6-10-96

SANDIA REPORT

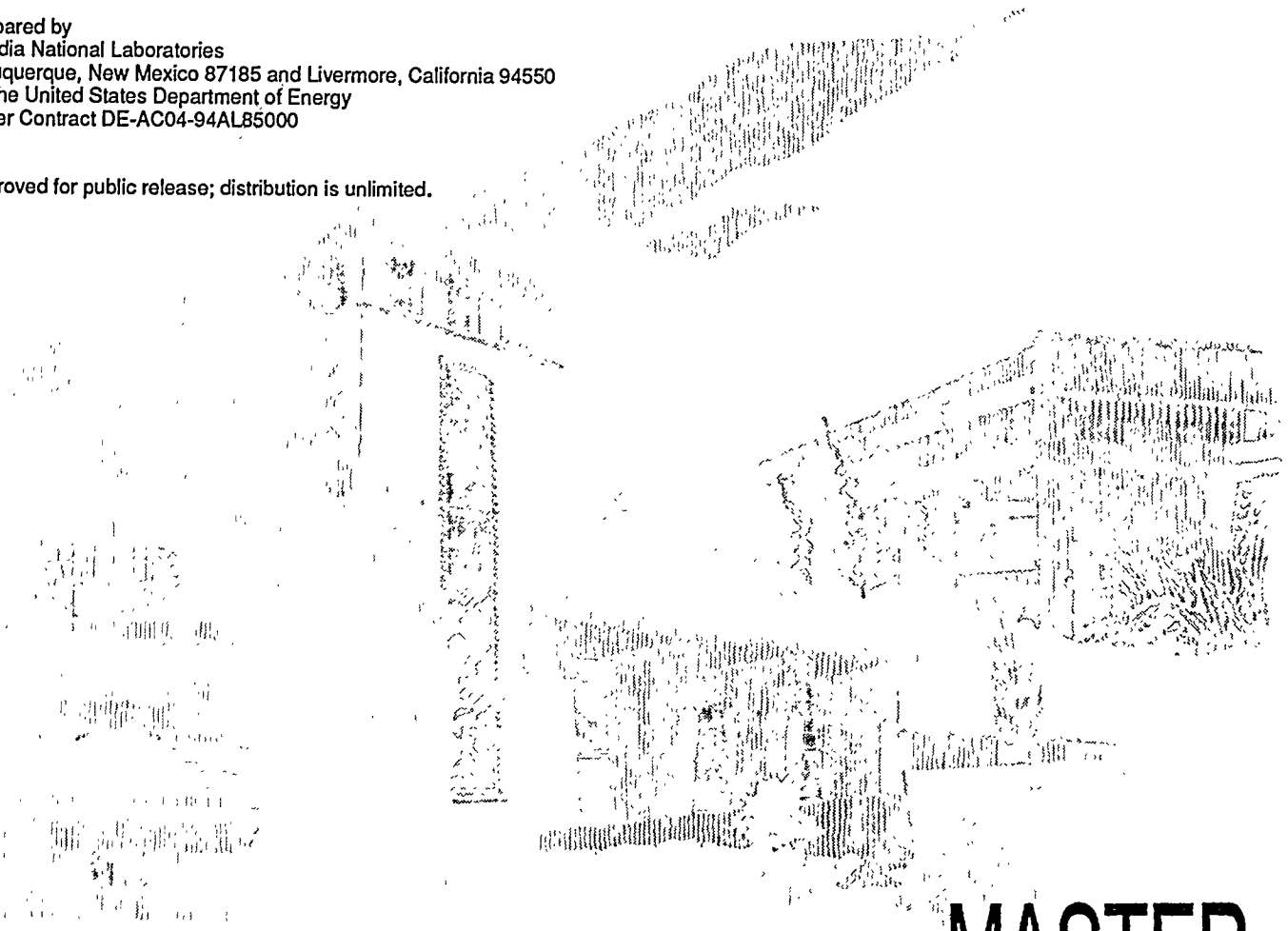
SAND96–1113 • UC–706
Unlimited Release
Printed April 1996

Sandia Airspace Recording System (SARS) Software Reference Manual

John L. Tenney

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-94AL85000

Approved for public release; distribution is unlimited.



Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
PO Box 62
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from
National Technical Information Service
US Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A05
Microfiche copy: A01

SAND96-1113
Unlimited Release
Printed April 1996

Distribution
Category UC-706

Sandia Airspace Recording System (SARS) Software Reference Manual

John L. Tenney
Assessment Technologies Department
Sandia National Laboratories
Albuquerque, New Mexico 87185

Abstract

SARS is a data acquisition system designed to gather and process radar data from aircraft flights. A database of flight trajectories has been developed for Albuquerque, NM and Amarillo, TX. The data is used for safety analysis and risk assessment reports.

To support this database effort, Sandia developed a collection of hardware and software tools to collect and post process the aircraft radar data. This document describes the data reduction tools which comprise the SARS, and maintenance procedures for the hardware and software system.

Acknowledgments

The author wishes to acknowledge the contributions Tom Lin and Norm Grandjean of Sandia National Laboratories made to the SARS project. They developed and tested many of the algorithms that make up the SARS.

Bob Roginski of Sandia National Laboratories provided sorting algorithms and technical consulting throughout the SARS project.

David Skogmo of Sandia National Laboratories provided hardware, software, and configuration support for the SARS project.

Table of Contents

List of Figures

List of Tables

1. Introduction

SARS is a system developed by Sandia for support in environmental studies at the Jupiter test facility in Albuquerque and the Pantex Plant in Amarillo, TX. Aircraft crash was determined to be a significant contributor to the environment if a crash occurred near a specific target. To support the analysis, Sandia developed a method to gather and analyze aircraft radar data in an efficient manner.

SARS is a hardware and software system which was developed to collect and analyze radar data. Although implemented for Amarillo and Albuquerque air traffic centers, SARS not site specific. The software and hardware configurations are the same for both airports.

This document describes the hardware and software system which comprise the SARS. It describes changes and enhancements made to existing software and provides documentation for new software. It is written for system personnel and programmers responsible for maintenance of the SARS system.

2. Overview of SARS

The production hardware ties directly into the radar facility at the local airport. The hardware responds to a input signal from the source radar and displays a point (or pixel) on the PC VGA screen. The dot represents the actual location of the aircraft as it flies overhead. A series of dots represents the entire flight path for the aircraft.

The hardware includes two PC's linked to two digitizers. The data is recorded on the local hard drive for each machine. Two machines are used a pseudo fault tolerant setup in case of failure.

Figure 2-1 depicts the interaction between the hardware, data collection software, and post-processing software.

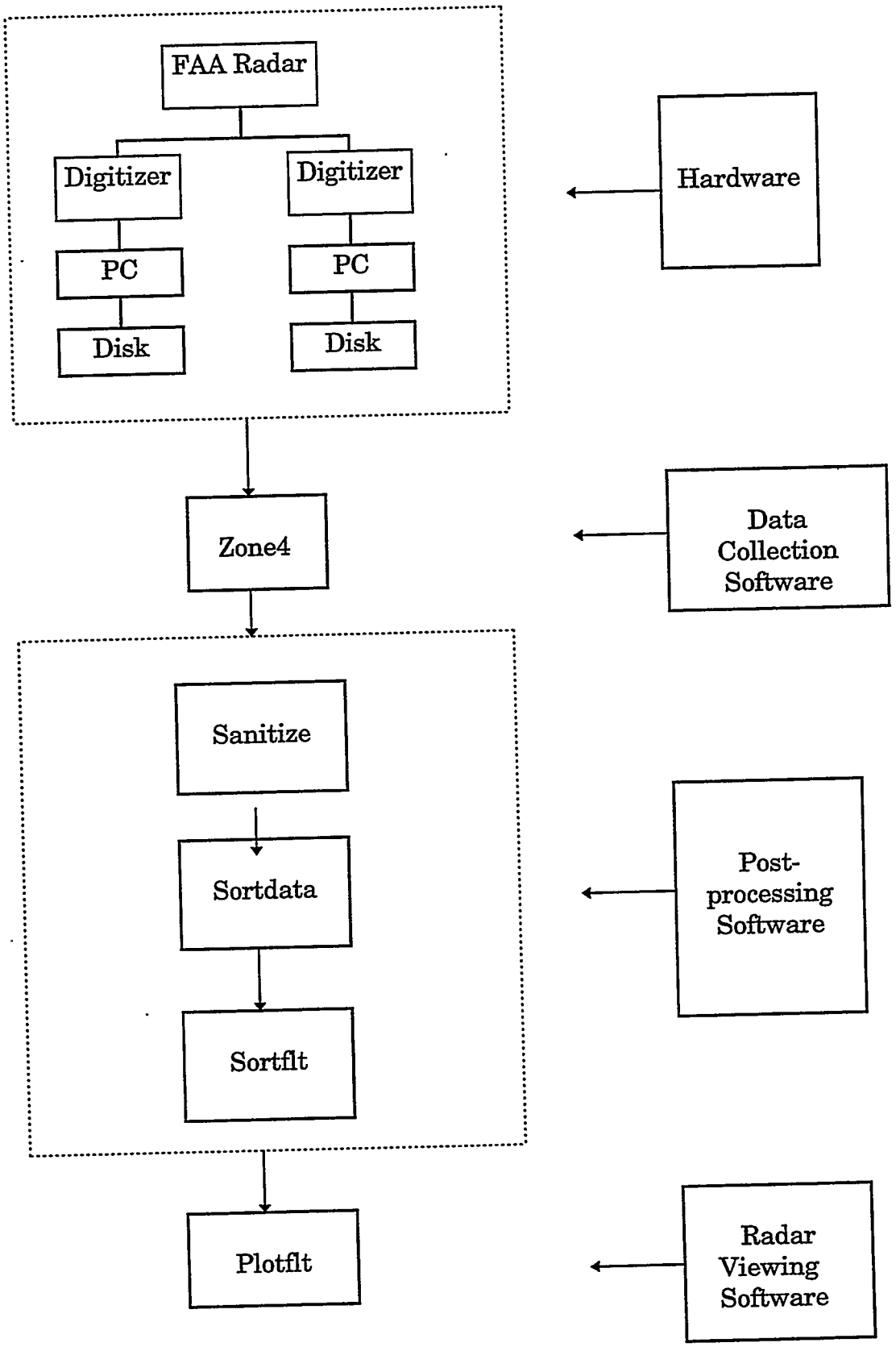


Figure 2-1. SARS Hardware, Data Collection and Post-processing System

The hardware interface diagram is shown in Figure 2-2. The interface is documented in [Ref. 1].

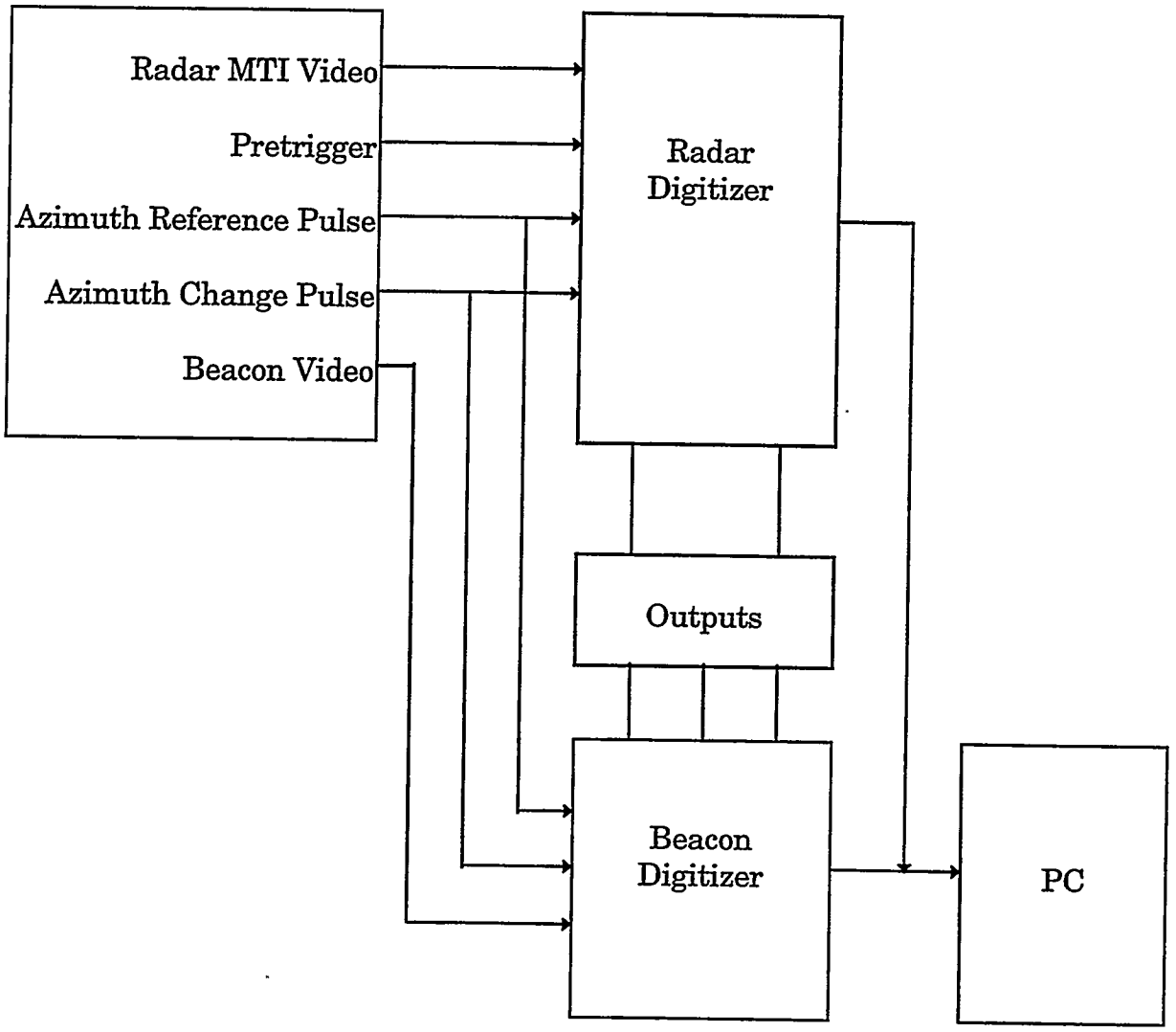


Figure 2-2. FAA/SARS Interface

Intentionally Left Blank

3. Data Collection Software

3.1. Zone 4 Software Description

Zone4 is the name of the primary data collection software. It is a modified version of the security detection software named RAMS (Radar Airspace Monitor System). This program processes radar signals from the FAA air traffic control center and converts the signals into graphics (see Figure 3-1). The flight paths consist of x,y,z coordinate information. The z component represents the elevation. These flight paths are saved to an ASCII formatted file while the graphics is displayed. The format is described in section 3-2.

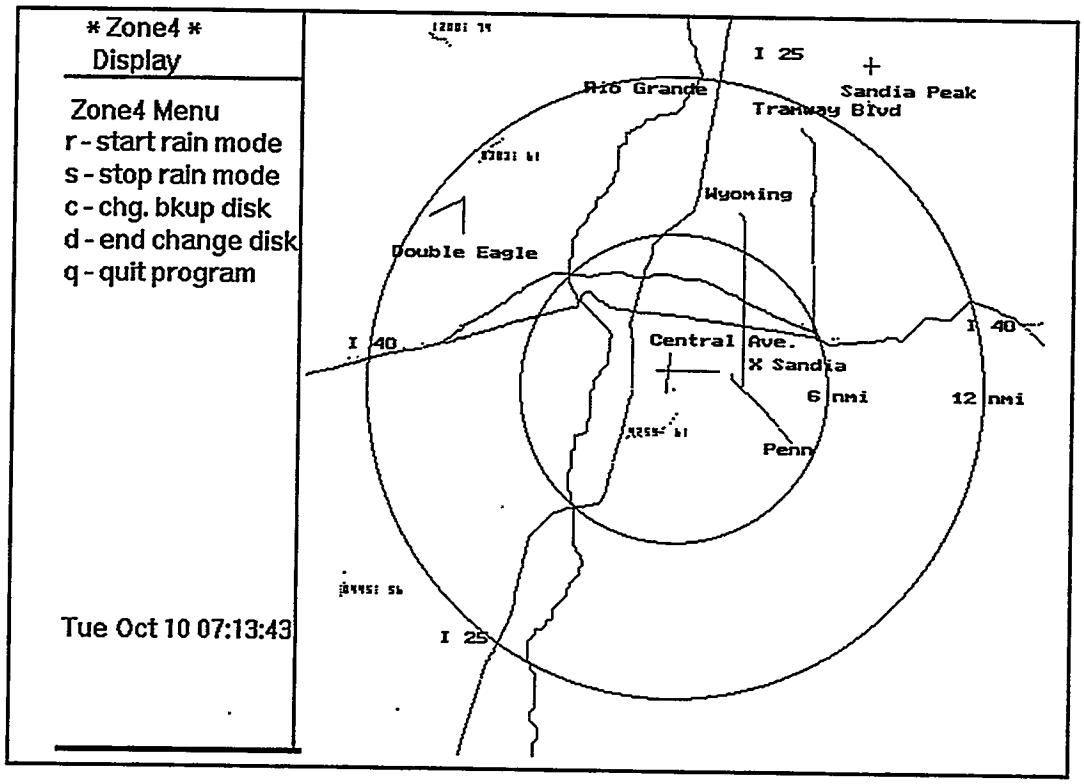


Figure 3-1. Zone 4 Output Screen

Figure 3-1 shows the flight trajectories plotted on the VGA display. This is a subset of actual flight data over Albuquerque on October 7, 1995. On the display, flight 4254:61 is plotted as a series of pixels or points near the center of the screen. The Beacon ID for this flight is 4254 and the elevation is 61 times 100 = 6100 feet. This flight is a takeoff heading to the southwest.

Zone 4 a real time program; the air traffic is plotted as it enters the airspace with minimum latency. The operator interface is shown on the left side of the screen. The additions and modifications to the original RAMS program are described in section 3.4.

3.2. Radar File Formats

Two types of radar are processed: Beacon and Primary. Beacon radar requires that the aircraft have a transponder in order to communicate with the air traffic tower. The Beacon code is a four digit number that is unique for a given time frame (e.g. 4254). Most commercial flights and large commuter aircraft use a Beacon ID's to identify the aircraft. Primary radar reflects off the skin of the plane. All aircraft, including commercial flights, are tracked by primary radar. Although not shown in Figure 3-1, primary radar will plot as a series of white dots near the trajectory of the Beacon radar path. The Beacon trajectory will plot as a series of blue dots. Small aircraft may or may not have a Beacon path, but it will always have the primary path plotted.

As a flight passes overhead, the radar data is captured to a file as it is displayed. At the end of the day, the file is closed and the next day is opened. Three types of radar files are saved for further analysis: Beacon, filtered primary, and unfiltered primary.

3.2.1. Beacon File Format

The Beacon radar file name use this format: BYYMMDD.DAT (e.g., B951107.DAT), where B is the first character, followed by the year, month, and day. An example of an unsorted Beacon file is shown is Table 3-1.

9511115145900S	145925	+1.33e+04	+1.79e+04	0452	64
9511115145901S	145925	+1.56e+04	-3.51e+04	0423	54
9511115145902S	145925	-1.50e+04	-2.65e+04	4357	77
9511115145903S	145925	-2.48e+04	+4.90e+03	0743	61
9511115145904S	145925	-4.47e+03	+4.98e+03	2602	50
9511115145905S	145925	-3.94e+03	+5.89e+03	7243	50
9511115145900P	145929	+1.33e+04	+1.89e+04	0452	64
9511115145901P	145929	+1.49e+04	-3.48e+04	0423	54
9511115145902P	145929	-1.63e+04	-2.62e+04	4357	75
9511115145903P	145929	-2.40e+04	+4.99e+03	0743	60
9511115145904P	145929	-3.88e+03	+4.83e+03	2602	50
9511115145905P	145929	-3.93e+03	+5.89e+03	7243	50
9511115145900P	145934	+1.32e+04	+1.97e+04	0452	65
9511115145901P	145934	+1.42e+04	-3.52e+04	0423	54
9511115145902P	145934	-1.80e+04	-2.57e+04	4357	73
9511115145903P	145934	-2.26e+04	+4.90e+03	0743	61
9511115145904P	145934	-2.78e+03	+4.39e+03	2602	50
9511115145905P	145934	-3.66e+03	+6.07e+03	7243	50
9511115145900P	145938	+1.38e+04	+2.11e+04	0452	66

Table 3-1. Beacon File Format

Each record in the Beacon file is fixed length 49 bytes.

The flight records are structured as follows:

Column (1:2)	Year
Column (3:4)	Month
Column (5:6)	Day
Column (7:8)	Start hour of flight (GMT)
Column (9:10)	Start min. of flight (GMT)
Column (11:12)	Flight sequence number (00 through 99)
Column (13:13)	(S)tart, (P)rogressive point , or (E)nd of flight
Column (15:16)	Time 1 - hour of flight
Column (17:18)	Time 1 - min. of flight
Column (19:20)	Time 1 - seconds of flight
Column (22:30)	X coordinate - true north (in feet)
Column (32:40)	Y coordinate - true north (in feet)
Column (42:45)	Beacon ID or squawk code
Column (47:49)	Elevation * 100 (in hundreds of feet)

Columns 1 through 13 are referred to as the unique ID of the flight. Every recorded flight will have one unique ID. This unique string is useful for sorting data.

Column 13 determines the complete path of a flight. The S(start), P..P..P..., E(end) sequence must be set for a good flight. The first 13 characters of each record (e.g., 940201101301S) are used throughout the programs as a unique identifier. The flight sequence number ranges from 00 through 99. This means 100 maximum flights can be recorded at one time.

The first record from Table 3-1 is identified by its individual components.

951115145900S 145925 +1.33e+04 +1.79e+04 0452 64

Column 1:2 - (95)	Year
Column 3:4 - (11)	Month
Column 5:6 - (15)	Day
Column 7:8 - (14)	Start hour of flight (GMT)
Column 9:10 - (59)	Start minute of flight (GMT)
Column 11:12 - (00)	Sequence number of flight
Column 13 - (S)	Start of flight
Column 15:16 - (14)	Time 1 recorded hour of data point (GMT)
Column 17:18 - (59)	Time 1 recorded minute of data point (GMT)
Column 19:20 - (25)	Time 1 recorded seconds of data point (GMT)
Column 22:30 - (+1.33e+04)	X coordinate true north (in feet)
Column 32:40 - (+1.79e+04)	Y coordinate true north (in feet)
Column 42:45 - (0452)	Beacon ID or squawk code of aircraft
Column 47:49 - (64)	Elevation * 100 = 6,400 ft.

3.2.2. Filtered Primary Radar Format

Usually all objects made of metal will be picked up by primary radar. Filtered radar data implies slow moving vehicles (e.g., trucks on a highway) are filtered from the raw data.

The filtered primary radar file name use this format: RYYMMDD.DAT (e.g., R951107.DAT), where R is the first character, followed by the year, month, and day. An example of an unsorted, filtered primary radar file is shown is Table 3-2.

951115145900S	145925	+1.36e+04	+1.84e+04	0000	0
951115145901S	145925	+1.55e+04	-3.49e+04	0000	0
951115145902S	145925	-1.50e+04	-2.67e+04	0000	0
951115145903S	145925	-2.51e+04	+5.00e+03	0000	0
951115145900P	145929	+1.36e+04	+1.92e+04	0000	0
951115145901P	145929	+1.50e+04	-3.52e+04	0000	0
951115145902P	145929	-1.64e+04	-2.64e+04	0000	0
951115145903P	145929	-2.41e+04	+5.00e+03	0000	0
951115145900P	145934	+1.37e+04	+2.03e+04	0000	0
951115145901P	145934	+1.41e+04	-3.50e+04	0000	0
951115145902P	145934	-1.82e+04	-2.58e+04	0000	0
951115145903P	145934	-2.29e+04	+5.24e+03	0000	0
951115145904S	145934	-5.46e+04	+4.63e+04	0000	0
951115145900P	145938	+1.37e+04	+2.11e+04	0000	0
951115145901P	145938	+1.36e+04	-3.54e+04	0000	0

Table 3-2. Filtered Primary Radar Format

The format of the filtered primary radar file is the same as the Beacon format. Each record is fixed length 49 bytes. This type of radar reflects off the skin of the aircraft; there will not be a Beacon ID or elevation associated with the flight. These values are set to 0 by default.

3.2.3. Unfiltered Primary Radar Format

Unfiltered radar will capture all moving objects, flocks of birds, and additional noise (e.g., radar antennas). These unfiltered files become very large. Eleven megabytes per day in Albuquerque is typical.

The unfiltered primary radar file name use this format: PYYMMDD.DAT (e.g., P951107.DAT), where P is the first character, followed by the year, month, and day. An example of an unfiltered primary radar file is shown is Table 3-3.

TIME=14:59:15	
658	261
347	373
961	577
309	636
691	728
795	1012
795	1630
442	1635
309	2076
397	2180
398	2184
743	2475
625	2756
392	3084
606	3460
903	3911
TIME=14:59:20	
660	273
351	356
946	571
309	636
697	726
807	1621
442	1648
309	2076
398	2216
397	2216
753	2497
623	2764
386	3089
903	3916

Table 3-3. Unfiltered Primary Radar Format

The format of this file is different from Beacon or filtered primary radar. Each flight starts with a TIME=HH:MM:SS syntax. The points that follow are the x and y coordinates of the flight path in screen pixels. The pixel values are converted within the Zone4 program from true north coordinates to screen coordinates.

3.3. Zone 4 Compile Instructions

The Zone4 software is controlled from a batch file name ZONE4A.BAT. It is executed from the command line in DOS: **ZONE4A**

A listing of ZONE4A.BAT for the Albuquerque radar follows:

```
@echo off
:start
CALLRAMS
if errorlevel 1 SETDIG
if errorlevel 1 ZONEA
if errorlevel 1 goto bottom
SLEEP 180
rem ctlaltdl causes a warm reboot of the computer
goto start
CTLALTDL
:bottom
CTLALTDL
echo on
```

The words in capital letters are programs that are invoked from the batch file. CALLRAMS dials up the digitizer at the local airport from a modem. SETDIG configures the digitizer for Beacon and primary radar. ZONEA is the main program for data collection. The SLEEP and CTLALTDL are programs that are used in case there is an error in the previous programs. Errorlevel 1 represents success; the next program will run only if the previous program finished successfully. If not, the computer will reboot.

ZONE4A is usually called from the autoexec.bat file. The software resides in a directory name zone4a on the recording computer. At the bottom of the autoexec.bat file there should be two entries:
cd zone4a
zone4a

The config.sys file in DOS should have the following entries:
files=30
buffer=30

Note: To prevent memory conflicts, there should be no memory managers (e.g., himem.sys) or other memory resident software (i.e., TSR's) loaded.

The Borland Turbo C++ 3.0 compiler for DOS was used to compile all of the programs associated with Zone4. A project file was required for the zonea.exe file. The path to \TC\BIN must be set before using the compiler.
The following procedure creates the Zonea executable file:

Type in: TC
Select the OPEN PROJECT menu option
Select the ZONEA.PRJ file
Select the COMPILE menu option
Select the BUILD ALL option

3.4. Additions/Modifications to Zone 4 Software

The Zone 4 software was modified from the original RAMS software for two specific reasons. First, the original RAMS software was not set up to handle a continuous, uninterrupted data stream. Zone 4 was created to handle user interrupts (i.e., disk change) without losing any real time data. Zone 4 also handles real time backup to multiple devices.

Second, the RAMS software was not set up to match the electronic data to the air traffic flight strips (i.e., green strips). The green strips contain pertinent information for safety analysts that is not contained in the electronic recorded data (e.g., aircraft type, aircraft ID). The information contained on the flight strips is required for safety analysis.

3.4.1. Green Strip/Electronic File Time Correlation

By default, the green strips are computer generated with GMT (Greenwich Mean Time) or UCT (Universal Coordinate Time). The local recorded data is adjusted to match the green strips. A sample green strip is shown is Figure 3-2.

SWA702	422	A1615	IFR			
T/B73S/A	SJN					
463	LAVAN		ABQ			

Figure 3-2. Sample Flight Arrival Green Strip

In this example, the aircraft ID is SWA702, the aircraft type is T/B73S/A, the Beacon ID is 4222 and the arrival time is 16 hours 15 minutes GMT.

Table 3-4 shows the correlation between the local time, the PC clock, and GMT. For the Albuquerque airport, green strips are picked up at 9 PM local time. In order to match this time with GMT, the PC clock and the Zone4 software is adjusted internally to match the green strip.

ABQ	Mtn.	Std.	Time	(Fall)			
G. S. Day	1	1	1	.....	1	1	1
Wall clock	9:00 pm	10:00 pm	11:00 pm	.....	6:00 pm	7:00 pm	8:00 pm
24hr. clock	21:00 pm	22:00 pm	23:00 pm	.....	18:00 pm	19:00 pm	20:00 pm
GMT G.S.	4:00 am	5:00 am	6:00 am	.....	1:00 am	2:00 am	3:00 am
PC clock	00:00 am	1:00 am	2:00 am	.....	21:00 pm	22:00 pm	23:00 pm
Offset add 4 hours	4	4	4	.....	4 Mod 24	4 Mod 24	4 Mod 24
Final Time	4:00 am	5:00 am	6:00 am	.....	1:00 am	2:00 am	3:00 am
ABQ	Daylight	Savings	Time	(Spring)			
G. S. Day	1	1	1	.....	1	1	1
Wall clock	9:00 pm	10:00 pm	11:00 pm	.....	6:00 pm	7:00 pm	8:00 pm
24hr. clock	21:00 pm	22:00 pm	23:00 pm	.....	18:00 pm	19:00 pm	20:00 pm
GMT G.S.	3:00 am	4:00 am	5:00 am	.....	00:00 am	1:00 am.	2:00 am
PC clock	00:00 am	1:00 am	2:00 am	.....	21:00 pm	22:00 pm	23:00 pm
Offset add 3 hours	3.	3	3	.....	3 Mod 24	3 Mod 24	3 Mod 24
Final Time	3:00 am	4:00 am	5:00 am	.....	00:00 am	1:00 am	2:00 am

Table 3-4 Green Strip/Electronic File Correlation

The objective in Table 3-4 is to match the green strip time to the final time. Since the green strip day ends on a different day than GMT, an adjustment is made to the data before it is saved to disk. For example, if we are in Albuquerque during DST and the wall clock is 9:00 PM, GMT time will be 6 hours ahead at 3:00 AM. Since 9 PM is the scheduled pick up time for the green strips, the Zone 4 program will close the current green strip day and open up the next day. The PC clock is set to 00:00 AM, then an offset of 3 hours is applied to the data before it is written to disk. The 9:00 PM local flight will

be recorded on disk as 3:00 AM GMT. This method will correlate the green strip time (GMT) with the recorded flight.

3.4.2. Zone 4 Display Screen

The output screen shown in Figure 3-1 was modified to accommodate the disk change utility. The menu choices are listed as follows:

r - start rain mode
s - stop rain mode
c - change backup disk
d - end change disk
q - quit program

The rain mode operation is useful if there is a rain or snow shower in progress. This will cause the primary radar to return unwanted noise thus increasing the primary radar file size. By pressing "r", start rain mode, the computer will not process primary radar until the "s", stop rain mode key is pressed, or by default, an elapsed time of two hours. At the end of two hours, the primary radar will begin to record.

The option for disk change is letter "c." This will close the file on the recording cartridge (typically a Bernoulli Lasersafe) and prompt the user to change the cartridge. A temporary file is created on the alternate backup device while the disk change occurs. The temporary file is copied back to the new cartridge after it is put in the Lasersafe drive. The "d" key is pressed after the cartridge is changed. This method prevents data loss while allowing the user to capture the recorded data.

The configuration options for the backup device and time zone offsets are read at startup from the configuration file SITE.CFG. It has the following format:

```
C:\ZONEDATA\     ;backup directory
1                ;backup flag 0-disable 1-enable
E                ;primary backup disk (E is Lasersafe Plus)
C                ;alternate backup device (for temp files)
4                ;Albuquerque offset hours DST=3 MST=4
```

The first line C:\ZONEDATA, specifies where the raw data files are stored. The second line specifies that the data will record to a backup device. A value of 1 enables backup while a value of 0 will record data on the C: drive only without a duplicate copy. The third line shows the device name if line 2 is enabled. In this example, the primary cartridge backup is logical device E:. The fourth line shows where the alternate backup device is as the disk change occurs. This is the device where temporary files are copied. The last line show the offset hours from the PC clock. If the PC clock is set to 6 am, the offset is applied to the recorded time and saved as 10 am if the offset is 4.

3.5. Zone 4 Software Diagram

Figure 3-3 shows the logical flow of the Zone 4 software. The process stays in an continuous loop once the connection is established to the digitizer.

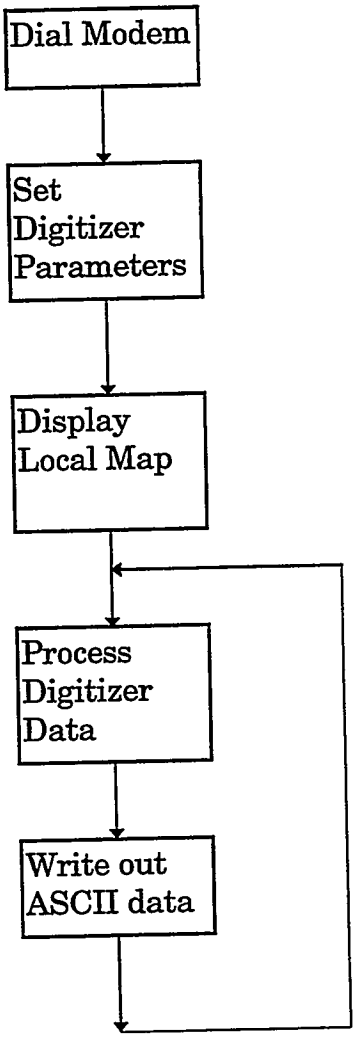


Figure 3-3. Zone 4 Software Diagram

3.6. Zone4 Module Description

The following list shows the entire module set for the Zone4 software. Two groups are listed: the software for the batch file and the software for the zonea executable. A brief description of each module is listed.

3.6.1. ZONE4A.BAT File

CALLRAMS.C - Callrams is a program that uses the Telebit T1000 modem to dial up the digitizer. It reads site specific information (e.g., phone number, access password) from the file PSMAP.CFG. The format of the PSMAP.CFG is listed:

```
0.00     ;xd=east/west coordinate in nautical miles from radar for program arb only
0.00     ;yd=north/south coordinate in nautical miles from radar
196.8    ;rcell=beacon range cell size in feet
196.8    ;rrcell=primary range cell size in feet
11.52    ;maxrg=coverage radius in statute miles
853.33   ;antrotrate=in azimuth ticks/sec=4096/period
1.0      ;tripradsq=radius of trip wire in miles
XXXX     ;password=password for digitizer
2        ;mode 0=primary only, 1=beacon only, 2=both
300      ;rgerr=beacon range error in range cells
-100     ;azerr=beacon azimuth error in azimuth ticks
245      ;rrgerr=primary range error in range cells
-100     ;razerr=primary azimuth error in azimuth ticks
ATDT2433178 ;modcmd=digitizer telephone number as a command to the modem
10.73    ;pfactor=10.73
cft/in@5kfeet;9.25@0ft;9.52@1kft;9.80@2kft;10.10@3kft;10.49@4kft
1        ;pressdefault. if 1 accept default pressure of 29.92 else stop and get pres.
367.0    ;maxtvel max threat speed in ft/sec. Faster than this is discounted.
59.0     ;mintvel min threat speed in ft/sec. Slower than this is discounted.
300.0    ;maxcrosstime in sec. Approach taking longer than this to cross are discounted
```

SETDIG.C
Setdig is a program used to configure the SRAMS digitizer.
It expects to find a disk file named PSMAP.DIG containing the information
to configure the digitizer. The file should appear as shown below.

```
2        ;mode 0=primary only,1=beacon only,2=both
3        ;beacon threshold. must have a value even if beacon not used.
6        ;primary threshold. must have a value even if radar not used.
1        ;pedit option: 1=beacon shares edit signal, else 0. must have value.
0        ;begin an edit parameter set 0 for clear 1 for set to 1. or end file.
0        ;AZST IN HEX, top three and bottom three bits=0
0        ;AZSP IN HEX, DITTO
0        ;RGST IN HEX, TOP 4 BITS AND BOTTOM 2 BITS =0
0        ;RGSP IN HEX, DITTO
1        ;NEXT EDIT PARAMETER SET HERE FOR 6 MILE BLOCK
BF8      ;AZST
F20      ;AZSP
130      ;RGST
30C      ;RGSP This line and all others must end with a line feed.
```

ZONEA.C
Program to track Beacon and Primary radar flight tracks. The data is saved into individual files.

SLEEP.C
If an error occurs with ZONEA, the program will put itself in suspend state with the sleep function. The program sleep will put the computer in a wait state for the specified number of seconds on the command line. For instance, the command sleep 180 will sleep for three minutes.

CTLALTDL.C
This program will causes a warm reboot of the computer. It is the equivalent of pressing the CTL-ALT-DEL keys. This function is used in case of a non-recoverable error.

3.6.2. Zonea Executable File

The following list shows the correlation between the PC filenames and the actual module names listed in the project file ZONEA.PRJ. There is a total of 24 modules in zonea.

File Name	Module Name
BTRKZONA.C	rtrack
BTRMZONE.C	btrimplt
CHGDISK.C	chgdisk
CHKDFREE.C	chkdfree
COMBINZ.C	combine
COPYTMP.C	copytmp
DELFILES.C	delfiles
ELEVZ.C	initelev
FILECPY.C	filecpy
GETDRLET.C	getdrlet
INIBOXZ.C	inibox
MONTHCVT.C	monthcvt
OPENBFIA.C	openbfil
OPENRFIA.C	openrfil
RCOMBINZ.C	rcombine
RTRKZONA.C	rtrack
RTRMZONE.C	rtrimplt
SITECFG.C	sitecfg
TBLINKZ.C	tblink
UNPKBZ.C	unpkb
UNPKRZ.C	unpkr
XSTRZ.C	xstr
ZONEA.C	zonea
ZONESCRN.C	safescrt

A brief description of each module is listed. More detail can be found in reference [Skogmo]. The modules in bold text are either modified or new additions to the original RAMS software.

BTRKZONA (module name: btrack) - This function provides tracking for the SRAMS Beacon radar data. It operates on a list of plane positions derived by the function btrimplt from Beacon radar data and converted x,y coordinates in feet from the site. The output of btrack is a list of plane positions at the plotting instant and their velocities.

BTRMZONE.C (module name: btrimplt) - This function accepts an array of range and azimuth Beacon radar reports and converts the array to Cartesian coordinates in feet from the position xd, yd. It trims out any points outside of the sector of interest defined by maxrg.

CHGDISK.C (module name: chgdisk) - This function swaps the primary and alternate backup devices. It also changes the input filename extension from .dat to .tmp or .tmp to .dat. For instance, if the primary disk is E: and the alternate is C:, the function will swap these values from E: to C:. This function is required for a disk change.

CHKDFREE.C (module name: chkdfree) - This function will check the available disk space on the selected drive. A warning beep will sound if the recording disk is more than 80% full.

COMBINZ.C (module name: combine) - This function combines multiple hits that are from the same plane. If two hits are within +/- 1 of the same range, with a certain span of azimuth, and don't have dissimilar id's, they are combined. This function is used for Beacon tracks.

COPYTMP.C (module name: copytmp) - This function copies the input file to the alternate backup device defined in SITE.CFG. This is required for a disk change. Calls routines chgdisk and filecpy.

DELFILES.C (module name: delfiles) - Delete all files in the C:\ZONEDATA directory that are older than 29 days. This prevents the C: drive from overloading. Calls routine monthcvt.

ELEVZ.C (module name: initelev) - This function initializes the constant arrays used by the function elevation. The elevation function is included in the ELEVZ.C file.

FILECPY.C (module name: filecpy) - This function copies one file to another and returns the number of bytes copied.

GETDRLET.C (module name: getdrlet) - This function returns the integer value for the disk device (i.e., A returns 0, B returns 1, ...)

INIBOXZ.C (module name: inibox) - This function checks to see if a point falls within a group of 4-sided polygons.

MONTHCVT.C (module name: monthcvt) - This function returns a two character month representing the integer for that specific month (i.e., JAN returns 01). Called from function delfiles.

OPENBFIA.C (module name: openbfil) - This function creates a descriptor for a plane based on the date, hour, and minute plus a two digit tag to allow for 100 concurrent planes. It writes the opening record for the plane into the Beacon file. This routine was modified to accommodate the offset hours to GMT. The following code segment shows the algorithm:

```
temphours = hours + offsethours
hours = temphours mod 24
write outputfile hours
```

The mod function is used to adjust for times greater than 24 hours (e.g., 24=0, 25=1).

OPENRFIA.C (module name: openrfil) - This is the same function as openbfil except it is used to open primary radar files (R files).

RCOMBINZ.C (module name: rcombine) - This function combines multiple hits that are from the same plane. If two hits are within +/- 1 of the same range, with a certain span of azimuth, they are combined. This function is used for primary radar tracks.

RTRKZONA.C (module name: rtrack) - This function provides tracking for the SRAMS primary radar data. It operates on a list of plane positions derived form the primary radar data and converted x,y coordinates in feet from the site. The output of rtrack is a list of plane positions at the plotting instant and their velocities.

RTRMZONE.C (module name: rtrimplt) - This function trims out points outside the sector of interest defined by maxrg. It operates on primary radar data.

SITECFG.C (module name: sitecfg) - This function reads site specific global variables from the SITE.CFG file. The backup flag determines if a copy of the data is needed for the alternate device.

TBLINKZ.C (module name: tblink) - This is a display function. It blinks the recent points plotted from their plot color to yellow in exclusive or mode. The state of the flash color (yellow or plot color) is given by the global variable flash, which is toggled by blink each time.

UNPKBZ.C (module name: unpkb) - This function moves the Beacon data from the receive buffer into the range, azimuth, id, and el buffers.

UNPKRZ.C (module name: unpkr) - This function moves the primary radar data fro the receive buffer into the range and azimuth buffers.

XSTRZ.C (module name: xstr) - This function draws the character string at point x, y in the passed color in exclusive or mode. This module contains the function xordot.

ZONEA.C (module name: zonea) - Zonea is the main program for data collection. It writes all processed data into the respective files (Beacon, primary, and unfiltered).

ZONESCRN.C (module name: safescrt) - This function displays the local site map.

In addition to the modules, there are common include files used in the data collection program.

COMNEWZ.C - This file contains the communications library for SRAMS.

GETCFG.C - Reads site specific information into global variables from the file PSMAP.CFG

READLIN.C - Reads characters from the passed stream into the passed buffer until the \n character is encountered. It terminates the buffer with a 0.

ZONEDEF.C - This file is the include file for all of the routines. It contains definitions and function prototypes.

Intentionally Left Blank

4. Postprocessing Software

4.1. Sanitize Software Description

The SANITIZE program does basic sanity checks on the input radar file. All the records in the file are 49 bytes long; if not, we print out the record number and the illegal string. The input radar file can be raw (.DAT) or sorted (.SRT) format.

Column 13 is checked for a valid S, P, or E character. If the character is missing or invalid, we have bad data - a message and the line number is printed.

Filename: SANITIZE.FOR

Compiler: Fortran Powerstation Ver. 1.0a for DOS

Compile instructions: FL32 SANITIZE.FOR

To execute: SANITIZE filename, e.g., SANITIZE B950112.DAT

The structure of the raw Beacon radar file is listed:
Each record is fixed length 49 bytes.

```
940201101000S 101017 +7.37e+03 +5.44e+04 1200  53
940201101001S 101017 +1.42e+04 +1.02e+04 0475  68
940201101002S 101017 +3.16e+04 +1.00e+04 0764 -20
                .
                .
```

The records are structured as follows:

- Column (1:2) Year
- Column (3:4) Month
- Column (5:6) Day
- Column (7:8) Start hour of flight
- Column (9:10) Start min of flight
- Column (11:12) Flight sequence number (00 thru 99)
- Column (13:13) (S)tart, (P)rogressive, or (E)nd of flight
- Column (15:16) Time 1 - hour of flight
- Column (17:18) Time 1 - min of flight
- Column (19:20) Time 1 - seconds of flight
- Column (22:30) X coordinate - true north
- Column (32:40) Y coordinate - true north
- Column (42:45) Beacon ID or squawk code
- Column (47:49) Elevation * 100

4.2. Sanitize Calling Sequence

Two subroutines are used in SANITIZE. A brief description of each subroutine follows:

READ_FILE - open up input file
CHECK_SPE - checks SPE count

The routines are all called from the main program. The calling sequence follows:

Read the input file from the command line
CALL READ_FILE (IFILE)

Do sanity checks
CALL CHECK_SPE (IFILE)

IFILE is a character variable read in from the command line. If the filename is not entered the following message will appear on the screen:

*** PLEASE ENTER FILENAME ON COMMAND LINE ***
SYNTAX: SANITIZE [radar_filename]
e.g., SANITIZE B950126.DAT

If the program completes successfully, the program will print out the number of records in the file. For example:

OUTPUT FROM SANITIZE:
FILE B950112.DAT CONTAINS 11706 RECORDS.

4.3. Sanitize Subroutine Description

SUBROUTINE READ_FILE (IFILE)
CHARACTER*(*) IFILE

Function name: READ_FILE
Purpose: Open up input file and count characters and total records
Parameters: IFILE (input file name - character string)
Sample call: CALL READ_FILE (IFILE)
Calls routine: none
Called from: main program
Return value: none

Messages returned:

If the input file cannot be found, the following message will appear on the display:
FILE B950112.DAT NOT FOUND

If a record is not equal to 49 bytes, the next message will appear:
Invalid record # = 1170

SUBROUTINE CHECK_SPE (IFILE)
CHARACTER*(*) IFILE

Function name: CHECK_SPE
Purpose: This routine does some basic sanity checks. First it determines if character position 13 of the input record has a S, P, or E in that column. At least 1 character (S, P, or E) must be found in column 13; if not a error message will be displayed on the screen.

The S,P,E characters represent the following:
S - start of flight
P - continuation point of flight
E - end of flight

The routine will check for a valid numeric in column 49 (elevation) This will confirm that the record is complete.

Parameters: IFILE (input file name - character string)
Sample call: CALL CHECK_SPE (IFILE)
Calls routine: none
Called from: main program
Return value: none

Messages returned:

If an invalid integer is found in column 49, the next error will appear:
Invalid integer in column 49 at line 1250

If an illegal character (other than S, P, or E) then the message will appear:
Illegal character in column 13 at line 1250

If there is not at least one record with an S, P, or E in column 13, then one of the following messages will appear:
Starting records not found - DATA ERROR
No primary records found - DATA ERROR
No ending records found - DATA ERROR

4.4. Sortdata Software Description

Filename: SORTDATA.FOR

This program reads and sorts the Albuquerque and Amarillo aircraft flight data. The program reads input from the Beacon radar file or primary radar (.DAT extension) and generates a sorted file (.SRT extension). It is expected that the SANITIZE program is run before this program is executed. SORTDATA uses an index array (INDX) for sorting.

Compiler: Fortran Powerstation Ver. 1.0a for DOS

Compile instructions: FL32 SORTDATA.FOR

To execute: SORTDATA filename, e.g., SORTDATA B950112.DAT

File dependencies: SORTDATA.INC is the required include file

The output file (.SRT extension) is sorted on two keys: the date/time stamp is the primary key(field 1, characters 1:13) and the secondary key is the start time of the flight (field 2, characters 15:20). Sample output is listed below:

```
940201101000S 101017 +7.37e+03 +5.44e+04 1200  53
940201101000P 101021 +7.95e+03 +5.52e+04 1200  52
940201101000P 101026 +7.80e+03 +5.59e+04 1200  51
940201101000P 101031 +7.45e+03 +5.63e+04 1200  51
940201101000P 101036 +8.10e+03 +5.76e+04 1200  50
940201101000P 101040 +8.35e+03 +5.83e+04 1200  50
940201101000P 101045 +8.64e+03 +5.92e+04 1200  50
940201101000E 101049 +0.00e+00 +0.00e+00 0000   0
```

4.5. Sortdata Calling Sequence

Three subroutines are used in SORTDATA. A brief description of each routine follows:

READ_FILE - open up input file
SORT1 - perform modified shell sort
WRITE_FILE - write out sorted file

All routines are called from the main program. The calling sequence follows:

Read the input file from the command line
  CALL READ_FILE

Sort the input file based on the date/time stamp

An index array is used for sorting.
CALL SORT1

Write out the sorted file
CALL WRITE_FILE

After the program finishes, the following message will print to the screen:
OUTPUT FROM SORTDATA:
A TOTAL OF 11706 RECORDS WRITTEN TO FILE B950112.SRT

All routines use a common include file named SORTDATA.INC. This file contains variables and arrays common for all routines. A listing follows:

Filename: SORTDATA.INC - include file for SORTDATA.FOR

Global variable definitions:

MXRECS - maximum number of records in radar file
LINE - array of 49 byte character strings
IFILE - input file string
OFILE - output file string (sorted file .srt)
NRECS - counter for number of records in file
INDX - index array (used for sort routines)

```
PARAMETER (MXRECS=300000)
CHARACTER LINE*49, IFILE*12, OFILE*12
INTEGER NRECS, INDX
COMMON NRECS, LINE(MXRECS), INDX(MXRECS), IFILE, OFILE
```

Two arrays are declared with a dimension of 300000. Currently, this value is large enough to hold all data records in memory for sorting.

4.6. Sortdata Subroutine Description

SUBROUTINE READ_FILE
Function name: READ_FILE
Purpose:
This routine reads the filename from the command line and populates the LINE and INDX global arrays.
Parameters: none
Calls routine: none
Called from: main program
Return value: none
Globals modified: NRECS, LINE, INDX

Messages returned:

The input file variable, IFILE, is declared as global and read in from the command line. If the filename is not entered, then the following message will appear:

*** PLEASE ENTER FILENAME ON COMMAND LINE ***
SYNTAX: SORTDATA [filename]
e.g., SORTDATA B950112.DAT

If the input file, IFILE, does not exist then the next message will appear:
FILE B950112.DAT NOT FOUND

If the input file exceeds the MXRECS of 300000, then the next message will appear:
*** EXCEEDED 300000 RECORDS ***

SUBROUTINE SORT1
Function name: SORT1
Purpose:
This routine uses a modified shell sort using 2 keys:
the 1st is the date/time stamp (1:12)
the 2nd is the flight time column (15:20)
This routine was obtained from Bob Roginski, 12333
Parameters: none
Sample call: CALL SORT1
Calls routine: none
Called from: main program
Globals modified: INDX
Return value: none

Messages returned: none

SUBROUTINE WRITE_FILE
Function name: WRITE_FILE
Purpose: write out the sorted filename (.SRT extension)
Parameters: none
Sample call: CALL WRITE_FILE
Calls routine: none
Called from: main program
Globals modified: none
Return value: none

Messages returned: none

4.7. Sortflt Software Description

Filename: SORTFLT.FOR

This program separates the good flight data from the unresolved flight data. Three output files are written: a .IDX file for the flight number, start, and end points for the good flight paths; a .BID file sorted by Beacon ID and a .RPT file for flight summary information. These files are required if there is a need to plot the flight paths. This program will process both Beacon and Primary radar files.

The good data has a continuos flight path; this is determined by the 13th column of the input file. The input file is assumed sorted (.SRT extension)

Compiler: Fortran Powerstation Ver. 1.0a for DOS

Compile instruction: FL32 SORTFLT.FOR

To execute: SORTFLT filename, e.g., SORTFLT B950112.SRT

Output files: Index files are required for viewing aircraft flight paths. The format of the three output files (.IDX, .BID, and .RPT) is described below.

4.7.1. Flight Index File Description

This file is fixed length 138 byte ASCII records and is created with a .IDX extension (e.g., B950112.IDX). It contains summary information for every flight in a given day. Start time, end time, start and end elevations and Beacon data is kept in the flight index file. This scheme allows a quick search of the sorted radar file. It also contains the updated information for the green strips. The index file is sorted by flight number (column 1). The flight number is a sequential number, starting at 1, which represents the order in which the flight appeared for a given day. The sequence pattern is 1, 2, 3, 4, ...,; each flight will be assigned a unique, sequential flight number. A sample index file (sorted by flight number) is shown in Table 4-1. The data is from Albuquerque on February 2, 1994.

1	1	8	940201101000	1200	101017	101045	53	50	0	1	3	#	############	############	#	####	0	###
2	9	68	940201101001	0475	101017	101458	68	121	0	1	1	#	############	############	#	####	0	###
3	69	98	940201101002	0764	101017	101226	51	65	0	1	1	#	############	############	#	####	0	###
4	99	130	940201101003	4240	101017	101235	79	111	0	1	1	#	############	############	#	####	0	###
5	131	137	940201101004	7262	101017	101040	280	280	0	1	1	#	############	############	#	####	0	###
6	138	163	940201101005	1200	101017	101208	64	54	0	2	3	#	############	############	#	####	0	###
7	164	179	940201101006	0473	101017	101121	77	60	0	1	1	#	############	############	#	####	0	###
8	180	193	940201101007	2667	101017	101113	111	117	0	1	1	#	############	############	#	####	0	###
9	194	205	940201101008	4234	101017	101103	95	95	0	1	1	#	############	############	#	####	0	###
10	206	218	940201101009	0461	101036	101126	20	30	1	2	2	#	############	############	#	####	0	###
11	219	252	940201101010	0741	101054	101322	65	134	0	1	1	#	############	############	#	####	0	###
12	253	264	940201101100	0510	101154	101240	260	260	0	1	2	#	############	############	#	####	0	###
13	265	314	940201101300	1200	101312	101653	58	55	0	3	3	#	############	############	#	####	0	###
14	315	348	940201101301	3241	101335	101603	279	260	0	1	1	#	############	############	#	####	0	###
15	349	357	940201101302	0510	101340	101412	260	260	0	2	2	#	############	############	#	####	0	###
16	358	392	940201101303	2660	101349	101621	83	55	0	1	1	#	############	############	#	####	0	###
17	442	482	940201101500	4206	101507	101808	70	56	2	1	2	#	############	############	#	####	0	###
18	483	503	940201101501	4206	101507	101635	120	86	0	2	2	#	############	############	#	####	0	###
19	504	523	940201101502	4250	101540	101703	49	49	0	1	1	#	############	############	#	####	0	###
20	561	568	940201101600	1310	101635	101703	388	391	0	1	1	#	############	############	#	####	0	###
21	569	585	940201101601	0461	101645	101755	50	60	0	2	2	#	############	############	#	####	0	###

Table 4-1. B940201.IDX Flight Index Table

The individual fields are defined below. The first record is used for reference.

Field 1(1:8) Flight number - this is the unique sequential number for the given flight. In the file listed above, the flight number for the first record is 1.

Field 2(9:15) Start of flight record; this column identifies the specific record for the start of flight in the corresponding radar file (.SRT file). In the first record, flight number 1 is the first recorded flight on February 1.

Field 3(16:24) End of flight record. The end record in B940201.SRT is 8.

Field 4(26:37) Unique ID for flight (e.g., 940201101000). This field matches the unique ID in the sorted radar file. The 94 represents the year, 02 the month, 01 the day, 10 the start hour of the flight, 10 the start minute, and 00 the sequence number for the flight. If two or more aircraft are flying at the same time, then the second flight will use a 01 suffix, the third a 02 suffix.

Field 5(39:42) Beacon ID or squawk code (e.g., 1200).

Field 6(44:49) Start time of the flight in HHMMSS (UCT or GMT) (e.g., 10:10:17)

Field 7(51:56) End time of flight in HHMMSS (GMT) (e.g., 10:10:45)

Field 8(58:60) Start elevation of flight (x 100) (e.g., 53*100=5,300 feet)

Field 9(62:64) End elevation of flight (x 100) (e.g., 50*100=5,000 feet)

Field 10(66:66) Flight indicator field. The indicator is defined as follows:

0 - good flight, no Beacon ID change
1 - Beacon ID changed from 1200 to another number
2 - Any Beacon Id number changed to any other number
3 - BID 1200 changed to another number, then changed again

In the sample flight, the flight indicator is 0 (no BID change). This information is important for attaching green strips to electronic data.

Field 11(71:71) Sequence number of Beacon ID (e.g., 1 of 3 total BID 1200 flights).

Field 12(76:76) Total count of Beacon ID's (e.g., 3 total BID 1200 flights).

The next group of columns are updated when the green strip and display program (PLOTFLT) is run.

Fld. 13(78:78) VFR/IFR/Unknown indicator. Three choices are allowed in this field: U - Unknown, I - Instrument, V - Visual

Fld. 14(80:91) Aircraft ID (e.g., AAR123)

Fld. 15(93:104) Aircraft type (e.g., B747)

Fld. 16(106:106) Time prefix; E(fly over), P(departure), or A(arrival)

Fld. 17(108:111) Time 24 hour format (GMT); (e.g., 1017)

Fld. 18(113:113) Match indicator field; 0-not matched, 1-matched. If the green strip matches the electronic data then set this field to 1, otherwise the default is 0.

Fld. 19(115:117) Initials of operator entering green strips (e.g., JLT).

Fld. 20(119:138) Blank spaces (reserved for future use).

4.7.2. Beacon Index File Description

The Beacon index file has the same format as the flight index file (.IDX). The only difference is the Beacon index file (.BID) is sorted and saved based on the Beacon ID (Field 5, values 0000 through 9999). Because of memory limitations on the PC, it was easier to process the BID disk file, and plot by Beacon ID, in the PLOTFLT program. The flight index file could have been used for plotting but this would require a sort inside of the PLOTFLT program. Hence, plot speed would have decreased and memory requirements increased. A sample Beacon ID index file from February 4 is shown in Table 4-2.

10	206	218	940201101009	0461	101036	101126	20	30	1	2	2	#	############	############	#	####	0	###
21	569	585	940201101601	0461	101645	101755	50	60	0	2	2	#	############	############	#	####	0	###
7	164	179	940201101006	0473	101017	101121	77	60	0	1	1	#	############	############	#	####	0	###
2	9	68	940201101001	0475	101017	101458	68	121	0	1	1	#	############	############	#	####	0	###
12	253	264	940201101100	0510	101154	101240	260	260	0	1	2	#	############	############	#	####	0	###
15	349	357	940201101302	0510	101340	101412	260	260	0	2	2	#	############	############	#	####	0	###
11	219	252	940201101010	0741	101054	101322	65	134	0	1	1	#	############	############	#	####	0	###
3	69	98	940201101002	0764	101017	101226	51	65	0	1	1	#	############	############	#	####	0	###
1	1	8	940201101000	1200	101017	101045	53	50	0	1	3	#	############	############	#	####	0	###
6	138	163	940201101005	1200	101017	101208	64	54	0	2	3	#	############	############	#	####	0	###
13	265	314	940201101300	1200	101312	101653	58	55	0	3	3	#	############	############	#	####	0	###
20	561	568	940201101600	1310	101635	101703	388	391	0	1	1	#	############	############	#	####	0	###
16	358	392	940201101303	2660	101349	101621	83	55	0	1	1	#	############	############	#	####	0	###
8	180	193	940201101007	2667	101017	101113	111	117	0	1	1	#	############	############	#	####	0	###
14	315	348	940201101301	3241	101335	101603	279	260	0	1	1	#	############	############	#	####	0	###
17	442	482	940201101500	4206	101507	101808	70	56	2	1	2	#	############	############	#	####	0	###
18	483	503	940201101501	4206	101507	101635	120	86	0	2	2	#	############	############	#	####	0	###
9	194	205	940201101008	4234	101017	101103	95	95	0	1	1	#	############	############	#	####	0	###
4	99	130	940201101003	4240	101017	101235	79	111	0	1	1	#	############	############	#	####	0	###
19	504	523	940201101502	4250	101540	101703	49	49	0	1	1	#	############	############	#	####	0	###
5	131	137	940201101004	7262	101017	101040	280	280	0	1	1	#	############	############	#	####	0	###

Table 4-2. B940201.BID Beacon Index File

4.7.3. Sortflt Report File Description

A report file is generated from the SORTFLT program. It has a .RPT extension. Pertinent information such as percentage of good flights, duplicate Beacon ID's, and flight 1200 count are saved. Flight 1200 is usually a unknown or default Beacon ID if the plane does not have a transponder. In Albuquerque, flight 1200 is also used by small planes and student pilots. A sample printout is shown below (B940201.RPT).

```
Input file = b940201.srt
Record count =    600
Scount =    25
Pcount =    554
Ecount =    21
Othcnt =    0
Total flight count =    21
Percent good records =  83.17%
Max distance =    5852.35 Feet
Max velocity =    831.30 MPH
Max distance flight num =    20
Number of flights which change Beacon IDs during flight =    1
*******************************************************************************
TOTAL FLIGHT COUNT:    21
Number of flights (unique BID =1):    12
Number of flights (non-uniq. BID >1):    9
Percent of file with unique Beacon IDs: 57.14%
Percent of file with duplicate Beacon IDs: 42.86%
*******************************************************************************

TOTAL FLIGHT COUNT =    21
```

Number of flights with flight count = 1 : 12
Number of flights with flight count = 2 : 6
Number of flights with flight count = 3 : 3
Number of flights with flight count = 4 : 0
Number of flights with flight count = 5 : 0
Number of flights with flight count = 6 : 0
Number of flights with flight count = 7 : 0
Number of flights with flight count = 8 : 0
Number of flights with flight count = 9 : 0
Number of flights with flight count = 10 : 0
Number of flights with flight count > 10 : 0
****

TOTAL COUNT OF 400 AND 1200 FLIGHT NUMBERS AND PERCENT: 7 33.33%
Total count of 400 flight numbers and percent: 4 19.05%
Total count of unique 400 flights (unique or = 1): 2
Total count of duplicate 400 flights (duplicate or >1): 2

Total count of 1200 flights and percent: 3 14.29%
****

TOTAL COUNT OF FLIGHTS W/O 400/1200 AND PERCENT: 14 66.67%
Number of flights with flight count = 1 : 10
Number of flights with flight count = 2 : 4
Number of flights with flight count = 3 : 0
Number of flights with flight count = 4 : 0
Number of flights with flight count = 5 : 0
Number of flights with flight count = 6 : 0
Number of flights with flight count = 7 : 0
Number of flights with flight count = 8 : 0
Number of flights with flight count = 9 : 0
Number of flights with flight count = 10 : 0
Number of flights with flight count > 10 : 0
****

Duplicate Beacon IDs

Beacon ID	Count
461	2
510	2
1200	3
4206	2

****

*** Records and flight numbers > MACH 3 ***
Record Flight number

4.8. Sortflt Include File

All subroutines in SORTFLT (except HIGHLOW) use an include file named SORTFLT.INC. This file defines global parameters, arrays, and record structures. A listing of the include file follows:

Filename: SORTFLT.INC - include file for SORTFLT.FOR

Global variable definitions:

MXRECS - Max number of records in radar file
MAXFLT - Max number of Beacon flights
MAXBID - Max number of unique Beacon ID's (0..9999)
LINE - Array of 49 byte character strings
RADARTYPE - Flag for (B)eacon or (P)rimary radar
IFILE - Input filename
NRECS - counter for number of records in file
FLIGHT - Array of start/end record structures
FLTCNT - Flight counter (S, P .. P .. P, E sequence)
FLTIDX - Integer array (0-unresolved 1-good flight 2-exceeds MACH3)
FLTNUM - Int array which correlates record numbers and flight numbers
TOTFLTCHG - Number of flights which chg Beacon ID within the flight
XVAL - X value array
YVAL - Y value array
BID - Beacon ID array of all records (MXRECS)
BIDARR - Cumulative counter of individual Beacon ID's
DIST - Distance array (dist between X, Y points)
VEL - Velocity array
MAXVEL - Maximum velocity of VEL array
MAXDIST - Maximum distance of DIST array
SCOUNT - Column 13 S counter
PCOUNT - Column 13 P counter
ECOUNT - Column 13 E counter
OTHCNT - Column 13 other counter
FLIGHT - Record structure of flight number, start and end recs.
MAXDISREC - Record number of max flight distance
MAXDISFLTNUM - Flight number of max flight distance
SWEEPTIME - 1 revolution of radar (4.8 secs)
MACH3DIST - Distance (ft.) traveled at mach 3 (740mph = mach 1)

```
PARAMETER (SWEEPTIME = 4.8)
PARAMETER (SECPERHR = 3600)
PARAMETER (FTPERMILE = 5280)
PARAMETER (MACH3DIST = 15645.145)
PARAMETER (MXRECS=300000, MAXFLT=20000, MAXBID=9999)
```

```
CHARACTER LINE*49(MXRECS), RADARTYPE*1, IFILE*12
INTEGER NRECS, FLTCNT, MAXDISREC, MAXDISFLTNUM
INTEGER SCOUNT, PCOUNT, ECOUNT, OTHCNT, TOTFLTCHG
INTEGER*1 FLTIDX(MXRECS)
INTEGER FLTNUM(MXRECS)
INTEGER BIDARR(0:MAXBID)
INTEGER BID(MXRECS)
REAL XVAL(MXRECS), YVAL(MXRECS), DIST(MXRECS), VEL(MXRECS)
REAL MAXVEL, MAXDIST

Record structure
STRUCTURE /IDX_FILE/
   INTEGER FLTNUM          ! Flight number (sequential)
   INTEGER SREC            ! Start of flight record number
   INTEGER EREC            ! End of flight record number
   CHARACTER PRIKEY*12     ! Primary key LINE(1:12)
   CHARACTER BEACON*4      ! Beacon ID
   CHARACTER STHR*2        ! Start hour
   CHARACTER STMIN*2       ! Start minute
   CHARACTER STSEC*2       ! Start second
   CHARACTER ENDHR*2       ! End hour
   CHARACTER ENDMIN*2      ! End minute
   CHARACTER ENDSEC*2      ! End second
   CHARACTER STELEV*3      ! Start elevation
   CHARACTER ENDELEV*3     ! End elevation
   INTEGER*1 FLTIND        ! Flight indicator field
   INTEGER BIDSEQ          ! Beacon ID sequence
   INTEGER BIDCNT          ! Beacon ID count
END STRUCTURE

Declare an array of flight structures
    RECORD /IDX_FILE/ FLIGHT(MAXFLT)

Common
    COMMON /ZONE4/ NRECS, FLTCNT, SCOUNT, PCOUNT, ECOUNT, OTHCNT,
   *                TOTFLTCHG
    COMMON FLTIDX, FLTNUM, LINE, RADARTYPE, IFILE, XVAL, YVAL, DIST,
   *          VEL, MAXVEL, MAXDIST, FLIGHT, MAXDISREC,
   *          MAXDISFLTNUM, BIDARR, BID
```

4.9. Sortflt Calling Sequence

Seven subroutines are called in SORTFLT. A brief description of each routine follows:

READ_FILE - Read input file from command line; populate global arrays
SPE_COUNT - Sum all S, P ,E characters in column 13
SORT_FLIGHT - Separated good data from incomplete data
DIST_VEL - Calculate max distance and max velocity between data points
WRITE_FLTNDX - Write flight index file (.IDX)
GEN_REPORT - Generate report file (.RPT)
SORT2 - Sort and generate Beacon ID index file (.BID)

All routines are called from the main program. The calling sequence follows:

Read input file
  CALL READ_FILE

Do sanity checks
  CALL COUNT_SPE

Separate the good flights from the unresolved flights
  CALL SORT_FLIGHT

Calculate distance and velocity between points
  CALL DIST_VEL

Write out start and end pts. of complete flights (.IDX)
  CALL WRITE_FLTNDX

Generate report file (.RPT)
  CALL GEN_REPORT

Sort data based on Beacon ID and write .BID file
  CALL SORT2

Upon completion of the program, the following confirmation message will appear:

```
OUTPUT FROM SORTFLT:
WRITING FLIGHT INDEX FILE: B940201.IDX
WRITING REPORT FILE: B940201.RPT
WRITING INDEX FILE: B940201.BID
```

4.10. Sortflt Subroutine Description

SUBROUTINE READ_FILE
Function name: READ_FILE
Purpose:
This routine reads the input file from the command line
and populates the global arrays.
Parameters: none
Sample call: CALL READ_FILE
Calls routine: none
Called from: main program
Return value: none
Globals modified: RADARTYPE, IFILE, LINE, XVAL, YVAL, BID
FLTIDX, FLTNUM, DIST, VEL, NRECS, BIDARR

Messages returned:

If the input file, IFILE, is not present on the command line, then the following message will appear:

*** PLEASE ENTER FILENAME ON COMMAND LINE ***
SYNTAX: SORTFLT [filename]
e.g., SORTFLT B950112.SRT

The first character in the input file must be an 'R', 'r', 'B', or 'b'. If not, the next message will appear:

First character in B940201.SRT must be B or R

If the input file does not exist, or if an invalid name is entered, then the next message will appear:

File B940201.SRT not found

If the read statement find invalid values for XVAL or YVAL , the next message will appear:

Invalid record found -> record # 1016

SUBROUTINE COUNT_SPE
Function name: COUNT_SPE
Purpose: Counts the S, P, and E characters in column 13
Parameters: none
Sample call: CALL COUNT_SPE
Calls routine: none
Called from: main program
Return value: none

Globals modified: SCOUNT, PCOUNT, ECOUNT, OTHCNT

S - start of flight
P - continuation point of flight
E - end of flight

If an illegal character or missing character is found in column 13, then the following message will appear:

Bogus data found at line 1227

SUBROUTINE SORT_FLIGHT
Function name: SORT_FLIGHT
Purpose:
This routine checks for the sequence of S, P..P..P, E for
a complete flight. The flight counter FLTCNT is incremented
if we complete a sequence.
Parameters: none
Sample call: CALL SORT_FLIGHT
Calls routine: none
Called from: main program
Return value: none
Globals modified: FLIGHT, FLTIDX, FLTNUM, BIDARR

Messages returned: none

SUBROUTINE DIST_VEL
Function name: DIST_VEL
Purpose:
This routine calculates the maximum distance and the maximum velocity between points.
Parameters: none
Sample call: CALL DIST_VEL
Calls routine: none
Called from: main program
Return value: none
Globals modified: MAXDIST, MAXDISTREC, MAXVEL

SUBROUTINE WRITE_FLTNDX
Function name: WRITE_FLTNDX
Purpose:
This routine writes out the flight number, start, and end records for the valid flights. The output index file is created with a .IDX extension
Parameters: none
Sample call: CALL WRITE_FLTNDX
Calls routine: none
Called from: main program
Return value: none
Globals modified: MAXDISFLTNUM

Messages returned:

OUTPUT FROM SORTFLT:
WRITING FLIGHT INDEX FILE: B940201.IDX

SUBROUTINE GEN_REPORT
Function name: GEN_REPORT
Purpose:
This routine writes out important parameters including:
record count, S, P, and E count and the flight count
The output file is created with a .RPT extension.
Parameters: none
Sample call: CALL GEN_REPORT
Calls routine: none
Called from: main program
Return value: none
Globals modified: none

Messages returned:

WRITING REPORT FILE: B940201.RPT

SUBROUTINE SORT2
Function name: SORT2
Purpose:
This routine does a modified shell sort using 3 keys:
The primary key is the Beacon ID (42:45)
the 2nd is the date/time stamp (1:12) and
the 3rd is the flight time column (15:20)
A index array (INDX) is used for sorting.
Once sorted, the output file is written with a .BID extension
The sorting algorithm was obtained from Bob Roginski, 12333
Parameters: none
Sample call: CALL SORT2
Calls routine: none
Called from: main program
Return value: none
Globals modified: none

This routine uses the flight index file for sorting (.IDX) and writes out a new Beacon index file (.BID). If the .IDX is deleted or missing, then the following message will appear:

File B940201.IDX not found

After the sort is finished, the following message will appear:

WRITING INDEX FILE: B940201.BID

SUBROUTINE HIGHLOW (ARRAY, START, END, HIGH, LOW, AVG)
Function name: HIGHLOW
Purpose:
This routine does basic statistics given an real array.
The routine calculates the high, low and average values
of a given range in a array. This routine was obtained
from the Numerical Recipes book.
Parameters: ARRAY - real array
START - start point in array
END - end point of array
HIGH - highest value (returned)
LOW - lowest value (returned)
AVG - average value of range (returned).
Sample call:
CALL HIGHLOW (DIST, SREC, EREC, MAX, MIN, AVG)
Calls routine: none
Called from: WRITE_FLTNDX
Return value: none
Globals modified: none

Messages returned:

If the starting record in the array is 0, the following message will be displayed:
Start value = 0 in HIGHLOW routine

Intentionally Left Blank

5. Aircraft Viewing Software

5.1. Plotflt Software Description

Filename: PLOTFLT.C

Function:
Plotflt is a DOS based plot program that allows analysts to replay recorded air traffic. This program permits replay of daily flights sequentially, by range, or by Beacon ID.

The plotflt program also allows users to attach green strips (from the local air traffic center) to the recorded electronic flight data.

Compile instructions: comp.bat (uses Turbo C++ 3.0 compiler)
Manual compile: tcc -ml plotflt.c graphics.lib
note: the tc\bin directory must be in the path

File dependencies: The PLOTFLT.H include file is required.

Command line execution instructions: plotflt filename (e.g. plotflt B940429.SRT)

Background:
This program was adapted from the RAMS program chektrak which was written by David Skogmo, 12/5/93.

Required input files:
PSMAP.CFG - Contains maximum radius for display
PSMAP.MAP - Graphics file for local airport and streets
PSMAP.TXT - Text file for labels
EGAVGA.BGI - Borland graphics driver
ABQPTX.CFG - Configuration file for data input mode
BYYMMDD.SRT - Sorted radar file
BYYMMDD.IDX - Flight index file
BYYMMDD.BID - Beacon index file

5.2. Main Program Description

The plotflt program consists of one main program and forty functions. The main program reads parameters (e.g. map coordinates) from configuration files and opens the radar files for viewing. The main program controls selection of both display and data input modes. Figure 5-1 shows the process.

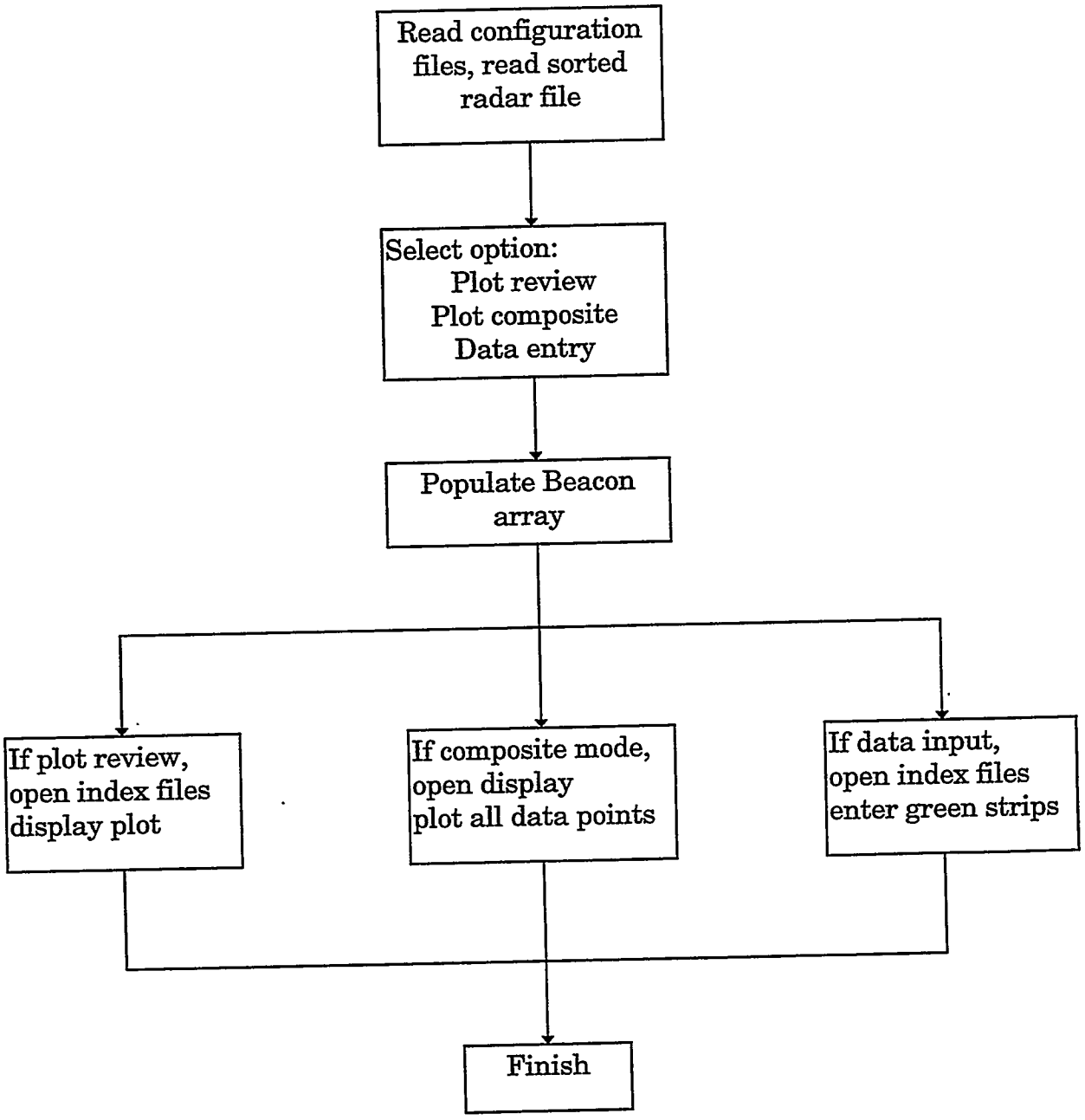


Figure 5-1. Main Program Diagram

5.3. Main Program Calling Sequence

The main program controls calling sequence for entire program. First, the abqptxcfg function is called to set up site specific parameters for Albuquerque and Amarillo. Next the open_radar_file is called; a valid .SRT file is expected. Select_mode is called next: the user selects plot display, plot composite, or data input mode. If plot display or data input mode are selected, then the user can change default plot options (i.e., plot speed). Next, the bid array, G_bid, is populated with Beacon ID integers from the Beacon ID file. This array is used to check if valid Beacon ID's are entered during plotting or data entry. A check is made to make sure we don't exceed the maximum number of flights per file. A limit was set because of memory concerns in DOS. Finally, depending on the mode selected, main will branch to display_plot, safescrt, or the input_data function. A diagram showing the calling sequence and pseudo code follows.

```
abqptxcfg();              ! Get Albuquerque and Amarillo data input parameters

open_radar_file();        ! Open sorted radar file

select_mode();            ! Review, composite, or data input mode

If mode = review or data input then
  select_plot_opt();      ! Select plot options
end if

pop_bid_arr();            ! Populate the bid array and calculate filesize

If mode = review then
  open_idx_files("rb");   ! Open index files read only
  display_plot();
else if mode = composite then
  safescrt();             ! Display screen
  trkplot();              ! Plot all flight paths
  closegraph();           ! Close graphics screen
else if mode = data input then
  open_idx_files("r+b"); ! Open files for update
  input_data();           ! Input green strip parameters
end if

end of main
```

Messages Returned:
If the maximum number of flights for a given day is exceeded, the following message will appear on the screen:

Max flights 2005 is greater than 2000.
Contact system personnel.

The maximum number of flights is currently set to 2000. This value is named MAXFLT and is defined in the include file PLOTFLT.H. This value has never been exceeded after processing two years of data at the Albuquerque airport.

5.4. Plot Display Mode

5.4.1. Overview

Plot display mode permits a user to view aircraft trajectories for a given day. The flights can be viewed sequentially (as they occurred during the day) or by Beacon ID. The flights are plotted on a VGA compatible screen as a series of dots. The plot speed is controlled by the user upon startup. Multiple flights can be displayed on the same screen. An sample plot is shown in Figure 5-2.

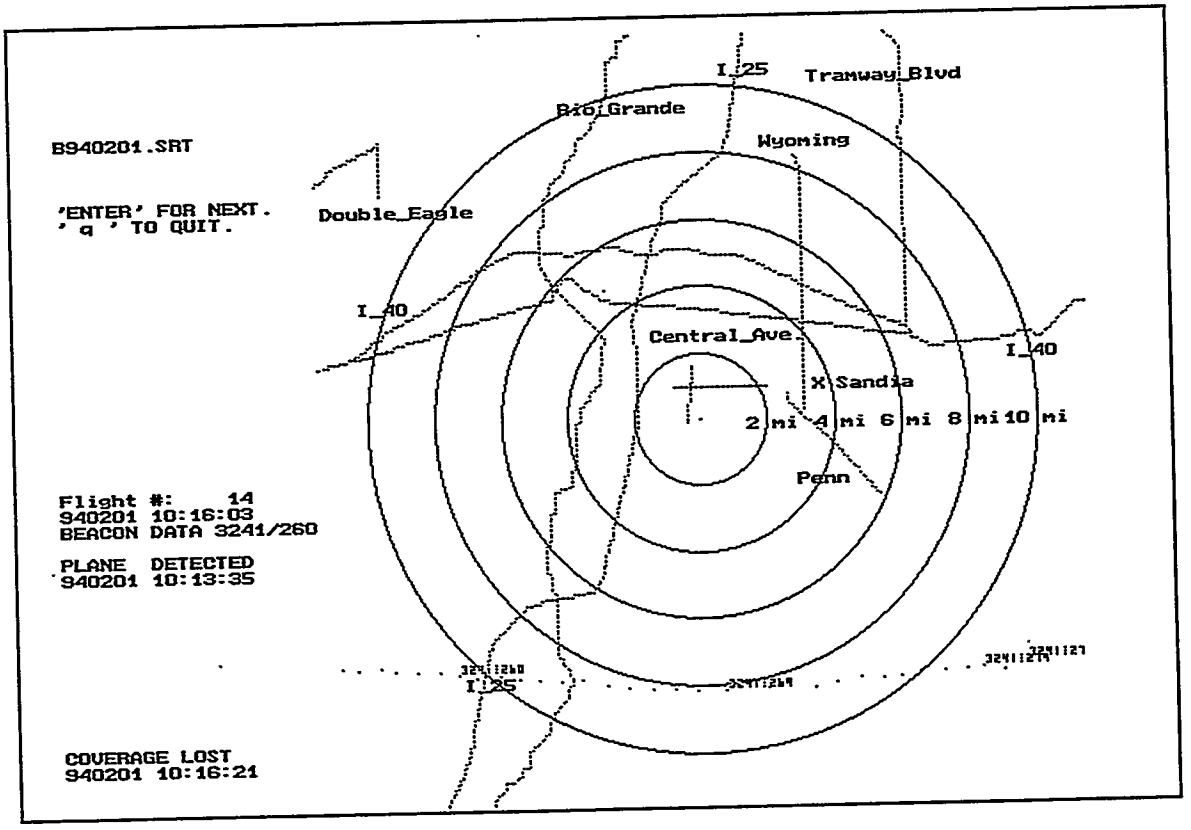


Figure 5-2. Sample Plot Using Display Mode.

This is a fly-over flight in Albuquerque on February 1st, 1994. The flight direction is from east to west. In Figure 5-2, the text in the left column represents the following information: B940201.SRT is the name of the radar file; 'Enter' for next will allow the user to view the next sequential flight in the file; 'q' will exit display mode; Flight #: 14 shows that this flight was the 14th flight of February 1st; 940201 10:16:03 represents the YYMMDD and the HH:MM:SS of the last plotted point of the flight; BEACON DATA 3241/260, 3241 is the squawk code or Beacon ID of the transponder and 260 is the

elevation times 100 (e.g. 26000 feet) of the latest plotted point; PLANE DETECTED 940201 10:13:35 is the YYMMDD and HH:MM:SS of the start of the flight; COVERAGE LOST 940201 10:16:21 is the YYMMDD and HH:MM:SS of the last point in the flight data. The trkplot function is used to plot the data points and the background display.

5.4.2. Plot Display Process

Figure 5-3 shows process flow and pseudo code for plot display mode. The function names are listed in ().

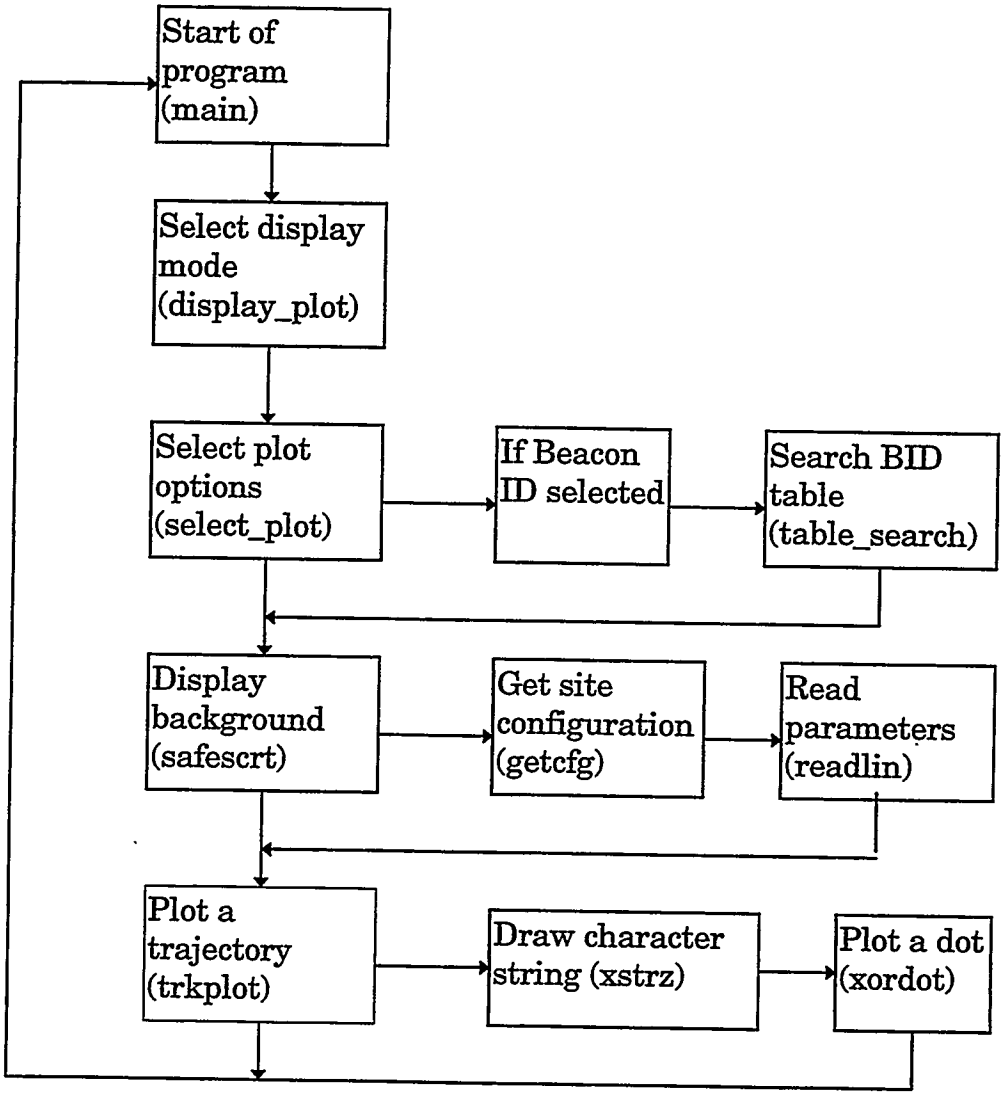


Figure 5-3. Display Mode Process Diagram

In Figure 5-3, the display_plot function is called from the main program. The select_plot function is called next. This option determines whether the user will select the flight based on Beacon ID, by a specific flight number, or sequentially. If the mode selected is Beacon ID, the BID array is searched for the entered BID. If a match is found, an offset is calculated for the radar file and the corresponding record is read from the radar file (.SRT). A pointer is positioned at the specific flight number and the points are plotted on

the screen. All Beacon numbers are plotted for the given flight. For example, if the user enters BID 4333, and if four flights are tagged with Beacon 4333 in the data file, then all four trajectories will plot on the screen sequentially. If the Beacon number is not found, the user is given another opportunity to enter a valid integer or quit.

If the mode selected is flight number or sequential, then the index file is positioned at the proper flight number. The record number, start, and end points of the flight is contained in the first three fields of the flight index file (.IDX). These fields are used as an index into the sorted radar file. The flight is plotted from the radar file beginning at the start point.

5.5. Data Input Mode

5.5.1. Overview

Data input mode permits users to attach air traffic control strips (see Figure 3-2) to electronic radar data. The green strips contain information (e.g., aircraft type and id) that is not captured or not available in electronic form. The aircraft type and related information is required for the current models used in aircraft crash analysis. Further information is contained in [Ref. 2].

Figure 5-4 shows the data input process. The functions are listed with (). The functions get_vfr_ifr and get_acft_id are bypassed if the switches are turned off in the ABQPTX.CFG file. This file is read upon startup; it is used to configure the data input environment. It was determined that the VFR/IFR and aircraft ID were not as important as the other variables listed on the green strips.

The process begins by asking the user a series of questions related to the flight strip. The time of flight, Beacon ID, and aircraft type are all captured. The entered Beacon ID is checked against the BID array to see if there is a match. If a match is not found, the entered data is saved to an unresolved file. If a match is found, all of the BIDS are plotted on the screen. At this point, the user is prompted to select which flight matches closely to the electronic data. At this time, this process is a visual comparison. If a selection is made, then the BID index file is updated with the green strip information. If the user chooses not to attach the flight strip to the data, the data is saved to an unresolved file.

5.5.2. Data Input Process

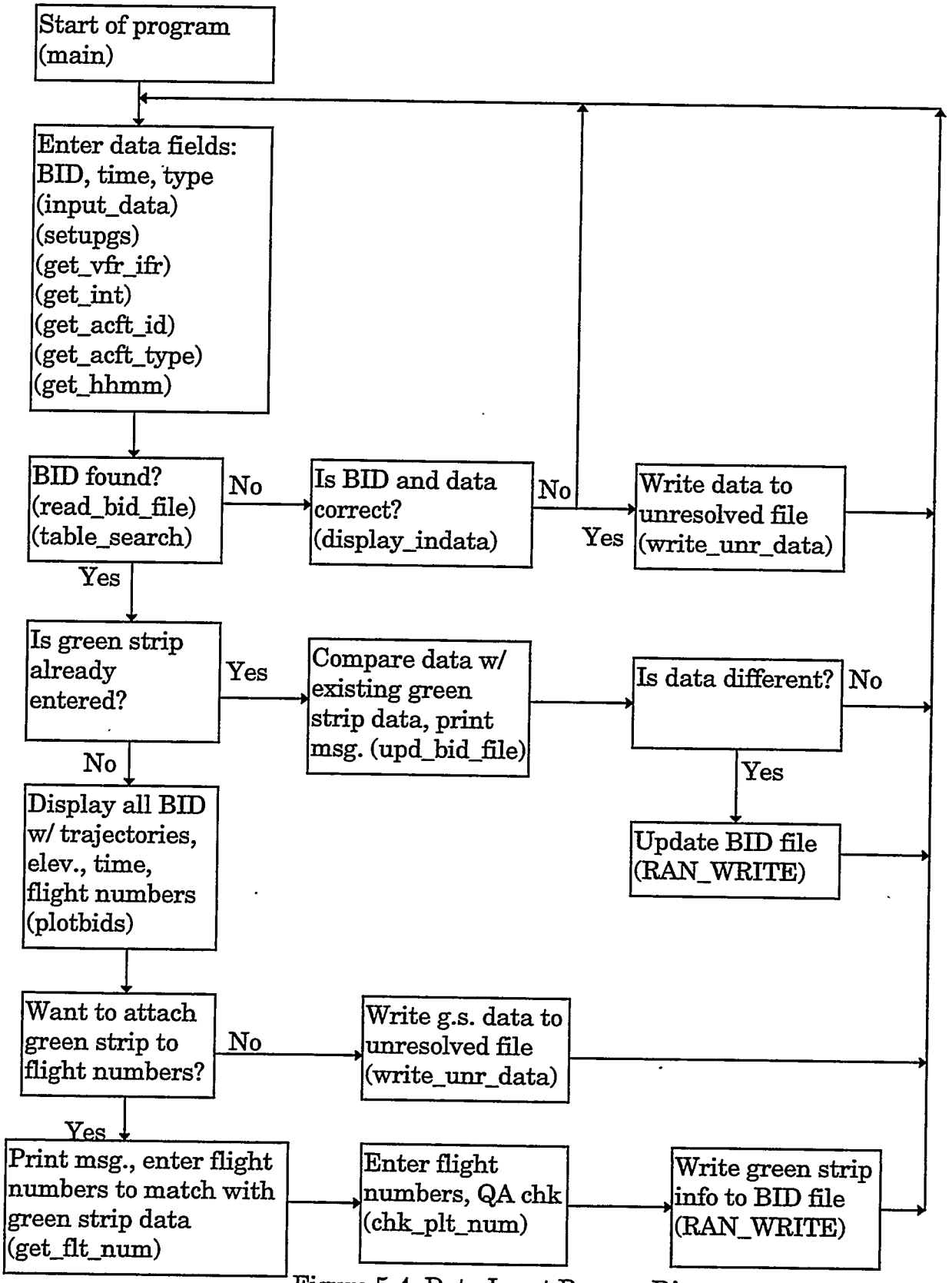


Figure 5-4. Data Input Process Diagram

5.6. Plotflt Include File

The plotflt program uses one include file named plotflt.h. This file contains the global variables, definitions, and function prototypes for the plotflt program. A listing follows:

```
/* plotflt.h - include file for plotflt.c */

/* include files */
#include <dos.h>
#include <graphics.h>
#include <math.h>
#include <stdio.h>
#include <bios.h>
#include <time.h>
#include <stdlib.h>
#include <io.h>
#include <string.h>
#include <conio.h>
#include <dir.h>
#include <ctype.h>
#include <malloc.h>
#include <sys\stat.h>

/* defines */
#define fpm 5280.                    /* feet per mile */
#define bkgd BLACK                   /* color for map background */
#define btc YELLOW                   /* color for beacon track */
#define rtc WHITE                    /* color for radar track */
#define btx btc^bkgd                 /* xor mode color to yield btc over bkgd. color */
#define rtx rtc^bkgd                 /* xor mode color to yield rtc over bkgd. color */
#define MAXFLT 2000                  /* max number of recorded flights per day */
#define TRUE 1
#define FALSE 0
#define ON 1
#define OFF 0
#define IDXBYTES 140L                /* length of .IDX file in bytes */
#define BIDBYTES 140L                /* length of .BID file in bytes */
#define RADBYTES 51L                 /* length of raw radar file in bytes */
#define MAXBYTES 256
#define STARS "**********"
#define BELL 7
#define EMPTY_STRING ""
#define MAXPLTNUM 100                /* max number of plots viewed on screen */
#define MAXENTRIES 30                /* max number of attached flight numbers */
#define BLANK ' '
#define PTX_RADAR_ELEV 3500L         /* Amarillo radar elevation in feet */
#define ABQ_RADAR_ELEV 5000L         /* Albuquerque radar elevation in feet */
```

```
/* plot speed codes in milliseconds */
#define DELAY_FAST 0L
#define DELAY_MED 50L
#define DELAY_SLOW 100L
#define SLEEP_FAST 0L
#define SLEEP_MED 1L
#define SLEEP_SLOW 2L

/* return error codes */
#define ZERO_LENGTH_STR -1
#define NO_ERROR 0
#define NO_RADAR_FILE 1
#define BID_NAME_ERROR 2
#define SCANFERR 3
```

5.7. Plotflt Function Descriptions

The plotflt program contains forty functions. They can be logically grouped into five categories. These are:

MAIN - functions called exclusively from the main program
PLOT DISPLAY - display mode only functions
DATA INPUT - data input functions
PLOT DISPLAY and DATA INPUT - functions contained in both plot and data input
GENERAL PURPOSE - functions that can be used in other programs

The functions are categorized below. The functions prototypes and a brief description of each routine is listed.

Main functions

```
void abqptxcfg(void);                   /* Read Albuquerque, Amarillo config file */
int open_radar_file(int numargs);       /* Open radar file */
void open_idx_files(char fmode[]);      /* Open idx file */
long pop_bid_arr(void);                 /* Populate the Beacon ID array */
int select_mode(void);                  /* Select plot mode */
void select_plot_opt(void);             /* Select plot options */
```

Plot display functions

```
void display_plot(void);                /* Main function for plot display mode */
int select_plot(void);                  /* Select plot options */
```

Data input functions

```
void input_data(void);                                  /* Main function for data input mode */
                                                        /* Check for valid plot number */
int chk_plt_num(int pltseq[], int intbuf[], int recarr[], int bidrecsave);
int display_indata(void);                               /* Display input data */
void get_acft_id(void);                                 /* Get aircraft id */
void get_acft_type(void);                               /* Get aircraft type */
void get_hhmm(void);                                    /* Get time */
int get_vfr_ifr(void);                                  /* Get vfr or ifr response */
int get_flt_num(int bidrec);                            /* Get flight number */
void plotbids(int bidrec);                              /* Plot by Beacon ID */
void read_bid_file(long bidloc);                        /* Read Beacon file */
int readinpfile(void);                                  /* Read input file */
int setupgs(void);                                      /* Setup green strip entry */
int upd_bid_file(int recarr[], int scanfcnt);           /* Update BID file */
void write_unr_data(void);                              /* Write data to unresolved file */
```

Plot display and data input functions

```
                                                        /* Adjust slant range of Abq data */
void abqadjust(double xf, double yf, double *newxf, double *newyf, long elev);
void getcfg(void);                                      /* Read global params from psmap.cfg */
                                                        /* Adjust slant range of Pantex data */
void ptxadjust(double xf, double yf, double *newxf, double *newyf, long elev);
void safescrt(void);                                    /* Prepare screen and get run parameters */
void trkplot (int fltnum);                              /* Plot a radar track */
void xstrz (char *strg, int x, int y, int color);       /* Low level plot routine */
void xordot (int xx, int xy, int xcol);                 /* Low level plot routine */
```

General purpose functions

```
                                                        /* Update a string */
int STRUPD(char source[], char replace[], int start);
                                                        /* Direct access read */
int RAN_READ(FILE *stream, long recnum, long bytes, char string[]);
                                                        /* Direct access write */
int RAN_WRITE(FILE *stream, long recnum, long bytes, char string[]);
int readlin (char buf[], FILE *strm);                   /* Read a line into a buffer */
long filesize (FILE *stream);                           /* Get file size in bytes */
int table_search(int a[], int n, int target);           /* Search BID array */
int get_int(char *prompt, int min, int max);            /* Get integer */
int get_reply(char *prompt);                            /* General purpose reply function */
                                                        /* Extract middle of string */
int mid_extract(char *src, char *dest, int start, int num_chars);
int char_count(char *string, char letter);              /* Character counter */
void squeeze(char s[], int c);                          /* Squeeze c from string s */
```

5.7.1. Main Function Headers

```
/*FF*************************************************************************
Function name: abqptxcfg

Purpose:
Get the Albuquerque and Amarillo site configuration. This routine opens up the
ABQPTX.CFG file and checks if the global flags for VFR/IFR and ACFT ID are set - if set
to 1 the question(s) the user is prompted for a response if set to 0, the questions are
bypassed.

The global variables G_vfrifr, G_acid, and G_miles are set in this routine. G_miles is an
integer which is used to eliminate flights outside this radius. This assumes that this
option is enabled in SORTFLT.FOR.

Called from: main

Calls routines: None

Sample call: abqptxcfg();

Return value: None

Messages returned:
  If the ABQPTX.CFG cannot be opened, the following message will appear:
    Cannot open ABQPTX.CFG file.

*************************************************************************/

/*FF*************************************************************************
Function name: open_radar_file

Purpose:
Make sure we have a valid Beacon input file. The file name must start with a "b" or a "B"
and the file extension should be .SRT

Called from: main

Calls routines: None

Sample call: rtnerr = open_radar_file(numargs);

Return value: integer representing one of the following conditions:
  NO_RADAR_FILE    -> no radar file present or invalid extension
  BID_NAME_ERROR -> file cannot be opened; 1st character must be B
  NO_ERROR             -> successful open
```

Messages returned:
If an invalid extension is used for a filename (other than .SRT) the following message will appear:
Please use .SRT extension for filename.
Enter name of the radar file for plotting (e.g., B940508N.SRT):

If the input file name does not exist or cannot be opened, the next message will appear on the screen:
FILE NAME = B940521.SRT
CANNOT OPEN FILE.

If the first character in the filename is not a b or a B, then next message will appear:
File name W940521.SRT must be a Beacon file.
The first character in the filename must be b or B.

**********/

/*FF**********
Function name: open_idx_files

Purpose:
Open the .IDX and .BID files. The filenames are constructed from the G_trkname filename.

Called from: main

Calls routines: None

Sample call: open_idx_files("rb");

Return value: None

Messages returned:
If an error occurs opening the flight index file, the following message will appear:
Inside open_idx_files module.
Error opening flight index file: B950521.IDX

If an error occurs opening the BID file, the following message will appear:
Inside open_idx_files module.
Error opening BID index file: B950521.BID

**********/

/*FF**********

Function name: pop_bid_arr

Purpose:
Populates the global Beacon ID array, G_bid. The .BID file is read sequentially and the 5th field is saved into the array. The .BID file is assumed sorted. This function also calculates the byte count of the .BID file

Called from: main

Calls routines: filesize

Definitions: MAXBYTES

Sample call: bytecnt = pop_bid_arr();

Return value: Long value which is the byte count of the BID file.

Messages returned:
Inside pop_bid_arr module.
Error opening sorted index file: B950521.BID

**********/

/*FF**********

Function name: select_mode

Purpose:
Select plot review, composite, or data input mode.

Called from: main

Calls routines: None

Sample call: choice = select_mode();

Return value: Integer value representing the following values:
0 -> Quit program
1 -> Plot Review mode
2 -> Plot Composite mode
3 -> Data Input mode

Messages returned: None

**********/

/*FF**********

Function name: select_plot_opt

Purpose:
Routine which allows user to change plot parameters and set up plot defaults for text, streets, and radii. If erase mode is turned on, the plot will erase the previous flight before it plots the next one. If erase mode is off, all flights will plot on the screen one at time.

Called from: main

Calls routines: None

Sample call: select_plot_opt();

Return value: None

Messages returned: None

**********/

5.7.2. Plot Display Function Headers

/*FF**********

Function name: display_plot

Purpose:
Allow the user to select a flight number(s) or BID to plot; select_plot is called first. If a flight number is input, the start point of the flight is computed for the index file (.IDX). The start point is used to fseek into the index file at the correct position. The index file is read and the record number for the specific flight (flight number, start and end pts.) are read into memory. These values are used to locate the proper flight to plot from the radar file. If a BID plot is chosen, then table_search is called to make sure the BID number is found. If found, then the start point is computed and the BID file (.BID) is used to locate the flight. A replay option is available after the points are plotted. The trkplot function is called to plot the points.

Called from: main

Calls routines:

select_plot	trkplot
table_search	safescrt

Sample call: display_plot();

Return value: None

Messages returned: None

**********/

/*FF**********
Function name: select_plot

Purpose:
Select the flight numbers or Beacon ID for plotting.

Called from: display_plot

Calls routines: None

Sample call: choice = select_plot();

Return value: Integer value representing the following:
0 -> Quit program
1 -> Plot all flights sequentially
2 -> Plot a specific flight
3 -> Plot a range of flights
4 -> Plot by Beacon ID

Messages returned: None

**********/

5.7.3. Data Input Function Headers

/*FF**********
Function name: input_data

Purpose:
This is the main routine for green strip input. The 1st routine called is setupgs which captures the initials of the user. A welcome message is printed and the following routines are called in sequence:
get_vfr_ifr - determine VFR or IFR flight
get_int - get Beacon ID

A check is made to see if the flight is within the radius specified in the abqptx.cfg. This global variable is named G_miles. Flights outside this range are ignored. This value is currently set to 0, which means no flights are ignored.

Next the aircraft id, aircraft type, and the time is captured from the green strip. These routines are named:

get_acft_id - get aircraft id
get_acft_type - get aircraft type
get_hhmm - get hours and minutes of flight

The table_search function is called next. If the Beacon number is found a plot is initiated (plotbids). If not found, the user has the option of changing the green strip information or saving the information to the unresolved file. In this case, the unresolved path is set to 1.

Batch mode is available but should not be used because it has been superseded with a electronic regional strip match utility. This new program negates the use of batch mode.

Called from: main

Calls routines:

setupgs	readlin
readinpfile	table_search
get_vfr_ifr	get_int
read_bid_file	RAN_READ
mid_extract	get_flt_num
get_acft_id	get_acft_type
get_hhmm	plotbids
write_unr_data	display_indata

Sample call: input_data();

Return value: None

Messages returned:

If the Beacon ID is not found in the file, the following message will appear:

BEACON ID 3271 NOT FOUND. Press return:

If the maximum number of flights that can be displayed on the screen is exceeded, the following message will appear:

A total of 120 BIDs found. This exceeds the maximum number of flights for plotting. No more than 100 flights can be viewed at one time.

**********/

/*FF**********

Function name: chk_plt_num

Purpose:
This routine checks for valid flight numbers. Positive numbers are allowed. Plot number 0 is illegal. The entries must match the displayed flight numbers. The number of entries is returned..

Called from: get_flt_num

Calls routines: char_count

Sample call: scanfcnt = chk_plt_num(pltseq, intbuf, recarr, bidrecsave);
Return value: Integer value which is number of entries read.

Messages returned:
If the number of entries is not between 1 and 30, the next message will appear:
You must enter between 1 and 30 digits.

If an invalid digit is typed in, the following message will appear:
Error reading input: enter digits only.

If the number of entries exceed the BID count, the next message will appear:
You entered more plot numbers than expected. Try again.

If negative numbers are entered, the following message will appear:
Enter positive numbers only. Please try again.

If plot number 0 is entered, the next message will appear:
You cannot enter plot number 0. Try again.

If the entered number does not match the flight numbers, the next message will appear:
Your selection(s) do not match the plot numbers. Try again.

**********/

/*FF**********

Function name: display_indata

Purpose:
Prints a confirmation screen after green strip information is entered; allows user to change all the data or none.

Called from: input_data

Calls routines: getreply

Sample call: rtnval = display_indata();

Return value: Integer value
0 -> No
1 -> Yes

Messages returned: None

**********/

/*FF**********
Function name: get_acft_id

Purpose:
Prompt for the aircraft ID (e.g., N321FM). This prompt can be bypassed by setting the aircraft id variable to 0 in the ABQPTX.CFG file. Enter 12 characters maximum.

Called from: input_data

Calls routines: None

Sample call: get_acft_id();

Return value: None

Messages returned:
If the aircraft ID is left blank, the following message will appear:
You must enter the aircraft ID. Please re-enter.

If the length of the aircraft ID exceeds 12 characters, the following message will appear:
You have exceeded 12 characters. Please re-enter.

**********/

/*FF**********
Function name: get_acft_type

Purpose:
Get the aircraft type (e.g. T/B73S/A). Enter 12 characters max.

Called from: input_data

Calls routines: None

Sample call: get_acft_type();

Return value: None

Messages returned:
If the aircraft type is left blank, the following message will appear:
You must enter the aircraft type. Please re-enter.

If the length of the aircraft type exceeds 12 characters, the following message will appear:
You have exceeded 12 characters. Please re-enter.

**********/

/*FF**********
Function name: get_hhmm

Purpose:
Get the hours and minutes from the green strip. Time must be in HHMM format. A minimum of four digits are required. If a leading prefix is used it must be "A" for arrival; "P" for departure; or "E" for fly over. HH must be in the range 00->23; MM must be in the range 00->59.

Called from: input_data

Calls routines: None

Sample call: get_hhmm();

Return value: None

Messages returned:
If the flight time is left blank, the following message will appear:
You must enter the aircraft flight time. Please re-enter.

If the time is less than 4 digits, the following message will appear:
Time too short; Use 4 digits minimum. Please re-enter.

If more than 5 digits are entered, the next message will appear:
Time too long; Use 5 characters max. Please re-enter.

If an invalid prefix is entered, the next message will appear:
Prefix must be A, E, or P. Please reenter.

If a negative number is entered, the next message will appear:
Please use valid positive digits for time. Please re-enter.

If the hour is not between 00 and 23, the next message will appear:
Hour is invalid: Use 00 to 23 for hour.

If the minute is not between 00 and 59, the next message will appear:
Minute is invalid: Use 00 to 59 for minute.

**********/

/*FF**********
Function name: get_vfr_ifr

Purpose:
Determine if this is a VFR, IFR, or unknown flight strip.

Called from: input_data

Calls routines: None

Sample call: rtnval = get_vfr_ifr();

Return value: Integer value 0, 1, 2, or 3
0 -> Quit data entry
1 -> IFR
2 -> VFR
3 -> UNKNOWN

Messages returned: None

**********/

/*FF**********
Function name: get_flt_num

Purpose:
The is a text mode version of plotbids without the graphics. The user is given options for attaching the green strips to the electronic flight data. The options are:

1. Quit
2. Attach green strip to flight numbers - the flight
   numbers are check and the Beacon file is updated
   chk_plt_num and upd_bid_file are called
3. Write green strip info to unresolved file

the unresolved file is updated with path=2
write_unr_data is called
4. Replay the plot

Like the plotbids function, if the BID changes during flight or if the flight is already attached the flight will display in yellow.

Called from: input_data

Calls routines:

RAN_READ	mid_extract
chk_plt_num	upd_bid_file
write_unr_data	

Sample call: rtnval = get_flt_num(found);

Return value: Integer
0 -> Quit, restart green strip input
1 -> Attach green strips to plot numbers
2 -> Write green strip information to unresolved file
3 -> Replay plot

Messages returned: None

**********/

/*FF**********
Function name: plotbids

Purpose:
Plot a flight track from the Beacon radar file. This routine Calls routines safescrt which toggles the text screen to graphics mode and sets up the screen colors. The scale factor for the plot is calculated from the global variable
G_maxrg. This variable is site specific.

A nested loop is started to plot the actual track. The G_bidcnt variable contains the total number of Beacon tracks found. This sets the limit on the number of flights plotted. The Beacon index file for the Beacon ID specified and the flight number, start and end times, start and end elevations are printed on the plot. If the flight is already attached to a green strip, the color is displayed as yellow for that flight. If the fltind variable is 2 or greater, then a * will be plotted next to the flight. This shows that the Beacon ID changed during flight or it may have changed multiple times.

The inner loop plots a data point from the radar file. The px and py are calculated in pixels and plotted. Adjustments are made to the x and y coordinates to account for the

slant range of the track. The abqadjust and ptxadjust routines convert slant range to projected ground coordinates.

Called from: input_data

Calls routines:
safescrt RAN_READ
ptxadjust abqadjust
mid_extract

Sample call: plotbids(bidrec);

Return value: None

Messages returned: None

**********/

/*FF**********
Function name: readinpfile

Purpose:
This routine is used exclusively for batch mode only. The response file is read and the global variables for Beacon ID, aircraft ID, aircraft type, and time are populated. Five fields must be present or an error message will print out.

Called from: input_data

Calls routines: readlin

Sample call: rtnval = readinpfile();

Return value: Integer value
SCANFERR -> invalid field count
1 -> success

Messages returned:
If an invalid field is found in the response file, the following message will appear:
Illegal record found in batch file B950521.RES

**********/

```
/*FF*************************************************************************
Function name: setupgs

Purpose:
Setup routine for green strip input; capture the initials for the data entry person. The
initials are used to update the BID index file. This routine is called only once upon start
up.

The bulk of this routine is programmed for batch mode which is obsolete. Batch mode will
use a response file which contains data like aircraft type and time. The sequence file is
used as an integer counter to keep track of which records in the response file have been
plotted. This permits Called from to stop and start the program gracefully.

Called from: input_data

Calls routines: None

Sample call: rtnval = setupgs();

Return value: Integer value
 1 -> Data from keyboard
 2 -> Data from batch file

Messages returned:
 If the response file (.RES) cannot be opened, the following message
 will appear:
  Inside setupgs module.
  Error opening input file B950521.RES

 If there is an error creating the sequence file, the following
 message will appear:
  Inside setupgs module.
  Error creating sequence file B950521.SEQ

 If there is an error opening the sequence file, the following
 message will appear:
  Inside setupgs module.
  Error opening sequence file B950521.SEQ

*************************************************************************/
```

```
/*FF*************************************************************************
Function name: read_bid_file

Purpose:
Read a line from the BID file; input the location and update the BID count for the given
flight.

Called from: input_data

Calls routines:
 RAN_READ          mid_extract

Sample call: read_bid_file(loc);

Return value: None

Messages returned: None
*************************************************************************/
/*FF*************************************************************************
Function name: upd_bid_file

Purpose:
Update the Beacon ID file with the green strip green strip information. The existing
information is read from the BID file. If the record has not been updated (matchind = 0),
then update the VFR/IFR, aircraft id, aircraft type, and time for the current record. If the
record has been updated already then compare the aircraft id, type, and time from the old
to the new. If there are differences then allow the user to save the old stuff or overwrite it
with the new information. If the user decides not to overwrite, then an option is available
to save the green strip information to the unresolved file with path=3.

Called from: get_flt_num

Calls routines:
 RAN_READ          mid_extract
 squeeze           STRUPD
 RAN_WRITE         write_unr_data

Sample call: rtnval = upd_bid_file(recarr, scanfcnt);

Return value: Integer value
 0 -> success
 1 -> Replay plot and re-enter flight numbers

Messages returned: None
```

```
**********************************************************************/

/*FF*****************************************************************
Function name: write_unr_data

Purpose:
Writes green strip information to unresolved file.

Called from:
 input_data          get_flt_num
 upd_bid_file

Calls routines: None

Sample call: write_unr_data();

Return value: None

Messages returned:
 If the unresolved file cannot be opened, the following message
 will appear:
  Error opening unresolved file : B950521.UNR

**********************************************************************/
```

5.7.4. Plot Display and Data Input Function Headers

```
/*FF*****************************************************************
Function name: abqadjust

Purpose:
Plot adjust routine for Albuquerque data; convert the x and y slant range back to projected
ground coordinates. No offset is necessary because the radar antenna is the 0,0 reference
point for all the raw data.

Called from:
 trkplot          plotbids

Calls routines: None

Sample call: abqadjust(xf, yf, &newxf, &newyf, elev);

Return value: None

Messages returned: None
```

**********/

/*FF**********

Function name: getcfg

Purpose:
Written by David Skogmo 4/5/91
adapted to read both Pantex and Albuquerque PSMAP.CFG files: J. Tenney 2/14/95

getcfg reads the file PSMAP.CFG and loads the site specific data into the SARS program global variables. The first five lines of the PSMAP.CFG file should look as shown below. It is essential that the items be in exactly this order and that all data entries be separated from any annotation by whitespace. All lines should terminate in \n. This is certainly true also of the last line.

```
0.00     ;xd=east/west coordinate in statute miles from radar
0.00     ;yd=north/south coordinate in statute miles from radar
196.8    ;rcell=beacon range cell size in feet
196.8    ;rrcell=primary range cell size in feet
11.52    ;G_maxrg=coverage radius in miles
```

Called from: safescrt

Calls routines: None

Sample call: getcfg();

Return value: None

Messages returned:
If the PSMAP.CFG file does not exist, the next message will appear:
Trouble with PSMAP.CFG file. Fatal error. Press any key.

**********/

/*FF**********

Function name: ptxadjust

Purpose:
Plot adjust routine for Pantex site. This routine converts the recorded slant range back to ground coordinates. The offset of x=38280 feet, y=38174 feet is used because these offsets are embedded in the recorded data. The radar antenna location is different from the Zone 4 recording site. The raw data is tracked from the Zone 4 coordinate. Therefore the offset must be applied to the data in order to calculate the correct position of the flight projection to the surface.

Called from: trkplot plotbids

Calls routines: None

Sample call: ptxadjust(xf,yf,&newxf,&newyf,elev);

Return value: None

Messages returned: None

**********/

/*FF**********
Function name: safescrt

Purpose:
Written by David Skogmo 3/20/91; adapted from autoscrn for zone4 program

safescrt prepares screen for RAMS display. It reads the site specific parameters from a file called PSMAP.CFG. It displays a map and its associated text file. The map file is expected to be a list of points (one point per line). Each line gives the x and y coordinates and the color. These numbers are separated by spaces. To make a black dot at 230,335; enter 230 335 0 etc. The text file should give the x and y coordinates then the color. This is followed by the text string to be placed at x,y. Since we use the function sscanf to read this string, it should contain no spaces. If you need a space, use the underline _ char. The map file should be named PSMAP.MAP and the text file should be named PSMAP.TXT.

Called from: main display_plot plotbids

Calls routines: getcfg

Definitions: bkgd

Sample call: safescrt();

Return value: None

Messages returned:
If the PSMAP.MAP file does not exist in the local directory, the following message will appear on the screen:
Cannot open PSMAP.MAP file. Press any key.

If the PSMAP.TXT file does not exist in the local directory, the following message will appear on the screen:
Cannot open PAMAP.TXT file. Press any key.
**********/

/*FF**********

Function name: trkplot

Purpose:
Plots a flight of data points one point at a time. The data is read from the Beacon radar file, G_trkfile. The Borland graphics library routines are used throughout this routine. Composite mode is supported also; this mode plots every data point in the file while review mode plots an individual track.

Called from:
main display_plot

Calls routines:
ptxadjust abqadjust

Sample call: trkplot(1);

Return value: None

Messages returned: None

**********/

/*FF**********

Function name: xstrz

Purpose:
xstr draws the character string at point x,y in the passed color in xor mode. The point x,y is taken as the bottom left corner of the string space. xstrz written by David Skogmo 11/26/93

Called from: trkplot

Calls routines: xordot

Sample call: xstrz(bufid, px+2, py-2, WHITE);

Return value: None

Messages returned: None

**********/

```
*FF**********************************************************************
Function name: xordot

Purpose:
xordot: plots a dot in xor mode at xx, xy

Called from: xstrz

Calls routines:
  getpixel          putpixel

Sample call: xordot(xdot,ydot,color);

Return value: None

Messages returned: None

**********************************************************************/
```

5.7.5. General Purpose Function Headers

```
/*FF**********************************************************************
Function name: STRUPD

Purpose:
Replace the source string with the replacement string.  Length of replacement string must
not exceed source string.

Called from: upd_bid_file

Calls routines: None

Sample call: STRUPD(string, vfrifr, 77);

Return value: Integer value
 0 -> success
-1 -> failure

Messages returned:
 If the source string is zero length, the next message will appear:
   Source string is zero length.

 If the replacement string is zero length, the next message will appear:
   Replacement string is zero length.
```

If the replacement string is longer than the source string, the
next message will appear:
Replacement string is longer than source string.

If the starting position is greater than the source string, the
next message will appear:
Starting location is greater than length of source string.

**********/

/*FF**********
Function name: RAN_READ

Purpose:
Do a random read of selected file; calculate the offset into the file; the file is assumed fixed length.

Called from:

read_bid_file	input_data
plotbids	get_flt_num
upd_bid_file	

Calls routines:

Sample call: RAN_READ(G_bidfile, bidloc, BIDBYTES, string);

Return value: Integer which contains number of characters read

Messages returned: None

**********/

/*FF**********
Function name: RAN_WRITE

Purpose:
Do a random write into the selected file; the file is
assumed fixed length.

Called from: upd_bid_file

Calls routines: None

Sample call: RAN_WRITE(G_bidfile, recarr[i], BIDBYTES, string);

Return value: Integer value which is numbers of characters written

Messages returned: None
**********/

/*FF**********
Function name: readlin

Purpose:
readlin reads characters from the passed stream into the passed buffer until the \n char is encountered. It terminates the buffer with a 0. It returns the number of chars read. If the end of file is encountered, it returns 0. Written by David Skogmo 4/5/91

Called from:

trkplot	safescrt
getcfg	display_plot
pop_bid_arr	input_data
plotbids	setupgs
readinpfile	

Calls routines: None

Sample call: i = readlin (string, G_bidfile);

Return value: Integer value: number of bytes read in

Messages returned: None
**********/

/*FF**********
Function name: filesize

Purpose:
This routine computes the file size in bytes - copied from the Borland library reference manual.

Called from: pop_bid_arr

Calls routines: None

Sample call: bytecnt = filesize(G_bidfile);

Return value: long value representing the byte count of the input file.

Messages returned: None

**********/

/*FF**
Function name: table_search

Purpose:
Do a linear search for the integer in the BID array.

Called from:
display_plot input_data

Sample call: loc = table_search(G_bid, G_maxflt, G_bid_choice);

Return value: Integer value: location of match in integer array
-1 is returned if no match is found

Messages returned: None

**/

/*FF**
Function name: get_int

Purpose:
Checks the input string for a valid integer value within min/max range.

Called from: input_data

Calls routines: None

Sample call: choice = get_int("Enter integer: ", 0, 4)

Return value: Integer value returned between min and max range

Messages returned: None

**/

/*FF**
Function name: getreply

Purpose:
Get a Yes or No response.

Called from: display_indata

Calls routines: None

Sample call: response = getreply(prompt);

Return value: Integer value
1 for Yes
0 for No

Messages returned: None

**********/

/*FF**********
Function name: mid_extract

Purpose:
Returns middle portion of string; like Quick BASIC's MID$ function

Called from:
read_bid_file input_data
plotbids get_flt_num
upd_bid_file

Calls routines: None

Sample call: mid_extract(instring, outstring, start_loc, num_of_chars)
num_of chars should start at 0 for counting

Return value: Integer -> number of characters copied or
ZERO_LENGTH_STR if input string is null

Messages returned: None
**********/

/*FF**********
Function name: char_count

Purpose: Count the number of letters in a string and return the count.

Called from: chk_plt_num

Sample call: spacecnt = char_count(dest, BLANK);

Return value: Integer value returns number of occurrences of match.

Messages returned: None

**********/

```
/*FF*************************************************************************
Function name: squeeze

Purpose: Delete all character c from string s

Called from: upd_bid_file

Sample call: squeeze("This is a test", BLANK);

Return value: None

Messages returned: None

*************************************************************************/
```

5.8. Pantex Plot Projection

One of the purposes of the plotflt program is to provide a visual representation of the ground projections of aircraft for areas of interest. Three data values are provided to this code in the radar input file (e.g., B950211.DAT). These reported data represent the x and y components of the translated slant range and the altimeter reading from the aircraft. To plot the actual ground projections, the aircraft altitude must be accounted for. This requires several steps. First, the translated components must be translated back into the radar coordinate system. This is done to insure stability in the solutions to the equations when solving for the true ground projections. Figure 5-5 shows the reported datum in the X, Y rectangular coordinate system (xloc, yloc). This coordinate system for the Pantex site is located 7.23 miles north and 7.25 miles east of the actual radar system (these values are contained in the PSMAP.CFG configuration file). The datum is first translated to the X', Y' coordinate system.

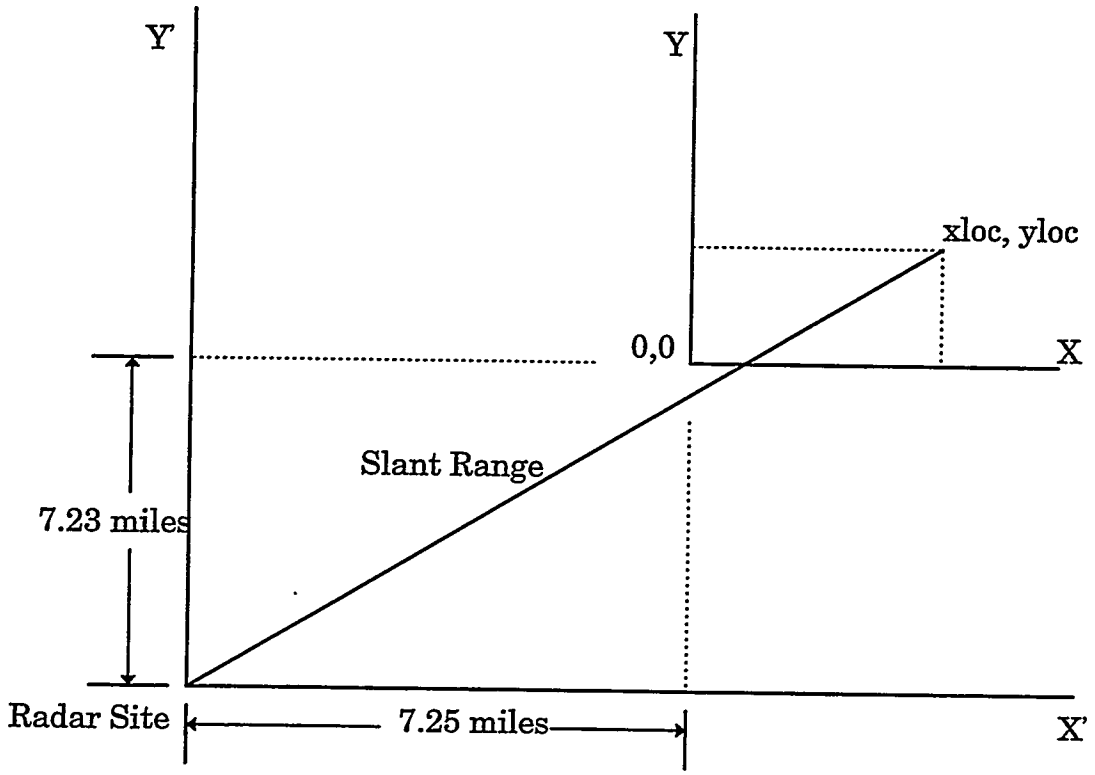


Figure 5-5. Radar Source and Recording Coordinate Systems

Once the datum is translated into the radar coordinate system, the following equations are solved to calculate the actual ground projections before plotting.

x' loc = xloc - (-38280 feet)

$y'\ loc = yloc - (-38174\ feet)$

$$\text{Slant Range} = \sqrt{x'\ loc^2 + y'\ loc^2}$$

The airport elevation of 3500 feet must be subtracted from the altitude to get the actual elevation from the ground to the aircraft. (h = h - 3500). Figure 5-6 shows the three dimensional coordinate system and the projected values.

The curvature of the earth is insignificant for the recorded distance; therefore it is ignored in the calculation.

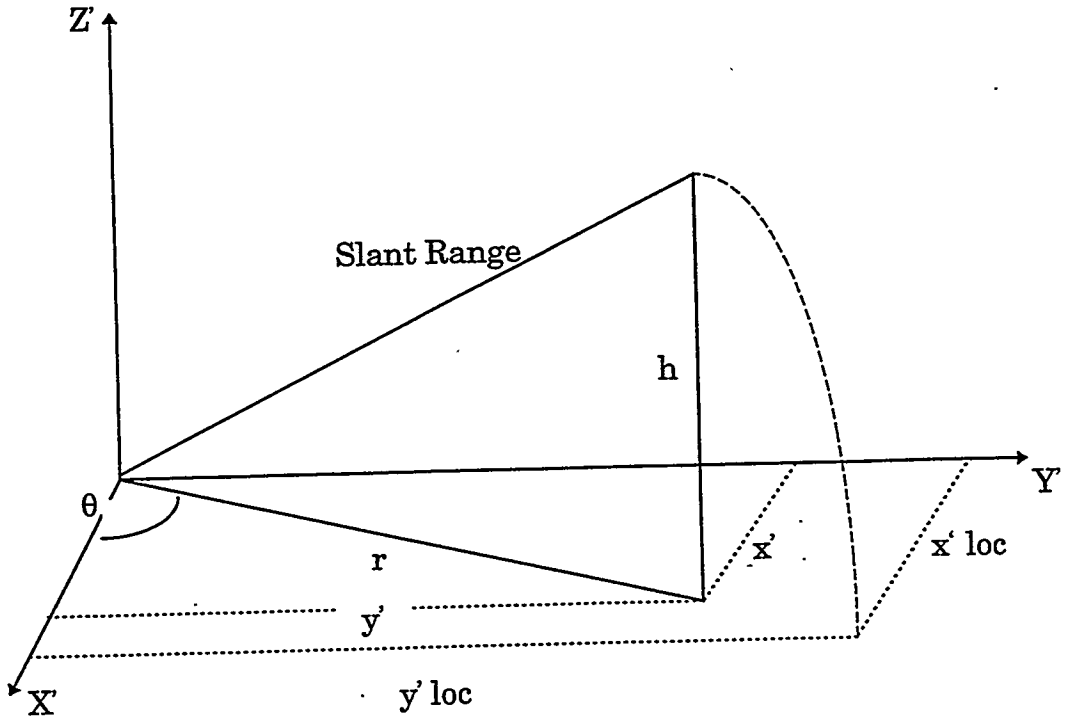


Figure 5-6. Pantex Slant Range Diagram

The X and Y coordinates represent the slant range vectors of the flight. x' and y' represent the actual ground projections of the aircraft. θ is the azimuth angle; h is the altimeter reading to sea level; r is the projected ground distance.

Solving for r, θ, x', and y':

$$\text{Slant Range} = \sqrt{x'loc^2 + y'loc^2} \quad = \quad \sqrt{r^2 + h^2}$$

Solving for r:

$$r = \sqrt{x'loc^2 + y'loc^2 - h^2}$$

Solving for θ:

$$\tan \theta = y' loc / x' loc$$

$$\theta = \tan^{-1} (y' loc / x' loc)$$

Solving for y':

$$y' = r \sin \theta$$

Solving for x':

$$x' = r \cos \theta$$

To obtain the true projection, the equations x - (-38280) and y - (-38174) are used. These reduce to x + 38280 and y + 38174. Next, the r value, azimuth angle, projected x and projected y values are computed and plotted.

The corresponding Fortran code to solve these equations is listed:

```
h = elev - 3500
if (h .lt. 0) h = 0
r1 = (x+38280.)**2 + (y+38174.)**2 - h**2
r = sqrt(r1)
angle = atan2 (y+38174., x+38280.)
x' = r * cos(angle) - 38280.
y' = r * sin(angle) - 38174.
```

5.9. VGA Pixel Coordinate System

The safescrt function displays the streets, labels, and radii around the local airport. The VGA screen is divided into pixels (640 in x, and 480 in y). At Pantex, two targets were selected (Zone 4 and Zone 12) and plotted on the screen in pixels. The calculations for these points are shown below. The data is recorded from coordinate 38280 east (7.25mi), 38174 north (7.23mi)(400x, 240y in pixels). This is the center of the screen in Figure 3-1.

The distance from the radar source at Pantex (0,0 coordinate) to the two targets [Ref. 3] is shown in Figure 5-5. Distance from radar source to Zone 4 is 37187 feet north (7.04 miles), 41915 feet east (7.94 miles); distance from radar source to Zone 12 is 29103 feet north (5.51 miles), 44912 feet east (8.51miles).

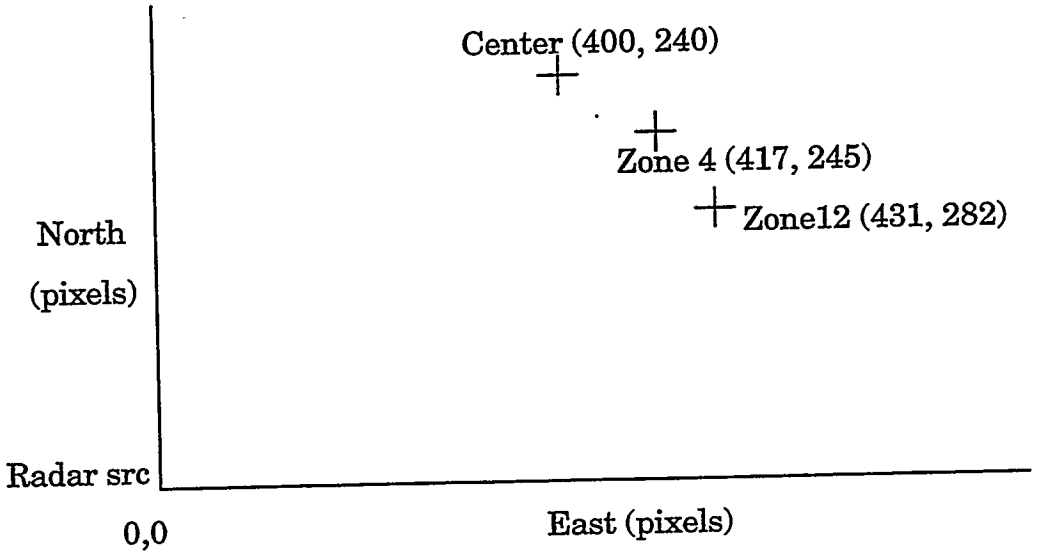


Figure 5-7. Pantex Pixel Coordinates

The coverage radius for the Amarillo radar data is 9.82 statute miles. This value is obtained from the PSMAP.CFG configuration file. The scale factor for the data can be calculated as follows:

scale factor = 240./max coverage radius

scale factor = 240./9.82 = 24.44 pixels/mile

Zone 4 distance from source is 7.94 mi. east, 7.04 mi. north

Offset from center point is:

7.94 - 7.25 = .69 x 24.44 pixels = 17 pixels east

7.23 - 7.04 = .19 x 24.44 pixels = 5 pixels south

Zone 4 coordinate is 400 + 17 east = 417 east (pixels)

Zone 4 coordinate is 240 + 5 south = 245 north (pixels)

Zone 12 distance from source is 8.51 mi east, 5.51 mi. north

Offset from center point is:

8.51 - 7.25 = 1.26 x 24.44 pixels = 31 pixels east

7.23 - 5.51 = 1.72 x 24.44 pixels = 42 pixels south

Zone 12 coordinate is 400 + 31 east = 431 east (pixels)

Zone 12 coordinate is 240 + 42 south = 282 north (pixels)

The target coordinates (417,245 and 431,282) are plotted in the safescrt function if the Pantex data is plotted.

The Albuquerque data is centered around the radar source at 0,0; therefore an offset calculation is not required. However, since the Albuquerque data is given in slant range coordinates, it is also projected to ground level coordinates. The abqadjust routine adjusts the data before the points are displayed.

Intentionally Left Blank

6. References

1. David Skogmo, Sandia National Laboratories, *Radar Airspace Monitoring System*, November, 1991.

2. John Tenney, Sandia National Laboratories, *Plot-Flight User's Manual Ver 1.0*, SAND95-1819, August, 1995

3. Tetra-Tech, *Distance from Radar Source to the Zone 4 and Zone 12 Points*, Internal Memorandum, March, 1995

Intentionally Left Blank

Distribution:

1	MS0405	M. P. Bohn, 12333
1	MS0405	D. D. Carlson, 12333
1	MS0405	T. R. Jones, 12333
1	MS0491	M. A. Dvorack, 12333
1	MS0491	N. R. Grandjean, 12333
1	MS0491	S. A. Kalumba, 12333
1	MS0491	M. C. Krawczyk, 12333
1	MS0491	Y. T. Lin, 12333
1	MS0491	R. J. Roginski, 12333
1	MS0491	R. E. Smith, 12302
10	MS0491	J. L. Tenney, 12333
1	MS9018	Central Technical Files, 8523-2
5	MS0899	Technical Library, 4414
1	MS0619	Print Media, 12615
2	MS0100	Document Processing, 7613-2 for DOE/OSTI