backed
A/D	16 bit, ±10 V, 16 Channel, 100 Ksps aggregate
Digital I/O	24 pins, 3.3 V and 5 V compatible
Supply Voltage	5V at 1.10 Amps
Temp. Range	-40^{o}C to +85^{o}C
Size	9.59 cm x 9.02 cm
Weight	84.6 g

hold its operating system and control programs. Since the system is capable of acquiring data at the rate of 9.6 MB/minute and a typically balloon flight last 45-60 minutes at least 576 MB of data storage is needed. The CF must also hold the OS which takes up about 64 MB.

The data storage device must be able to operate in sub-freezing temperatures and low pressure readings. It must also have very low power requirements and be able to sustain the shocks of landing and stress. In order to meet these requirements a 1 GB Sandisk CF card was chosen.

The only limit encountered with the cards was a slow data write transfer rate of 1.6 MB/s. This limited the sample rate of the A/D converters. There was also a problem with the A/D converter driver having buffer overflows if the system was writing the CF at the same time data acquisition was occurring.

Figure 2.20: Prometheus CPU board mounted to plate.

Figure 2.21 on page 33 shows a picture of the CF card and CF to 44 pin IDE adapter card mounted to the bottom of plate 6.

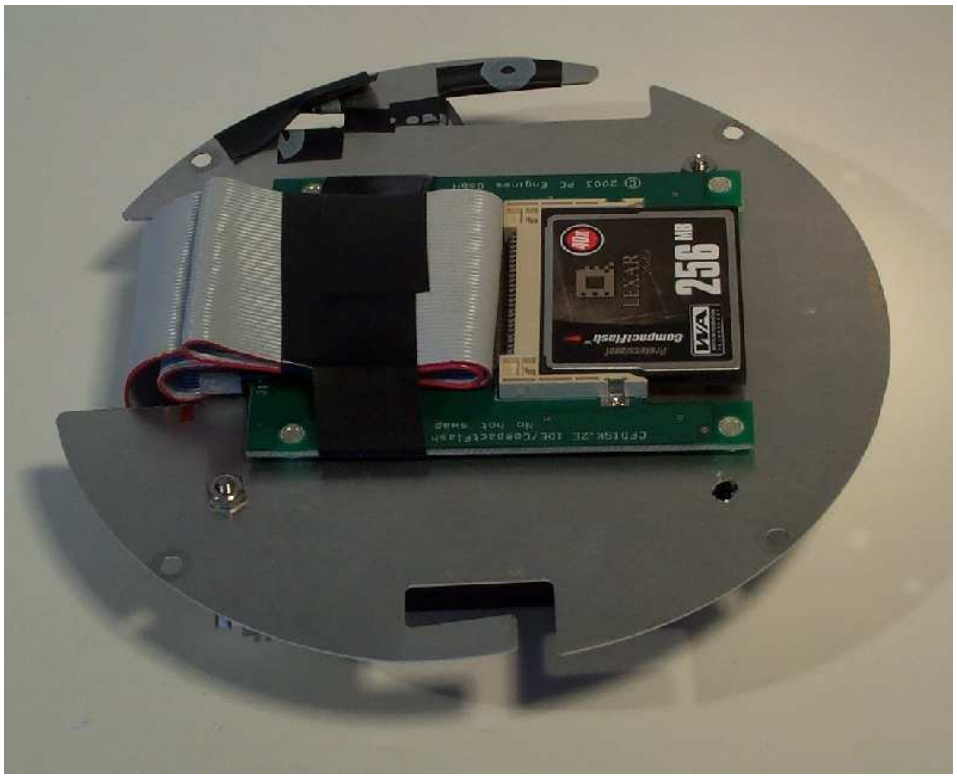


Figure 2.21: Compact Flash and IDE interface board mounted to plate.

2.9 Magnetometer

The magnetometer was used in the E-sonde in order to determine its azimuthal orientation and how much the z-axis was tilted from perpendicular to the Earth's surface. The magnetometer did not have a g-force sensor so the magnitude of the tilt was used to indicate how reliable the azimuthal orientation was. If the there was a large deviation in the z-axis value of more than a few degrees then the azimuth could not be calculated from the x-axis and y-axis values. Figure 2.22 on page 35 shows a picture of the magnetometer mounted on the E-sonde plate. The magnetometer was place near the top of the E-sonde in order to reduce electromagnetic interference from other circuits.

The magnetometer was sampled at the same rate as the electric field sensors so there would be good time correlation between the magnetometer values and the e-field values. Table 2.3 on page 36 shows the specifications for the magnetometer. The magnetometer has a response of 400 Hz. This is slower then the 10,000 kHz sample rate, but is fast enough to detect significant rotation of the sonde.

On earlier flights, in post data analysis, noise from the PPS input was showing up on the magnetometer x-axis. A 1 uF capacitor was added in parallel on the signal conditioning board. The addition of the capacitor was successful in eliminating the PPS input noise.

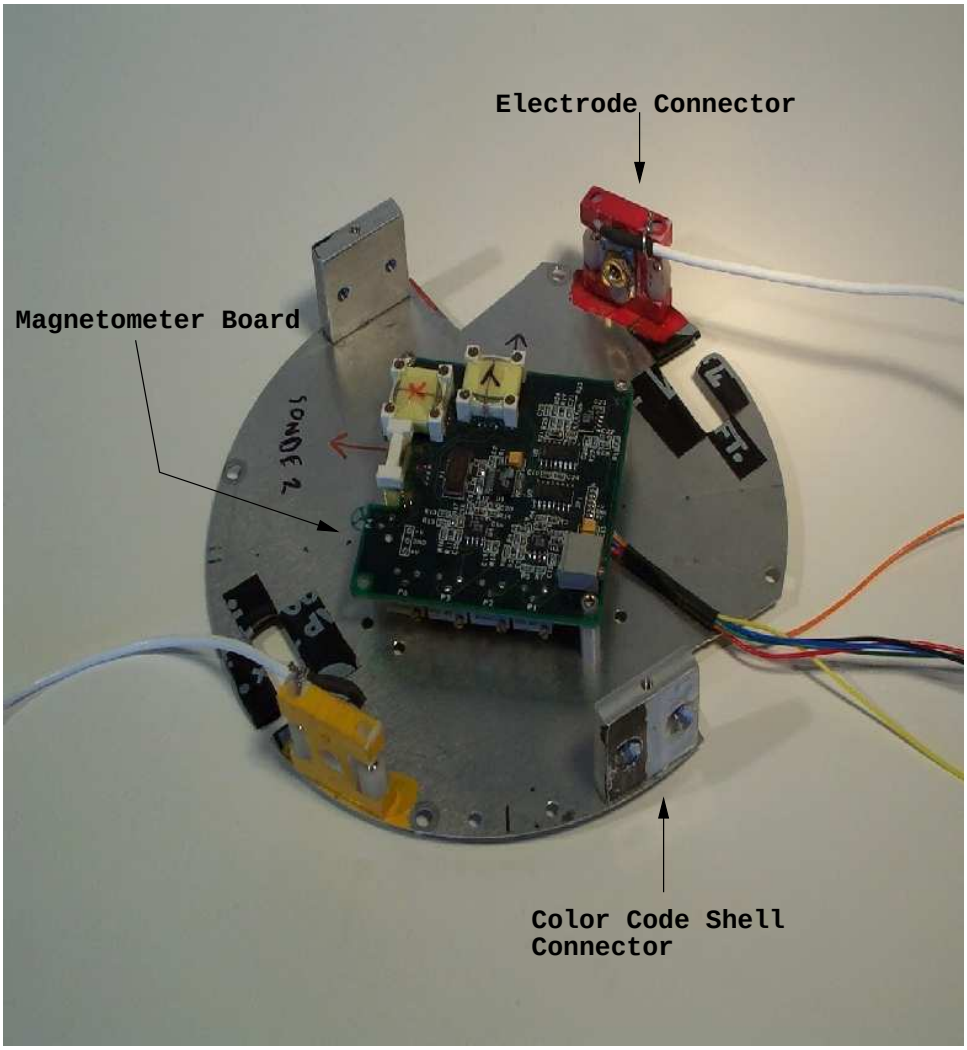


Figure 2.22: Magnetometer board mounted to plate.

Table 2.3: Magnetometer Specifications.
Applied Physics Model 113 PC Board-level 3-Axis Fluxgate Magnetometer

Noise level	$3x10^{-6}$ G RMS/$Hz^{1/2}$
Frequency response	DC to 400 Hz (-3dB)
Linearity	$\pm 0.1\%$ of Full Scale
Drift in zero with temperature	$< 10^{-4}$ G/^{o}C
Drift in scale factor with temperature	$< 0.05\%$ FS/^{o}C
Sensitivity	4.00 V/G
Orthogonality	$\pm 2^o$
Alignment of sensor package with reference surfaces	$\pm 2^0$
Size	2.5"x2.5"x0.75"
Weight	30 g
Power input	+5VDC at 28ma, -5VDC at 7ma

2.10 GPS

The GPS (Global Positioning System) device is an essential part of the E-sonde system. It has dual purposes of tracking time and position. The time output is used in a high resolution way of feeding the PPS output to the A/D so the signal is recored along with the data. This allows for aligning the data with the UTC time to the nearest 100 uS. The system also uses the serial output of the GPS to synchronize the system time to the GPS UTC time with the NTP daemon. Table 2.4 on page 38 shows the specifications for the GPS. Both the NTP daemon and the balloon_tx program use the NMEA GPGGA string to extract their time and position information. All other strings were turned off and the GPS was programmed to transmit its GPGGA string every second.

The GPS sends the position information through the serial port as well. The position information contains latitude, longitude and altitude. It is important to know where the E-sonde is so that the data e-field vector can be correlated with the LMA data.

Figure A.1 on page 82 shows how the GPS is wired to the CPU and what ports are used by the balloon_tx program to read the GPS. The balloon_tx program can pole the GPS at any interval,but 17 seconds was chosen for the three initial flights.

Figure 2.23 on page 38 shows the GPS mounted on the E-sonde top cover. The GPS was placed on the top so that its antenna would have the maximum view of the sky. The GPS case is waterproof and made of tough plastic so that rain and small hail would not affect it.

Table 2.4: GPS Specifications.
Garmin model GPS35-LVS

Weight	124.5 g
Size	57 mm(w)x96 mm(l) x 27 mm(h)
Operating temperature	-30^{o}C to +85^{o}C(internal)
Voltage	+3.6VDC to 6.0VDC 150mV ripple
Current	120 mA typical 140 mA max.
Satellite tracking	12 or 11 with PPS active
Position update rate	1 sec.
Acquisition time	45 sec. cold to 5 min. sky search
Position accuracy	15 meters RMS
PPS accuracy	$\pm$1 uS at rising edge of PPS pulse
Interface	RS-232 compatible level with baud of 300 to 19200
Data format	NMEA 0183 ver. 2.0 ASCII output

Figure 2.23: Garmin GPS-35 LVS mounted to sonde top cover.

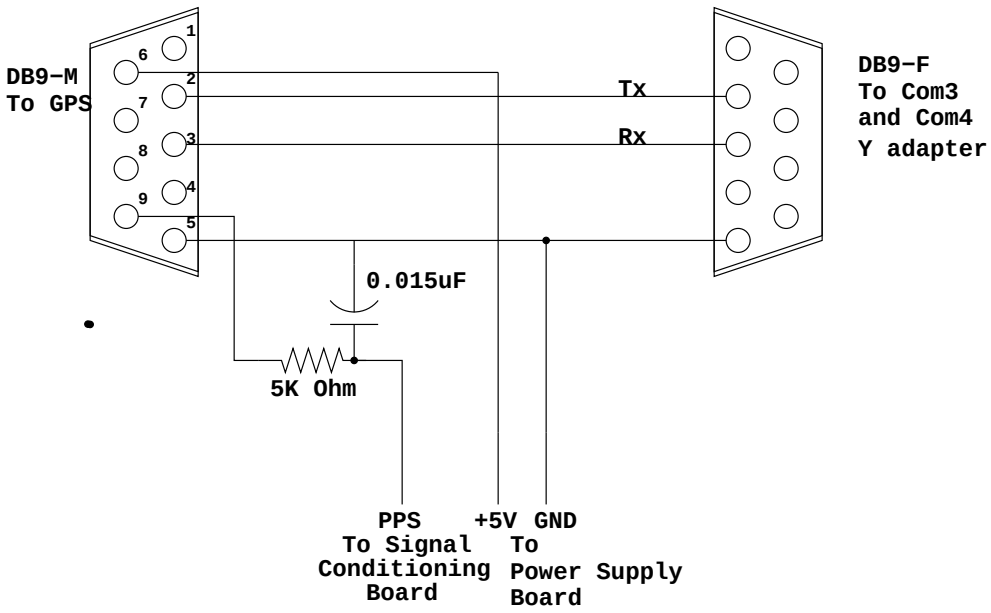


Figure 2.24: Schematic of GPS connection to system with PPS filter circuit.

The GPS PPS signal passes through an RC circuit to convert the pulse into an exponential ramp. Figure 2.24 on page 39 shows the circuit used to modify the PPS signal. This modification was done so that it would be easier to extract which sample the PPS edge was on. The post analysis data program uses the time constant of the RC to do an exponential fit of the data and determine exactly which sample the PPS pulse started on. The PPS pulse is a square wave with a period of 100 mS on and 900 mS off. The edge of the off to on determines when the second began that is transmitted in the next serial output message.

2.11 Power Control Board

The power control board regulates and distributes power from the battery packs to the other subsystems. Figure 2.25 on page 40 shows a picture of the board mounted to plate 5. The board holds four voltage regulators, a set of voltage monitor select jumpers, and connectors that run to battery packs and subsystems. The three five volt positive regulators are mounted to a large aluminum heat sink. The jumpers select whether the DAQ system monitors the battery voltage or the regulated voltage.

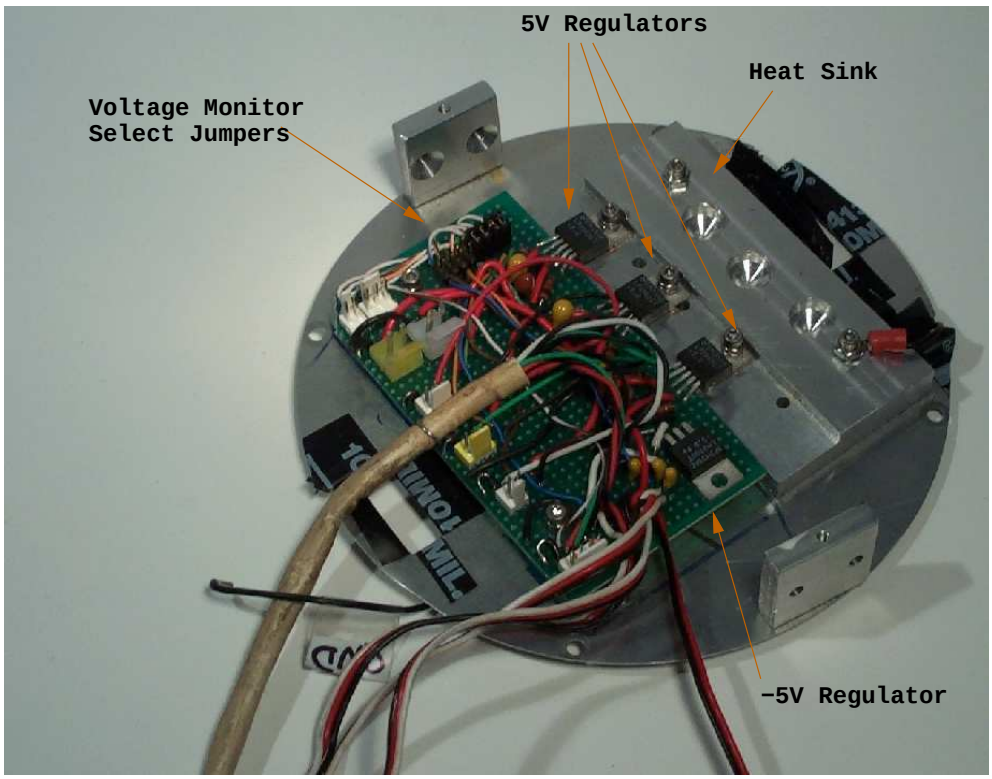


Figure 2.25: Power regulation and distribution board mounted to plate.

Figure 2.26 on page 42 shows a schematic of the power board. In this diagram the power distribution scheme for the battery packs can be seen. Due

to the higher current draw of the radio amplifier and the CPU two battery backs are attached to each regulator. The individual 5V positive regulators (LP3963) are capable of handling 3.0 A each. The switch for the system is used to turn on the three positive regulators and connect the 9 V batteries to the power board. When the switch is in the off position the -9 V and +9V batteries are disconnected. The LP3963's have there shutdown pin shorted to ground which turns off the output voltage. When the switch is in the on position the CPU battery pack provides a 6V signal to the shutdown pin through a 10K ohm resistor. In the actually construction of E-sonde 3 the 3PDT switch was replaced with two single pole switches. One switch was wired to the 9V batteries and the other controlled the 5V regulators.

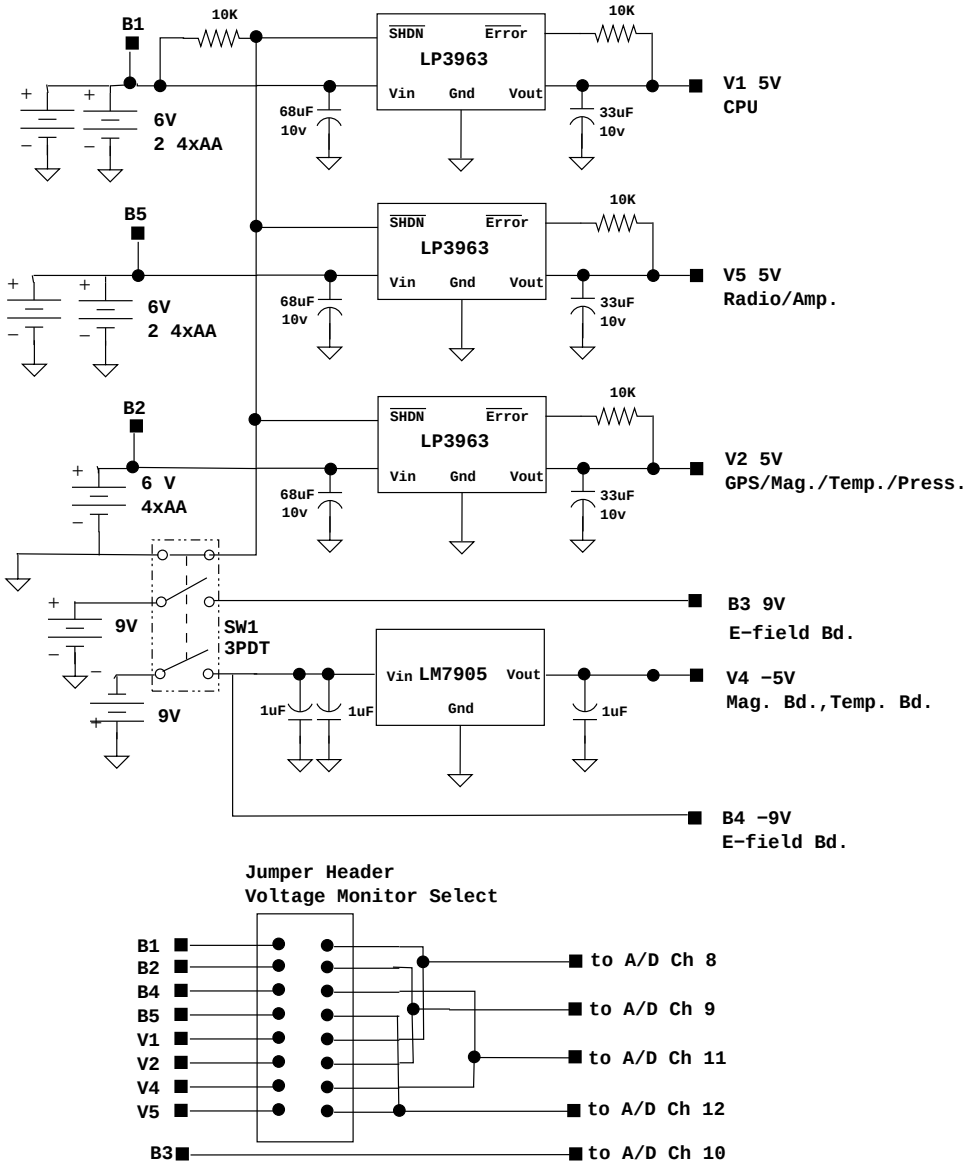


Figure 2.26: Schematic diagram of power control board..

Table 2.5: Radio modem specifications
Maxstream 9XStream OEM RF module

Outdoor Range	11 km w/dipole ant.
Data Throughput	19,200 bps
Transmit Power	100 mW(20 dBm)
Receiver Sensitivity	-110 dBm
Frequency Range	902-928 MHz
Spread Spectrum	Frequency Hopping, 7 sequences, 25 freqs.
Power	5 VDC($\pm$0.25V) 50-150 mA (RX vs.TX)
Size	4.06 cm x 7.18 cm x 0.89 cm
Weight	24 g
Operating Temp.	-40^{O}C to 85^{o}C
Ant. Connector	RPSMA 50 Ohms

2.12 Radio TX/Sensor Board

The radio transmitter and sensor board has two functions that are independent and combined on one board only for conveniences. The board is shown in figure 2.27 on page 45 mounted to plate 1. In the picture the Maxstream $9XStream^{TM}$ 900 MHz radio modem can be seen plugged into the board. The radio board contains a circuit to convert from the 5V logic levels of the modem to the RS-232 level of the CPU. A MAX232 IC from Maxim Semiconductor was used to perform the conversion. Figure 2.28 on page 46 shows a detailed schematic of the circuit for converting the signals. The connector pin out detail for the Maxstream is also shown. Since there is a 100 mA difference in supply current between receive and transmit mode there was an addition 33 uF capacitor added to the voltage supply pins at the connector. Table 2.5 on page 43 shows a listing of the radio modems specifications.

Since the radio only had a maximum throughput of 19,200 bps or

approximately. 2 KB/s it was not feasible to transmit the higher sample rate data which occurred at 160 KB/s during the time of the flight. The modem was used to transmit the position and slow A/D data as well as the maximum voltages for the fast A/D channels. The transmit and receive modems used in the E-sonde systems were set up for one way data communication. However, in the future it is possible to use the modems in bidirectional data communication.

The other part of the radio board is a temperature sensor and pressure sensors. A remote temperature sensor that is located on the bottom of the E-sonde also connects to this board. The schematics for the pressure and temperature sensors is show in figure 2.28 on page 46 and the pressure sensor can be seen on the picture in figure 2.27 on page 45.

The pressure sensor used was a Motorola 5100 AP which gave a voltage output proportional to absolute air pressure. The sensor was temperature compensated, but since its temperature sensor was inside the E-sonde it could not fully compensate for outside air temperature changes. The sensor has a range of 15 to 115 kPa. The sensor was not calibrated before the flights.

The temperature sensor was a National Semiconductor LM34CAZ. It has a voltage output proportional to 10 mV/^{o}F and will output a negative voltage for values less than 0^{o}F. This negative output voltage required a -5V supply to be routed to the sensor. The sensors were not calibrated before the flights.

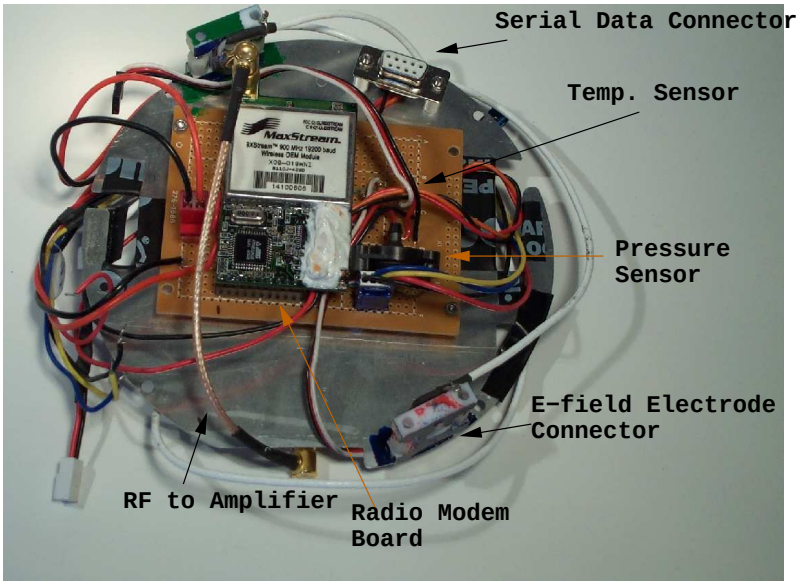


Figure 2.27: Radio modem, pressure and temperature sensor board mounted to plate.

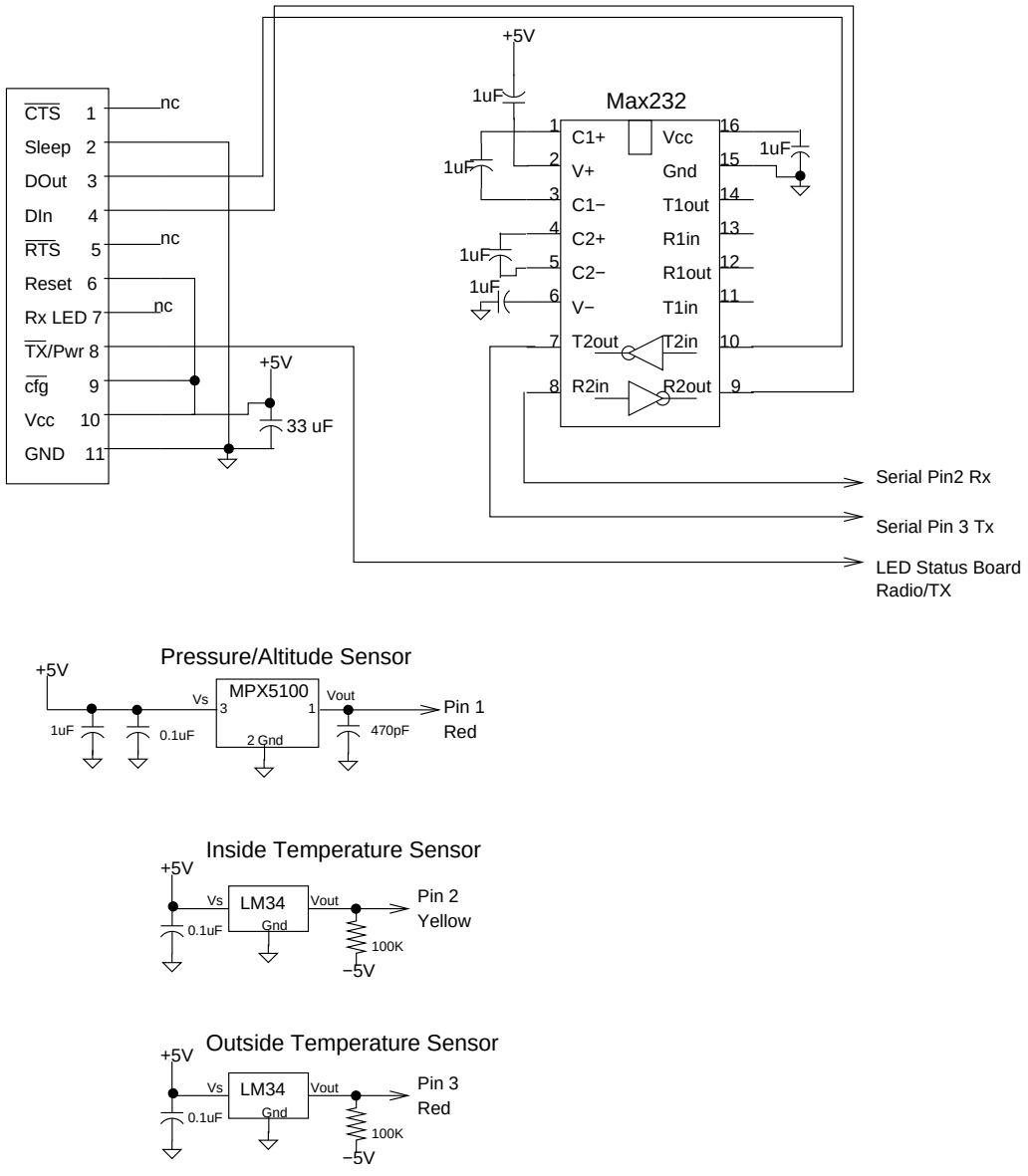


Figure 2.28: Schematic of radio modem, pressure and temperature sensor board.

2.13 Amplifier Board

The purpose of the amplifier board is to boost the signal of the radio modem to 1W of output power. The amplifier board is show in the figure 2.29 on page 48. The board is a Maxim MAX2602 evaluation kit. With the amplifier the E-sonde is capable of transmitting data reliably over 100 km. The board can draw over 1 A of current at 5V so the radio modem and this board were given there own voltage regulator. Since data packets are transmitted only every 18 seconds and the power draw is very low most of the time. The main problem encountered with the board was that Maxim does not want to see the evaluation kits in quantities greater than two. Table 2.6 on page 47 shows the specifications of the board.

Table 2.6: Radio amplifier specifications
Maxim MAX2602 Evaluation Kit

Output Power	1W (30dBm) at 836 MHz
Power Supply	2.7-5.5V at 1.1 A
Connectors	SMA 50 Ohm input and output
Temp. Range	-40^{o}C to 85^{o}C

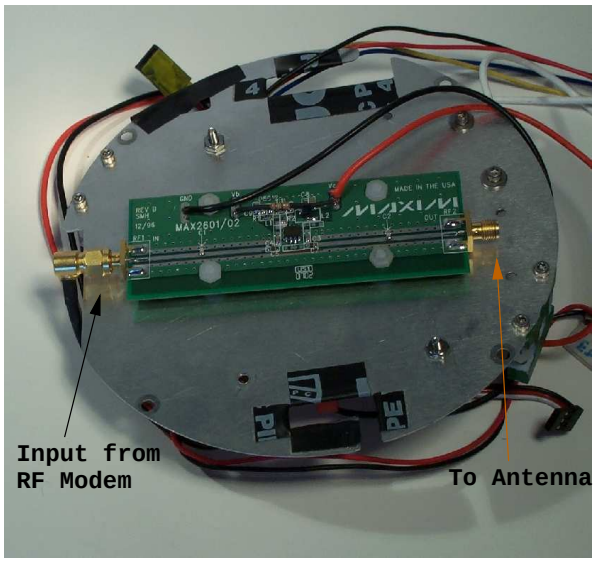


Figure 2.29: 1 Watt RF amplifier board mounted to plate.

2.14 Batteries

The batteries are critical part of the E-sonde design. They had to meet the environmental and current capacity criteria while being as light as possible. Tables 2.7 and 2.8 show the specifications for the batteries. Although expensive the Lithium batteries provided the best energy to weight ratio. They also produce quite a bit of heat when under a heavy current load and help keep the internals of the E-sonde warm.

Table 2.7: AA Lithium Battery Specification

Chemistry	Lithium/Iron Disulfide
Battery Voltage	1.5 Volts
Weight	14.5 g
Oper. Temp.	-40^{o}C to 60^{o}C
Capacity	3000 mAh (to 1.0 V)
Max. Discharge	2.0 Amps

Table 2.8: 9V Battery Specification

Chemistry	Alkaline (Zn/MnO_2)
Battery Voltage	9.0 Volts
Weight	45.6 g
Oper. Temp.	-18^{o}C to 55^{o}C
Capacity	625 mAh (to 4.8 V)

Four batteries were tied in series to form a battery pack with a plastic battery holder. Figure 2.31 on page 51 shows both the 9V and AA batteries in their pack mounted to the plate. In order to keep the batteries in place during flight wide electrical tape was place over the plastic battery carrier as show in figure 2.30 on page 50. This figure also shows the double AA battery pack on the plate. One major drawback of this system is that the entire E-sonde must

be taken apart to change the batteries out. This proved to be a major endeavor while in the field and resulted in a few missed storm opportunities.

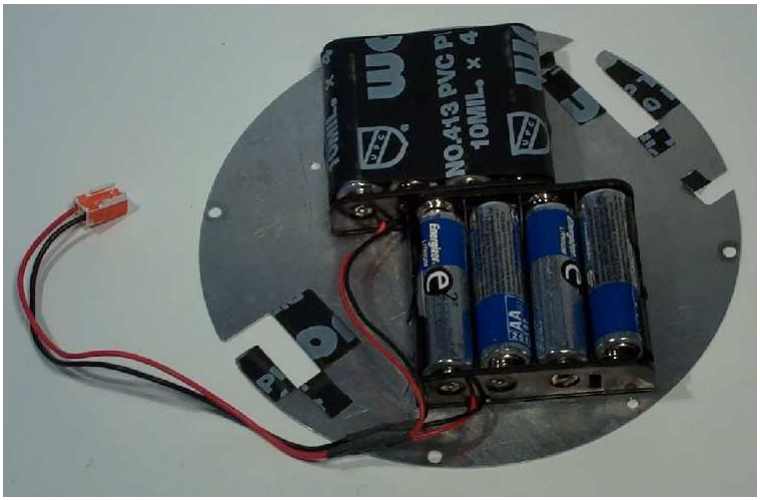


Figure 2.30: Two 4xAA 6V Battery Packs.

The CPU drawing from 700 mA to 1.1 A of current was the most demanding on system. In order to ensure adequate preflight time, fight time, and post flight search time two battery packs were tied together in parallel to give 6 Ah of current. In testing the system was able to run for about seven hours before it stopped transmitting. Although flight times were less than two hours, this gave time to try and find the package once it had landed.

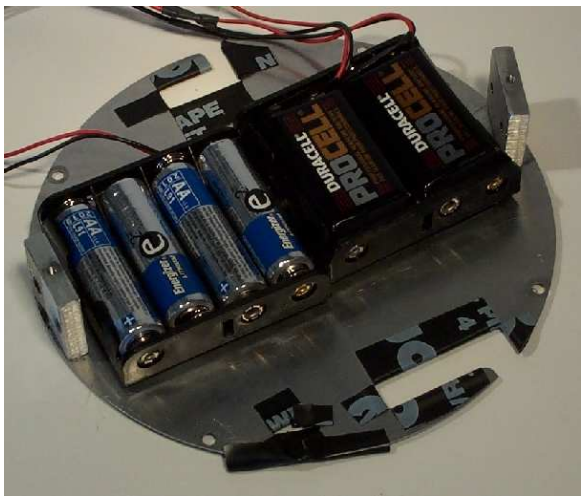


Figure 2.31: 9V and 4xAA 6V Battery Packs.

2.15 Interface and Control

The interface and control section shows the top of the bottom plate of the E-sonde. From the picture in figure 2.32 on page 53 the various components can be located. There are three mini-boards on the plate. The LED status board hold the resistors and connectors for the system status LED's. The power switch board takes wires from the switch and converts them into plugs which then connect to the power board. The plugs are of different sizes so they cannot be hooked up to the wrong connector. The temperature board holds the circuit show in the schematic of figure 2.28 on page 46 for the outside temperature sensor. The serial console connector and antenna connector and cable can be seen as well. There is also a hole where the Ethernet connector should have been. The Ethernet connector had a damaged wire and so it was removed and the hole covered with tape.

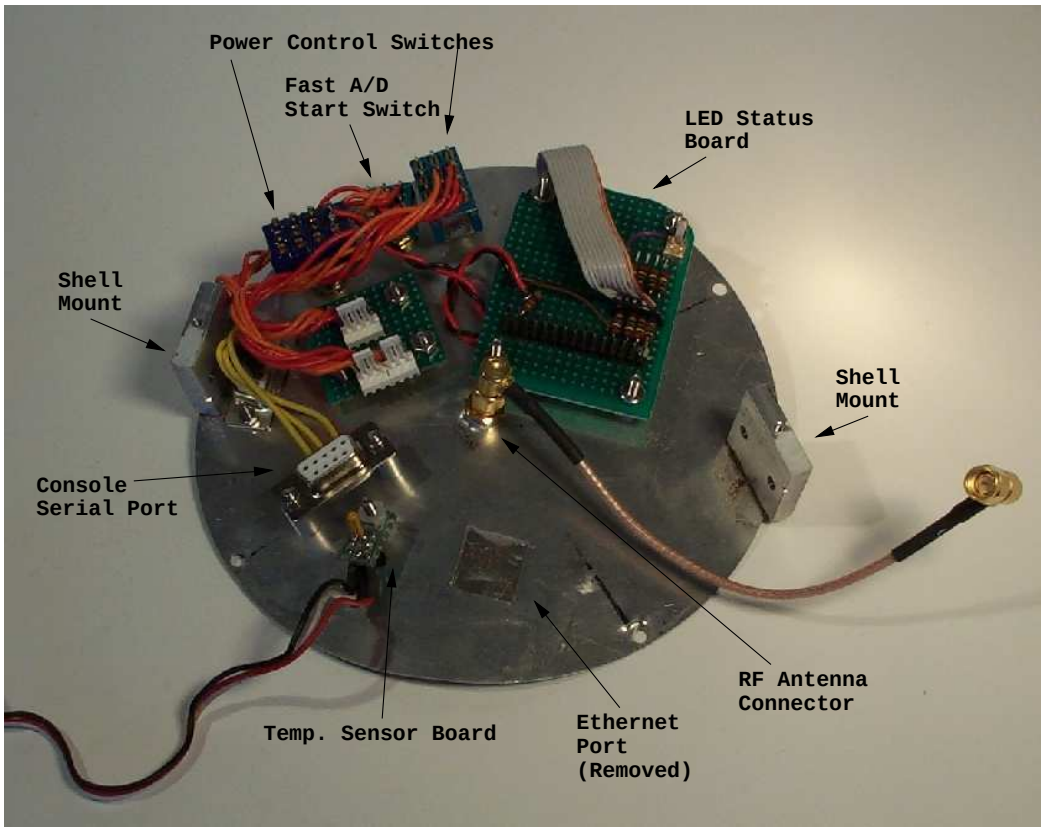


Figure 2.32: Picture of the inside of the E-sonde's bottom plate.

CHAPTER 3

Software

3.1 Operating System

Although it would be possible to run the E-sonde without the use of an operating system it would require writing code for all system functions from scratch. The decision to use an existing OS saved a tremendous amount of time and allowed for the coding effort to be focused on the control program balloon_tx. The decision was made to go with a Linux based distribution due to Linux's open source nature, reliability, no cost, and there were drivers available for the Prometheus CPU board peripherals.

The Pebble Linux distribution from NYCwireless was chosen to for the E-sonde because it was originally designed to be run from a compact flash card and was based on the Debian distribution. The Debian nature of the distribution allowed for easy adding and removing of additional software packages. The Debian configuration and layout is also well documented. Since the distribution was designed to run from a compact flash it was design to minimize file writes by putting all the log files in RAM disk. It also keeps the filesystem partition in read only mode so sudden power failures do not corrupt the OS partition on the flash.

Since the distribution was originally intended to run in a wireless router box it had many firewall features enabled that were not needed for the

E-sonde. It also only had a limited version of the NTP daemon installed that did not have a driver for a GPS. These issues were easy to fix and the installation and configuration of the Pebble distribution of the E-sonde is detailed in appendix B on page 84.

3.2 System Time NTP Daemon

On the E-sonde system it is very important to correlate the data it took with the lightning mapping array data. In order to do this the system has to know what the UTC time is. The purpose of the NTP daemon is to synchronize the system time with UTC time and keep them locked.

The NTP daemon does this by reading data from the serial port from the GPS that has the time encoded. It uses several algorithms to figure out how much time has occurred between the start of the data string and when it processes the data. It also monitors how much the system clock is drifting between successive reads. The NTP daemon nudges the system clock toward the GPS time instead of setting it at once from a single reading.

It can take several minutes for the two to become synchronized. If the time between the UTC and system time is off by too much the NTP daemon will abort. During preflight the system clock was set to the same time as UTC time.

It is important to allow time for GPS to locate itself and figure out the UTC time and then wait for the NTP daemon to synchronize the clocks. In appendix D on page 97 there are detailed instructions for checking that the two are synchronized. Instructions for setting up the NTP configuration are given

in Appendix B on page 84. Before launch the system time was compared to a clock that was set by the WWIV radio signal to make sure it had synchronized.

The NTP daemon keeps the the system clock synchronized to the GPS with better than 100 ms. Since the fast scanned data is encoded with the PPS signal only 1 s of time correlation is needed. It is possible to modify the kernel and use the PPS driver to synchronize the kernel time to a much higher accuracy of around 100 us. There was not enough time before the summer launches to work this mode but remains a good approach for future flights.

3.3 Transmit Control Program

3.3.1 Program description

The program responsible for controlling the E-sonde data collection and transmission is called balloon_tx. The program is written in the C language and composed of several files. The progam is built with passing the balloon_tx argument to the makefile. The makefile is detailed in appendix F on page 104. The appendix also documents where specific functions used in the program can be found and the file dependecies of the makefile.

The program has several important functions for the success of the flight. The first fucntion is to find the voltage offsets of the A/D and store them. The program preforms this calibration operation right after it initializes the data aquisition system. All sixteen A/D channels are sampled 10,000 times at 10 Khz. Each channel is then analyzed for mean and standard deviation. The voltage values of the mean and standard deviation are written to a file of the format Inst_ID_HH_MM.cal. Where Inst_ID is the name given to the instrument

in the command line options when starting the program and HH_MM are the hours and minutes in UTC time of the system when the calibration step was done.

The purpose of the calibration routine is to help in processing the data later on. The mean is useful for the offsets of the electric field channels. The standard deviation gives a general idea of how much noise is on each channel. Another way to use this feature is to ground all the input channel and then run the balloon_tx program. The calibration file will then show the offsets internal to the A/D converter and give an idea of how much noise the A/D contributes to the sytstem itself.

For the balloon_tx program to work properly it needs to have the system time set to UTC. The system time is used for logging the time of the slow data scans. The NTP daemon does the work of reading the GPS time and setting the system time. The balloon_tx program does read and store the GPS time for the GPS transmit string. By comparing the GPS position string time with the slow A/D transmission string the operator can determine if the system time is synchronized to the GPS UTC time within one second.

Of prime importance for recovering the E-sonde is knowing where it landed. This is done by the program reading the GPS serial data string and transmitting this data to the radio modem. The data is also logged to the compact flash drive. The details of the file naming convention and format can be found in appendix E on page 99. The rate at which the program reads the GPS postion and transmits it is determined by the a command line parameter or if not specified uses the default value of 30 seconds. If the debug and vebose

is turned on the program will display the string to the terminal. The status indicator light for GPS lock is turned on. See the figure 2.7 on page 12 for a diagram of where the LED is located on the bottom panel.

During preflight it is useful to know what the status of the system is. Before a flight, the launch person needs to know that the battery voltages are good, the GPS has a lock, the balloon_tx program is running and taking data, and the transmittr is receiving data from the program. All the above conditions are controlled by the balloon_tx program controlling the status LED's except for the radio modem receiving data.

The progam has two different modes of collecting the sensor data. The first mode is referred to as the slow scan mode. In this mode the program scans all sixteen A/D channels 20 times and averages the result for each channel. This number is set in the balloon.h file under the variable SLOW_SCAN_AVG_NUM. There is no set sample rate for this scan. The program calls the sigle channel scan function in a loop until the programmed number of scans is completed.

The second mode for data aquistion is fast scan mode. In this mode the first eight channels are scanned at 10 KHz for two seconds. The data from the first four channels is converted to voltages and checked agianst the current maximum voltage value. If a value is greater in absolute magnitued the new signed value is stored. The data from the scan is stored to the compact flash immediately after the scan in the the orignal binary format from the A/D converter in a structure that also holds a before and after scan time stamp. The original intent was to have the scans continuing while the data was being written to storage, but the A/D system was getting buffer overflows while trying

to scan while data was being written out. Buffer overflow conditions mean that the A/D subsystem could not tranfer data fast enough and incoming data samples had to be discarded. There is no way of knowing how many sample were dropped. This is also flagged in the fast data file structure, but was not seen to happen once A/D scans and data writing were done seperately. The fast scan files each hold about one minutes worth of data as the files name is determined by the current time of the system. The fast scan data files end with the extension ".edat".

The program then assembles the GPS postion information and the slow A/D data scans along witht the maximum electic field channel values into text strings and transmits them to the radio modem. After transmitting the data the maximum values stored are reset. The data strings are also written to a text based log file. The log files for the transmitted data end in the extension ".log" The are named by the instruments given ID an the system date.

The above process of getting data, storing data, and transmitting data is repeated until either the system runs out of power or the program is killed. If the program fills up the compact flash disk files of length zero are created.Figure 3.2 on page 61 shows a flow chart of the balloon_tx main control loop. Figure 3.1 on page 60 shows a breakdown of the main program function names and what purpose they serve.

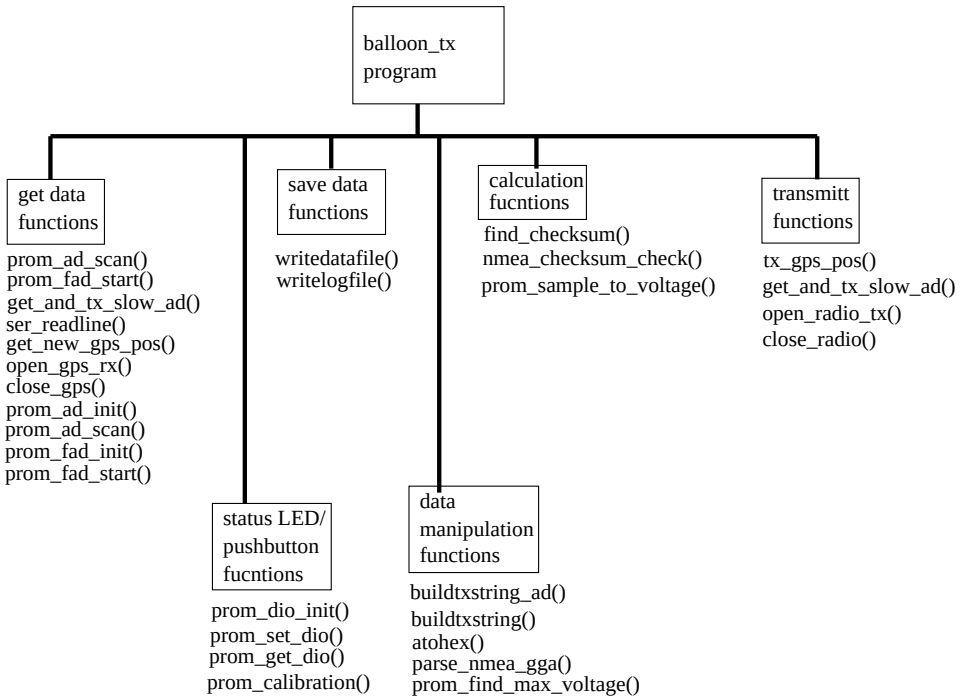


Figure 3.1: Block diagram of balloon_tx program function names and purpose.

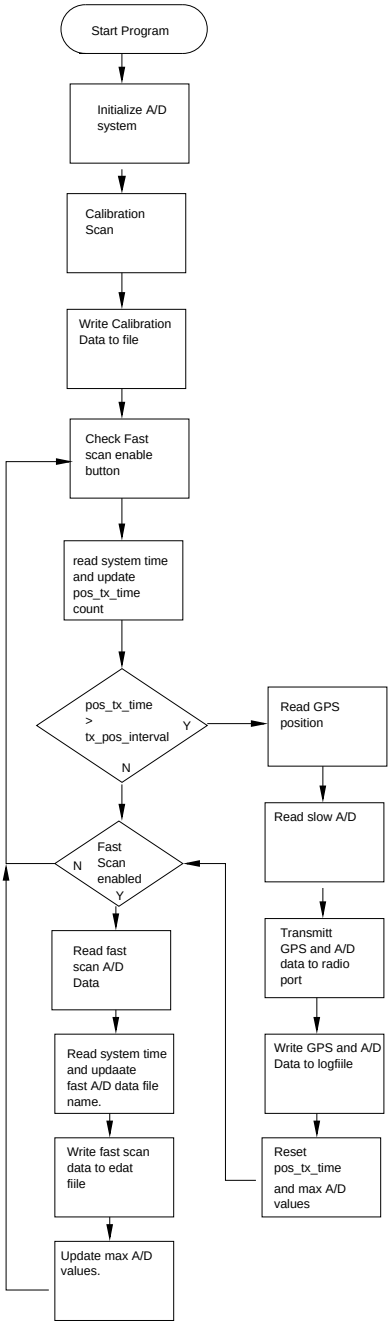


Figure 3.2: Flowchart diagram for balloon_tx program.

3.3.2 Program command line options

The listing below shows the programs command line options and what their usage is. The -i option must be set before flight to give the E-sonde a unique identification for the transmitt strings and the log files. The -t option was used on the initial flights to set the transmitt interval to 18 seconds. This was done in case there problems with receiving the balloons position. A shorter interval gives more position strings sent, but reduces the number of fast data scans. The -x, -p, -v,and -d options were used in early debugging of the software. The -b option is used to indicate if the jumpers on the power board are set to monitor the battery or regulator voltages. This setting changes the internal values used to light the LED voltage status indicators. The values are set in the balloon.h file.

```
Usage: balloon_tx -hvda <-l filename.log> <-r comm port radio>
                  <-g comm port gps> <-f alt.log file> <-i InstrumentID>
                  <-t TX interval>

-h               Display this help screen.
-v               Verbose mode: display data to screen
-d               Debug enable mode
-l               Turn on logging default is ON
-r               Radio Comm. Port Default:/dev/ttyS0
-g               GPS Comm. Port Default:/dev/ttyS1
-f logfilename   Use alternative logfilename dflt is InstID_MMDDYYYY.log
-i instr name    Name of Instrument default hostname
-t TX interval   Time delay in seconds  between position and
                 slow A/D scan data  default=30s
-x number        Disable Fast A/D scanning/logging
                 bit 4 2 1
                     | | |
                     | | - Fast A/D scan
                     | --  Slow A/D scan
                     ----- GPS position
                 Example -x 3  Disable Fast & slow A/D
-p               Disable push button Fast A/d datalogging start.
-b               Enable monitoring of battery instead of regulator voltages.
```

3.3.3 Program revision control

The balloon_tx program files have a simpe revisions control system called RCS. A “man rcsintro” will bring up a man page describing this particular RCS system. The RCS data base files are stored in a subdirectory called RCS. The command “ci -l ¡filename¿” is used to check-in a file and add comments to it. The -l keeps a locked copy of the file out in the main directory. The source code files include special tags in the comment sections at the beginning and end of the file to show the RCS version and comment information.

3.3.4 Program dependencies

In addition to the progam file dependencies shows in appendix F.1 on page 105 the balloon_tx program is linked against several libraries. Most are standard libraries, but the dscud5 library is from the Prometheus software package and must be installed first. One thing to note in the makefile on page 104 in the same appendix is that the math library “-lm” is placed last in the file. The order of the libraries is dependent. If changed the file may to compile.

3.4 Receive Program

3.4.1 Program Purpose

The primary purpose of the receive program is to receive incoming data packets and display them on the screen. The program is also repsponsible for presenting a real-time display of the E-sonde flight status and log the packets to the hard disk. The program is also capable of automatically pointing an

antenna mounted on the Meade LX-200 mount at the E-sonde for maximum signal gain and reading the last packet received signal strength from the radio modem. A diagram showing the functional breakdown of the program can be found in figure 3.3 on page 64.

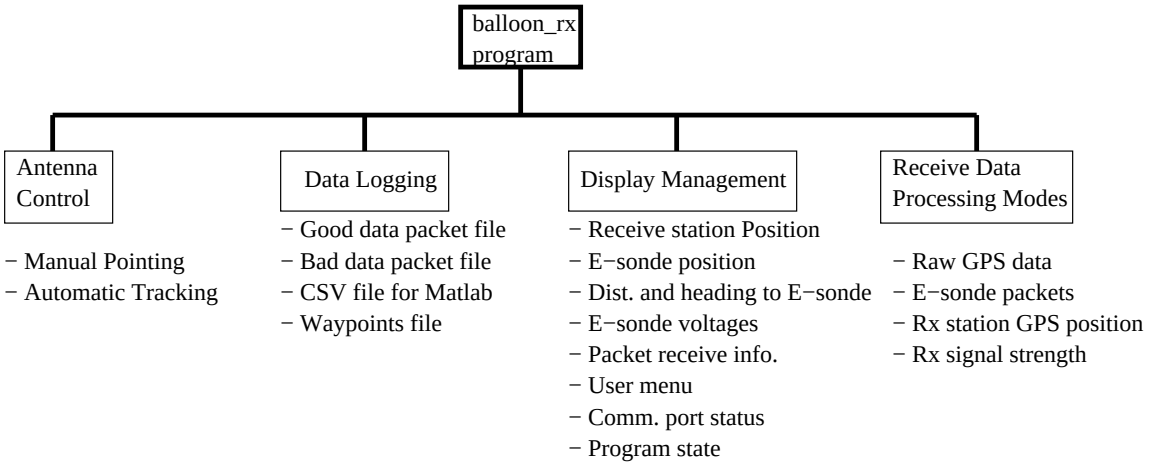


Figure 3.3: Block diagram of balloon_rx program showing basic functionality.

The program is capable of receiving data in three differnent modes. The first mode was to receive raw GPS data strings transmitted by an earlier balloon package that had a GPS tied straight to the radio modem without an intervening CPU. The program can log these data packets in a format shown in appendix E on page 99 under the GPSRAW catagory. Currently, the program does not provide antenna pointing in this mode, but this is an easy modification to code to support this in the future.

Similar to this mode the program can read the position from a local GPS attached to a system serial port. The purpose for this is to provide an easy way to update the location of the receive station which is needed for antenna pointing along with distance and bearing calculations to the E-sonde. A future

goal is to expand this capability to provide real-time position updates for a vehicle tracking the E-sonde. On the NMEA GPGGA strings are considered valid. Other strings will be flagged as bad packets if received from the GPS.

The third receive mode is to look for instrument data packets and decode them. This mode provides the most information for the user during a flight. The user can monitor E-sonde slow scan voltages as well as the position information. This mode receives data packets formatted by the CPU on the E-sonde. When the packets arrive a checksum at the end of the packet is re-calculated and compared to determine if the packet is good. Once a packet is determined to be good the display is updated with the new packet information.

While receiving packets of data logging can be turned on or off. The purpsoe for this is so the operator and choose to not have packets recorded which testing or in pre-flight that would otherwise just clutter the log files. Log files names are created based on the instrument ID and the system date. A duplicate file for the good data packets is also created when in instrument receive mode that modifies the format of the data so that Matlab can easily load the data into memory. The file ends with the ".csv" for comma seperated values. In this file there are no text strings so the data can be loaded into a matrix. The type of string is represented numerically and based on the first column can be sorted. Another file created is the way.txt file which contains the waypoints used by the GPSDRIVE program to plot the path of the E-sonde and its altitude. The program encodes the altitude as part of the waypoint label. The details for the format of the log files are discussed in appendix E on page 99 in the receive program section.

The receive program displays information on the screen in a terminal window of 80x25 size. The display is updated as the data packets arrive. The display has several sections. A screenshot of the display can be found in figure 3.4 on page 67. There are three sections on the top of the terminal showing the receive station's position, the balloon's position, and the distance and orientation to the balloon. The instrument position section also shows the speed and heading of the balloon. The orientation section also contains information about the antenna's current mode of operation and position. The instrument section in the middle shows the time of last update in position from the E-sonde.

Below the RX Postion section is the voltages from the E-sonde and the time of last update of the voltages. The voltage updates come in sets of four voltages and get updated as each type of voltage packet arrives. So the update shows one of the four groups of voltages were updated, but not which one. Under the instrument position is the system menu that will be describe in section 3.4.2 on page 67. Under the antenna information section on the top left is a section showing the GPS time from the system and the system time along with the ID of the last instrument packet sent.

At the bottom of the screen on the left is the status of logging. On indicates that logging of packets will occur. Off indicates that logging will not occur. The status display shows which state the main receive loop of the system in in. The radio and gps sections show whether or not a communication port is open. The system time is continuosly updated and shows the time of the receive system.

The section in the lower right corner shows how many packets were received and how many were succesfully decoded. The last packet heading shows when the last packet was received regardless of whether it had a good or bad checksum and type. The last line in the lower right shows the receive power level of incoming packets and what rate the level will be checked. Since the program can not receive incoming packets while the power level is being checked the rate can be disabled.

3.4.2 Program Operataion

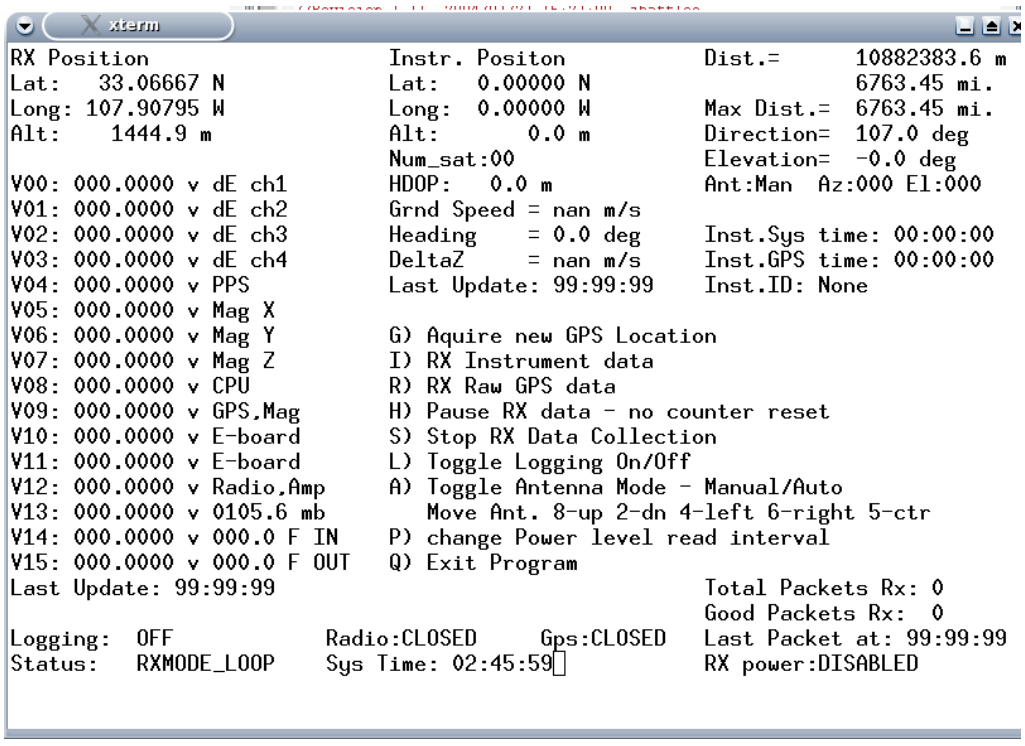


Figure 3.4: Screenshot of balloon_rx program running..

In this section the details of how to operate the balloon_rx section will be covered. The first thing the user wants to do when starting the program

is fill out any appropriate command line options. The most common would be -h to get what the options were. The result of typing the -h option is shown in section 3.4.3 on page 71. The command line options can change the communication ports used by the program, the default log file name, and turn on loggging automatically.

Once the program is up and running the firs thing to do is make sure the RX Position section in the upper left shows the current information for the location of the receive station. There are two ways to change this information. The first is to edit the values in a file called position.cfg and restart the program. The second way is to hook a GPS up to the comm. port designated for the GPS and press "G" on the menu. This will tell the program to read the position in from the GPS and save the new value to the position.cfg file. If the position information is wrong for the receive station the on-screen calculations for distance and heading will be wrong. If antenna tracking is planned the antenna will probably point in the wrong direction as well. The received data and logs will not be affected by this information being incorrect.

After setting the receive stations position, the next step is to manually align the antenna and turn on automatic tracking. The program starts with the antenna in manual mode. In this mode by pressing the 2,4,6,8 or 5 keys the antenna's azimuth and elevation can be set. The 5 key centers the antenna at zero elevation and zero altitude. Each press of the other keys increases or decreases the elevation or azimuth by ten degrees. Once the antenna is pointing at the balloon launch location the automatic antenna tracking can be turned on by pressing the "A" key. Pressing "A" again will place the antenna back in

automatic mode. The antenna position indicator will update automaticlly in auto mode each time the program decides to update the values. The program will update the values and send the new position command to the antenna when there is a difference between the current setting and the new setting of one degree or more.

While in manual mode each time a key is pressed a new altitude or azimuth command is sent to the LX200 with the new value. If the old command is not finished being processed by the LX200 it will beep and ignore the new command. If this happens the user can simply press the opposite movement command and then press the desired one again once it has moved.

Once the system is ready and the E-sonde is turned on then the user can select the "I" command to begin receiving telemetry packets. As the packets come in information on the screen should begin to update with the new time stamps adn the total and good packet counts should start to increase. If the "H" key is pressed the receive mode will pause, but the counts and last displayed data will remain. If the "S" key is pressed the counts will be reset.

On the display the voltage values can be used to check out the what the E-sonde is experiencing envriomentally and how the battery voltages are holding up. The instrument position section will give an indication of how the GPS is doing. In particular the Num_sat and HDOP section show how well the GPS is locked by giving the number of satellites and horizonal precision information. Also of use is the two time indicators in the middle right part of the screen. From looking at these two times the user can quickly see if the GPS and system time have been synchronized by the NTP daemon.

Before launch the user wants to make sure that logging has been turned on. If the command line option was not given to force it on then by presing “L” the logging will be turned on. Its status is showin in the lower left corner of the screen.

Once the E-sonde is launched the user can montitor its ascent rate by checking the DeltaZ field in the upper center part of the screen. The ground speed and heading are useful for tracking where it is going. When the balloon breaks the DeltaZ will change signs and increase in magnitude to about 10 m/s. For the three flights in the summer of 2004 the average ascent speed was about 6 m/s.

The “P” command is used to set the interval at which the radio modem’s received power level is checked. This was useful during intial testing of the system while in the raw GPS mode, but was not used during subsequent flights. The feature was used while driving the package around to find out what the maximum distance was. Since the user could see the value decreasing before it started losing packets, its was easier to determine where the cut off happened. The problem is that the program will drop packets while trying to get the power value from the modem. During actual storm flights it was undesireable to lose any packets.

Once the flight has ended pressing the “Q” key will exit the program. Another useful thing to do is to open a terminal in the same directory as the balloon_rx program and execute a “tail -f ¡logfile¿” to view the log file as it is written. What’s useful is to watch for the MAX1 strings and look for larger than normal voltage values to see if the E-sonde is seeing large e-field chanes.

3.4.3 Program command line options

The listing below shows the programs command line options and what their usage is.

```
Usage: balloon_rx -hd <-l filename.log> <-c comm. port>
                  <-g gps port> <-a lx200 port>
-h              Display this help screen.
-d              Debug enable mode
-l              Turn on logging default is InstID_MMDDYYYY.log
-f logfilename  Use a different logfile name.
-c comm. port   Port to use for radio.
                default is /dev/ttyS0
-g gps port     Port to use for GPS postion updates.
                default is /dev/ttyS1
-a lx200 port   Port to use for antenna
                default is /dev/ttyS1
```

The options are self-explanatory and describe ealier. The one option that is obsolete is the -d debug option. The problem is that the program uses the ncurses library to control the display and the debug statements are printed to standard out. So as soon as the screen is updated all the information is overwritten.

3.4.4 Program Function map

Figure 3.5 shows a list of the program function names and what there general purpose is. In appendix F on page 106 a listing showing what file each particular function can be found in. The progam is designed to reuse as much code as possible from the balloon_tx program. In addittion to not having to rewrite much of the code, the functions are consistent in format for both programs.

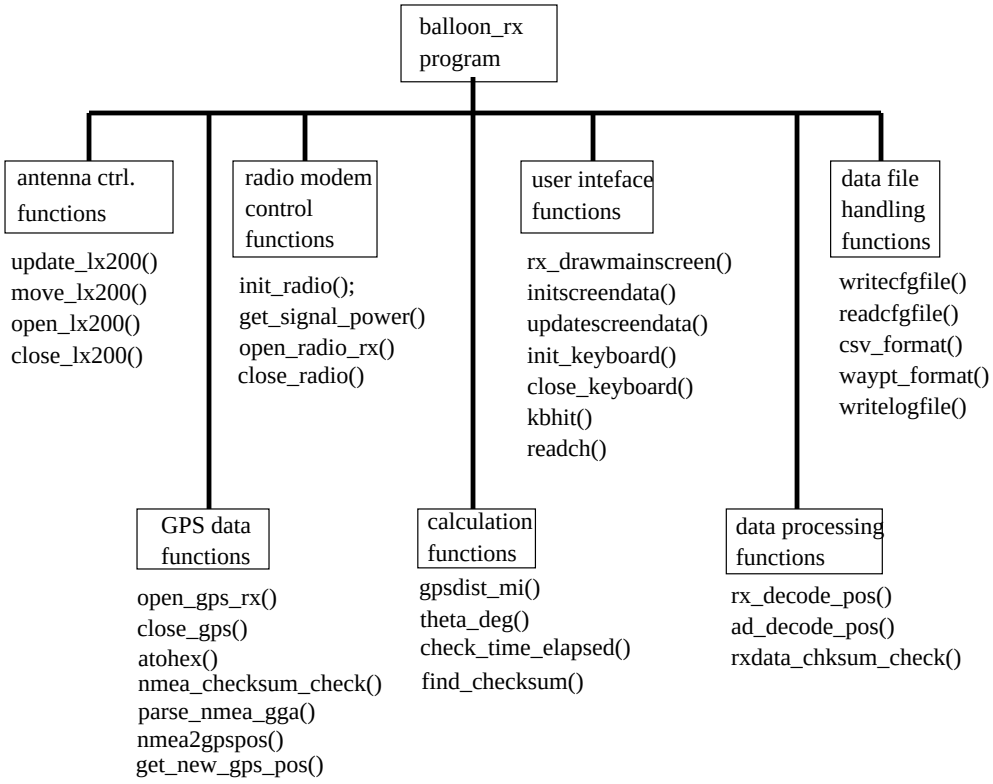


Figure 3.5: Block diagram of balloon_rx program function names and purpose.

3.4.5 Program revision control and compile instructions

The balloon_rx program files have the same RCS system used in the balloon_tx program. Details can be found in section 3.3.3 on page 63. The program is built with the same makefile used by balloon_tx which can be found in appendix F on page 104 along with a diagram of the dependencies.

3.5 Instrument Test Programs

Describe 1s data capture test program and what it can be used for.

3.6 Graphical Balloon Tracking

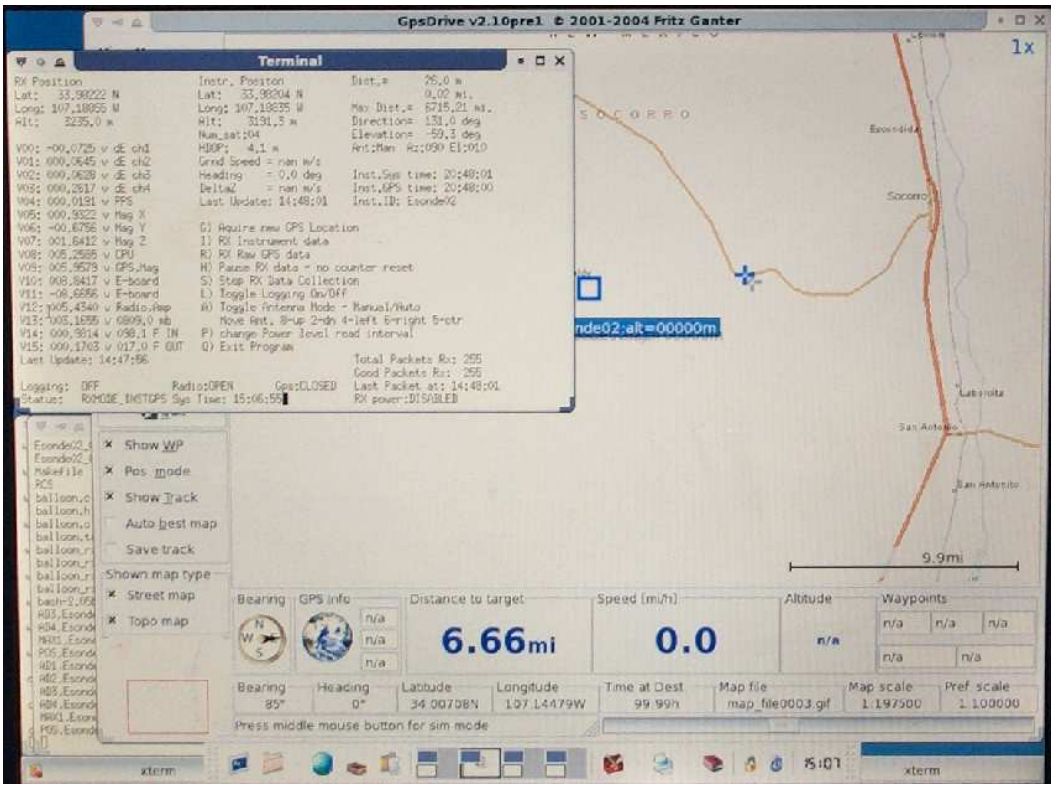


Figure 3.6: Telemetry and tracking map screen during a flight.

Describe GPS drive and how it is used to track the balloon.

3.7 Preflight Procedures

Discussion of things to do before launch.

3.8 Data Analysis Programs

Describe the python programs to extract time from PPS signal and create a CSV file with the voltages aligned to time.

Describe Mike's weird time system.

Describe the time_check.c program an why it's useful in checking the results of the python program.

CHAPTER 4

Testing and Calibration

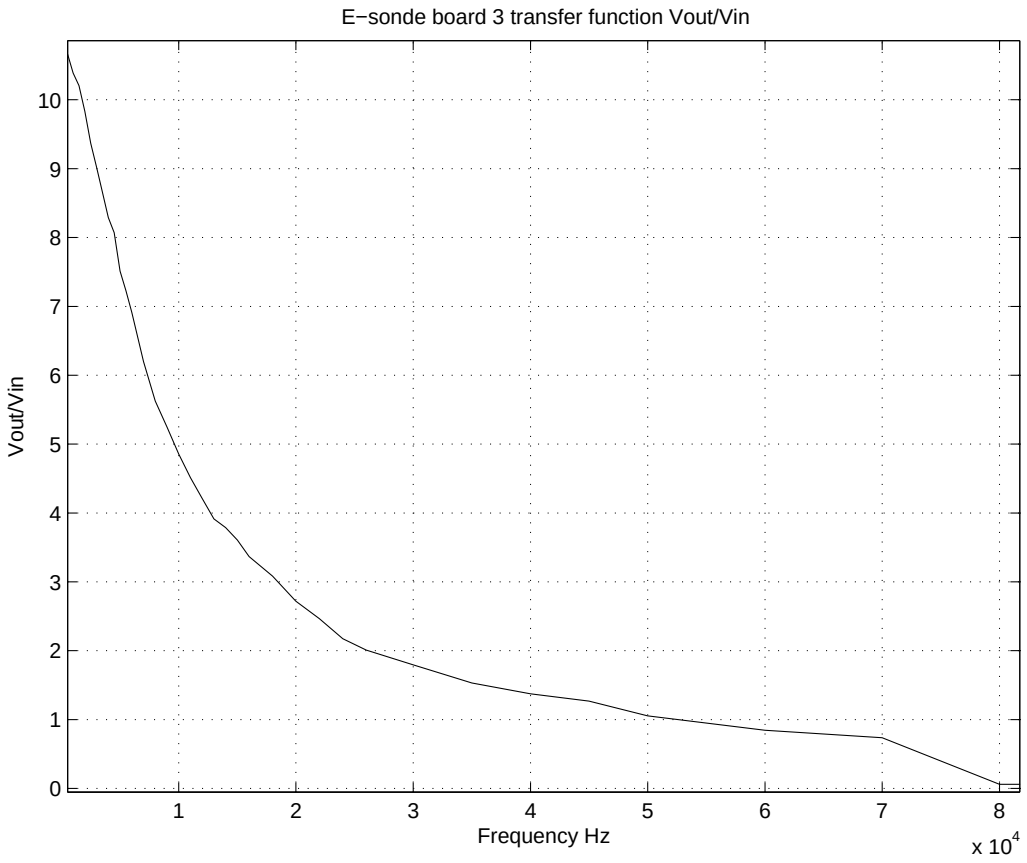


Figure 4.1: Plot of transfer function for e-field sensor board.

Describe the frequency response for the E-field board and how it was tested.

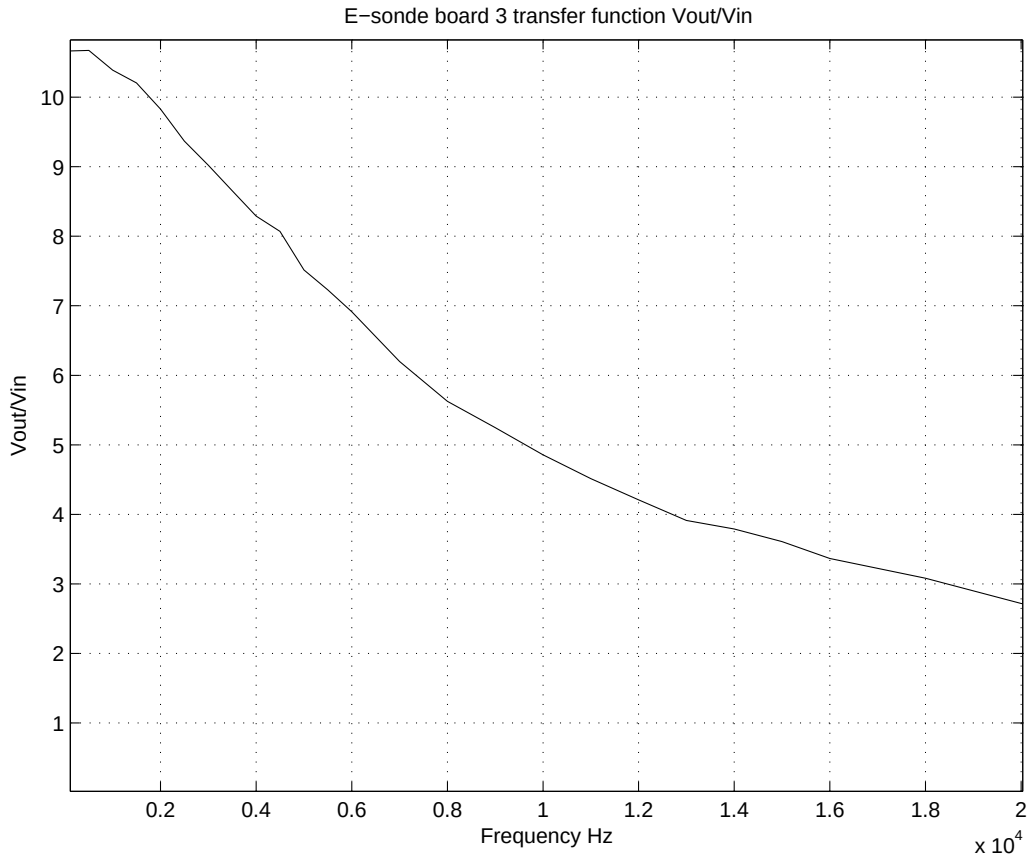


Figure 4.2: Transfer function plot of e-field board showing detail up to 20 kHz.

CHAPTER 5

Flight Results

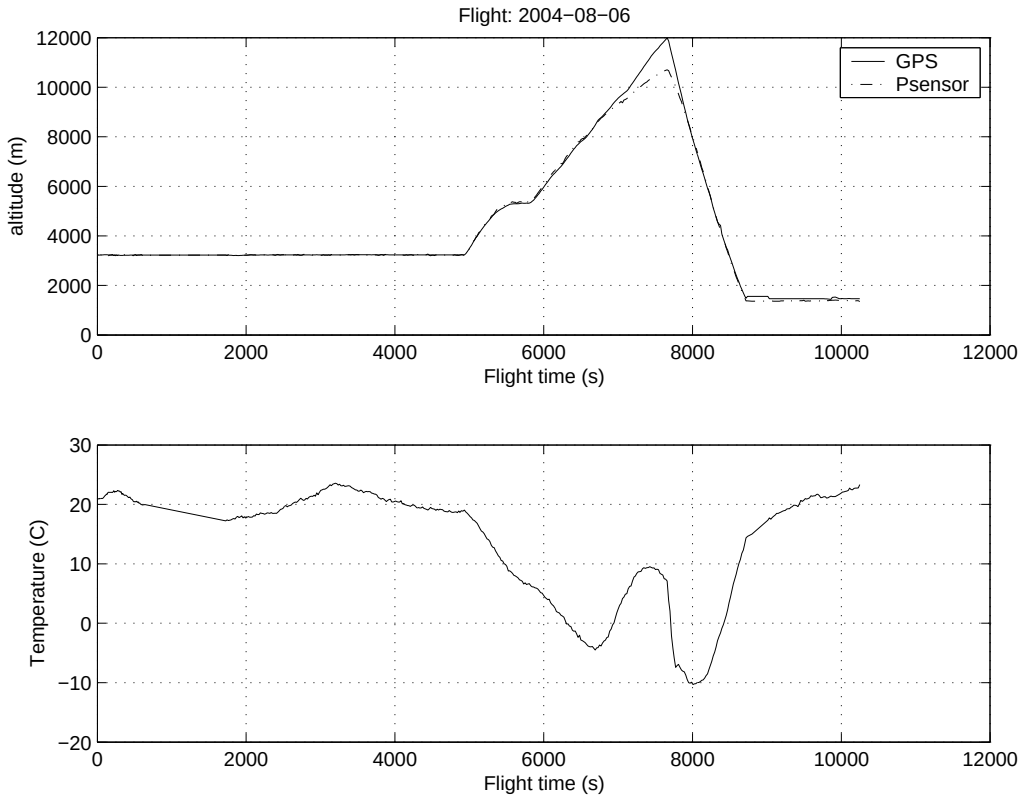


Figure 5.1: Comparison of altitude from GPS and pressure sensor with temperature.

Describe correlation between GPS alt. and press. sensor calculated alt.

Show formulas to calculate altitude from Pressure sensor.

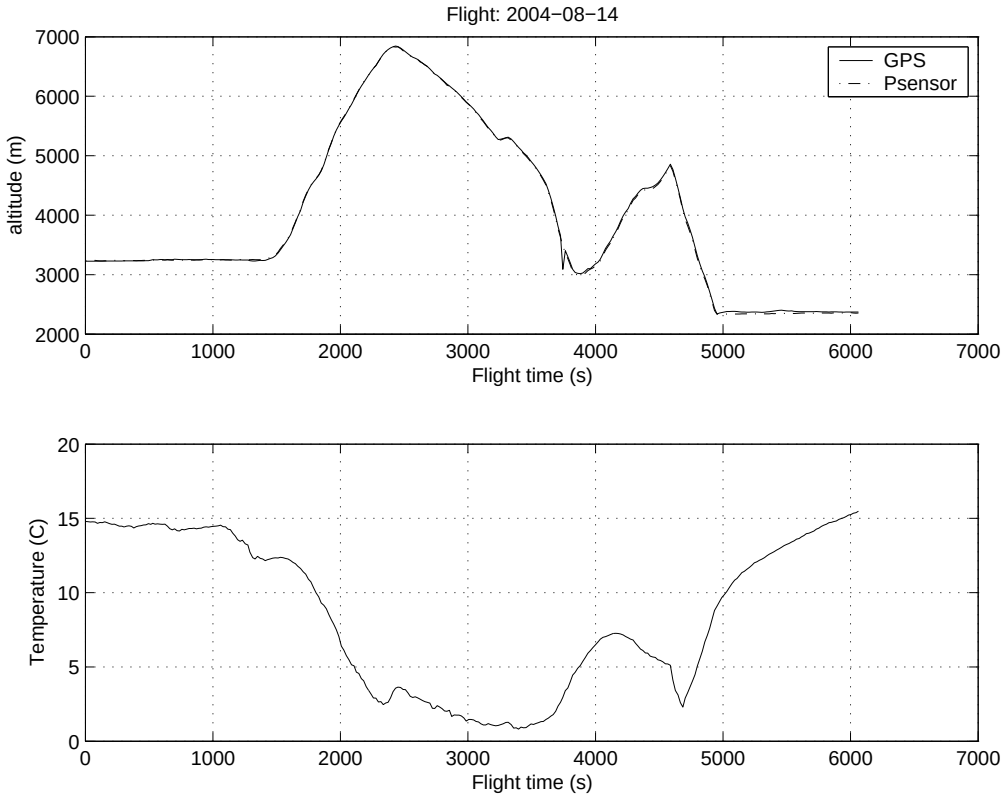


Figure 5.2: Comparison of altitude from GPS and pressure sensor with temperature.

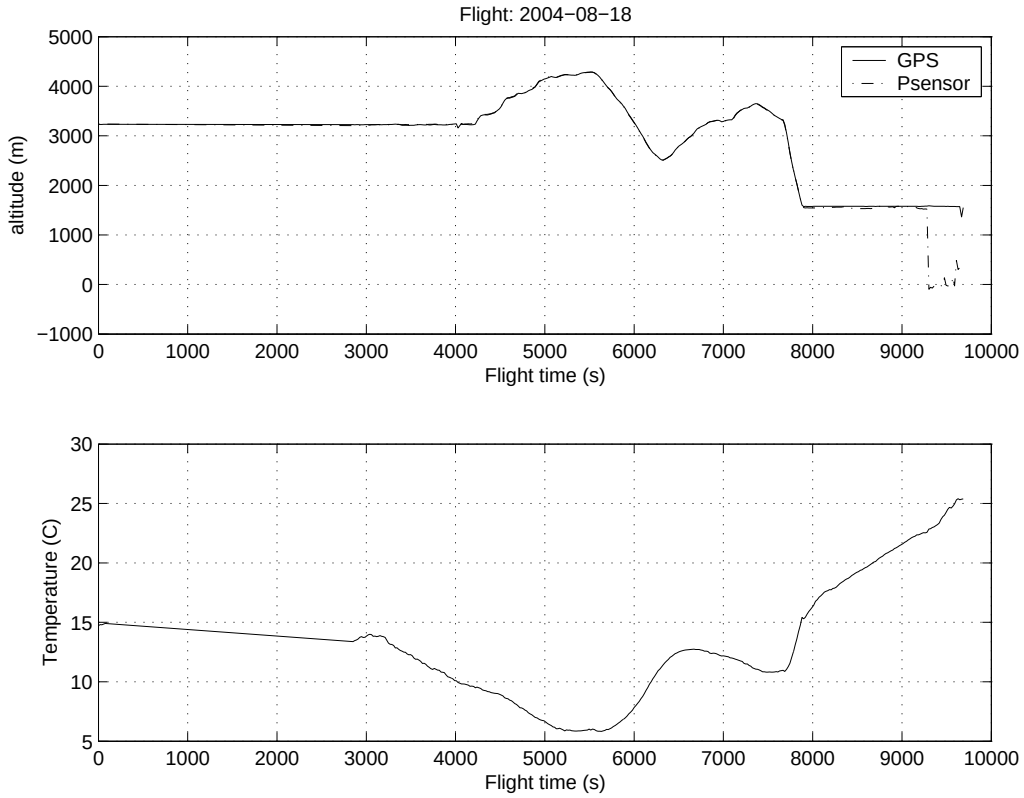


Figure 5.3: Comparison of altitude from GPS and pressure sensor with temperature.

Describe discrepancy in 2004 08 06 graph.

CHAPTER 6

Data Analysis

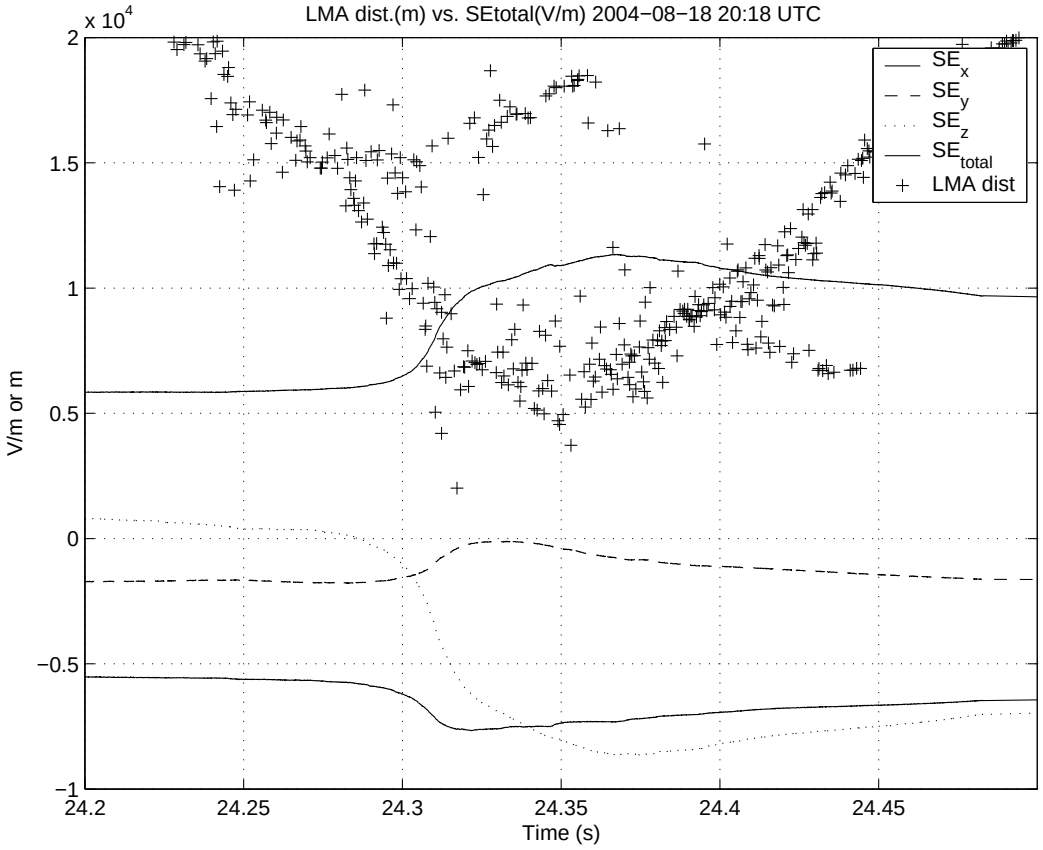


Figure 6.1: E-fields vs. LMA distance for August 18,2004 20:18:24.

Find a few more example of LMA data correlating with sonde e-field changes.

Describe what is happening in each picture.

APPENDIX A

System Configuration

A.1 Communication Ports

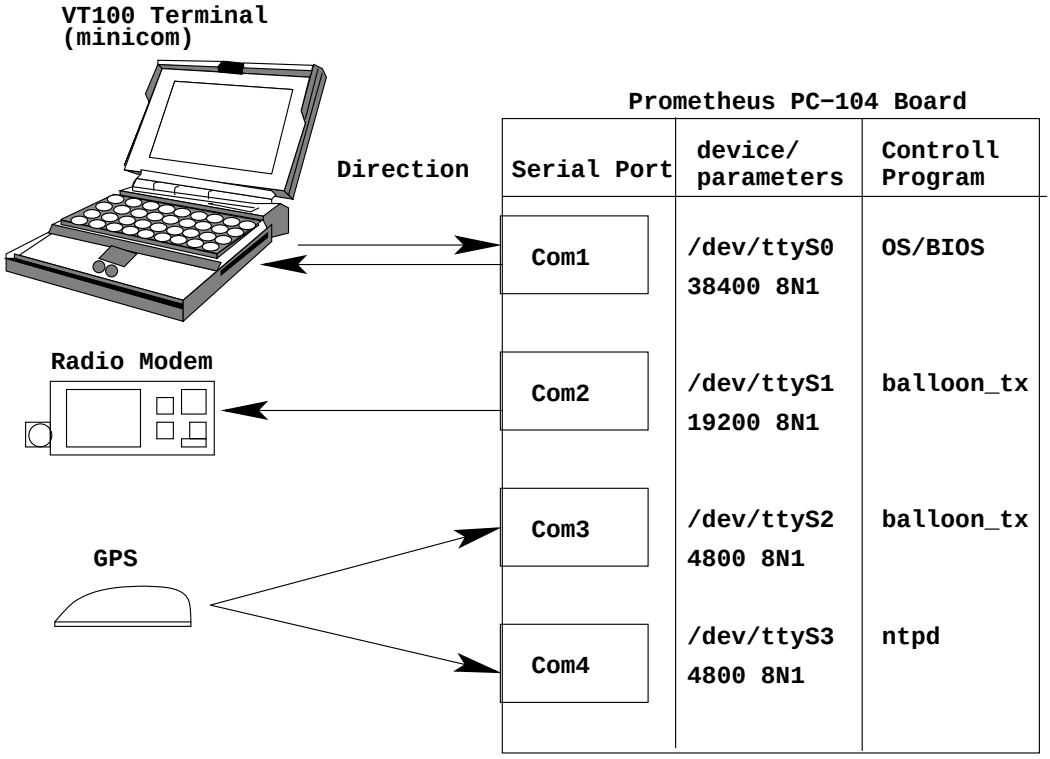
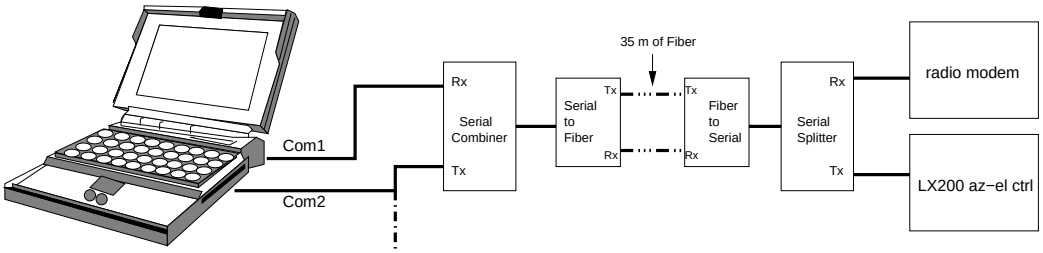


Figure A.1: Serial port configuration diagram for E-sonde.



Port	Device/ Parameters	Functiion
Com1	/dev/ttyS0 19200,N,8,1	Receive telemetry data from sonde
Com2	/dev/ttyS1 9600,N,8,1	Control ant. positiion
Com2	/dev/ttyS1 4800,N,8,1	Read GPS position

Figure A.2: Serial port configuration diagram for receive station.

APPENDIX B

Pebble Linux Installation and Setup

```
VERSION 1.5  APRIL 28, 2005

   Pebble Install and Configuration for Prometheus E-balloon System
   ====== ======= === ============= === ========== ========= ======

Overview:
      This document describes how to install the Pebble Linux
      distro. on the compact flash. It then explains how to
      further configure the system for balloon flight operations.

Why Pebble Linux?

    Pebble Linux was chosen for several reasons.  The two main ones
    are that it's small and it's designed to run on a compact
    flash(CF).  Compact Flashes contain a limited number of write
    cycles, so it is important to not have the filesystem continually
    writing small bits of log data to the flash.  Pebble creates a ram
    disk for the log files and mounts the flash as read-only.

    The down side of pebble is that it's designed for a router
    and has many security modules and firewalling stuff loaded by
    default that are not needed for the balloon package.  The other
    problem is that it contains the 2.4.xx kernel by default and the
    balloon system would be better off with the real time features of
    the 2.6.x kernels.

    The good news is that Pebble is a Debian based distro. and the
    packages on it can be easily updated/removed/added with the
    apt-get command.

    One word of caution though is that when apt-get is run it can pull
    down 100 Meg of package listings.  So if you put it on a small
    partition less than 64 Meg and try to update the database it will
    fill up the flash.

    Pebble can be found at www.nycwireless.net/pebble/
```

```
    Theses installation instructions were based on the pebble.README
    file.

Section 1: Installation: (best done as root user)

1.1) Put the CF disk into the USB CF reader.  The disk will show up
   under linux typically as /dev/sda, but it depends what other drives
   are on the system.  It could show up as /dev/sdc or other.  For
   this reason, the commands below show /dev/sdX.

 WARNING: It is extremely important to get the right device, as you
   will be partioning and formatting it.  If you get the wrong device,
   you will be partitioning it.  In the worst possible case, you may
   reformat and repartion your main hard-drive.  Linux will let you do
   this, and it will keep running ... for a while!  You can find out
   what device the flash is mounted as from doing a dmesg command or
   looking at /var/log/messages to see what device the CF was assigned
   to.  At this point if the CF is new there will be no partitions on
   it.  So the first thing to do is run something like

       cfdisk /dev/sdX or fdisk /dev/sdX

   Create 2 partitions. For a 1 GB we used something like

      /dev/sdX1 80M
      /dev/sdX2 920M

   Set the boot flag on /dev/sdX1 so lilo can boot from it.


Here is what Prometheus flash partition table looks like on a 1 GByte
flash.

(fdisk with the -p option)
Disk /dev/sdc: 1024 MB, 1024966656 bytes
32 heads, 63 sectors/track, 993 cylinders
Units = cylinders of 2016 * 512 = 1032192 bytes

Device Boot         Start         End      Blocks   Id  System
/dev/sdc1               1          79       79600+  83  Linux
/dev/sdc2              80         993      921312   83  Linux

1.2) Create a file system for each partition.  Currently ext2 was used
   for simplicity, but there is a new filesystem called JFFS2 designed
   for compact flashes that may work better.  (Sticking w/ ext2 until
   further notice)
```

```
       mkfs.ext2 /dev/sdX1
       mkfs.ext2 /dev/sdX2
       Alternately, mkfs -V /dev/sdX1 will work.  It defaults to ext2.

1.3) Mount the new partitions to the host system.
     mount /dev/sdX1 /mnt/cf1
     mount /dev/sdX2 /mnt/cf2

   It helps to make sure you already have the mount point directories
   on the host system.

    [cd /mnt; mkdir cf1; mkdir cf2; chmod 755 cf?]

   Can try chmod 777 cf? to see if you can do more without being root.
   It seems, for example that the tar command doesn't work well when
   you're not root.  So the 777 is probably useless and 755 is good
   enough.

   Type "mount" afterwards to check that it worked.
   The "mount" command will only work if you have the /mnt/cf? entries
   in your /etc/fstab file.

   To make mounting easier, you can modify /etc/fstab by adding these lines:
   This is done on the host system not the prometheus system.

# USB PROMETHEUS FLASH
/dev/sdX1  /mnt/cf1        auto            defaults     0 0
/dev/sdX2  /mnt/cf2        auto            user,rw,noexec,noauto   0 0


Note that sdX1 is mounted w/ defaults.  If you mount it noexec, then
no commands can be executed on it.  Therefor, for example, you can't
run lilo on it, which you'll need to do.  The error in this case is
exceedingly unhelpful "Permission denied".  Even if you have the
directories on cf1 set to rwx and even if you are root, the "noexec"
option in fstab trumps all of that and prevents lilo (or any other
command on the flash) from running.

1.4) Unpack the pebble tar file onto the new mount point.
     (This needs to be done as root)
     cd /mnt/cf1
        [change to the directory into which you want to untar the file]
     tar --numeric-owner -jxvf /<downloaded dir path>/pebble.vXX.tar.bz2
         e.g., as of April 2005, it's
     tar --numeric-owner -jxvf /<downloaded dir path>/pebble.v41.tar.bz2
```

```
    This will place all the pebble files into their correct sub
     directories on the compact flash.  Note --numeric-owner is part
     of the incantation to make this happen.

1.5) Update and install lilo on the CF
      cd /mnt/sdX1/etc
      cp lilo-standard-hdc.conf lilo.conf

         (Note) for console output to com1 there is a lilo-serial.conf
         file you can use instead.  (Additional note) --
         /etc/lilo.conf is standard with any lilo distro.  However
         lilo-serial.conf is not generic.  It is included with pebble,
         so when you untar pebble onto the flash, you'll find
         /etc/lilo-serial.conf In fact, since you are hand-editing
         lilo to do the right thing, which .conf file you start with
         doesn't matter.  When you finally have the .conf file in the
         form you like it, it must be copied to to /etc/lilo.conf on
         the flash before it can be used by lilo.  Lilo doesn't know
         about lilo-serial ... it's just there as an example.

Here is a working lilo.conf for serial port boot:

#############################################################################
#
# NOTE:  boot and disk below must be set to wherever your distro
# is currently mounted (while you're installing LILO).  Don't mess
# with anything else unless you really know what you're doing.
#

boot = /dev/sdX
disk = /dev/sdX
bios = 0x80
compact
delay = 1
serial=0,38400n8

image = /boot/vmlinuz-2.4.26-pebble
root = /dev/hda1
append="console=ttyS0,38400n8"
label = pebble
read-only

# NOTE2: The append="console..." and serial commands make lilo
# correctly output boot messages to the serial port.  That way, you
# can use a remote serial cable to watch the sonde boot once it's
# buttoned up # Without this, you can't tell what the sonde is doing
```

```
# until the ethernet comes up.  Which means you can't debug it at all!
# April 05.  There is the potential that some of these settings are
# still not quite right.  We find when the flash boots that sometimes
# it spews garbage out the serial port.  This can be caused by an
# error in lilo.conf or by having the laptop listening into the sonde
# set to wrong baud rate (or other serial parameter)
# ########################################################################

  chroot /mnt/cf1 lilo

  This runs lilo on the CF.  Without the chroot it will run it on your
  host system!  The chroot command is tricky.  Read info coreutils
  chroot for more discussion.  Here is a spontaneous essay about
  chroot, so you use it wisely.

  Chroot mystifies some people.  You may think that it literally
  changes what kernel you are running or something really tricky.

  In fact, it only redefines the "root" of your filesystem.  An
  example of a use for this command helps it make sense.  Normally
  "Root" is the directory /.

  However, let's say you are working off of a compact flash on
  /mnt/flash but booted from the hard drive at /.  You can change your
  "root" from harddrive to flash by chroot /mnt/flash Then cd /etc
  actually puts you in /mnt/flash/etc rather than /etc.

  Why is this necessary?  Let's say you want to run lilo to make a
  compact flash bootable.  Could do /mnt/flash/sbin/lilo.  However, lilo
  by default looks for /etc/lilo.conf.  Thus, even though you were
  running lilo on the flash, it would still pull the lilo.conf from the
  hard-drive.  Doing a chroot /mnt/flash /sbin/lilo runs the lilo on the
  flash and pulls lilo.conf from /etc on the flash.

1.6) The system should now boot and configuration from the console can
   be done.

   umount /mnt/cf1
   umount /mnt/cf2

   remove the CF and put it into the balloon sonde.
   At this  point it should boot and you should see a login prompt.
   Login as root.

1.7) Configuration after booting and logging in.  There should be no
   root password so just log in as root and begin editing the
```

```
following files.

vi /etc/network/interfaces
   comment out the #iface eth0 inet dhcp
   add
 iface eth0 inet static
       address 10.0.100.xxx   -where XXX is 56 testing,
                  181-186 esonde_x
       netmask 255.255.255.0
       network 10.0.100.0
       broadcast 10.0.100.255
       gateway 10.0.100.1

 vi /etc/resolve.conf
    add
    nameserver 10.0.100.1

 vi /etc/modules
    add
  natsemi
  dscudkp

   comment ALL the other modules.  (If you don't you'll get a lot
   of errors on boot and won't get to the login prompt)
   The dscudkp usually doesn't load
   because you need to use the -force on the insmod usually due to
   it being compiled on a different kernel.  The dscudkp is the
   driver for the A/D D/A IO on the prometheus.  The specifics for
   this driver will be discussed later in this doc.

 vi /etc/ntp.conf

    For the gps timekeeping add the following line.

    server 127.127.20.0 mode 2 burst prefer.

    This IP address should be checked.  The point is that it will
    synch the sonde clock with an NTP server if the sonde is
    connected via ethernet.  This means it will bring the sonde
    close to correct time.

 ln -s /dev/ttyS3 /dev/gps0

    this creates a link between the gps driver for NTP and the
    serial port COM4.
```

```
   vi /etc/inittab
      remove the nocat lines

1.8)Run
    passwd         - to set the root password
    ssh-keygen     - sets up sshd keys
    mkdir /wxdata
    chmod ugo+rw /wxdata

    vi /etc/fstab

       /dev/hda1   /           ext2    defaults,noatime,ro         0   0
       proc        /proc       proc    defaults                    0   0
       tmpfs       /rw         tmpfs   defaults,size=10M           0   0
       /dev/hda2   /wxdata     ext2    defaults,rw,exec,noatime    0   0

    apt-get install ntp minicom setserial

       This will install the full ntp packge with the gps drivers.
       minicom and setserial are also installed.


    shutdown -r now   or reboot

1.9) The system should now be accessible from across the network.

   ping 10.0.100.xxx
   scp balloon_tx root@10.0.100.xxx:/wxdata
   ssh root@10.0.100.xxx

      Where xxx is the assigned ip address.

1.10) Building the DSCUD drivers (from the host machine,not esonde)

      NOTE: We are now recommending that you use DSCUD58.  The
      procedure in dscud_build.txt supercedes the procedure in section
      1.10 here.

   a) untar the driver package LINUX-DSCUD57.tar.gz
    (available at http://www.nmt.edu/~rsonnenf/research/research.html)
    This runs on the Pebble 2.4 kernel.


   b) go into the gcc3 directory and run install.sh
    this will build the files and put the library files in
         /usr/local/dscud5 on the host machine
```

```
        /lib/modules/misc

   c)scp /lib/modules/misc/dscudkp root@10.0.100.xxx:/lib/modules/misc

   d) scp /usr/local/dscud5/load.sh root@10.0.100.xxx:/etc/init.d/dscud

   e) ssh to the e-sonde and run depmod -a

1.11) log into the esonde either by SSH or console

    cd /etc/init.d
    chmod +x dscud
    cd /etc/rc2.d
    ln -s /etc/init.d/dscud S99dscud


    * This will allow the driver to load when the system is booting
    * You may want to clean out all the S?? files that you don't want
      running at runlevel 2 as well.
    * The S99 sets the run priorities.  The init scripts start with S1
      and work up through 99 being the last.

1.12) Serial Port setup

    vi /etc/setserial.conf

       /dev/ttyS0 uart 16550A port 0x03f8 irq 4 \
            baud_base 115200 spd_normal skip_test
       /dev/ttyS1 uart 16550A port 0x02f8 irq 3  \
            baud_base 115200 spd_normal skip_test
       /dev/ttyS2 uart 16650V2 port 0x03e8 irq 9 \
            baud_base 115200 spd_normal skip_test
        /dev/ttyS3 uart 16650V2 port 0x02e8 irq 15 \
            baud_base 115200 spd_normal low_latency

    Make sure /etc/rc2.d has a S13setserial and it's executable.

** You are now ready for flight **

Currently (Summer2004) the balloon_tx program was started by hand
before flight. Mostly this was done because the startup parameters
were being tweaked on each flight.  This should be automated though.
There is a danger that the balloon might get launched without this
program running, in which case tracking becomes more difficult.

Section 2: Duplicating a working system onto other compact flash
```

cards:

2.1) It is fairly easy and straightforward to create a duplicate of a
   CF. There is a command called 'dd' which will create a byte by
   byte image of a CF.

   In order to do this use the following command:
      dd if=/dev/sda of=image_file.out bs=512

   The if= is the input device to be copied.
       of= is the output device or file.
       bs is the number of blocks to copy at a time. 512 is standard
       for most hard drives and CFs.

   - The above command will create a file of the same size of the
     compact flash.

   - It is best to do this as the root user.

   - /dev/sda is the device where the compact flash is.

   - The device should not be mounted.

   - The process can take a long time on a 1GB flash hooked to USB 1.1

   - Make sure you have enough room on the local hard drive to store
     the file image.

   - It is recommended that you encode the file with the date of the
     image, such as pebble_20050203.img

2.2) The process can be reversed by issuing the following command:
   dd if=image_file.out of=/dev/sda bs=512

   The CF can be un-formatted, but must be the same size or greater.
   The boot sector and partition table are part of the image so it
   should be rady to boot as soon as the command returns.

Section 3: Advanced methods of creating images

You want to save space on the local hard drive or time by only copying
part of the CF. You may also want to move to a CF of different size.

3.1) You can copy only a single partition with the dd command
        with the followling:

```
    dd if=/dev/sda1 of=image_sda1_20050203.img bs=512

    This creates an image file with just the first partion.
    On the e-ballon the first partion may only be 64 MB.

3.2) Now you want to create a CF that is only 256 MB.

    a) Put the new CF in the CF reader/writer.

    b) use cfdisk /dev/sda
       - create the new partiotion to the size you want.
       - set the first one as bootable.

    c) use lilo to make the CF bootable as outlined in section 1.

    d) use mkfs.ext2 /dev/sda2 to format the second/data partition

    e) mount the second partition: mount /dev/sda2 /mnt/cf2

    f) copy any files such as balloon_tx to /mnt/cf2

3.3) This will build a smaller or large CF with the partion one
        setup with the pebble boot image.  When it boots it should
        mount the second partition from the fstab table in partition
        one.

3.4)  Beware that not all 1GB flashes are the same exact size even
         though they say they are 1GB.  When copying images from one
         brand to another make sure they have the same byte counts
         when dd runs.

3.5)  If they are different then the technique in this section can
         be used instead of duplicating the entire CF.

Revison History:
========================================================================
   Version 1.0 by J.Battles
               (created original document)
   Version 1.1 by R.Sonnenfeld
   Version 1.2 by W.Walden
   Version 1.3 by R.Sonnenfeld
   Version 1.4 by R.Sonnenfeld
               (adding versioning info on DSCUD)
   Version 1.5 by J.Battles
               (Reformatted code. Clarified and cleaned up a few issues.)
```

APPENDIX C

DSCUD Driver Installation

```
##############################################################################
# Instructions for DSCUD 5.8 Installation on Prometheus
##############################################################################


1) Download LINUX-DSCUD58.tar.gz file from
   http://www.diamondsystems.com/support/software

2) Download a kernel soucre from
   http://www.kernel.org/pub/linux/kernel/v2.4/
   that matches the current version on pebble.
   Currently 2.4.26 so you would get linux-2.4.26.tar.bz2

note:Switch to root after step 2.

3) Move the kernel to the /usr/src directory
   mv linux-2.4.26.tar.bz2 /usr/src
4) Untar the kernel in /usr/src on the host system.
   cd /usr/src
   tar -jxvf linux-2.4.26.tar.bz2

5) Next we are going to start a kernel build, but not finish it so the
   driver install program can find what it needs.
   cd linux-2.4.26
   make menuconfig
        when the screen comes up select Exit and then say Y to save.
   make dep

6) cd /(where you put the DSCUD file and want to unpak it.

7) tar -zxvf LINUX-DSCUD58.tar.gz

8) cd dscud-5.8

9) cd gcc3   (This is assuming you have gcc version 3.x.x on your
   system.  You can check with gcc --version.  If you 2.x.x then do a
   cd gcc2 instead.
```

```
10) tar -zxvf dscud-5.8.tar.gz

11) cd dsucd5

12) ./install.sh

13) Hit enter.

14) Now you will see a list of kernel version on your system. Enter
    the number that corresponds to
    # ) Kernel 2.4.26 (/usr/src/linux-2.4.26)
    Hit enter again.

15) It should now compile and build the driver.  You should see
    something like:


--> Compiling kernel module for your system <--
rm -f dscudkp.o
gcc -c -o dscudkp.o dscudkp.c -O2 -D__KERNEL__ -DMODULE  \
    -I/usr/src/linux-2.4.26/include -D__SMP__ -DSMP
--> Installing module dscudkp.o in /lib/modules/misc <--
mkdir -p /lib/modules/misc
cp dscudkp.o /lib/modules/misc/

Step Three: Final Instructions
------------------------------

The dscudkp kernel module has been installed in
/lib/modules/misc/.  You must copy this
file to the same location on your target system.

The load.sh script will load the kernel module so
that it can be used by the driver.  You must run
this script each time the Linux system boots.  See
the README file for help with this.

Driver installation complete.

16) Copy the file to the Prometheus target with:

    scp /lib/modules/misc/dscudkp root@10.0.100.xxx:/lib/modules/misc
    scp /usr/local/dscud5/load.sh root@10.0.100.xxx:/etc/init.d/dscud

17)) ssh to the e-sonde and run depmod -a
```

This finishes installing and building the new drivers.

A few other files were built that need to be installed on the host development system.

18) mkdir /usr/local/dscud5
    cp dscud.h /usr/local/dscud5
    cp libdscud5.a /usr/local/dscud5

19) In the Makefile for any programs using the dscud5 lib the order of the libs must be changed so that -lm is after the other libraries. Otherwise, you will see compile erros like the following:
    : undefined reference to ‘pow’
    collect2: ld returned 1 exit status

    The make file should look have a line that looks like this:
    LIBS = -L/usr/local/dscud5 -ldscud5 -lrt -lm

    The reason for this is that the new dscud5 library makes use of the pow() funciton defined in the math library and so must be linked before the math library.

    This problem only occurs on programs linked with the -static option to the gcc compiler as in the case of balloon_tx.

20) Version of tools used during this instruction manual:
    Development platform: Slackware 10.1 running Linux 2.6.10.6
    gcc: gcc (GCC) 3.3.5
    GNU ld version 2.15.92.0.2 20040927
    glibc-2.3.4
    GNU Make 3.80
    binutils-2.15.92.0.2

APPENDIX D

NTP Daemon Checkout Procedures

```
NTP Pre flight check procedure.


1) Make sure gps is attached and sending good data to COM4 /dev/ttyS3

2) Type ntpq at command prompt as root

3) type the peers command at the ntpq> prompt.

ntpq> peers

remote       refid    st t when poll reach delay   offset  jitter
==============================================================================
*GPS_NMEA(0) .GPS.     0 l   8   64   37    0.000  -754.53  49.693
 LOCAL(0)    LOCAL(0) 10 l   3   64   37    0.000    0.000  0.001


You should get something similar to the above.  The thing to check is that
        1) the GPS_NMEA(0) line is present.
        2) The stratum (st) is set to 0
        3) The jitter column is less than 4000 ms.  4000ms is the max.
jitter. the deaomon will handle before ignoring the clock driver.
values of 50-200 are in the normal range.


The next command to check is
ntpq>clockvar

status=0000 clk_okay, last_clk_okay,
device="NMEA GPS Clock",
timecode="$GPGGA,224346,3403.9667,N,10654.4480,W,1,07,1.7,1453.7,M,-23.7,\
M,,*4F",
poll=3, noreply=0, badformat=0, baddata=0, fudgetime1=0.000, stratum=0,
refid=GPS, flags=0

Here you should see the last line from the gps under "timecode=$GPGGA.."
You can check the status fields for any erros in helping to debug.
```

```
noreply=  gives a count of how many times the ntpd tried to poll the gps
and didn't get a response.

Notes:
- The 'date' command can be used to see what the current time is.

-Use 'date -s hh:mm'  to set the time as close as possible to UTC.  This
will speed up the time lock on.

About ntp.conf in /etc

Should have a config line that looks like:

server 127.127.20.0 mode 2 burst prefer

       127.127.20.0 - sets ntpd to use NMEA serial gps /dev/gps0
            gps0 should be a link to /dev/ttyS? wher ? is 0-3

        mode 2 - Use NMEA GGA string for time.
    burst - when checking time use 2s burst intervals.  Time is set faster.
    prefer - Use as statum 0 (highest) level clock driver.
```

APPENDIX E

Log and Data File Naming Convention and Format

```
--------------------------------------
Log file format for balloon_tx program
--------------------------------------

The program creates two type of log file.  One master file for all the
position and slow scan data and one for fastscan data.

GPS positon and slow A/D format (stored on balloon device compact flash)

file naming convention:

     hostname_MMDDYYYY.log

    MM - Month starting at January=01
    DD - Day starting at 01
    YYYY - Year in 4 digits (ex. 2004)

Example of GPS log file:
     pebble_02152004.log

     Log file for pebble machine started on Feb. 15, 2004

     <<Data in Logfile with loss of GPS signal note no 2nd POS line>>
# Mon Feb 16 21:26:49 2004

POS,pebble,21,26,49.00,34.066216,N,106.907402,W,1446.9,1,09,01.1,2c^M
AD1,pebble,21,26,49.03687,-00.3708,000.0009,000.0674,000.0043,16^M
AD2,pebble,21,26,49.03687,000.0745,000.0302,-00.0677,-00.0247,17^M
AD3,pebble,21,26,49.03687,-00.0049,-00.0070,000.0037,-00.1279,10^M
AD4,pebble,21,26,49.03687,-00.1294,-00.2075,-00.2426,-00.1593,1b^M
AD1,pebble,21,27,26.03746,-00.3647,000.0012,000.1266,000.1657,14^M
AD2,pebble,21,27,26.03746,000.1492,000.1868,000.3711,000.6699,30^M
AD3,pebble,21,27,26.03746,-00.0018,-00.1923,-00.3259,-00.4291,0f^M
AD4,pebble,21,27,26.03746,-00.4071,-00.5048,-02.1924,-03.3752,18^M
    <<end of logfile>>

    <<Data Logfile with multiple program restarts on same day>>
```

```
# Mon Feb 16 21:26:49 2004

POS,pebble,21,26,49.00,34.066216,N,106.907402,W,1446.9,1,09,01.1,2c^M
AD1,pebble,21,26,49.03687,-00.3708,000.0009,000.0674,000.0043,16^M
AD2,pebble,21,26,49.03687,000.0745,000.0302,-00.0677,-00.0247,17^M
AD3,pebble,21,26,49.03687,-00.0049,-00.0070,000.0037,-00.1279,10^M
AD4,pebble,21,26,49.03687,-00.1294,-00.2075,-00.2426,-00.1593,1b^M
AD1,pebble,21,27,26.03746,-00.3647,000.0012,000.1266,000.1657,14^M
AD2,pebble,21,27,26.03746,000.1492,000.1868,000.3711,000.6699,30^M
AD3,pebble,21,27,26.03746,-00.0018,-00.1923,-00.3259,-00.4291,0f^M
AD4,pebble,21,27,26.03746,-00.4071,-00.5048,-02.1924,-03.3752,18^M
# Mon Feb 16 21:39:06 2004

POS,pebble,21,39,11.00,34.066174,N,106.907433,W,1431.7,1,08,01.4,26^M
AD1,pebble,21,39,11.03608,-00.3683,000.0012,001.3995,002.1896,21^M
AD2,pebble,21,39,11.03608,002.8766,003.9557,005.5457,009.0012,39^M
AD3,pebble,21,39,11.03608,-00.0034,000.3992,000.7239,000.6100,11^M
AD4,pebble,21,39,11.03608,000.7703,001.3589,002.6794,007.7908,41^M
POS,pebble,21,39,38.00,34.066174,N,106.907463,W,1428.3,1,08,01.4,34^M
AD1,pebble,21,39,38.03628,-00.3671,000.0009,000.1178,000.0867,20^M
AD2,pebble,21,39,38.03628,000.1584,000.1364,000.0632,000.1282,1c^M
AD3,pebble,21,39,38.03628,-00.0037,-00.1682,-00.2487,-00.3552,18^M
AD4,pebble,21,39,38.03628,-00.3555,-00.4477,-00.5612,-00.3174,1f^M
    <<end of logfile>>

Log string formats:
-------------------
POS or position:


POS,Inst ID,UTC Hour,Min,Secs,Latitude,Hemisphere,Longitude,Hemisphere,
         Altitude (m),gps fix,num sats,hdop,checksum

*The time for the POS string comes from the GPS.


A/D string:
AD?,Inst. ID,UTC Hour,Min.,Sec.,Voltage channel[0,4,8,12],V ch.[1,5,9.13],
                                V ch.[2,6,10,14],V ch.[3,7,11,15],checksum

*The time for the ADx strings come from the CPU system time.


Maximum Votage string:

MAX1,Inst ID,UTC hour,min,sec,max ch0.,max ch1.,max ch2.,max ch3.,checksum
```

*The max value represents the maximum voltage (either positive or negative)
that was encoutered while reading the fast A/D channels. The time stamp
is the time it was transmitted and not the precise time where it occured.
The values are reset after transmission.

Fast A/D format (stored on balloon device compact flash) ending in .edat

file naming convention:

hostname_MMDDYYYY_HHMM_VVV.edat

MM - Month starting at January=01
DD - Day starting at 01
YYYY - Year in 4 digits (ex. 2004)

HH - Hour (0 -23 UTC)
MM - Minute (0-59 UTC)

VVV - 3 digit file format version number. Currently 001.

Files are currently about 8.5M in size.

A new file is generated for each minute.

The file is composed of multiple two second data records
created with the following structure.

```
#define FAD_SAMPLES 20000
#define FAD_SAMPLE_RATE 10000

struct ad_capture{
  struct timespec time_start;
  struct timespec time_stop;
  short overflow;
  DSCSAMPLE ad0[FAD_SAMPLES];
  DSCSAMPLE ad1[FAD_SAMPLES];
  struct timespec time_start1;
  struct timespec time_stop1;
  DSCSAMPLE ad2[FAD_SAMPLES];
  DSCSAMPLE ad3[FAD_SAMPLES];
  struct timespec time_start2;
  struct timespec time_stop2;
  DSCSAMPLE ad4[FAD_SAMPLES];
```

```
  DSCSAMPLE ad5[FAD_SAMPLES];
  struct timespec time_start3;
  struct timespec time_stop3;
  DSCSAMPLE ad6[FAD_SAMPLES];
  DSCSAMPLE ad7[FAD_SAMPLES];
  };


time_start 8 bytes
time_stop  8 bytes
overflow   2 bytes
ad[0-7] 2 bytes * FAD_SAMPLES = 40000 each or 320000 total

--------------------------------------
Log file format for balloon_rx program
--------------------------------------

Raw GPS receive mode:

#POS, GPSRAW, hrZ,min,sec.xx,Lat,N,Long,W,Alt(m),rx mode,num sats,
      hdop,checksum

Examples:

POS,GPSRAW,21,13,05.00,33.981312,N,107.187920,W,3232.2,6,00,12.5,7e
POS,GPSRAW,21,13,16.00,33.981503,N,107.187973,W,3232.2,6,00,12.5,7c
POS,GPSRAW,21,13,17.00,33.981522,N,107.187973,W,3232.2,6,00,12.5,7e

The time comes from the GPS UTC time.
rx_mode is 0= no fix available and 1 = non-differential GPS fix
num sats is the number of satellites the GPS is tracking.
hdop is the horizontal dilution of preciscion in meters.

* There appears to be a bug in the balloon_rx program.  The rx_mode
should be either a 0 or 1 not a 6.  Also the hdop of 12.5 meters would
correspsond to a satellite of 6 not 0.  So there apears to be a problem
in the raw gps GGA string decoding and log writing.

Other file formats are the same for POS,AD?,and MAX1 strings as above.

Way Point file used by GPSDRIVE program to track the balloon:

Instr. ID:alt=<altitude>m latitude longitude

Examples:
```

```
Esonde03:alt=03231m +33.982182 -107.188141
Esonde03:alt=03231m +33.982182 -107.188141
Esonde03:alt=03232m +33.982162 -107.188156

*The latitude and longitude are converted to there proper plus and
minus values based on the hemispher character.  The first field is
actually the waypoint label as far as the GPSDRIVE program is
concerned.

Matlab output files:

Since matlab can't handle characters very well when trying to
convet the entire file into a matrix the text in the string type were
replaced with numbers, the instr. ID's removed,and the lat. and long.
were converted to their plus and minus values.  Other than those
changes,the stirngs remianed the same.

String coversion list:
0 = POS
1 = AD1
2 = AD2
3 = AD3
4 = AD4


Examples:

0,18,39,22.00,33.982197,-107.188133,3229.3,1,07,01.5
1,18,39,22.009,-00.0459,+00.0937,+00.0816,+00.3004
2,18,39,22.009,+00.0206,-00.7253,+00.5827,+01.6620
3,18,39,22.009,+05.1613,+05.8903,+08.4318,-08.6043
4,18,39,22.009,+05.2209,+03.1706,+00.8654,+00.5862

Naming convension:
Instr ID_MMDDYYYY.mat

Instr ID is the name of the transmitting instrument.
MMDDYYY is the the month, date and year of the start of
        when the file was written.

Esonde03_08182004.mat
```

APPENDIX F

Makefile, Function locations, and File Dependencies

```
CFLAGS = -I/usr/local/dscud5 -D_THREAD_SAFE -pthread -ggdb \
         -D_LINUX -march=i486 -static
LIBS = -L/usr/local/dscud5 -ldscud5 -lrt -lm

CFLAGS2 = -ggdb -D_LINUX
LIBS2 = -lm -lrt -lncurses

all: balloon_rx balloon_tx

balloon_rx : balloon_rx.o nmea.o balloon.o
    gcc -o balloon_rx balloon_rx.o nmea.o balloon.o $(CFLAGS2) $(LIBS2)

balloon_tx : balloon_tx.o nmea.o prometheus.o balloon.o
    gcc -o balloon_tx balloon_tx.o nmea.o prometheus.o balloon.o \
        $(CFLAGS) $(LIBS)

balloon_rx.o : balloon_rx.c
    gcc -c balloon_rx.c $(CFLAGS)

balloon_tx.o : balloon_tx.c
    gcc -c balloon_tx.c $(CFLAGS)

nmea.o: nmea.c nmea.h
    gcc -c nmea.c $(CFLAGS)

prometheus.o: prometheus.c prometheus.h
    gcc -c prometheus.c $(CFLAGS)

balloon.o: balloon.c balloon.h
    gcc -c balloon.c $(CFLAGS)

clean :
    rm balloon_tx *.o
```

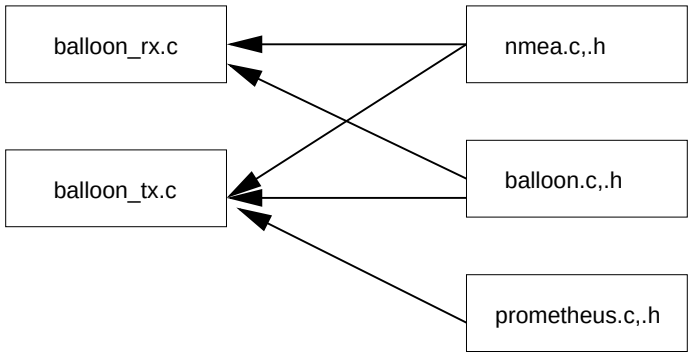


Figure F.1: Block diagram of file dependencies.

```
Function Locations for balloon_tx program
file: balloon_tx_funcs.txt
----------------------------------------

balloon_tx.c
 writedatafile()
 buildtxstring_ad()
 tx_gps_pos()
 get_and_tx_slow_ad()

balloon.c
 writelogfile()
 ser_readline()
 get_new_gps_pos()
 find_checksum()
 buildtxstring()

 open_gps_rx()
 open_radio_tx()
 close_gps()
 close_radio()

nmea.c

 atohex()
 nmea_checksum_check()
 parse_nmea_gga()

prommethues.c

 prom_ad_init()
```

```
 prom_ad_scan()
 prom_fad_init()
 prom_fad_start()

 prom_dio_init()
 prom_set_dio()
 prom_get_dio()
 prom_calibration()

 prom_sample_to_voltage()
 prom_find_max_voltage()


Function file locations
file: balloon-rx_funcs.txt
----------------------------

balloon_rx.c:
//Antenna pointing
 update_lx200()
 move_lx200()

//Program data handling functions
 rx_decode_pos()
 ad_decode_pos()
 nmea2gpspos()
 gpsdist_mi()
 theta_deg()
 check_time_elapsed()

 writecfgfile()
 readcfgfile()

 rxdata_chksum_check()

 csv_format()
 waypt_format()


// Program GUI functions
 rx_drawmainscreen()
 initscreendata()
 updatescreendata()
 init_keyboard()
 close_keyboard()
 kbhit()
```

```
 readch()


balloon_h:

 writelogfile()
 ser_readline()
 get_new_gps_pos()
 find_checksum()
 buildtxstring()
 hexstr2int()

//Comm related functions
 init_radio()
 get_signal_power()
 open_gps_rx()
 open_radio_tx()
 open_radio_rx()
 close_gps()
 close_radio()
 open_lx200()
 close_lx200()


nmea_h
 atohex()
 nmea_checksum_check()
 parse_nmea_gga()
```

APPENDIX G

Balloon E-field,B-field, and Coordinate System

Definition of Variables and Directions
Data flow from Raw to processed delta E-field data.

(Version 1.0 by Gao-Peng Lu / February 2005)
(Version 2.0 modified by Richard Sonnenfeld / March 2005)

1. Raw Data and Corrections thereto

The actual raw data values are A/D counts on the 16-bit A/D used by the Diamond Prometheus sonde mainboard. The board and driver have several settings. The settings selected for the Esonde project have the board/driver (Prometheus/DSCUD) provide integers directly into system memory on each data acquisition cycle. The integers (-9999 to 9999) represent –9.999 to 9.999 volts. We represent these integers by N_0 - N_3. Different settings of the board and driver could result in a different range of N, but we assume the board setup remains consistent., and thus it is accurate to refer to the N's as "raw-data". The actual raw-data (16 bit A/D counts) are never seen at any stage of our program.

The Balloon_TX program begins each data acquisition run by acquiring a set of offsets. It obtains these by cycling through all 16 channels of the board and recording the A/D value. The only sense in which these "offsets" are accurate is if one can assume that all the analog sensors are reading 0. This assumption in invalid for all the sensors but E-field. (For example, the Earth's B-field is certainly present at power on, and the temperature outside is likely not 0 C, nor is the pressure 0 torr.). In the future, we might obtain accurate offsets for some purposes by grounding all inputs of the A/D board before running the offset acquisition program. However, for the purpose of calculating E-fields, the offsets have some value. Because the delta-E board returns to baseline several seconds after a field change, and because the ambient field is likely to be constant when the sonde is first powered on, the offsets measured capture a combination of actual A/D offset and (the dominant factor) the DC offset shifts of the A/D board itself. Because of the circuit design, these shifts can be as large as 0.2 V (200 A/D counts). Balloon_TX creates a NNNNNNN_HH_MM.cal file. The cal file contains the offsets ($o_0 - o_{15}$) as decimal volts. There is (as yet) no gain calibration file, but one can imagine gain factors for the E-field board ($g_0 - g_3$). The gain factors are predicted to be in the range 0.98-1.02 based on the accuracy specifications of the components used in the E-field circuit and some bench testing of the response of several boards to sine-wave charge inputs.

Finally then, we calculate the V's in terms of the N's as follows.

$$V_I = g_I \times \left(\frac{N_I}{10000} - o_I \right)$$

2. Multiple sets of variables for same quantity

Table 1 shows 4 sets of variables for E-field. This is for clarity and consistency in data processing. The first 2 sets refer to the raw signals measured at individual sense plates. Sets 3 and 4 are both obtained by processing the raw data. The goal is ultimately to get

vector E-field information relative to the fixed Earth reference frame, but the first information is always sonde referenced (thus SE_X, SE_Y, SE_Z).
After calculating the sonde referenced values, we correct for the absolute sonde orientation (using the B-field information) to yield Earth referenced data.

3. Definition of positive direction for E-fields

Our individual amplifier channels were designed to be consistent with the 'lightning' or 'charge' convention, in that the sign of the output is the same as the size of the dominant charge in the neighborhood of the electrode. Thus, buffed Teflon (negatively charged) creates a negative voltage out of the amplifier when brought near. This is a useful fact to know. However, we have elected to represent electric fields using the "physics convention" rather than the old atmospheric-electric or charge convention. Thus, since a the presence of negative charge above the electrode induces positive charge on the electrode and has the same effect as a positive electric field (that is a field vector emanating from the electrode and pointing away from it), we wish to flip the sign of our outputs in additional calculations.

This can only be completely clear if you look at the definitions in the table. Defining:

$SE_X = (V_1 - V_0) \cdot \frac{\Gamma_{IP}}{p \cdot G_{xcal}}$ We see that if the E-field (physics convention) points along the arrow labeled SE_X in figure 1, then the function $SE_X(V_1,V_0)>0$. Similarly $SE_Y(V_3,V_2)>0$ if the E-field vector points as specified under SE_Y. Finally, if the E-field vector points up (fair weather field), $SE_Z(V_0,V_1,V_2,V_3)>0$. Note that this is a right-handed coordinate system, as it should be.

To unify the variable definition in later data processing, we define the variables associated with electric field as listed in Table 1. With the definition of the positive direction for electric fields in the Sonde frame, for example, if we have a positive SE_x, then the dominant charge nearby Sensor 0 is negative, which leads to a negative output for channel 0, while the channel 1 output is positive. Thus, to assure that the positive electric field corresponds with a positive value, the formula to get the electric field with the combination of channel data should be as shown below.

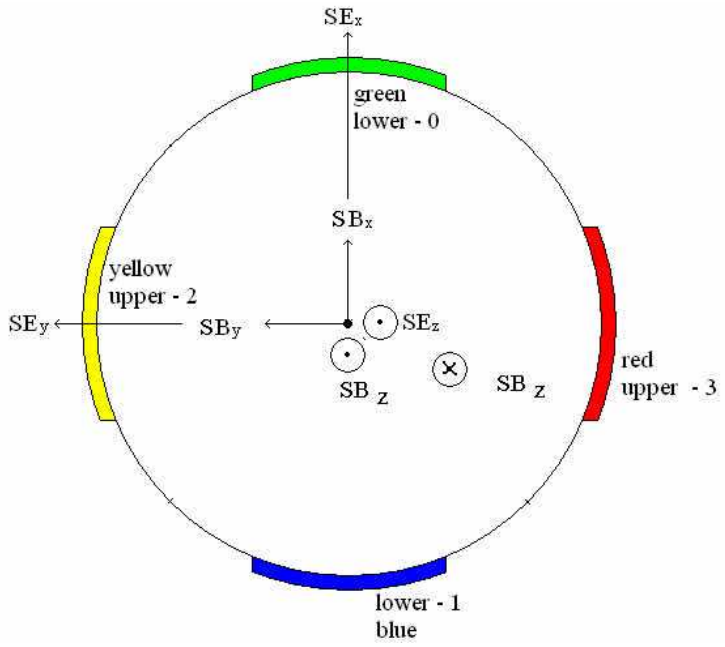


Figure 1. The top view of Sonde.

Set No.	Variable	Physical meaning
0	N_0	Integer value (-9999-9999) A/D channel 0
	N_1	Integer value (-9999-9999) A/D channel 1
	N_2	Integer value (-9999-9999) A/D channel 2
	N_3	Integer value (-9999-9999) A/D channel 3
1	V_0	Corrected Output voltage on ch 0
	V_1	Corrected Output voltage on ch 1
	V_2	Corrected Output voltage on ch 2
	V_3	Corrected Output voltage on ch 3
2	SE_x	x component in Sonde reference frame
	SE_y	y component in Sonde reference frame
	SE_z	z component in Sonde reference frame
	$SE_\perp$	$\sqrt{SE_X^2 + SE_Y^2}$
	$S\phi_E$	The sonde-referenced angle the E-field vector makes w.r.t. x in Sonde x-y plane

	$S\theta_E$	The sonde-referenced angle the E-field vector makes w.r.t. z in Sonde x-z plane
3	E_x	x component in Earth reference frame
	E_y	y component in Earth reference frame
	E_z	z component in Earth reference frame

Table 1. Definition of variables (E).

For single plate in the ideal buried guarded plate geometry, there is a simple relation (Γ_{IP}, for "ideal-plate gain") between the output voltage and the ambient electric field as follows,

$$V_{out} = \varepsilon_0 EA\frac{G_2}{C_1} = \Gamma_{IP} \times E \Rightarrow E/V_{out} = \Gamma_{IP} = 24.8\times10^3\,\mathrm{m}^{-1}.$$

What should be noted is that when using the combination of several plates to determine the ambient electric field, the transformation factor should be modified by the number (p) of the plates included, that is

$$SE_x = (V_1 - V_0)\cdot\frac{\Gamma_{IP}}{p\cdot G_{xcal}} = (V_1 - V_0)\cdot\frac{24.8\ \mathrm{kV/m}}{2\cdot1}\ ;\ \text{and}$$

$$SE_y = (V_3 - V_2)\cdot\frac{\Gamma_{IP}}{p\cdot G_{ycal}} = (V_3 - V_2)\cdot\frac{24.8\ \mathrm{kV/m}}{2\cdot1} = (V_3 - V_2)\cdot12.4\ \mathrm{kV/m}\ ;\ \text{and}$$

$$SE_z = (V_0 + V_1 - V_2 - V_3)\cdot\frac{\Gamma_{IP}}{p\cdot G_{zcal}} = (V_0 + V_1 - V_2 - V_3)\cdot\frac{24.8\ \mathrm{kV/m}}{4\cdot1} = (V_0 + V_1 - V_2 - V_3)\cdot6.2\ \mathrm{kV/m}$$

where G_{xcal}, G_{ycal} and G_{zcal} are correction factors due to distortion by the configuration of Sonde.
In the above calculations, $G_{xcal} = G_{ycal} = G_{zcal} = 1$. In actual fact, these gain factors aren't 1. They are probably closer to 2, and $G_{xcal} \neq G_{zcal}$

4. Definition of Positive Direction for B-fields

The 3-axis B-field board is installed into the sonde with sensors aligned as shown. It is installed such that V5-V7>0 if Earth's B-field is as shown in Fig. 1. Because Earth's B-field has a declination below the horizontal, generally V7 should be <0. For Sonde3 and earlier sondes, the B-field board was mounted upside down, thus making V7>0, but giving the board a left-handed coordinate system.

Set No.	Variable	Physical meaning
0	N_5	Integer value (-9999-9999) A/D channel 5
	N_6	Integer value (-9999-9999) A/D channel 6
	N_7	Integer value (-9999-9999) A/D channel 7
1	V_5	Corrected Output voltage on ch 5

	V_6	Corrected Output voltage on ch 6
	V_7	Corrected Output voltage on ch 7
2	SB_x	x-comp of B_{EARTH} in Sonde reference frame
	SB_y	y-comp of B_{EARTH} in Sonde reference frame
	SB_z	z-comp of B_{EARTH} in Sonde reference frame
3	B_x	x component of B in Earth reference frame
	B_y	y component in Earth reference frame
	B_z	z component in Earth reference frame

When SB_x is maximum and SB_y=0, we define the x direction of the sonde to be pointing north. Thus, if the field vector for E was along SE_x at this time, we would say the E-field was North in Earth based coordinates. When SB_y is maximum and SB_x=0, we define the y direction of the sonde to be pointing north, and the x-direction to point East. Thus, if the field vector for E was along SE_x at this time, we would say the E-field was East in Earth based coordinates.

The NASA earth Model says that Bz= …. By= …., Bx=….

5. Cartesian to Spherical Coords

The reason for all this work is to easily understand the E-field vector in spherical coordinates where it is useful. We use the coordinate definitions common in most physics texts, with theta as angle from North pole and Phi as positive CCW from X-axis. Since much data analysis will be done with MATLAB, we reference the "ATAN2" function which does an unambiguous 4-quadrant decoding of X and Y components of a vector.

With these preliminaries. $S\phi_E = \text{atan2}(SE_y, SE_x)$

$$S\theta_E = \text{atan2}(SE_Z, SE_\perp)$$

$\phi_B = \text{atan2}(SB_y, SB_x)$, $\theta_B = \text{atan2}(SB_Z, SB_\perp)$

$\phi_E = S\phi_E + \phi_B$

REFERENCES

[Beasley et al., 2000] Beasley, W. H., kenneth B. Eack, Morris, H. E., Rust, W. D., and MacGorman, D. R. (2000). Electric-field changes of lightning observed in thunderstorms. *Geophysical Research Letters*, 27(2):189–192.

[Coleman et al., 2003] Coleman, L. M., Marshall, T. C., Stolzenburg, M., adn P. R. Krehbiel adn W. Rison, T. H., and Thomas, R. J. (2003). Effects of charge and electrostatic potential on lightning propagation. *Journal of Geophysical Research*, 108(D9):12–1 thru 12–27.

[Cooray, 2003] Cooray, G. V. (2003). *The lightning flash*, volume 34 of *IEE power series*. Instittution of Electrical Engineers, London, United Kingdom.

[Krehbiel et al., 1979] Krehbiel, P. R., Brook, M., and McCrory, R. A. (1979). An analysis of the charge structure of lightning discharges to ground. *Journal of Geophysical Research*, 84(C5):2432–2456.

[MacGorman and Rust, 1998] MacGorman, D. R. and Rust, W. D. (1998). *The electrical nature of storms*. Oxford University Press, New York, NY.

[Marshall et al., 2005] Marshall, T., Stolzenburg, M., Maggio, C. R., Coleman, L. M., adn T. Hamlin, P. R. K., Thomas, R. J., and Rison, W. (2005). Observed electric fields associated with lightning initiation. *Geophysical Research Letters*, 32(L03813).

[Marshall et al., 1995] Marshall, T. C., Rison, W., Rust, W. D., Stolzenburg, M., Willett, J. C., and Winn, W. P. (1995). Rocket and balloon observations of electric field in two thunderstorms. *Journal of Geophysical Research*, 100(D10):20815–20828.

[Rakov and Uman, 2003] Rakov, V. A. and Uman, M. A. (2003). *Lightning: physics and effects*. Cambridge University Press, New York, NY.

[Shao and Krehbiel, 1996] Shao, X. M. and Krehbiel, P. R. (1996). The spatial and temporal development of intracloud lighting. *Journal of Geophysical Research*, 101(D21):26641–26668.