DATA LINK TEST AND ANALYSIS SYSTEM/ ATCRBS TRANSPONDER TEST SYSTEM

Technical Reference

John Van Dongen

May 1990

DOT/FAA/CT-TN90/7

Technical Report Documentation Page

1. Report No. DOT/FAA/CT-TN90/7	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle DATA LINK TEST AND ANALYSIS SYSTEM/ATCRBS TRANSPONDER TEST SYSTEM TECHNICAL REFERENCE		5. Report Date May 1990 6. Performing Organization Code ACD-320
7. Author(s) John Van Dongen		8. Performing Organization Report No. DOT/FAA/CT-TN90/7
9. Performing Organization Name and Address Federal Aviation Administration Technical Center Atlantic City International Airport, N.J. 08405		10. Work Unit No. (TRAIS) 11. Contract or Grant No. T2001F
12. Sponsoring Agency Name and Address U.S. Department of Transportation Federal Aviation Administration Technical Center Atlantic City International Airport, NJ 08405		13. Type of Report and Period Covered Technical Note 14. Sponsoring Agency Code
15. Supplementary Notes		
16. Abstract This document is reference material for personnel using or making software changes to the Data Link Test and Analysis System (DATAS) for Air Traffic Control Radar Beacon System (ATCRBS) transponder testing and data collection. This is one of a series of documents to be published on DATAS.		
17. Key Words Transponder (testing of) Data Link Data Link Test and Analysis System (DATAS)	18. Distribution Statement This document is available to the public through the National Technical Information Service, Springfield, VA 22161	
19. Security Classif.(of this report) Unclassified	20. Security Classif.(of this page) Unclassified	21. No. of Pages 152 / 22. Price

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

EXECUTIVE SUMMARY

This document provides reference material for persons using the Data Link Test and Analysis System (DATAS) for testing Air Traffic Control Radar Beacon System (ATCRBS) transponders.

Included in this document is a brief overall description of the DATAS, a brief description of the ATCRBS, a thorough description of how to operate the DATAS ATCRBS transponder analysis system (User Guide), a detailed description of the DATAS ATCRBS transponder analysis programs, and a thorough description of each of the transponder field tests.

INTRODUCTION

BACKGROUND.

Since the beginning of the federal operation of the air traffic control (ATC) system in 1936, the amount of air traffic has increased dramatically. In 1981 the Federal Aviation Administration (FAA) assessed their current status and it was predicted that by the year 2000 air traffic would double. Faced with this situation, the FAA conceived the National Airspace System (NAS) Plan. The NAS Plan included replacing the entire existing air traffic system with new automated and computer enhanced systems. These systems included an Advanced Automation System (AAS) that will provide computer assistance to air traffic controllers, increased reliability via a digital data link between ground sensors and aircraft which will be provided by a Secondary Surveillance Radar (SSR) known as the Mode Select Beacon System (Mode S), which will replace the current Air Traffic Control Radar Beacon System (ATCRBS). Delivery of the Mode S sensors will be in the 1990's.

The FAA Technical Center has a major role in the implementation of the NAS Plan. Many of the systems are being designed and/or tested at the Center. From the design and integration of such complex systems spawns the growth of small test systems such as the Data Link Test and Analysis System (DATAS).

The DATAS was originally conceived at the Technical Center by members of the Data Link project whose responsibilities include verification of Data Link systems reliability, interface protocols and system capacities. The DATAS was designed and fabricated at the FAA Technical Center to provide such test and analysis capabilities.

DATAS is capable of testing all components of the Data Link system. These components include: ATCRBS and Mode S transponders, avionics Data Link processors (ADLP), and all Data Link system interfaces. It will also provide the capability of Mode S sensor simulation and 1030 and 1090 megahertz (MHz) radio frequency (RF) environment analysis for all beacon transmissions including Traffic Alert and Collision Avoidance Systems (TCAS). The DATAS has the capability of RF signal analysis within the frequency range of 950 to 1200 MHz.

This document accompanies the completion of the first phase of the development of the DATAS, the ATCRBS transponder test system. Certain sections of this document are intended to provide unfamiliar users of the system with enough information to perform data collection operations. Other sections provide detailed information on the system software in order to make software maintenance easy and to aid development of future DATAS services.

DESCRIPTION OF EQUIPMENT

HARDWARE.

Figure 1 shows a very general block diagram of the DATAS in the ATCRBS transponder field data collection configuration. This diagram displays the full system. Some of the components shown are optional since they are not required when the system is used strictly for data collection purposes, but are used for data analysis functions.

Physically, the system consists of three racks: the RF unit, the DATAS hardware section (digital components), and the computer rack (each of which is approximately 19 inches x 12 inches x 19 inches), a dish antenna (which is approximately 50 inches in diameter), and, at minimum, one video display terminal.

The Motorola 68020 computer uses a Motorola MVME 135A central processing unit (CPU). This processor card consists of a 68020 32-bit microprocessor operating at 20 MHz. There is 4 megabytes of on-board random access memory (RAM). The card also contains a 68881 floating point coprocessor. The memory management unit has been disabled because there is too much overhead involved for some of the memory input/output (I/O) functions that the system will be required to perform with the Mode S testing functions.

The computer contains an MVME50 card for timing functions. The card provides up to 1 microsecond resolution.

The computer system uses a WYSE 75 (or compatible) terminal and any centronix interface printer. The lab system for software development uses a Fujitsu M304X series printer. The terminal is required to operate the system, but the printer is only required if the user wishes to obtain hardcopies of the transponder test summaries in data collection situations.

The DATAS may also contain an optional data presentation subsystem for data storage, data reduction, and report generation. The presentation subsystem is actually an independent personal computer (PC) system that plugs into the Versa Module Extended (VME) Bus rack and interfaces to the VMEbus. A Xycom XVME-682 VMEbus PC advanced Technology (AT) processor module is used for this purpose. This subsystem requires its own graphics display terminal. Optional peripherals include a floppy disk drive and a Hewlett Packard (HP) Laser Jet II printer or equivalent.

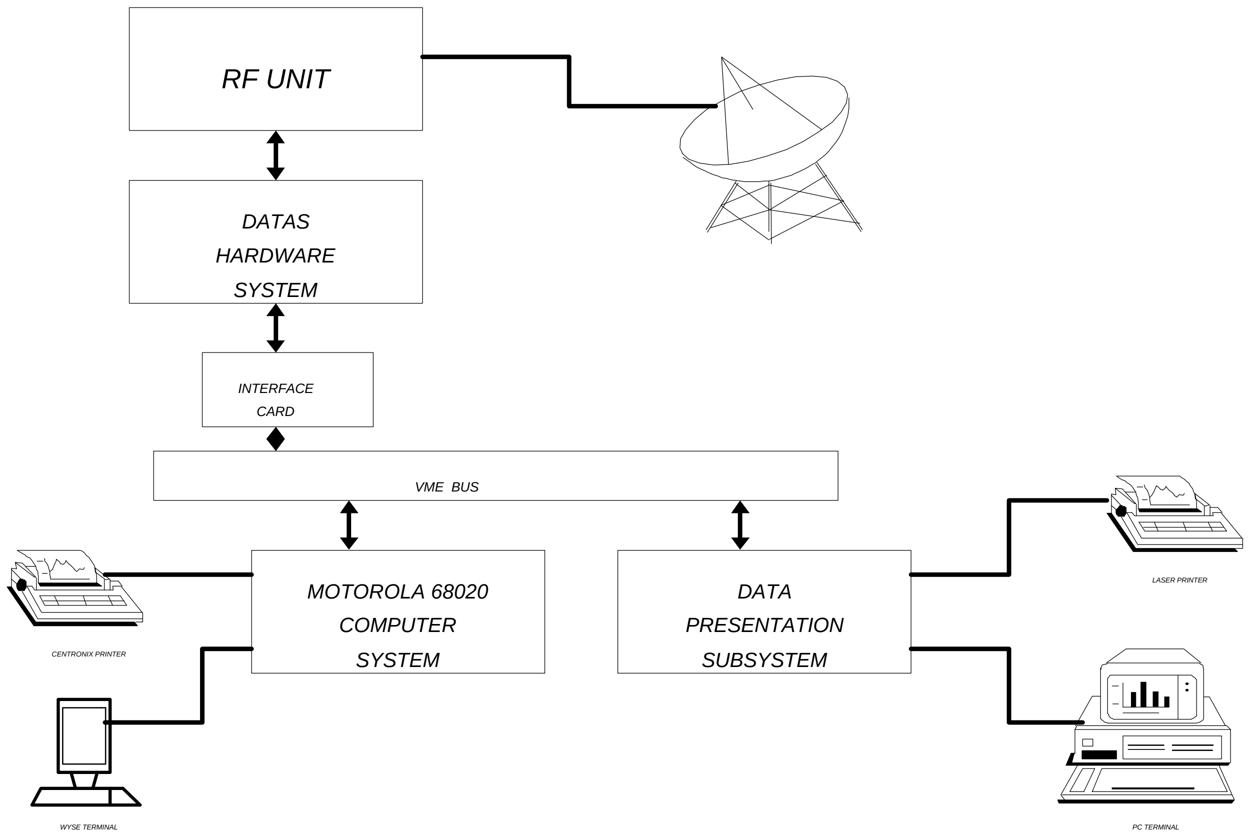


FIGURE 1. DATAS

SOFTWARE.

The Motorola computer systems were used for the software development of the DATAS. The operating system of the DATAS is Motorola VERSADOS, version 4.6. VERSADOS operates on the various CPU boards and systems offered by Motorola. VERSADOS is a real-time operating system that offers the facilities to operate in a real-time domain.

The DATAS programs were written in C language, except for a few of the lower level interrupt handling functions which were written in assembly language. The compiler used was the Alcyon C68 C compiler.

ATCRBS GENERAL DESCRIPTION

The ATCRBS is a secondary surveillance radar system that was designed to provide ATC with more information about aircraft within controlled areas. It is a cooperative system that consists of a ground-based rotating directional antenna, interrogator/receiver, signal processing equipment, and active aircraft transponders (figure 2). In operation, an interrogation pulse group is transmitted from the directional antenna and triggers each airborne transponder located in the directional main beam as the antenna rotates or scans by the aircraft. Measurement of the round-trip transmit time (the interrogation followed by the reply) determines the range; the mean direction of the interrogator antenna during aircraft replies determines the azimuth of the replying aircraft. This range/azimuth information is used to track the location of aircraft within controlled areas.

The ground-based interrogations are very specific, rigidly controlled RF pulse groups (code trains) transmitted by the ground equipment (on 1030 MHz) to interrogate all aircraft in the area of coverage. When the aircraft transponder receives an interrogation, it responds with a coded reply pulse train back to the ground station on a different frequency (1090 MHz). This reply contains the aircraft identity selected by the pilot, or, when the aircraft is properly equipped, contains the aircraft altitude (figure 2). The ground-based equipment can then automatically decode the information and determine the aircraft range, azimuth, identity, and altitude. Discrete aircraft emergency codes are also provided in the ATCRBS system.

INTERROGATION.

The basic interrogation consists of pulse pairs (P1 and P3) which are transmitted by the rotating directional antenna, normally at a repetition rate of several hundred pairs per

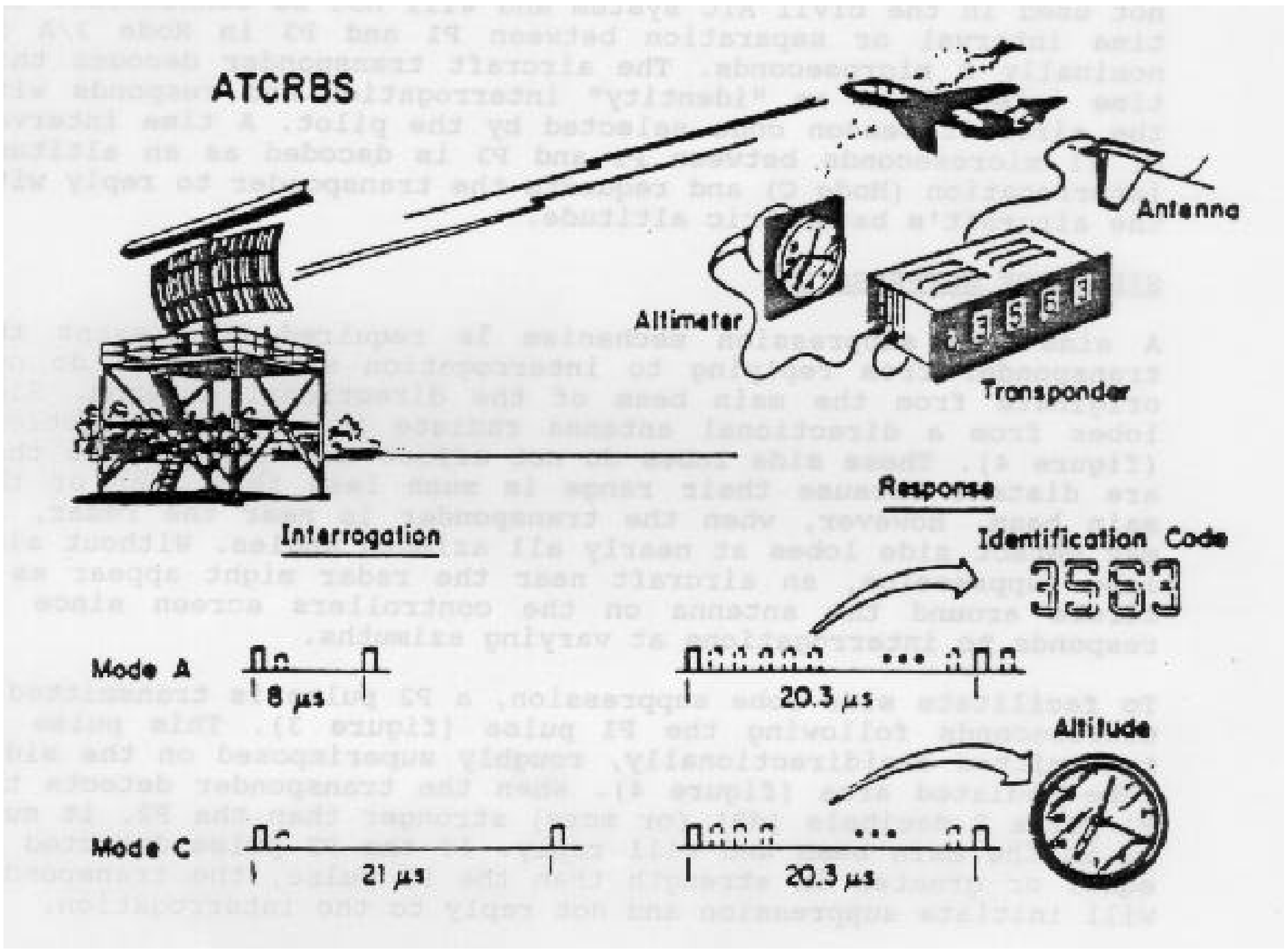


FIGURE 2. ATCRBS

second (figure 3). These pulses are nominally of equal amplitude and are separated by time intervals dependant on the use of the mode of interrogation. The primary Modes of interest are Mode 3/A (identity) and Mode C (altitude). Other modes such as Mode 4 are not used in the civil ATC system and will not be considered. The time interval or separation between P1 and P3 in Mode 3/A is nominally 8 microseconds. The aircraft transponder decodes this time interval as an "identity" interrogation and responds with the aircraft beacon code selected by the pilot. A time interval of 21 microseconds between P1 and P3 is decoded as an altitude interrogation (Mode C) and requests the transponder to reply with the aircraft's barometric altitude.

SIDE-LOBE SUPPRESSION.

A side-lobe suppression mechanism 'is required to prevent the transponder from replying to interrogation signals that do not originate from the main beam of the directional antenna. Side lobes from a directional antenna radiate in various directions (figure 4). These side lobes do not affect the transponders that are distant because their range is much less than that of the main beam. However, when the transponder is near the radar, it may detect side lobes at nearly all azimuth angles. Without side lobe suppression, an aircraft near the radar might appear as a circle around the antenna on the controllers screen since it responds to interrogations at varying azimuths.

To facilitate side lobe suppression, a P2 pulse is transmitted 2 microseconds following the P1 pulse (figure 3). This pulse is transmitted omnidirectionally, roughly superimposed on the side-lobe radiated area (figure 4). When the transponder detects the P1 pulse 9 decibels (dB) (or more) stronger than the P2, it must be in the main beam and will reply. If the P2 pulse detected is equal or greater in strength than the P1 pulse, the transponder will initiate suppression and not reply to the interrogation.

REPLY.

When the proper interrogation is detected, the transponder generates and transmits a reply pulse train encoded with either the aircraft's identity or pressure altitude, depending on the mode of the interrogation. The reply consists of 2 framing pulses (F1 and F2) spaced 20.3 microseconds apart; 12 data pulses and 1 special purpose pulse position (X pulse), all spaced in increments of 1.45 microseconds from the first framing pulse; and a special position identification pulse (SPI) spaced 4.35 microseconds after the F2 framing pulse (figure 5).

In the case of the Mode 3/A interrogation, aircraft identity is encoded in the ABCD pulses providing a capability of 4,096 discrete codes in the reply. For Mode C, the aircraft altitude is

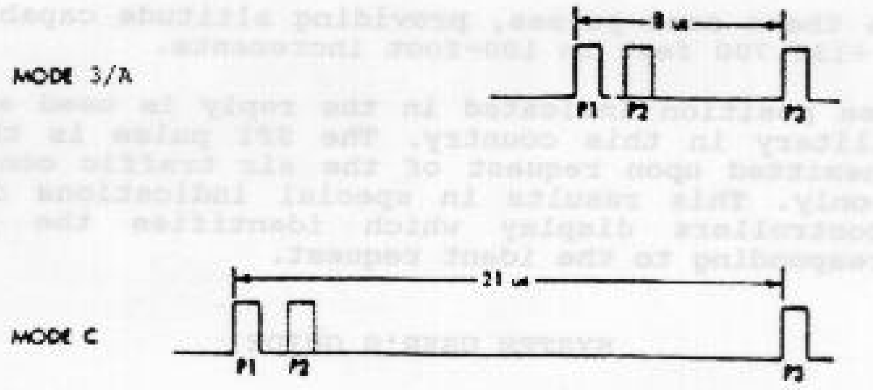


FIGURE 3. ATCRBS INTERROGATION

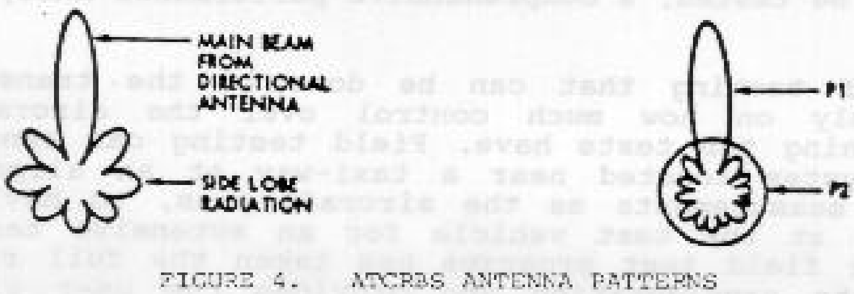


FIGURE 4. ATCRBS ANTENNA PATTERNS

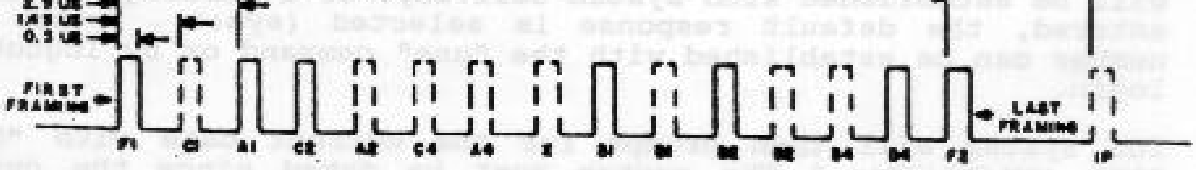


FIGURE 5. ATCRBS REPLY

encoded in these same pulses, providing altitude capability from -1,000 to +126,700 feet in 100-foot increments.

The X pulse position indicated in the reply is used exclusively by the military in this country. The SPI pulse is the "IDENT" pulse transmitted upon request of the air traffic controller in Mode 3/A only. This results in special indications on the air traffic controllers display which identifies the particular aircraft responding to the ident request.

SYSTEM USER'S GUIDE

This section describes the operation of the various programs that may be run when DATAS is used for ATCRBS transponder testing in the field data collection environment.

ATCRBS TRANSPONDER FIELD TESTING.

DATAS can be used to test the operation of beacon transponders without physically connecting to the hardware in any way. Although this does impose some limitations on the extent to which the units may be tested, a comprehensive performance analysis can be made.

The extent of testing that can be done on the transponders depends largely on how much control over the aircraft the personnel running the tests have. Field testing can range from having the system located near a taxi-way at an airport and making quick measurements as the aircraft pass, to having the aircraft stop at the test vehicle for an extensive test. The design of the field test programs has taken the full range of situations into consideration and provides the user with the ability to take full advantage of his particular test situation.

SYSTEM START-UP. When the system is powered on it will perform the boot procedure and activate VERSAdos which, upon initialization, will prompt the user for several inputs. The first is "ENTER DEFAULT SYSTEM VOLUME:USER=." Here the user should respond with the volume and user number that contains the ATCRBS field testing programs. An example response is "sys:321." The volume name is "sys" and should not change. The user number "321" is where the ATCRBS field test software was developed and should be the one where it will run. If there are any changes, it will be established with system delivery. If a carriage return is entered, the default response is selected (sys:0). A new user number can be established with the "use" command or by logout and login.

The system will then prompt for the current date with "ENTER DATE (MM/DD/YR)=." The system must be dated since the current date is used as part of the file names created by the programs. The test date is required as an entry from programs that reproduce test data previously acquired. For this reason, the date of data collection should be recorded.

The system will then prompt for the current time with "ENTER TIME (HR:MIN)=." The time of day is not a critical entry for the ATCRBS testing programs, but may be with some of the other DATAS software packages. The correct time of day should be entered.

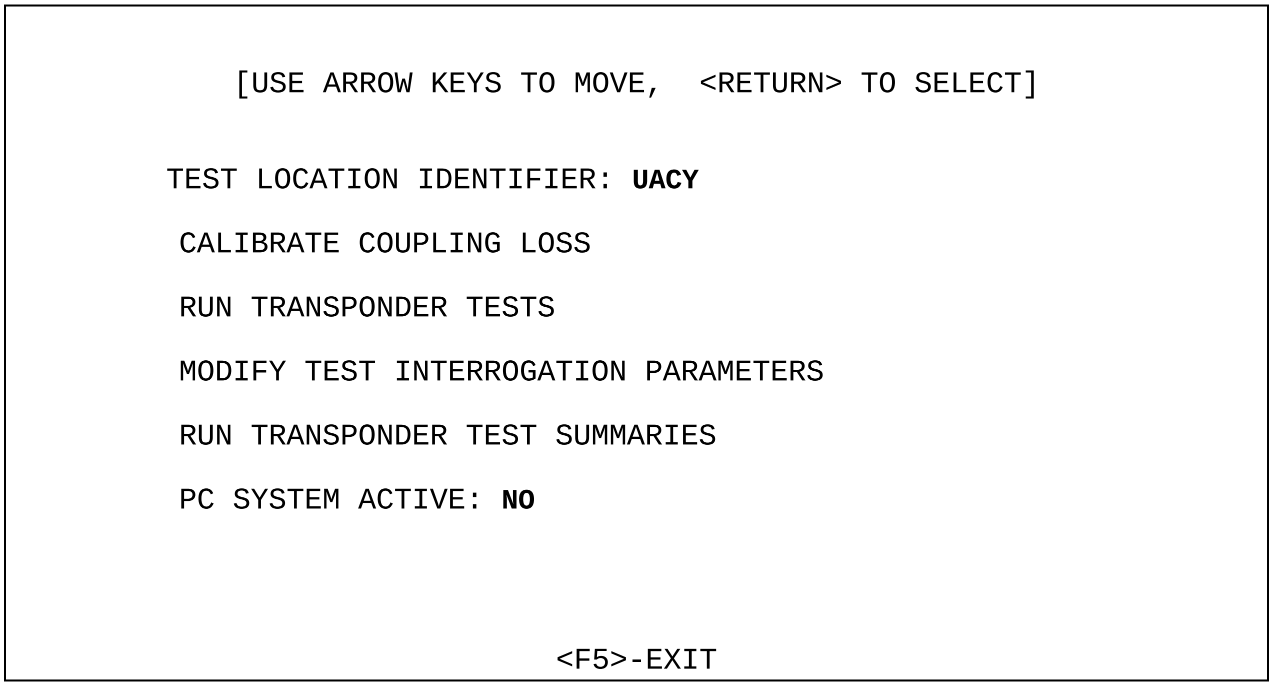
When the system completes the bulletin and several chain files, the "=" prompt will appear. VERSAdos is now active. Any VERSAdos command may be entered or any of the DATAS application programs that reside in the current user number may be run.

DATA COLLECTION PROCEDURE. There are several basic steps involved in conducting a transponder field test. First a location for the system must be chosen, then the coupling losses must be measured between the system and the transponder's antenna, a test scenario must be generated, and the program must be run that will conduct the actual field testing.

MAIN MENU. The ATCRBS Transponder Field Test program is activated with the command "afexec". Figure 6 shows screen #1 of the ATCRBS transponder field testing program. This screen provides a menu for the user to select which of the higher level functions of the program to perform. The current menu choice is highlighted in reverse video, and the arrow keys are used to move between choices. The function key F5 will exit the program.

1. "TEST LOCATION IDENTIFIER." This is not one of the functions of the program, but is an important item. An entry here is required only if transponder data collection will be conducted. Up to four characters may be entered at this prompt. A carriage return is required to accept the entered characters. It is suggested that the standard four character identification code for the airport (or the nearest airport to the test location) is entered here, although anything the user desires up to four characters may be entered. Whatever is entered here must be remembered since this character field is used as part of the data file name. The same response entered here when data are collected will be required by the functions that reduce or reproduce previously recorded data.

2. "CALIBRATE COUPLING LOSS." If this prompt is selected with a carriage return, the coupling loss calibration function is performed. The purpose of this function is to measure the difference between the original calibrated RF point of the system with the location where the aircraft will be tested. These differences are stored as offsets for test data acquired during testing. Those measurements affected by coupling losses are sensitivity (which requires an accurate system transmit power level), reply power, and reply delay. The coupling loss program has not been developed as of yet since the need for the system as a transponder field tester has not arisen.



```
            [USE ARROW KEYS TO MOVE,  <RETURN> TO SELECT]

        TEST LOCATION IDENTIFIER: UACY

         CALIBRATE COUPLING LOSS

         RUN TRANSPONDER TESTS

         MODIFY TEST INTERROGATION PARAMETERS

         RUN TRANSPONDER TEST SUMMARIES

         PC SYSTEM ACTIVE: NO



                              <F5>-EXIT
```

FIGURE 6. SCREEN #1

3. "RUN TRANSPONDER TESTS." This selection will activate the transponder test programs. These programs require a test sequence file and a calibrated system. The description of these programs is later in the section (Test sequence files).

4. "MODIFY TEST INTERROGATION PARAMETERS." This selection will allow the user to modify how the tests themselves will run. These programs allow the user to change things such as P1 and P3 width, interrogation power level, and/or the number of samples to take, etc., for each of the transponder tests. This selection should be made when the user wishes to reduce data collection time, modify the tests performance, or increase data collection for a more detailed analysis. The description of these programs is discussed later in this section.

5. "RUN TRANSPONDER TEST SUMMARIES." This selection is used to reproduce test results of previously recorded data. Test summaries or test plots can be produced. Test plots require the

PC subsystem, therefore, the PC system activation must be performed prior to making plots of test results. These programs are described later in this section.

6. "PC SYSTEM ACTIVE." This prompt is used to establish communications between the PC subsystem and the Motorola 68020 computer. This communication path is required for plotting data on the PC system screen or printer, or storing the data on the PC disk. Figure 7 shows the subsequent prompts that follow a "YES" response. The "SCREEN PLOT (PC)" prompt allows data to be plotted after each series of tests is completed ("ON POST-TEST"), as the data is acquired (live) ("ON LINE"), or no plot on the screen ("OFF"). The "HARDCOPY PLOT (PC)" prompt will select whether or not to produce a hardcopy of the plots on the PC system. The "STORE DATA IN PC (PC)" prompt will select whether or not to store the data on disk in the PC system.

```
PC SYSTEM ACTIVE:       YES            (NO)
SCREEN PLOT (PC):       ON POST-TEST  (ON LINE, OFF)
HARDCOPY PLOT (PC):     NO             (YES)
STORE DATA IN PC (PC):  NO             (YES)
```

FIGURE 7. PC SYSTEM ACTIVATION

TEST OPTIONS MENU. Figure 8 shows the Test Options menu. This menu allows the user to select between various options that control how the system will conduct the transponder tests. Again, the current menu choice is highlighted in reverse video, and the arrow keys are used to move between choices. The function key F5 will return to screen #1 and F6 will advance to the transponder test screen.

1. "HARDCOPY OF TEST RESULTS." This prompt can be answered either "YES" or "NO." The selection is made with the space-bar key (actually, any key will work for any of the space-bar selection prompts). This will select whether or not a printed summary page is automatically produced for each transponder tested. If "NO" is selected a printed summary could still be produced from the command keys during testing.

2. "POWER MODE." This prompt can be answered either "MANUAL" or "AUTOMATIC." The selection is made with the space bar key. If "MANUAL" power mode is selected the interrogation power of the system can be adjusted manually by the user from the transponder test screen. If "AUTOMATIC" power mode is selected the system will adjust the interrogation power automatically between tests in order to maximize reply efficiency and minimize interference

from other transponders. Automatic power mode has not yet been implemented.

3. "INITIAL POWER LEVEL decibels referenced milliwatt (dbm)." This prompt allows the user to enter the starting interrogation power level in -dBm. The prompt will accept up to two numeric digits and requires a carriage return. The level will be adjusted by the system if power mode is "AUTOMATIC" or can be adjusted by the user if power mode is "MANUAL."

```
           DATAS TRANSPONDER FIELD DATA COLLECTION PROGRAM
                          TEST OPTIONS MENU


        SELECT USING ARROW KEYS.............   <CR> TO ENTER

     HARDCOPY OF TEST RESULTS:    NO

                   POWER MODE: MANUAL

    INITIAL POWER LEVEL (dBm): -45

           TEST SEQUENCE FILE: STDSEQ

                   STORE DATA: YES

           DATABASE FILE EXT.: AA NEW FILE


            F5 TO EXIT -------------- F6 TO START SYSTEM
```

FIGURE 8. TEST OPTIONS MENU

4. "TEST SEQUENCE FILE." The test sequence file name is entered at this prompt. The prompt will accept up to eight characters and requires a carriage return. Entry will be tested by the program to see if it is a valid file name. If it is not, error messages will appear on the screen. The test sequence file contains a list of transponder test numbers that control which tests and in what order they will be run.

S. "STORE DATA." This prompt can be answered either "YES" or "NO." The selection is made with the space-bar key. If the user elects to store data he will be prompted for a data file name extension. The data file will contain test data and aircraft information for all transponders tested. The data should be stored whenever the system is used in field data test situations.

The only time no data storage should be selected is during debug situations.

6. "DATABASE FILE EXT.." This prompt will accept up to two characters and requires a carriage return. This prompt is only available when the user has selected data storage in the previous prompt. The entry will be the file extension for the data file. If the data file does not exist, "NEW FILE" will be displayed. If the data file exists "WARNING: FILE EXISTS.. WILL APPEND NEW DATA" will be displayed.

TRANSPONDER TEST SCREEN. Figure 9 shows the transponder test screen. This is how the display will appear after the F6 key is entered from the previous screen shown in figure 8. From here the transponder tests are conducted.

```
        DATAS ATCRBS FIELD DATA COLLECTION SYSTEM  08-04-1989
                   IDLE INTERROGATION RUNNING
  TEST STATUS     POWER     DELAY     OUTPUT POWER (DBM):- 45.0
_______________________________________________________________








    AIRCRAFT ID:             AIRFRAME HEIGHT(FT):0
  AIRCRAFT TYPE:                      POWER MODE:MANUAL
TRANSPONDER TYPE:
COMMENT FIELD:

            <F5>-EXIT    <F7>-INCREASE <F8>-DECREASE POWER
                   <F9>-RUN TEST    <F10>-STOP IDLE
```

FIGURE 9.   TRANSPONDER TEST SCREEN

The date is displayed at the top right corner of the screen. Just below that line is the system status message line. Here the current status of the system is displayed, such as "IDLE INTERROGATION RUNNING" or "FIELD TEST IN PROGRESS," etc.

The interrogation power of the DATAS system is displayed in the upper right corner. This level can be adjusted using the F7 and F8 function keys if the system is in manual power mode.

The center portion of the screen is the scrolling region for the ATCRBS tests. As each test completes, the test number, completion message, reply power, and reply delay are sent to the screen. See figure 10 as an example of a field test in progress.

```
         DATAS ATCRBS FIELD DATA COLLECTION SYSTEM  08-04-1989
                      FIELD TEST IN PROGRESS
TEST STATUS      POWER       DELAY       POWER (DBM):- 45.0
________________________________________________________________
1-TEST 10....    COMPLETE 55.7 dBm  3.1 us
2-TEST 17....    COMPLETE 55.8 dBm  3.0 us
3-TEST 19






    AIRCRAFT ID:           AIRFRAME HEIGHT(FT):0
  AIRCRAFT TYPE:                    POWER MODE:MANUAL
TRANSPONDER TYPE:
COMMENT FIELD:
                             <F9>-STOP TEST
```

FIGURE 10.   TRANSPONDER TEST SCREEN WITH
FIELD TEST IN PROGRESS

Below the scrolling region is the aircraft information entry area. Here the Aircraft ID (Tail number), aircraft type, transponder type, and a comment field entry areas are provided as a means of identifying the test data. The cursor is moved between these prompts using the arrow keys; the entries must be ended with a carriage return. This information is stored with each series of completed tests. The aircraft information can be entered prior to testing and after the test has completed. It cannot be entered during the test sequence or after the test data has been stored. See figure 11 for an example screen with aircraft information entered. There is also a prompt for airframe height and power mode within this area. The cursor is on the airframe height prompt when this screen is initially displayed so that it can be entered prior to the start of the test. The measured reply power and the interrogation power delivered to the aircraft's antenna are affected by the height of the antenna from the ground. The approximate height of the antenna in feet (usually the height of the underside of the aircraft) should be entered here prior to the start of the test. There is also a prompt that allows the user to switch between MANUAL and AUTOMATIC power mode. (See the description of "POWER MODE" after figure 8 "Test options menu.")

```
         DATAS ATCRBS FIELD DATA COLLECTION SYSTEM  08-04-1989
                    ENTER AIRCRAFT INFORMATION
TEST STATUS      POWER       DELAY        POWER (DBM):- 45.0
___________________________________________________________
1-TEST 10... COMPLETE 55.7 dBm 3.1 us
2-TEST 17... COMPLETE 55.8 dBm 3.0 us
3-TEST l9... COMPLETE 56.0 dBm 3.1 us
   SENSITIVITY 70 - 71  POWER 55.6 - 56.0  DELAY 3.30 - 3.00
  TEST ENDED AFTER SEQUENCE #3 ... TEST #19

        AIRCRAFT ID:N1234          AIRFRAME HEIGHT(FT):0
      AIRCRAFT TYPE:CESSNA 150              POWER MODE:MANUAL
   TRANSPONDER TYPE:KING KT-76
      COMMENT FIELD:NO MODE C CAPABILITY

            <F9>-STORE TEST RESULTS   <F10>-ABORT
```

FIGURE 11. TRANSPONDER TEST SCREEN WITH FIELD TEST COMPLETE

Below the aircraft information entry area is the function key definition area. This area displays the currently active function keys. The function keys will not be the same all the time, they depend on the current status of the system. For example, if the system is running, the only active function is F9 which is to stop testing (see figure 10). Figure 9 shows the active function keys when the idle interrogation is running. The F5 key will exit (return to screen #1), F7 and F8 are to increase or decrease the interrogation power level. They are only active when MANUAL power mode is selected. The power level is made variable so that the user can make the power level high enough so that the aircraft being tested will reply, but low enough so that any other aircraft nearby will not. The F9 key is used to start the test sequence. The F10 key will stop the idle interrogation. When the idle interrogation is stopped the F10 key can be used to start it again.

Figure 10 shows the transponder test screen with the transponder testing in progress. At the top of the screen, the system status shows "FIELD TEST IN PROGRESS." The test scrolling region shows that test 10 is complete and the highest reply power reported during the test was 55.7 dBm and the shortest reply delay was 3.1 microseconds. Test 17 is also complete, and the improvement shown in both the reply power and delay may indicate that the aircraft is moving towards the center of the beam. The only active function key is F9 which can be used to stop the test.

If the test sequence is not stopped prior to completion with the F9 key, the system will continuously monitor the transponders sensitivity, reply power, and reply delay until the user stops the system with the F9 key. At this point the screen will no longer scroll, but will continuously display the current sensitivity, power, and delay as compared to the best of each measured so far (see figure 11). The display will show "current value--best value" for each of the three measurements. If the current value is an improvement over the best value, it will be displayed in reverse video along with the word "INCREASING" left of the delay. The best value will then be updated. The idea behind this is so that these measurements can be made while the aircraft rolls through the calibrated main beam.

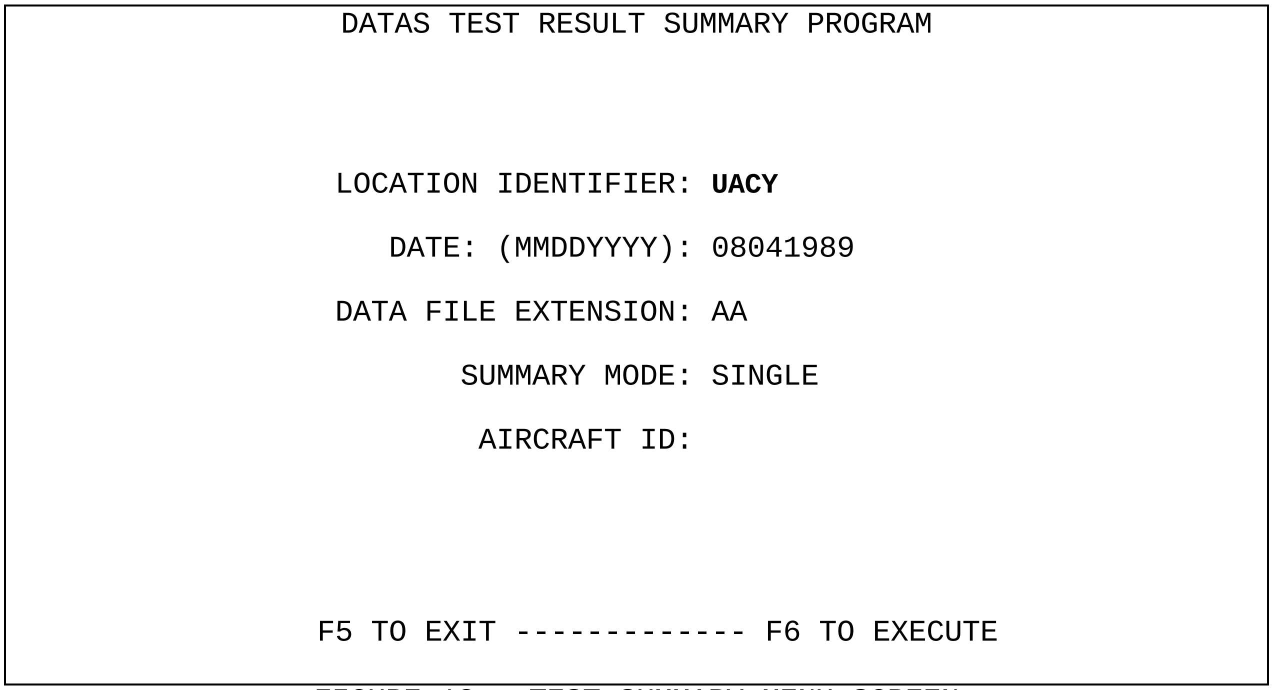
Figure 11 shows the transponder test screen after the test sequence is complete and the user has stopped the test with the F9 key. The system status message is reminding the user to enter the aircraft information. In this example screen the aircraft information has already been entered.

In the scrolling region, it shows that three tests have run, test 10, 17, and 19. The highest sensitivity measured was -71 dBm, the highest reply power was 56.0 dBm and the shortest reply delay was 3.0 microseconds. The highest reply power and/or shortest delay can be measured in any of the transponder tests since they all record that information.

The active function keys at this point are F9 to store the test results and F10 to abort the test. If F9 is entered, the data from all the tests and the aircraft information will be stored in the data file and a summary of the test results will be displayed (see figure 14). If F10 is entered, the data will be discarded and the screen will return to original transponder test screen as shown in figure 9.

<u>TEST SUMMARIES</u>. Figure 12 shows the test summary menu screen which is called when the "RUN TRANSPONDER TEST SUMMARIES" menu selection is made from the main menu (screen #1). This menu allows the user to locate data files on the disk in order to produce test summaries or plot the data using the PC subsystem. The three file location entries are the location identifier, date, and data file extension. The arrow keys are used to move
between selections. The location identifier, date, and data file extension must be the same as what was entered when the test data was originally collected.

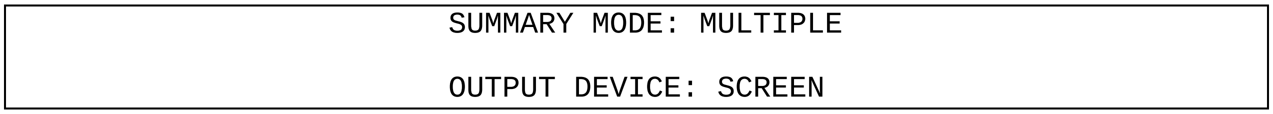
The "SUMMARY MODE" prompt allows the selection of whether to produce the results of a single aircraft or all aircraft in the file. The mode selection is switched using the space bar key. If the mode is "SINGLE" the program will retrieve the test data for a single aircraft. A subsequent prompt requesting the "AIRCRAFT ID" accompanies the SINGLE summary mode. The identification of the aircraft is required to find its test results. The aircraft ID must be entered identical to what was stored with the data, and must be terminated with a carriage return.



```
                    DATAS TEST RESULT SUMMARY PROGRAM



                    LOCATION IDENTIFIER: UACY

                       DATE: (MMDDYYYY): 08041989

                    DATA FILE EXTENSION: AA

                           SUMMARY MODE: SINGLE

                            AIRCRAFT ID:



                 F5 TO EXIT ------------- F6 TO EXECUTE
```

FIGURE 12. TEST SUMMARY MENU SCREEN

If the summary mode selected is "MULTIPLE" the program will retrieve all of the data in the file one aircraft at a time (see figure 13). A second prompt requesting an output device appears with the multiple summary mode. The selections available are for "SCREEN" or "PRINTER." A space bar is used to change the selection.



```
                        SUMMARY MODE: MULTIPLE

                       OUTPUT DEVICE: SCREEN
```

FIGURE 13. SUMMARY MENU SCREEN "MULTIPLE SUMMARIES" SELECTED.

Figure 14 shows an example test summary page from a transponder test. This screen can be called from two different places. First, it could be displayed following the completion of a transponder test. After the user elects to store the test data (F9 entered figure 11), the summary of the tests is displayed. Second, the screen could be called from the test summary menu (F6 key figure 12).

```
                         DATAS FIELD TEST SUMMARY

  AIRCRAFT ID: N1234        AIRCRAFT TYPE: CESSNA 150
  TRANSPONDER TYPE: KING KT-76
  COMMENT: NO MODE C CAPABILITY

  10 - Minimum RF Output Power = 52.3 dBm
  17 - Reply Rate Limit = 975 PRF
       Sensitivity Reduction - Mode A 98.0%   Mode C 97.0%
  19 - Maximum pulse position error (us) F1 ref = -1.02
                               Pulse-pulse ref = -.98




      <F5>-EXIT <F6>-PAGE <F7>-PRINT LINE/ALL <F8>-PLOT LINE/ALL
```

FIGURE 14.   TEST SUMMARY PAGE

At the top of the screen is the aircraft information. The center area of the screen shows the test summaries. In the example shown, tests 10, 17, and 19 were conducted on the transponder. At the bottom of the screen are the various function keys that are available.

The F5 key will exit the screen. If the screen was called from a Summary program, it will return to the Summary menu. If the screen was entered following a test sequence, it will return to the transponder test screen.

The F6 key will page through the test summaries if they take more room than will fit on one page. In the example shown, there is less than one screen of test summaries.

The F7 key will print the summary for the test which is displayed in reverse video (in the example it is in bold). The arrow keys may be used in order to select another test summary to print. The F7 key will always print the test summary which is displayed in reverse video. If SHIFT F7 is entered, all test summaries for the transponder will be sent to the printer.

The F8 key will plot the test results for the current test displayed in reverse video. If SHIFT F8 is entered, all tests for the transponder will be plotted. For anything to be plotted, the PC system must be on line (see main menu figure 7).

If the screen was called from the summary menu in "MULTIPLE" summary mode, an additional function will appear. F9, the "MORE" key, is used to advance to the following transponder test within the file.

<u>PARAMETER MODIFICATION</u>. Figure 15 shows screen 1 of the parameter modification programs. This screen is called when the "MODIFY TEST INTERROGATION PARAMETERS" option is selected from the main menu (figure 6). The test parameters are variables that can be changed by the user to alter the way a transponder test works. There are two types of parameters: interrogation parameters which define the interrogation used by the test, and test parameters which enable test algorithms to be modified. The interrogation parameters are the same for all of the tests, but all of them are not used by every test. Appendix C describes how each test uses the interrogation parameters.

The first screen is simply a prompt for the test number of which the user wishes to modify. If a "0" is entered, the interrogation parameters for all tests can be changed. This is useful if, for example, the user wishes to run all tests at a lower PRF, he would enter a 0 and change the PRF in the following screen.

A single function key, F5 is provided which is used to return to the main menu.

```
                    --------- PARAMETER MODIFICATION ---------
```

```
 ENTER TEST NUMBER -

   - TO SET GLOBAL PARAMETERS FOR ALL TESTS





                              <F5>-EXIT
```

FIGURE 15.   PARAMETER MODIFICATION MENUS
SCREEN #1

Figure 16 shows an example of an interrogation parameter menu for test 3. There are two sets of parameters: one for Mode A and the other for Mode C. A new value for a parameter is entered from the keyboard followed by a carriage return. The arrow keys are used to select which parameter to change. The currently selected parameter is displayed in reverse video (in the example it is shown in bold print "P1 WIDTH (us)   0.800"). Once a parameter is changed to a value other than its default value it will be displayed in reverse video. Each parameter has a limited range and resolution. If a value is entered that is inappropriate for one of the parameters, an error message is displayed and the entry is rejected. Table 1 is a table of the interrogation parameters, their acceptable ranges, and resolutions.

```
                      PARAMETER MENU FOR TEST -3


                             -----MODE A-----

Pl WIDTH (us)     0.800           P2 WIDTH (us)            0.800
P3 WIDTH (us)         0.800           Pl-P2 SPACING (us)       2.000
Pl-P3 SPACING (us)    8.000           INT POWER OFFSET(dB)  0.000
P2 POWER OFFSET(dB)   -40.000         PRF                   450
FREQUENCY (MHz)       1030.000


                             -----MODE C-----

Pl WIDTH (us)         0.800           P2 WIDTH (us)            0.800
P3 WIDTH (us)         0.800           Pl-P2 SPACING (us)       2.000
Pl-P3 SPACING (us)    21.000          INT POWER OFFSET(dB)  0.000
P2 POWER OFFSET(dB)   -40.000         PRF                   450
FREQUENCY (MHz)       1030.000




    <F5>-EXIT <F6>-SET ALL TO DEFAULT <F7>-SET CURRENT TO DEFAULT
```

FIGURE 16. INTERROGATION PARAMETER MENU

TABLE 1. INTERROGATION PARAMETERS

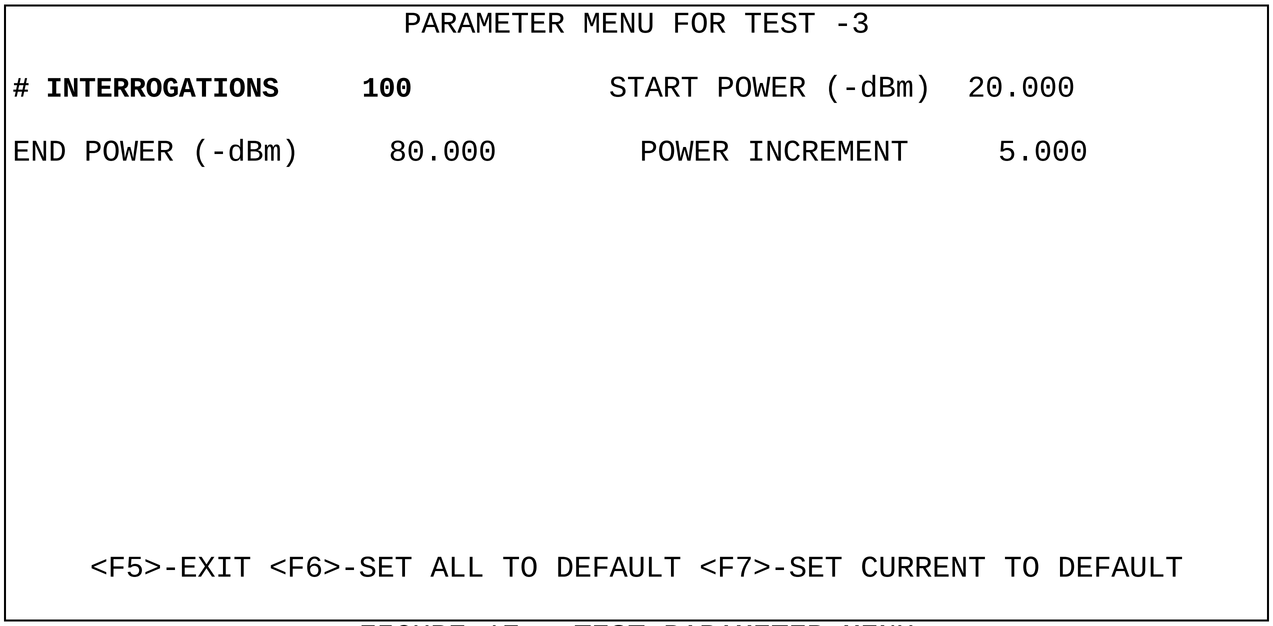
Parameter	Default Value	Minimum Value	Maximum Value	Resolution
P1 WIDTH (us)	0.800	0.100	7.975	0.025
P2 WIDTH (us)	0.800	0.100	7.975	0.025
P3 WIDTH (us)	0.800	0.100	7.975	0.025
P1-P2 SPACING (us)	2.000	0.100	80.000	0.025
P1-P3 SPACING (us)	*8.000,**21.000	0.200	80.000	0.025
INT POWER OFFSET(dB)	0.000	-90.000	+90.000	0.1
P2 POWER OFFSET(dB)	-40.000	-90.000	+90.000	0.1
PRF	450	1	2000	50
FREQUENCY (MHz)	1030.000	900.000	2000.000	0.500

*-Mode A, **-Mode C

The pulse widths are from lead edge to trail edge in microseconds. The pulse spacings are from lead edge to lead edge in microseconds. The power offsets are the number of dB's from normal. For example, if INT POWER OFFSET for a test was +3.000, that would mean that the interrogation power is 3 dB higher than all other tests at default. These power levels are relative to the current system power. In~order to introduce suppression into a test, P2 POWER OFFSET could be set to 0.000. This would mean that P2 power is equal to Pl,P3 power. PRF is the number of interrogations per second. Frequency is the RF transmit frequency in MHz.

The function keys for both the interrogation and test parameter menus are the same. The F5 key will exit and store the parameters in the menu. The F6 key will return all parameters on the screen to their default values. The F7 key will return only the selected parameter (reverse video) to its default value.

Figure 17 shows an example of a test parameter menu for test 3. The menu works the same as the interrogation parameter menu. The test parameters are not the same for all tests. They are user changeable parameters designed to make the transponder test procedure more flexible. Their primary importance is in accommodating the speed of each test to the present test situation. If there is an unlimited time to test each transponder, the values could be made to acquire the largest sample of data with the smallest granularity possible, in order to make each test as accurate as possible. If the desire is to keep test time to a minimum, the samples could be reduced and the granularity made greater. The default values for each test are usually somewhere in the middle.



```
                    PARAMETER MENU FOR TEST -3

# INTERROGATIONS     100           START POWER (-dBm)  20.000

END POWER (-dBm)     80.000        POWER INCREMENT     5.000





    <F5>-EXIT <F6>-SET ALL TO DEFAULT <F7>-SET CURRENT TO DEFAULT
```

FIGURE 17. TEST PARAMETER MENU

TEST SEQUENCE FILES.

Test Sequence Files are user generated files that determine which transponder tests are run and in what order when the transponder test program is activated. A valid test sequence file name is required at the start of the transponder test program. The DATAS system provides an editor called "SEQ" to generate or modify test sequence files.

GENERAL DESCRIPTION. A test sequence file contains a list of numbers which determine which transponder tests are run. The order of the tests are determined by the order of the numbers in the file. The purpose of test sequence files is to allow a predetermined series of tests to be run without user interaction during time critical testing situations.

TEST NUMBERS. Most of the transponder test procedures come from the ATCRBS/Mode S transponder MOPS section 2.4.2 "Detailed Test Procedures." The test numbers are assigned in the order that they appear in the MOPS. The reason the test numbers are not consecutive is because the MOPS test procedures intermix ATCRBS and Mode S tests, and this document describes only the ATCRBS transponder tests. However, it is not required that this numbering scheme be strictly adhered to since some tests may be combined into one test, or new tests may be designed to meet specific user requirements. Table 2 lists the available ATCRBS transponder tests.

SEO - TEST SEQUENCE FILE EDITOR. The SEQ editor allows the user to generate new files, modify existing files, or modify existing files and store them with new file names.

The SEQ editor is menu driven and utilizes four different menu screens that provide the following functions:

1. File access - used to read files for editing and write files for disk storaqe.

2. File display - displays the contents of a test sequence file with the test numbers referenced with the Minimum Operational Performance Standard (MOPS) test description.

3. Test list - displays a list of available transponder tests in numerical order with reference to the MOPS test descriptions.

4. Edit screen - displays the contents of the file as test numbers for editing purposes.

TABLE 2. ATCRBS TRANSPONDER TESTS

TEST NUMBER	MOPS REFERENCE	DESCRIPTION
2	2.4.2.1 Step 2	Sensitivity
3	2.4.2.1 Step 3	ATCRBS Dynamic Range
9	2.4.2.2.1	Reply Transmission Frequency
10	2.4.2.2.2 Step 1	ATCRBS Power Output
17	2.4.2.2.5 Step 1	Determination of Reply Rate Limit

19	2.4.2.3.1 Step 1	ATCRBS Reply Pulse Spacing
20	2.4.2.3.1 Step 2	ATCRBS Reply Pulse Shape
22	2.4.2.3.2 Step 1	Reply Delay
25	2.4.2.4 Step 1	SLS Decoding
27	2.4.2.4 Step 3	SLS Pulse Ratio
29	2.4.2.4 Step 5	Suppression Duration
30	2.4.2.4 Step 6	Suppression Reinitiation
31	2.4.2.4 Step 7	Recovery After Suppression
34	2.4.2.5 Step 3	Pulse Level Tolerances (ATCRBS)
36	2.4.2.5 Step 5	Pulse Duration Tolerances (ATCRBS)
41	2.4.2.5 Step 10	Simultaneous Interrogations
42	2.4.2.6 Step 1	ATCRBS Single Pulse Desensitization and Recovery
44	2.4.2.8	Undesired Replies
55	Not from MOPS	Suppression Test

CALLING THE SEO EDITOR. In order to run the Test Sequence File editor, enter "seq" at the VERSAdos prompt. The display will appear as in figure 18. The first display in the test sequence editor is the file access screen. At this point the user may read an existing file, write to a file whether it exists or not, or exit the program. The read/write function is switched using the space-bar key. The function key F5 will exit the program.

READING FILES. The user may accept the default file name "TSTSEQ" by entering a carriage return or enter a file name of his choice followed by a carriage return. In either case, if the file exists, the screen will display the contents of the file as shown in the example display in figure 19. If the file does not exist, a blank edit screen similar to the one shown in figure 20 will be displayed.

```
    DATAS TEST SEQUENCE FILE EDITOR

    ENTER FILE NAME: STDSEQ
            READ FILE
     (SPACE BAR TO CHANGE)






  F5-EXIT
```

FIGURE 18. SEQ DISPLAY #1 FILE ACCESS

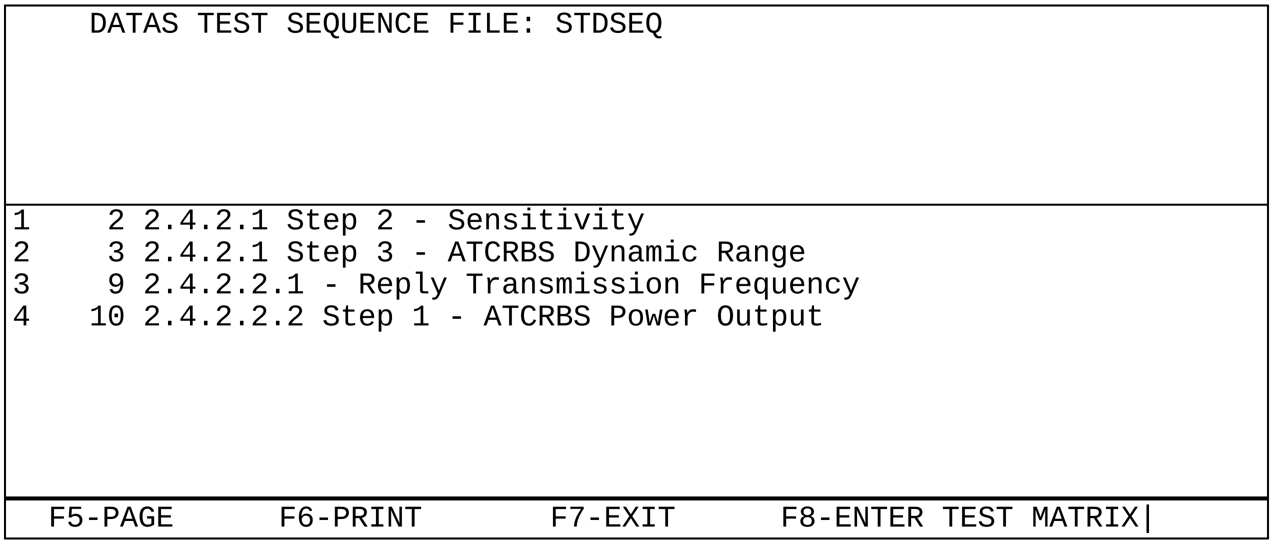
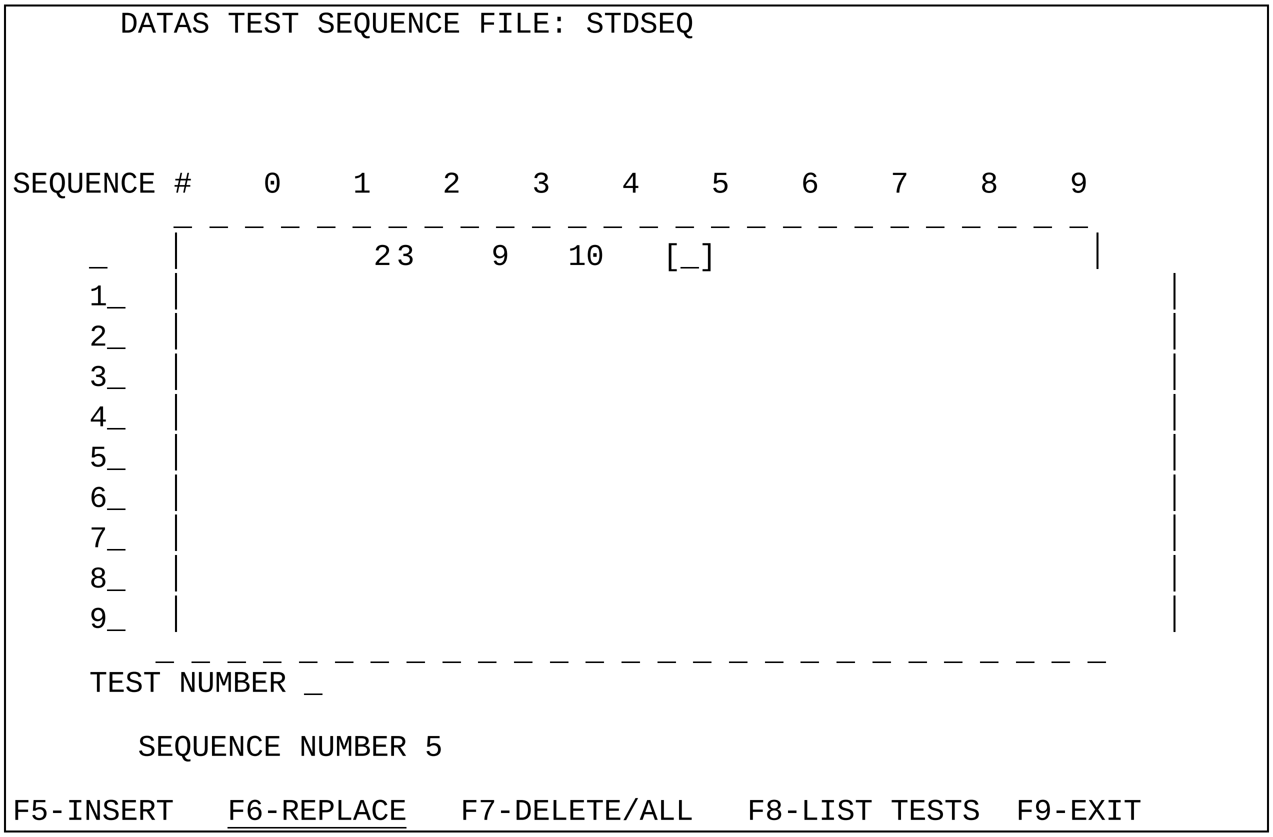

```
    DATAS TEST SEQUENCE FILE: STDSEQ

1     2 2.4.2.1 Step 2 - Sensitivity
2     3 2.4.2.1 Step 3 - ATCRBS Dynamic Range
3     9 2.4.2.2.1 - Reply Transmission Frequency
4    10 2.4.2.2.2 Step 1 - ATCRBS Power Output

  F5-PAGE      F6-PRINT       F7-EXIT      F8-ENTER TEST MATRIX|
```

FIGURE 19. SEQ DISPLAY #2 EXAMPLE FILE DISPLAY



```
      DATAS TEST SEQUENCE FILE: STDSEQ



SEQUENCE #    0    1    2    3    4    5    6    7    8    9
         _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
    _    |          23    9   10   [_]                  |
    1_   |                                                  |
    2_   |                                                  |
    3_   |                                                  |
    4_   |                                                  |
    5_   |                                                  |
    6_   |                                                  |
    7_   |                                                  |
    8_   |                                                  |
    9_   |                                                  |
        _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
    TEST NUMBER _

       SEQUENCE NUMBER 5

F5-INSERT   F6-REPLACE   F7-DELETE/ALL   F8-LIST TESTS  F9-EXIT
```

FIGURE 20. SEQ DISPLAY #4 EDIT SCREEN

Figure 19 shows an example display of the contents of a test sequence file named "STDSEQ." The file will cause the transponder test program to run the following four transponder tests: test 2 - Sensitivity, test 3 - ATCRBS Dynamic Range, test 9 - Reply Transmission Frequency, and test 10 - ATCRBS Power Output. The decimal numbers refer to the MOPS section which defines the test.

The F5 key will page through the list of tests if there is more than one screen is able to display. The F6 key will send a complete list of the tests in the file to the system printer. The F7 key will return to display #1, the file access screen. The F8 key will advance to the edit screen.

EDITING FILES. Figure 20 shows the test matrix screen where the actual file editing takes place. The file name is displayed at the top of the screen. In this example the file name is "STDSEQ."

There is a 10x10 matrix which shows the 99 possible test numbers in their sequential order. There is a cursor within the matrix which highlights the current sequence position. In the example it is in sequence order 5 since this is the next available sequence position. The screen entry cursor can be moved between two prompts using the arrow keys: the "TEST NUMBER" prompt and the "SEQUENCE NUMBER" prompt. The "TEST NUMBER" prompt is where the actual test numbers are entered and stored in the position determined by the number in the "SEQUENCE NUMBER" prompt. The user may change the sequence number to allow insertions and changes to the file. When a test number is entered the program will automatically advance to the next sequence number. The sequence number cannot be advanced beyond the location following the last test number.

The F5 and F6 keys are used for "INSERT" and "REPLACE" modes, respectively. When the insert mode is selected, any test numbers entered will be stored at the current sequence number position and any following test numbers will be advanced one position. When the replace mode is selected the test number at the current sequence number position will be replaced by the new one entered.

The F7 key is used for deleting tests from the sequence. If the F7 key is pressed, the test at the current sequence number will be deleted. If SHIFT F7 is pressed, all of the tests in the file will be deleted.

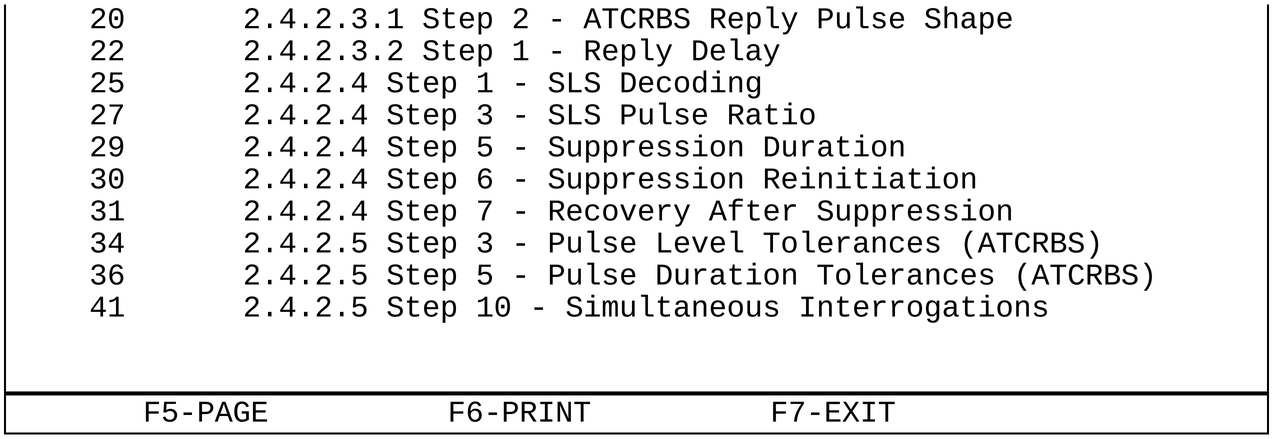
The F8 key is used to display a list of available tests. An example display is shown in figure 21.

The F9 key will exit the screen and advance to the file access screen where the file may be stored.

Figure 21 shows an example of a display of all available transponder tests. The F5 key is used to page through the list and the F6 key is used to print a copy of the list. F7 will return to the edit screen.

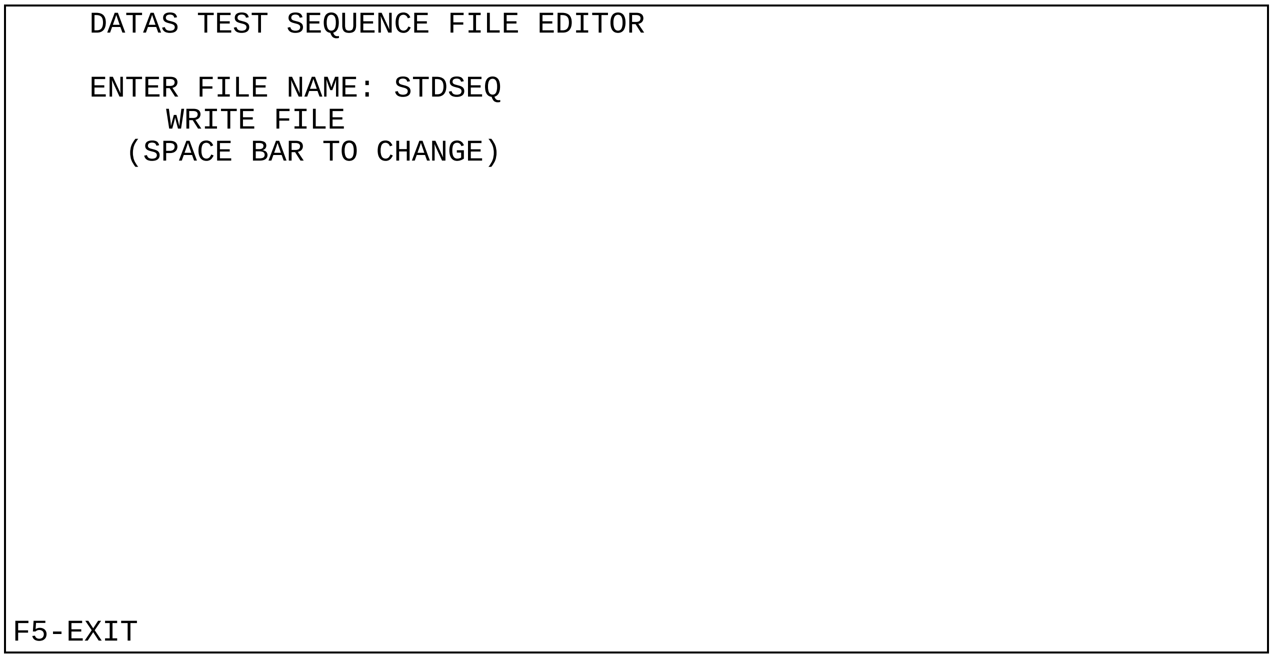
WRITING FILES. If the user wants to save the current test sequence file, he must write the file to the disk. The file access screen screen is used to write files to the disk (figure 22). He may accept the default file name or type in any valid file name followed by a carriage return. If the file does not already exist it will be written to the disk and the prompt will switch to read the file. If the file already exists, the user will be warned and will have the option to either name it something else or replace it with the current file. To exit the editor enter the F5 key. Note: If the F5 key is pressed before the file is written to the disk, the file will not be stored.

```
                     DATAS TRANSPONDER TESTS

     2      2.4.2.1 Step 2 - Sensitivity
     3      2.4.2.1 Step 3 - ATCRBS Dynamic Range
     9      2.4.2.2.1 - Reply Transmission Frequency
    10      2.4.2.2.2 Step 1 - ATCRBS Power Output
    17      2.4.2.2.5 Step 1 - Determination of reply rate limit
    19      2.4.2.3.1 Step 1 - ATCRBS Reply Pulse Spacing
```



```
    20      2.4.2.3.1 Step 2 - ATCRBS Reply Pulse Shape
    22      2.4.2.3.2 Step 1 - Reply Delay
    25      2.4.2.4 Step 1 - SLS Decoding
    27      2.4.2.4 Step 3 - SLS Pulse Ratio
    29      2.4.2.4 Step 5 - Suppression Duration
    30      2.4.2.4 Step 6 - Suppression Reinitiation
    31      2.4.2.4 Step 7 - Recovery After Suppression
    34      2.4.2.5 Step 3 - Pulse Level Tolerances (ATCRBS)
    36      2.4.2.5 Step 5 - Pulse Duration Tolerances (ATCRBS)
    41      2.4.2.5 Step 10 - Simultaneous Interrogations

       F5-PAGE          F6-PRINT          F7-EXIT
```

FIGURE 21. SEQ DISPLAY #3 TEST LIST



```
    DATAS TEST SEQUENCE FILE EDITOR

    ENTER FILE NAME: STDSEQ
        WRITE FILE
      (SPACE BAR TO CHANGE)

F5-EXIT
```

FIGURE 22. SEQ DISPLAY #1 FILE ACCESS IN WRITE MODE

BIBLIOGRAPHY:

Baker, John L., Philip N. McCabe, Kenneth V. Byram, Journal of ATC, October - December 1985.

Federal Aviation Administration, National Airspace System Plan, September 1989.

Radio Technical Commission for Aeronautics, Minimum Operational Performance Standards for Traffic Control Radar Beacon System/Mode Select (ATCRBS/Mode S) Airborne Equipment, March 1983.

APPENDIX A

ATCRBS FIELD TEST FUNCTIONS

GENERAL DESCRIPTION

The Air Traffic Control Radar Beacon System (ATCRBS) Field Executive program is the actual transponder testing program. It is a menu-driven multiscreen program that gives the user complete control over his testing environment. The major subfunctions of the program are: calibration of the coupling loss, control of individual as well as global test parameters, display of individual transponder test results summaries, control of the personal computer (PC) subsystem interface, and execution of the transponder tests.

SOURCE FILES.

INCLUDE FILES (.h). The following is a list of the include files used by the functions of the ATCRBS field executive program and a brief description of their contents:

300.include.calib.h - definitions related to system calibration. Includes the calibration file names and file structure definitions.

300.include.dataloc.h - definitions for the storage locations for the three transmit control channels.

decode.h - definitions for the hardware decoder addresses and structures to store the decoder information.

300.include.displib.h - definitions for WYSE terminal display attributes and escape sequences.

idle.h - contains the idle loop interrogation definition.

300.include.mmap.h - definitions for the memory locations of the DATAS hardware.

parminit.h - declaration and initialization of all transponder test parameters.

parmlib.h - definitions used by the parameter modification programs.

pcinc.h - definitions used for PC-68020 communications. Communication control and buffer structure definitions.

seqflib.h - definitions related to the test sequence file.

300.Include.stdlib.h - standard library provides common definitions such as TRUE, FALSE, etc.

testdef.h - function array definitions for the ATCRBS transponder tests.

tstinc.h - common transponder testing library.

SOURCE FILES (.cc) - The afexec program may be linked by running the command file afexec.cf. A cross reference between function names and file names follows:

Major functions are displayed in bold print.

Aftsts()	**aftsts.cc**	
aftst[x]()	**aftst[x].cc**	
bldmenu()	bldmenu.cc	
calread()	calread.cc	
chk_channels()	wf2chann.cc	
chk_dflt()	parmscrn.cc	
chk_level()	chklev.cc	
chk_name()	chkname.cc	
ck_format()	chkformt.cc	
ck_range()	chkrange.cc	
ck_resolu()	chkresol.cc	
clk_set()	timer.cc	
clk_sp()	timer.cc	
clk_strt()	timer.cc	
cnt_pulses()	repulse.cc	
convt_ans()	convtans.cc	
count_pulses()	wf2count.cc	
curpos()	disputil.cc	
dbfile()	aftsts.cc	
dfname()	dfname.cc	
d_fun_keys()	tstdrvr.cc	
erase()	disputil.cc	
extrct_data()	repulse.cc	
fatal()	pcinit.cc	
fatal_error()	ferror.cc	
funct[x]()	chkformt.cc	
gen_action()	wf2acti.cc	
gen_atten()	wf2atte.cc	
gen_pattern()	wf2patt.cc	
gen_pulse()	wf2puls.cc	
gen_time()	wf2time.cc	
gen_transm()	wf2transm.cc	
get_decl()	disputil.cc	
get_input()	disputil.cc	
getmtl()	getmtl.cc	
getsd()	getsd.cc	
getyorn()	disputil.cc	
hwinit() interference()	hwinit.cc	
int_err()	intfer.cc	
int_loop()	wfmod.cc	
main()	intlp.cc	
mdec_acr()	**afexec.cc**	mdecode.cc
mparms()	**mparms.cc**	
mtl_fun()	getmtl.cc	**parmscrn.cc**
parmscrn()	pcinit.cc	**plotdat.cc**
pcinit()	sumscrn.cc	sumscrn.cc

plotdat()	pmsg.cc
plotmsg()	prhead.cc
plotque()	print.cc pwthresh.cc
pmsg()	odatf.cc
pr_head()	sumdat.cc rdsfile.cc
prt_init() pw_thresh()	sfread.cc reptest.cc
odatf()	repulse.cc
rddat()	parmscrn.cc
rds_file()	setfreq.cc setidle.cc
read_seqf_file()	setlev.cc disputil.cc
rep_test()	spandd.cc disputil.cc
repulse()	storeans.cc
set_dflt()	storeans.cc
setfreq()	sapx.cc
set_idle()	**sum[x].cc**
setlev()	**sumdat.cc** **summenu.cc**
setscroll()	**sumscrn()** disputil.cc
spandd()	**tstdrvr.cc** **tstseq.cc**
stcenter() store_ans()	vfcreat2.cc vfopen.cc
store_ptr() suc_apprx()	wfmod.cc
sum[x]()	wrtdat.cc
sumdat()	
summenu()	
sumscrn()	
toupper()	
tstdrvr()	
tstseq()	
vfcreat2()	
vfopen()	
wfmod()	
wrtdat()	

Figure A-1 shows a hierarchical relationship between the more important functions within the transponder test program. These and all of the functions written for use with the transponder test programs are contained in this appendix.

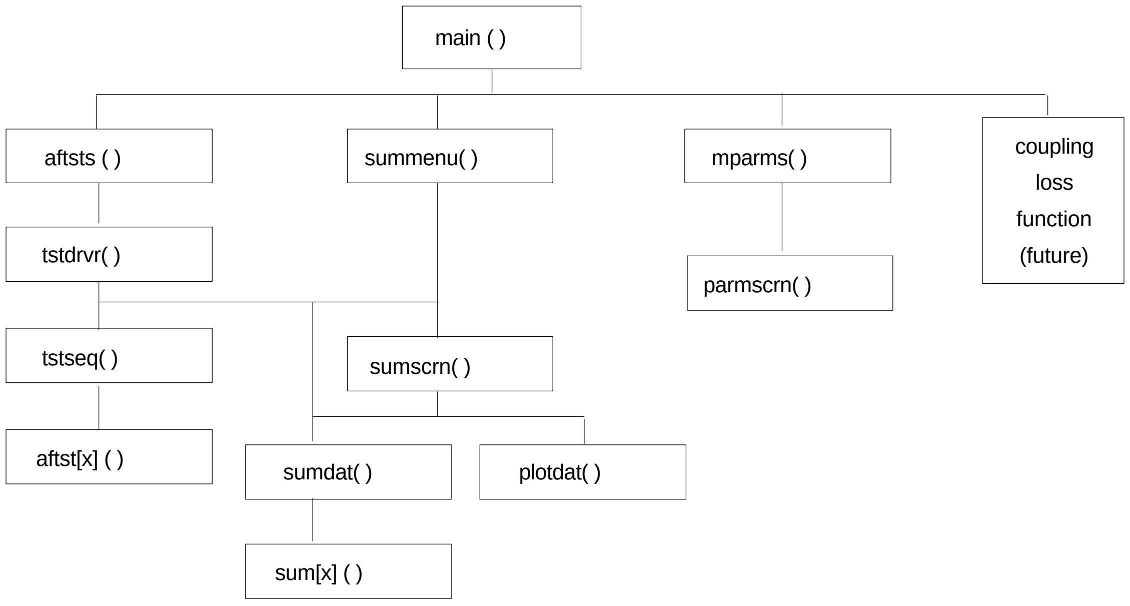


FIGURE A-1. MAJOR FUNCTION RELATIONSHIPS

<u>aftsts()</u>

NAME

aftsts - ATCRBS field testing options menu. Source file-aftsts.cc.

FUNCTION CALL

aftsts().

GLOBAL VARIABLES

The following are global variables that are used throughout the program that are declared in aftsts.cc.

int i_power - Interrogation power level.
int p_mode = 0 - Interrogation power mode, default is "MANUAL" (0), other choice is "AUTOMATIC" (1).
int ps_mode = NO - Hardcopy flag, indicates whether or not to automatically print a summary of the test results. Default is "NO" (0), other choice is "YES" (1).

int store = YES - Flag to indicate whether or not to store the test data on disk. Default is "YES" (1), other choice is "NO" (0).

DESCRIPTION

This function permits the user to select various options to control how the system will function when running transponder tests. The options included are: hardcopy of test results or not; interrogation power mode--whether it is manual or automatic adjust; initial power level; test sequence file name; whether or not to store data on disk, and if so, the database file name.

DETAILED DESCRIPTION

The function builds the display, maintains an error array (e_flg[]) that indicates error conditions in menu options, and calls the function dbfile() to open data and key files. If there is an error condition for a prompt, there is an indication next to the prompt. When the user selects the prompt with the error condition the error is explained at the bottom of the screen. The program then waits for input from the user. If F5 is entered the data files and key files are closed and the program returns. If F6 is entered the program advances to the test function. Arrow keys will select between options. ASCII characters are accepted as input as file names or switches for flags.

<u>aftst*()</u>

NAME

aftst* - ATCRBS field test number *. Source file-aftst*.cc.

FUNCTION CALL

```
aftst*(t_data, best_power, best_delay, p_data_loc)
struct data_share *t_data;   /* Test data        */
int *best_power;             /* Best power measured    */
int *best_delay;             /* Best delay measured    */
struct data_destination p_data_loc; /* Hardware addrs. */
```

GLOBAL VARIABLES

The following variables must be declared external to most ATCRBS field test functions:

<u>struct INT_TYPE int_cur[]</u> - Current interrogation parameters.
<u>struct TST_TYPE tst_cur</u> - Current test parameters.
<u>struct CAL_TYPE calib</u> - Calibration tables.
<u>int i_afloss</u> - Airframe loss.
<u>int i_Power</u> - Default interrogation power.
<u>int_tx_Port</u> - Transmitter port in use.

DESCRIPTION

This represents the calling structure of a typical ATCRBS field test function. The ATCRBS tests are described in detail in appendix C. The test results are stored in the structure "t_data." The highest reply power measured during the test are returned in the variable "best_power." The lowest reply delay is returned in the variable "best_delay." "p_data_loc" provides the addresses of the hardware locations to store the interrogation waveforms.

bldmenu()

NAME

bldmenu - Build menu for terminal screen. Source file-bldmenu.cc.

FUNCTION CALL

```
    bldmenu(parm_io,  prmt_ord,  prmt_flg,  strc_indx,  prmt_cnt,
    prm_ques, spaceb)
    struct PARM_IO_TYPE *parm_io; /* Parameter IO structs  */
    int prmt_ord[];            /* Array of prompt order */
    int prmt_flg[];            /* Prompt flags       */
    int strc_indx;                 /* Index of struct    */
    int prmt_cnt;                  /* Number of prompts       */
    char *prm_ques[]           /* Prompt questions   */
    char *spaceb[];                /* Spacebar answers   */
```

DESCRIPTION

This function is used to set the order of the test or interrogation parameters for display on the screen. The parameters are displayed as question and answer and they are positioned in two columns on the screen when possible. The main value of this function is to accommodate parameters that require more than half of the screen and, therefore, must be displayed on a screen row by themselves.

DETAILED DESCRIPTION

The variable "prmt_cnt" is used to determine how many prompts to position on the screen. The pointer "*parm_io" is the location of the actual prompts. The array "prmt_ord[]" will contain the order number to display each prompt, a -1 indicates an unused position. The array "prmt_flg[]" will contain "TRUE" for all prompts that are used and "FALSE" for those that aren't. For each prompt the question and answer lengths are combined to determine the overall length. If the length is greater than half the screen it must be positioned on the left side, and the adjacent right side must be made unavailable. The function will return a success condition (0) if all prompts can be displayed, or a fail condition (1) if there is not room for all prompts on the screen.

calread()

NAME

calread - Calibration file read function. Source file-calread.cc.

FUNCTION CALL

```
     calread(calib, mode)
     struct CAL_TYPE *calib;  /*  Pointer to calibration
                                  parameters */
     int mode;                    /* Read mode */
```

DESCRIPTION
    This function opens and reads the calibration file that is selected by the variable "mode." The include file "calib.h" must be included for #defines for the mode as well as file names and calibration structures. The possible modes are: MTF, CTF, MCF, and CCF for Master Table File, Current Table File, Master Coupling File, and Current Coupling File.

chk_channels()
NAME
    chk channels - Check channels. Source file - wf2chan.cc.

FUNCTION CALL
```
    chk_channels(p_wfm_def, p_maxl, p_max2, p_max3)
    struct wave_parm *p_wfm_def;  /* Waveform struct  */
    int *p_maxl;                  /* Locations used ch. 1  */
    int *p_max2;                  /* Locations used ch. 2  */
    int *p_max3;                  /* Locations used ch. 3  */
```

DESCRIPTION
This function determines the number of waveform events used in each of the three channels. The end of a series of events must be flagged in the waveform structure with a zero stored in the channel member.

chk_level()
NAME
    chk_level - Check for valid interrogation power level.
Source file - chklev.cc.

FUNCTION CALL
```
    chk level(p_level, port, atten)
    int p_ievel,             /* Power level           */
    int port;                /* Interrogation port (0-2)   */
    int atten;               /* Attenuator selected (0-2)  */
```

DESCRIPTION
    This function is used to check if a desired power level can be achieved. The function will subtract coupling loss and airframe loss from the desired interrogation level and test if the transmitter can maintain that level at the calibration point. If the level is too high "P_TOO_HI" is returned, if it is too low "P_TOO_LO" is returned, otherwise "SUCCESS" is returned. The ;n~ de files "calib.h" and "stdlib.h" are used in this function.

chk_name()
NAME
    hk name - Check file name. Source file chkname.cc.

FUNCTION CALL
```
     chk name(filename, count)
     char *filename;  /* Filename to check  */
     int count;           /* Number of characters */
```

DESCRIPTION

Tests a character string to see if it is a valid VERSAdos file name.

DETAILED DESCRIPTION

The purpose of this function is to test for a valid filename before it is combined with user number and catalog, etc., for file open. The name is stored in the character string "filename" and the length of the string is passed in the variable "count." The first character is tested to be an alphabetic character and the following characters are tested to be either alphabetic or numeric. If either of these conditions are violated, a "1" is returned to indicate illegal characters. If count is greater than eight, a "2" is returned to indicate the file name is too long. If the file name is valid, the function returns a "0."

ck_format()
NAME

ck_format - Check input format. Source file - chkformt.cc.

FUNCTION CALL
```
     ck_format(p in_data, parm_io)
     char *p_in_data;             /* Pointer to input data */
     struct PARM_IO_TYPE *parm_io; /* Parameter IO structure*/
```

DESCRIPTION

This function will test a character string to see if it is in a predefined format. The character string is passed in the variable "p_in_data" and the format is defined in the structure "parm_io." There are seven possible formats to test, which are #defined in tstlib.h. If the string is in the correct format, a success condition is returned (0); if it is in an incorrect format, an error condition which is defined in "parmlib.h" is returned. The possible formats are: decimal, binary, octal, hexadecimal, character, real, and undefined (for expansion purposes).

ck_range()
NAME

ck_range - Check that a test parameter is within a valid range. Source file - chkrange.cc.

FUNCTION CALL
```
    k_range(parm_io, temp_ans)
    struct PARM_IO_TYPE *parm_io; /* Parameter IO structure*/
    long temp_ans;                /* Parameter value       */
```

DESCRIPTION

This function will test that a long variable "temp_ans" is within its valid range. This is used in conjunction with the test parameter modification routines. The range of the variable is defined in the structure "parm_io" (parm_io->min and parm_io->max).

ck_resolu()
NAME
ck_resolu - Check that a test parameter is at a valid resolution value. Source file - chkresol.cc.

FUNCTION CALL
```
    ck_resolu(parm_io, temp_ans)
    struct PARM_IO_TYPE *parm_io; /* Parameter IO structure*/   lonq
temp ans;                /* Parameter value       */
```

DESCRIPTION
This function will test that a long variable "temp_ans" is at the proper resolution. This is used in conjunction with the test parameter modification routines. The resolution of the variable is defined in the structure "parm_io" (parm_io->res). The function uses the MOD function to test for the proper resolution.

clk_set()
NAME
clk set - Clock set function. Source file - timer.cc.

FUNCTION CALL
```
    clk_set(nm_ticks)
    long nm ticks;          /* Number of ticks for interrupt*/
```

DESCRIPTION
Utilizes on board timer. Timer rate is based on 2 megahertz (MHz) clock rate. "nm_ticks" is the number of microseconds between interrupts.

clk_sp()
NAME
clk sp - Clock stop function. Source file - timer.cc.

FUNCTION CALL
```
     clk sp()
```

DESCRIPTION
Stops the on board timer and disables interrupts.

clk_strt()
NAME
clk strt - Clock start function. Source file - timer.cc.

FUNCTION CALL
```
     clk_strt()
```

DESCRIPTION
Starts the on board timer and enables interrupts. The timer interrupts occur at the rate established via call to "clk_set."

__**cnt_pulses()**__
NAME
cnt_pulses - Count reply pulses. Source file - repulse.cc.

FUNCTION CALL
```
     cnt_pulses()
```

DESCRIPTION
This function will return the number of reply pulses indicated by the raw status word.

__**convt_ans()**__
NAME
convt_ans - Convert and display a test parameter. Source file - convtans.cc.

FUNCTION CALL
```
     convt_ans(parm_io, spaceb)
     struct PARM_IO_TYPE *parm_io; /* Parameter IO structure*/
     char *spaceb[];               /* Space bar answers     */
```

DESCRIPTION
This function will display a parameter which is stored in the parameter structure (*parm_io->p_cur) as a character string in its proper format on the display screen.

__**count pulses()**__
NAME
count_pulses - Count interrogation pulses (events). Source file - wf2coun.cc.

FUNCTION CALL
```
    count_pulses(p_wfm_def, p_countl, p_count2, p_count3)
    struct wave_parm *p_wfm_def; /* Waveform struct*/
    int *p_countl;                /* Count for ch. 1   */
    int *p_count2;                /* Count for ch. 2   */
    int *p_count3;                /* Count for ch. 3   */
```

DESCRIPTION
This function will count the number of pulse lead-edges and trail-edges for each of the three channels. The end of a channel is determined when the channel member contains a zero.

__**curpos()**__
NAME
curpos - Cursor position. Source file - curpos.cc.

FUNCTION CALL

```
     curpos(ro, co)
     int ro;                  /* Screen row        */
     int co:                  /* Screen column */
```

DESCRIPTION

This function will position the cursor at the specified row and column on a wyse 75 compatible terminal.

<u>dbfile()</u>

NAME

dbfile - Data file access function. Source file - aftsts.cc.

FUNCTION CALL

```
    dbfile(cur_ques, df_lun, kf_lun, dfd, kfd)
    int cur_ques;             /* Current question number*/
    char *df_lun;             /* Data file LUN      */
    char *kf_lun;             /* Key file LUN       */
    int *dfd,                     /* Data file descriptor */
    int *kfd:                     /* Key file descriptor  */
```

DESCRIPTION

This function provides access to the data and key files, and displays the status of these files to the user at the data file extension prompt on the screen displayed by aftsts(). This function was designed to work with the function aftsts().

DETAILED DESCRIPTION

Displays the answer to the store option (YES or NO). If the current question is the store option, the answer will be in reverse video. The data file extension prompt and answer is then erased. If the store option is YES, the prompt and answer are redisplayed. The file is opened with a call to odatf(). The file status will be displayed next to the file extension prompt. The file status possibilities are: the file already exists, the file is new, or there was an error in the file open.

<u>dfname()</u>

NAME

dfname - Create data file name. Source file - dfname.cc.

FUNCTION CALL

```
    dfname(f_str, f_type, f_date, s_loc_id)
    char *f_str;          /* File name string       */
    int f_type;               /* File type         */
    char *f_date;             /* Date                    */
    char *s_loc_id;           /* Location Identifier    */
```

DESCRIPTION

This function will create a file name from the location identifier and the date for either the data file or the key file depending on the "f_type" variable (f_type=1 - data file, else key file). The location identifier is used as the catalog and the date is part of the file name. Formats: "(s_loc_id).D(f_date)." for data file and "(s_loc_id).K(f_date)." for key file.

<u>**d_fun keys()**</u>
NAME
d fun keys - Define function keys. Source file - tstdrvr.cc.

FUNCTION CALL
```
     d_fun_keys(sys_status)
     char sys_status;         /* System status    */
```

DESCRIPTION
Will display the specific function keys and their action for the "tstdrvr" function.

<u>**erase()**</u>
NAME
erase - Erase characters on the display. Source file- erase.cc.

FUNCTION CALL
```
    erase(n_chars)
    int n_chars;             /* Number of characters to erase*/
```

DESCRIPTION
Will erase n_chars characters starting at the current cursor location. If n_chars is 0, will erase entire line from the cursor.

<u>**extrct data()**</u>
NAME
extrct data - Extract reply data. Source file - repulse.cc.

FUNCTION CALL
```
     extrct_data(pulse, p_addr)
     struct RAW_PULSE *pulse;     /* Reply pulse data */
     unsiqned long *p_addr;           /* Hardware address */
```

DESCRIPTION
This function is used to separate the various reply data for the reply pulse at the address indicated by "p_addr." See the function "repulse()" for more information.

<u>**fatal_error()**</u>
NAME
fatal error - Fatal error message. Source file - ferror.cc.

FUNCTION CALL
```
    fatal error(err num, func_name)
    int err_num;        /* Error message number       */
    char *func_name;    /* Name of function error occurred */
```

DESCRIPTION

Will display a message on the screen describing an unrecoverable error that occurred in a function.

qen_action()

NAME

gen_action - Generate waveform action field. Source file-wf2acti.cc.

FUNCTION CALL

```
    gen_action(p_wfm_def, p_data_pnt)
    struct wave_parm *p_wfm_def;  /* Waveform struct  */
    unsiqned long p_data_pnt;     /* Hardware address */
```

DESCRIPTION

This function will load the action field in its proper binary format into the hardware waveform definition locations. Action fields consist of items such as DELAY, PAM, PAM with SYNC PHASE etc.

gen_atten()

NAME

gen_atten - Generate waveform attenuator selection. Source file - wf2atte.cc.

FUNCTION CALL

```
    gen_atten(p_wfm_def, p_data_pnt)
    struct wave_parm *p_wfm_def;  /* Waveform struct  */
    unsiqned long p data_pnt;     /* Hardware address */
```

DESCRIPTION

This function will load the attenuator selection into the proper hardware waveform definition locations.

gen_pattern()

NAME

gen_pattern - Generate waveform pattern. Source file-wf2patt.cc.

FUNCTION CALL

```
     gen_pattern(p_wfm_def, p_data_loc, mode_s_offset)
     struct wave_parm *p_wfm_def;  /* Waveform struct  */
      struct data_destination *p_data_loc;    /* Hardware locs.*/
     unsigned long mode_s_offset;  /* Data block offset     */
```

DESCRIPTION

This is the high level function for generating interrogation waveforms in binary format from waveform structures.

DETAILED DESCRIPTION

This function first determines the number of pulse events and stores this in "p_data_loc->num_of_pulses." This can be used later by functions that need to modify the interrogation as it is stored in hardware. For each of the channels, the waveform information is encoded into its correct binary format and stored in the hardware addresses determined by "p_data_loc." The variable "mode_s_offset" is used to properly position the Mode S data block with respect to the P6 pulse.

gen_pulse()

NAME

gen_pulse - Generate pulse fields. Source file-
wf2puls.cc.

FUNCTION CALL

```
    gen_pulse(p_wfm_def, p_data_pnt)
    struct wave_parm *p_wfm_def; /* Waveform struct*/
    unsigned long *p_data_pnt; /* Hardware address */
```

DESCRIPTION

This function encodes the pulse information (lead-edge, trail-edge, etc.) into the proper binary format required by the hardware. The pulse information is stored in the waveform structure "p_wfm_def."

gen_time()

NAME

gen time - Generate time field. Source file - wf2time.cc.

FUNCTION CALL

```
    gen_time(p_wfm_def,   p_data_pnt,   p_strt_s_blk,
    cal_blk_offset, p error)
    struct wave_parm *p_wfm_def;  /* Waveform struct*/
    unsigned long *p_data_pnt;      /* Hardware address      */
    unsigned int *p_strt_s_blk;     /* Start of Mode S block */
    unsigned long cal_blk_offset; /* Data block offset     */
    unsigned int *p_error;          /* Position error          */
```

DESCRIPTION

This function will encode the time delay values contained in the waveform structure into the binary format used by the hardware. It will automatically use more than one hardware address if the time delay requires more than the maximum value provided by one location.

gen_transm()

NAME

gen_transm - Generate transmit field. Source file-
wf2tran.cc.

FUNCTION CALL

```
    gen_transm(p_wfm_def, p_data_pnt)
    struct wave_parm *p_wfm_def;  /* Waveform struct  */
    unsigned long *p_data_pnt;    /* Hardware address */
```

DESCRIPTION
This function encodes the transmit information into the proper format required by the hardware.

get_decl()

NAME
get_decl - Get decimal numbers as input. Source file-getdecl.cc.

FUNCTION CALL
```
    get decl(p_data, row, col, max_in)
    char *p_data;        /* Pointer to input data  */
    int row;         /* Current row position       */
    int col;         /* Current column position    */
    int max_in;          /* Max digits allowed     */
```

DESCRIPTION
Accepts only numeric characters as input from keyboard.

DETAILED DESCRIPTION
This function is intended to be called only after the first numeric character has been read in. Its purpose is to accept only numeric characters (0 - 9) until a carriage return is entered, arrow keys are entered, or the buffer is full. If any other characters are entered they will be ignored except the backspace and delete keys, which are recognized and will erase characters that were entered. The input characters are stored in the array p_data up to the maximum input length indicated in max_in. Row and Col are used to position the echoed characters on the screen. This is a modified version of get_dec(), it was updated to exit if an arrow key is entered.

get_input()

NAME
get_input() - Get keyboard input. Source file - getinl.cc

FUNCTION CALL
```
    get input(p_data, count, row, col, mx_in, e_len, opt)
    char *p_data;            /* Pointer to input data      */
    int *count;              /* # characters read in           */
    int row;             /* Current row position           */
    int col;             /* Current column position        */
    int mx_in;               /* Max character input allowed    */
    int e_len;               /* # characters to erase      */
    int opt;             /* Option, 0 exit, 1 pause max    */
```

DESCRIPTION
This is a general purpose function that filters only printable characters from keyboard input and echoes them on screen and stores them in a fixed length buffer.

DETAILED DESCRIPTION

This function is intended to be called only after the first character has been read in. Its purpose is to test this character and test and accept any following characters until the buffer is full or a carriage return is entered. If any invalid characters are entered (not between ascii 0x20 and 0x7f) the function returns a "1" as an error condition. If all characters are valid and a carriage return is entered, the function will return a "0" as a success condition. The backspace and delete keys are recognized and will erase characters that were entered. The input characters are stored in the array p_data up to the maximum input length indicated in mx_in. Row and Col are used to position the echoed characters on the screen. Count will contain the number of characters read in.

This is a modified version of get_input, it has the addition of "e_len" which determines how many characters to erase before echoing input, and "opt" which determines what to do if the maximum input is exceeded. If opt is "0" the program will exit, if opt is "1" it will pause.

getmtl()

NAME

getmtl - Acquire minimum transmit level (MTL) of transponder. Source file - getmtl.cc.

FUNCTION CALL

```
    getmtl(mtl, p_data_loc)
    int *mtl;           /* Minimum transmit level          */
    struct data_destination *p_data_loc;    / *    Hardware
                                                locations */
```

DESCRIPTION

This function will determine the minimum RF signal level required to produce a 90 percent reply efficiency. The measurement is made with a standard Mode A interrogation using a successive approximation algorithm.

DETAILED DESCRIPTION

This function requires the include files: tstlib.h, stdlib.h mmap.h, and calib.h. It requires the external variables: struct CAL_TYPE calib for access to the calibration tables, int i_afloss for airframe loss, and tx_port for the current transmitter port in use. It uses a standard Mode A interrogation with the reply window adjusted for coupling time. The PRF used is 500.

A subfunction called mtl_fun() is included in the function to do the actual interrogation algorithm. This function initially sets the interrogation power to be -88 dBm, the power variation to 8 dBm, and the number of interrogations to be 20. It will continuously interrogate with a call to the function "suc_apprx()" until the variation in power is down to 1 dB or the interrogation power is out of range. If the transponder replies at least 90 percent, the interrogation power is decreased by the power variation, if less than 90 percent, the power is increased by the power variation. Once the transponder replies at least 90 percent to a series of interrogations, the number of interrogations will be doubled for the next run and the variation will be divided in half.

The measured MTL is returned in the variable "mtl" in dB * 10 .
* Potential improvement - The function returns the MTL in a pointer variable, but it should also return a status to indicate if MTL was found. i.e., if (was_90) return(SUCCESS).

<u>**getsd()**</u>

NAME

getsd - Acquire suppression duration of transponder. Source file - getsd.cc.

FUNCTION CALL

```
getsd(sd, mode, p_data_loc)
long *sd; /* Suppression duration */
int mode;           /* Interrogation mode, 0=A 1=C     */
struct data destination *p_data_loc;    / *    Hardware
                                       locations */
```

DESCRIPTION

This function will determine the suppression duration of the transponder. Suppression duration is defined as the length of time the transponder is suppressed following a P1-P2 suppression pair. The measurement is made with a standard P1-P2 pair followed by a standard interrogation. The interrogation mode is determined by the variable "mode." If "mode" is a "0," Mode A is used, if it is a "1," Mode C is used. The function varies the spacing between the Pl-P2 pair and the interrogation using a successive approximation algorithm to find the 10 percent reply point.

DETAILED DESCRIPTION

This function requires the include files: tstlib.h, stdlib.h, decode.h, mmap.h, and calib.h. It requires the external variables: struct CAL_TYPE calib for access to the calibration tables, int i_afloss for airframe loss, i_power for standard interrogation power, and tx_port for the current transmitter port in use. It stores the interrogation with the reply window adjusted for coupling time in channel 1 and the P1-P2 pair in channel 2. The PRF used is 450 and the frequency is 1030 MHz.

The function sets the interrogation power for both channels the same as the standard interrogation level in the global variable i_power. If it is unable to set the level of both channels the same, it will abort. The initial spacing between the P1-P2 pair and the interrogation is set to 50 microseconds, the spacing variation to 10 microseconds, and the number of interrogations to be 10. It will continuously interrogate with a call to the function "suc_apprx()" until the variation in spacing is down to 25 nanoseconds or the pulse spacing is less than the spacing variation. If the transponder replies at least 10 percent, the spacing is decreased by the spacing variation, if less than 10 percent, the spacing is increased by the spacing variation. Once the transponder replies less than 10 percent to a series of interrogations, the number of interrogations will be doubled for the next run and the variation will be divided in half.

The measured suppression duration is returned in the variable "sd" in nanoseconds.

getyorn ( )

NAME

getyorn - get "y" or "n" from the keyboard. Source file-getyorn.cc.

FUNCTION CALL

```
    getyorn();
```

DESCRIPTION

Waits indefinitely for a "y" or "n" from the keyboard. Lower case and upper case are allowed. Returns a "1" if a "y" is entered, returns a "0" if "n" is entered.

hwinit()

NAME

hwinit - Hardware initialization. Source file - hwinit.cc.

FUNCTION CALL

```
    hwinit()
```

DESCRIPTION

This function initializes the important DATAS hardware locations. It will reset the system clock, RF control, local oscillator is set to 1030 MHz, the interrupt control is set to 0, the transmit controller is set to stop all channels, the external output controls are set, the threshold is set to minimum, and all decoder variables are set.

interference()

NAME

interference - interference detection function. Source - intfer. cc.

FUNCTION CALL

This function has not yet been developed.

int_err()

NAME

int_err - interrogation setup error message. Source file-wfmod.cc.

FUNCTION CALL

```
     int_err(e_code, msg)
     int e_code;          /* Error code        */
     char *msg;               /* Message string    */
```

DESCRIPTION

This function will decode and form an error message describing the error code returned from the function "wfmod()."

int_loop()

NAME

int_loop - Interroqation loop algorithm. Source file-intlp.cc.

FUNCTION CALL

```
    int_loop(num_ints, reply, xmit_chan)
    int num_ints;                /* Number of interrogations*/
    struct reply_format *reply;  /* Reply information struct*/
    int xmit chan;               /* Transmit channels     */
```

GLOBAL VARIABLES

The following variables must be declared external to this function:

```
     struct CAL_TYPE calib;          /* Calibration tables  */
     int i_afloss;                   /* Airframe loss       */
     int tx_port;                /* Current transmit port */
```

DESCRIPTION

This function is used as a general purpose transponder interrogation algorithm. It is used by the transponder tests that require only a percent reply or some other general reply information. The function will send the number of interrogations defined in "num_ints," run the transmit channels defined in "xmit_chan," and store the reply information in the struct "reply."

DETAILED DESCRIPTION

The reply information supplied by the function is contained in a data structure. The structure format of the "reply" struct is defined in tstlib.h and provides the following information: reply->power.min - The lowest reply power of the **last reply pulse** (F2) measured out of all replies. Is stored in dB's * 10. All calibration offsets are added including airframe loss and receiver coupling loss.

reply->power.mean - The mean reply power of the **last reply pulse** (F2) measured in all replies. Is stored in dB's * 10. All calibration offsets are added including airframe loss and receiver coupling loss.

reply->power.max - The highest reply power of the **last reply pulse (F2)** measured out of all replies. Is stored in dB's * 10. All calibration offsets are added including airframe loss and receiver coupling loss.

reply->frequency.min - The lowest RF transmitter frequency of the first reply pulse measured out of all replies. It is stored in kHz.

reply->frequencv.mean - The mean RF transmitter frequency of the first reply pulse measured out of all replies. It is stored in kHz.

reply->frequency.max - The highest RF transmitter frequency of the first reply pulse measured out of all replies. It is stored in kHz.

reply->delay.min - The least reply delay measured out of all replies. It is stored in nanoseconds. This is the delay from the lead edge of the reply window. It assumes that the reply window has been adjusted for coupling time.

reply->delay.mean - The mean reply delay measured out of all replies. It is stored in nanoseconds. This is the delay from the lead edge of the reply window. It assumes that the reply window has been adjusted for coupling time.

reply->delay.max - The maximum reply delay measured out of

all replies. It is stored in nanoseconds. This is the delay from the lead edge of the reply window. It assumes that the reply window has been adjusted for coupling time.
reply->atcrbs_code - ATCRBS identification code (4096). This code comes from the system reply decoder and is in whatever format the decoder provides. In order for this variable to be stored it must be the same for at least five consecutive replies. If the code changes during the interrogation loop, this variable will contain whatever the code was for the latest five consecutive replies.
reply->stdev_pls_cnt - Standard deviation of reply pulses. This is provided so that reply reliability can be determined.
reply->mean_pulse_count - Mean number of reply pulses per interroqation.
reply->intfere_ratio - Ratio of replies that were rejected because of interference to accepted replies. This is provided to indicate data reliability.
reply->reply_cnt - Reply count. The number of replies detected.

The variable "xmit_chans" determines from which channels to transmit. Each bit position corresponds to a channel; a "1" means to transmit and a "0" means not to transmit. There are currently three channels in the system, meaning a total of three active bits (e.g., 0x0005 means to transmit on channels 1 and 3). The timing must be set for the channels prior to calling this function.
This function will start the system clock, but the PRF must be set prior.
A status (integer) will be returned when the function terminates. The following are return possibilities which are defined in tstlib.h: T_ABORT (-1), PRF_ERROR (2), or T_PASS (0). A status of T ABORT will be returned if the acceptable level of interference is exceeded. PRF_ERROR will be returned if the system timer interrupt is already set when it is first read at any time during the loop. T_PASS will be returned in all other cases.

main()
NAME
main - Transponder field testing main function. Source file - afexec.cc.

FUNCTION CALL
main()

GLOBAL VARIABLES
The following is a list of global variables that are used throughout the afexec program. They are declared in afexec.cc.

char asq_buf[l00] - Asynchronous service queue buffer.
char date[9] - Current date.
char dat file[STR_LEN] - Data file name. STR_LEN is defined in tstlib.h (24).
char df_ext[EXT_LEN+l] = (**"AA")** Data file extension. EXT_LEN is defined in tstlib.h (2).
char f_date[9] - Current date in file format.

char key_file[STR_LEN] - Key file name.
char loc_id[LID_LEN + l] = ("UACY") - Location identifier. LID LEN is defined in tstlib.h (4).
char seq_file[STR_LEN] = {"STDSEQ"} - Test sequence file name.
char tsknam[4] = ("AFEX"} - Task name used for asynchronous service queues.
char util_buf[REP_BUF_LEN] - "REP_BUF_LEN" is defined in tstlib.h (5000). This is a utility buffer intended to be used as a general purpose array.
int hardcopy - Hardcopy flag.
int i_afloss - Airframe loss.
int pc_active - PC system active flag.
int pc_store - Flag to indicate if data should be stored in PC .
int plot - Plot mode flag.
int test_array[T_LIMIT] - Test array from test sequence file. T_LIMIT is defined in tstlib.h (100).
int tx port = ANTENNA - Current transmission port in use. ANTENNA is defined in calib.h.
long pc_buf_loc[BUF_CNT] - Array of PC buffer addresses. BUF_CNT is defined in pcinc.h (10).
struct CAL_TYPE calib - Calibration tables.
struct seqf_head seq_data - Test sequence file header.

parminit.h

Parminit.h is one of the most important files in the afexec program. It contains definition and initialization of the global test parameters, interrogation parameters, and structures that support these parameters for all of the transponder tests. This file is included in main() (afexec.cc), and any modifications to these parameters, or any time a test is added, parminit.h must be updated and afexec must be recompiled.

The major components of this include file are the current and default test parameters, and the current and default interrogation parameters. The test parameters are variables that control how the test will operate (i.e., number of interrogations to run, etc.). Two identical structures maintain the default test parameters, which can only be changed with a program modification, and the current test parameters, which are equated to the default values but may be changed by the user at run time. The interrogation parameters control the waveforms for the interrogations that are common to all tests (i.e. Pl width, P1 to P3 spacing, etc.). There is a default structure that can only be modified with a program change that stores the initial value for all parameters. There is a current structure array (one element for each test) that the user may change at run time. The current parameters are stored in an array so that all tests may be modified with one change if so desired.

These structs and their supporting structures that are defined in parminit..h are defined below:

struct INT_TYPE int_cur[MAX_NM_TESTS]
struct INT_TYPE int_def - These are the interrogation parameter structures for the current and default interrogation parameters respectively. The structure format INT_TYPE is defined in tstlib.h and has the following format:

```
#define NUM_PARMS 18
struct INT_TYPE
```

```
    {
        long parm[NUM_PARMS];
    } ;
```
There are 18 parameters for each test. The current parameters are stored in an array with enough elements for the number of existing tests, "MAX_NM_TESTS" which is also defined in tstlib.h. At program startup, the current interrogation parameter array is filled with the values of the default interrogation for each member of the array.

struct TST_TYPE tst_cur

struct TST_TYPE tst_def - These are the test parameter structures for the current and default test parameters respectively. The structure format TST_TYPE is defined in tstlib.h and has the following format:

```
    struct TST_TYPE
    {
        struct T2_TYPE   tst2;
        struct T3_TYPE   tst3;
            -
            -
        struct TN TYPE   tstN;
    };
```
There is a substructure corresponding to each existing test.These substructures are also defined in tstlib.h.

There exists a structure that governs the users ability to modify the structures at run time. The purpose of this is to check the validity of a parameter value that is entered by the user and to contain a pointer to the parameter value. This structure definition is as follows:

```
    struct PARM_IO_TYPE int_io[]
    struct PARM_IO_TYPE tst_io[]
```
The structure format PARM_IO_TYPE is defined in tstlib.h and has the following format:

```
    struct PARM_IO_TYPE
    {
        long *p_cur;            /* Current value */
        long *p_def;            /* Default value */
        long min;                   /* Minimum value */
        long max;                   /* Maximum value */
        long res;                   /* Resolution of value*/
        int max_inp_char;       /* Max input of chars */
        int variabl_type;       /* Variable type code */
        int input_format;       /* Input format code  */
        int ques_index;             /* Index to question  */
    };
```
The interrogation parameter IO array (int_io[]) contains one structure for each of the 18 parameters. Because there is an array of these structures, the pointer to the interrogation for a particular test can be loaded at run time. The test parameter IO array (tst_io[]) contains one structure for each test parameter that exists. The pointer to each parameter is stored in parminit.h because they could not be found at run time. That is the reason for the extensive initialization in parminit.h

Since the current test parameters may be modified by the user at run time, the program has to be able to locate each parameter based on the test number alone. To accomplish this, a lookup table is also defined and initialized in parminit.h. The lookup table has the following format:

```
    int p_lkup[MAX_NM_TESTS][2] = {( 0,0 },
    ( 0,-1 },( 0,2 },....... };
```

For each test there is two integer values; the first is the index into the io_struct location and the second is the number of parameters for that test.

DESCRIPTION

Initializes the system, reads in the calibration files, and generates the first display screen. The menu selections for this screen are: Location ID entry, Calibrate Coupling Loss, Run Transponder Tests, Modify Test Interrogation Parameters, Run Transponder Test Summaries, PC System Activation.

DETAILED DESCRIPTION

Calls hwinit() to set system hardware to starting point. Calls prt_init() to initialize printer device. Allocates an asynchronous service queue for event timing purposes. Acquires system date and time. Calls calread() to read in calibration files. Builds the display screen. If the PC system active flag is true, will attempt to initialize the PC system by calling pc_init(). The function will then wait for input from the keyboard. The F5 key will exit the program, arrow keys will select different menu options (the current menu option is displayed in reverse video), a carriage return will execute the current menu option, and ASCII characters are accepted as input for the location ID and to switch the PC option answers.

When a menu option is selected with a carriage return, the data and key file names are created using the location ID and the system date via a call to the function dfname().

The PC active flag can be switched using the space bar or any other key. When it is switched on (YES), three additional prompts will appear to allow the user to select what the PC system will do with the data. The options include whether or not to display on the PC screen, whether or not to produce a hardcopy, and whether or not to store the data in the PC system. The global variables that contain this information are: plot, hardcopy, and pc_store, respectively. The global variable for the overall PC system is pc_active. Each of these variables contain a "l" when active or a "0" when inactive accept for the variable plot, which contains a "0" for inactive, a "1" for active plot after the test is complete, and a "2" for active on-line with the test.

<u>mdec_acr()</u>

NAME

mdec_acr - Multiple decode of ATCRBS replies. Source file-mdecode.cc.

FUNCTION CALL

```
    mdec_acr(p_a_status, p_a_data, max_dec, max_win)
    struct MA_REP_STAT *p_a_status;    /* Pointer to ATCRBS
                                        reply status struct */
    struct MA_REP_DATA *p_a_data;      /*  Pointer  to  ATCRBS
                                           reply data struct    */
    int max_dec;        /* # of decode data structs available*/
    int max_win;        /* # of reply windows allowed        */
```

DESCRIPTION

This function will process the decoded reply data from the hardware memory area. The decoded reply status will be stored in the struct "p_a_status"; the decoded reply data will be stored in the array of structs "p_a_data." The number of data structs available is passed in the variable "max_dec." The function will return three possible status conditions, "NO_REPLY" (1) if the decoder indicates there was no reply, "DEC_ERROR" (-1) if there is an error in the decoder data, or a "0" if there was a successful decode.

DETAILED DESCRIPTION

If the number of decodes indicated in hardware is "0," a no reply status is returned. If the decode error word in hardware indicates a fatal decode error, then a decode error status is returned. Otherwise the decode information is stored in a status structure and a reply data structure.

Relevant information regarding the decode data is stored in a status structure called "p_a_status." The structure is of the type "MA_REP_STAT" which is defined in decode.h. The following information is stored in the decode status struct:

p_a_status->nm_replies - The total number of ATCRBS replies decoded in all reply windows. Initialized to "0," this variable is incremented with each occurrence of a time of arrival (TOA) word where the decode type was ATCRBS.

p_a_status->nm_windows - The number of reply windows. Initialized to 1, this variable is incremented with each occurrence of an interim status word. It is assumed that there must have been at least one reply window.

p_a_status->rep_w_map - This is a pointer variable that contains the location of an array which will store the number of replies in each window. The contents at the pointer is incremented with each occurrence of a TOA and the pointer itself is incremented with each occurrence of a interim status word.

p_a_status->d_error - Decode errors. This variable contains the decode errors provided by the hardware status as well as errors that occurred when checking through the decoded data. Each bit in the word corresponds to an error. The following are errors from the hardware errors at location EF824024:

```
        Bit  0 - Decode 1 counter overflow
             1 - Decode 2 counter overflow
             2 - Decode 3 counter overflow
             3 - Decode 4 counter overflow
             4 - Decode 5 counter overflow
             5 - Decode 6 counter overflow
             6 - Decode 7 counter overflow
             7 - Decode 8 counter overflow
             8 - Total decode overflow
             9 - TIMLOSS-lost a time word
            10 - ATCLOSS-lost ATCRBS data
            11 - MSLOSS- lost Mode S data
```

12 - TOVFLO- Range counter overflow

The following are errors that can occur when checking the decoded data:

Bit 13 - Missed decode data
14 - Illegal decode, decode not ATCRBS
15 - Reply window overflow
16 - Reply overflow, not enough data loc.

Information regarding each reply is stored in an array of structs called "p_a_data." Each reply is stored in its own structure element. If there are more replies than the array can hold, the error bit 16 "Reply Overflow" is set in the error word. The structures are of the type "MA_REP_DATA" which is defined in decode.h. The following describes the contents of each data struct:

p_a_data->a_time - The time field from the TOA word from hardware.
p_a_data->a_code - The code data from the Decode Status word from hardware.
P a_data->wind_nm - The reply window that this data came from.

mparms().

NAME

mparms - Modify parameters. Source file - mparms.cc.

FUNCTION CALL

mparms()

GLOBAL VARIABLES

The following external variables are required by this function:
char *prm_ques[] - The array of parameter questions.
char *spaceb[] - The array of answers or space bar selectable parameters.
struct INT_TYPE int_cur[] - The array of current interrogation parameters.
struct INT_TYPE int_def - The structure of default interrogation parameters.
struct TST_TYPE tst_cur - The structure of current test parameters.
struct TST_TYPE tst_def - The structure of default test parameters.
int p - lkup[] [2] - The lookup array for tne test parameters locations.
struct PARM_IO_TYPE int_io[] - The array of user I/O information for the interrogation parameters.
struct PARM_IO_TYPE tst_io - The user I/O information for the test parameters.

DESCRIPTION

This is the main function for the parameter modification area of the DATAS transponder testing program. It builds the screen that prompts the user for the test number for which parameters to access. It will accept only numeric characters as input for the test number or the F5 function key to exit. If an invalid test number is entered, an error message will be displayed. If a "0" is entered, it will call the appropriate functions to modify the interrogation parameters for all tests. Otherwise, it will call the function "parmscrn()" to access the interrogation parameters and again to access the test parameters. The function will then prompt again for another test number. It will be active until the F5 key is entered.

There are two arrays that determine the position on the screen for each parameter. The array "sp_sc" is for the interrogation parameters, and the array "db_sp" is for the test parameters.

parmscrn()

NAME

parmscrn - parameter modification screen. Source file- parmscrn.cc.

FUNCTION CALL

```
    parmscrn(parm_io,  index,  num,  prm_ques,  mov_dir, title,
    tl_row, tc, tst_num, int_cur, spaceb)
    struct PARM_IO_TYPE *parm_io; /* Parameter IO info      */
    int index;                    /* Parm IO index (start)    */
    int num;                      /* Number of parameters */
    char *prm_ques[];         /* Parameter questions  */
    int mov_dir[][2];             /* Prompt locations     */
    char *title[];                /* Title messages           */
    int tl_row[];                 /* Location for titles  */
    int tc;                   /* Title count          */
    int tst_num;                  /* Test number          */
    struct INT_TYPE *int_cur;     /* Interrogation parameters*/
    char *spaceb[];               /* Space bar answers      */
```

DESCRIPTION

This is the display function for the parameter modification routines. It displays the parameters on the screen and allows the user to change them. The user selects between parameters using the arrow keys. The current parameter as well as those that have been set to other than default are displayed in reverse video. The function keys that are active are: F5 to exit, F6 to set all values displayed to their default values, and F7 to set only the current parameter to its default value.

DETAILED DESCRIPTION

The function will first determine the order and location of the parameter prompts and answers with a call to the function "bldmenu." If there is not enough room for all parameters the program will terminate with a fatal error. The parameters and their prompts are displayed on the screen with the parameters whose values are other than default displayed in reverse video. If the current parameter is one that is altered by entering the space bar key, an appropriate message directing the user to do so is displayed on the screen.

The active function keys are F5 to exit, F6 to set all of the parameters displayed to their default values, and F7 to set only the current parameter to its default value. The arrow keys are used to select between the parameters.

When an answer to a prompt is entered, it will be put through several checks before it is accepted. Most of the checks are made against constraints stored in the parameter IO struct. First the format is checked with a call to the function "ck_format." If the format is ok, the parameter string that was entered is converted into its variable type. The value is then checked that it is in range with a call to "ck_range." and that it is in the proper resolution with a call to "ck_resolu." If there are any errors, the answer is rejected and a message is displayed.

In the case of a spacebar parameter, the value is simply incremented to the next value. If it is at the maximum value it is simply returned to the starting value.

When the answer has been fully tested the actual parameter value is replaced with the new value. If all test parameters are chanqed, then it will update those of all tests.

pcinit()
NAME

pcinit - PC system initializer. Source file - pcinit.cc.

FUNCTION CALL

```
pcinit(msg, pc_buf_loc)
char *msg;              /* Status message */
long *pc_buf_loc;       /* Array of PC data buffers */
```

GLOBAL VARIABLES

The following are external variables required by this function:
char asq_buf[l - Asynchronous service queue buffer.
char tsknam[] - Task name.

DESCRIPTION

This function will establish the communication link between the 68020 computer and the PC subsystem. It will store the location of the communication buffers and attempt to bring the two systems to an on-line state. A success (0) or fail (1) condition is returned and if the initialization did fail, the variable "msg" will contain an error message.

DETAILED DESCRIPTION

The include file "pcinc.h" is required by this function to establish PC system addresses, relevant #defines, and communications buffer structures. There are two primary goals of this function. The first is to bring the 68020 and the PC to an on-line state, and the second is to store the addresses of the PC data buffers into an array.

The two systems are brought "on-line" through a mutual handshaking process. The process is controlled by the 68020 system which acts as the master system. The control information is stored in a structure which contains the following fields:

p_Pc_ctrl->h_id - Host identifier.
p_pc_ctrl->h_state - Host state.
pc-ctrl->pc-id - PC system identifier.
p_Pc-ctrl->pc-state - PC state.
p_pc_ctrl->mode - Operation mode.
p_Pc_ctrl->buf_num - Number of buffers.
p_pc_ctrl->block_addr - Entry block address.
p_pc_ctrl->buf_addr - Buffer base address.
p_pc_ctrl->buf_size[BUF_CNT] - Array of buffer sizes.

This structure is stored in the PC memory. When the PC system is started, it will fill in the information that it is responsible for including pc_id, pc_state, buf_num, block_addr, buf_addr, and buf_size[]. The state of the two systems is then controlled by the 68020 system with the following scenario:

    -Host state is off-line (h_state = S_OFFLINE (0))
    -Host waits until PC state is equal to host state
    -Host state is PC request (h_state = S_PC_REQ (1))
    -Host waits until PC state is equal to host state
    -Host state is on-line (h_state = S_ONLINE (2))
    -Host waits until PC state is equal to host state

The time that the Host system will wait is controlled by the service queue. If the PC does not respond in time or if any other failure occurs, a message will be stored and the function will return a failure condition.

Once the link is established, the array "pc_buf_loc" is filled with the starting address of each of the PC buffers.

plotdat()

NAME

plotdat - Plot reply data on PC system. Source file-plotdat.cc.

FUNCTION CALL

```
    plotdat(test_data, b_hdr_in, entry_pos)
    struct data_share *test_data; /* Test reply data       */
    struct BUF_HDR *b_hdr_in;     /* Buffer header info    */
    int *entry_pos;               /* Block entry position  */
```

GLOBAL VARIABLES

The following external variables are required by this function:
long pc_buf_loc[l - Array of buffer addresses.
int pc_active - PC status 0=off 1=on.
int plot - Plot flag.
int hardcopy - Hardcopy flag.
int pc store - Store data flag.

DESCRIPTION

This function is used to transfer the results of a transponder test to a buffer location in the PC system. The PC system is used to represent the transponder test performance in graphic form. The data can be sent there to be plotted on the screen, plotted on a hard copy, stored in the PC memory, or any combination of all three.

DETAILED DESCRIPTION

In order for this function to work, the PC system/68020 communication link must have already been established. First, the function will check that the pc_active flag is true, check that the pc_id is correct, and check that the pc_state is on-line. If any of these conditions are not met, a fail condition is returned.

Next, the function will have to find an available buffer that will fit the transponder test data block. The data buffers are of various sizes, therefore, the function will search for the smallest one that will fit the test data. Since the last buffers are the smallest and the first buffers are the largest, the search is done from the last buffers to the first. If no buffers are available or none that will fit the data are available, a buffers-full condition is returned. Otherwise, the buffer header and the data are transferred using the memcpy() function. The buffer header precedes the data in each buffer and contains some important information. The structure is described below:

b-hdr-out->bit.blk-tY~e - Data block type, 0=data 1=ASCII.
b_hdr_out->bit.store - Data store flag.
b_hdr_out->bit.hdcpY - Hardcopy flag.
b_hdr_out->bit.display - Display plot flag.
b_hdr_out->bit.link - 0=single buffer 1=multi-buffer.
b_hdr_out->bit.mode - 0=Block 1=itterative.
b_hdr_out->bit.flaq - 0=Buffer available, l=not available.
b_hdr_out->bit.buf_id - ID of next buffer (link=1).
b_hdr_out->test_num - Test number.
b_hdr_out->data_ct - Size of data.
b_hdr_out->point_ct - Number of points so far, if itterative mode.

This buffer header information is supplied to this function, it is not supplied by this function.

Once the data is transferred to the buffer, it is then required that the entry is made known to the PC system by making an entry in the buffer entry queue. The information in this queue is constantly monitored by the PC and contains a flag indicating a used buffer as well as the location of the buffer. If there are no positions available in the queue, a buffers full status is returned .

Plotmsg ( )

NAME

plotmsg - plot system status message. Source file- sumscrn.cc.

FUNCTION CALL

```
plotmsg(status, row)
int status;          /* Status code */
int row:                 /* Display row */
```

DESCRIPTION

This function will display a message describing the status of the PC system on the screen at the row provided.

plotque ( )

NAME

plotque - Plot system queue. Source file - sumscrn.cc.

FUNCTION CALL

```
plotque(data, pc_buf_head, entry_pos)
struct data_share *data;      /* Data to be plotted    */
struct BUF_HDR *pc_buf_head;  /* Buffer header info    */
int *entry_pos;               /* Block entry position  */
```

DESCRIPTION

This function will periodically try to transfer data to the PC plot system. If no buffers are available, a message will be displayed and the function will wait until one becomes available or an escape character is entered.

pmsq ( )
_NAME
pmsg - Print error message. Source file - pmsg.cc.

FUNCTION CALL
```
 pmsg(e_code, str)
 int e_code;     /* Error code */
 char *str;          /* Error message array */
```

DESCRIPTION
This function will display an error message for each bit that is set in the integer "e_code." Each bit position is tested and if it is a "1" an error message indexed by the bit position will be displayed.

pr_head()
NAME
pr_head - Print header information. Source file - prhead.cc.

FUNCTION CALL
```
pr_head(hdrinfo, io_buf, num_lines)
struct HEADER *hdrinfo;      /* Header information    */
struct data_share *io_buf;  /* Output buffer         */
int *num lines;             /* Number of lines used  */
```

DESCRIPTION
This function formats and stores the transponder under test header information into a buffer for display purposes. This function is used by the summary programs to display unit under test information with the test results. The header information includes the aircraft ID, aircraft type, transponder type, and the comment field.

Print ( )
NAME
print - Send string to printer device. Source file-print.cc.

FUNCTION CALL
```
 print(s_ptr, nm_prt)
 char *s_ptr;        /* String pointer     */
 int nm_prt;         /* Number of characters    */
```

DESCRIPTION
This function will send the number of characters specified in "nm_prt" from the string pointed to by "s_ptr" to the printer device. The lwrite() function is used. The status of the lwrite is returned.

prt_init()
NAME
prt_init - Initialize printer device. Source file-print.cc.

FUNCTION CALL
```
    prt init()
```

DESCRIPTION
This function will assign a Logical Unit Number to the printer device. The lassign() function is used. The logical unit number is stored in the global variable "plun." Currently, an escape sequence is sent to initialize an HP Laser printer, but this may be temporary. If the assign is successful a PASS condition is returned, otherwise, a FAIL condition is returned.

pw_thresh()
NAME
pw_thresh - Pulse width threshold set function. Source file - pwthresh.cc.

FUNCTION CALL
```
    pw_thresh(p_data_loc)
    struct data destination *p_data_loc;    / *    Hardware
                                               locations */
```

DESCRIPTION
This function will set the pulse width measurement threshold to the optimum level which is determined by the reply power level. The value that must be stored in the hardware to set the level correctly is stored in the calibration tables and can be referenced based on the reply power.
This function will generate 100 Mode A interrogations in order to find the mean reply power of the first reply pulse. If there are at least 10 replies, the threshold will be set and a SUCCESS condition is returned, otherwise, a FAIL condition is returned.

odatf()
NAME
odatf - Open data file. Source file - odatf.cc.

FUNCTION CALL
```
    odatf(dat_file, df_ext, df_lun, dfd, key_file, kf_lun, kfd,
     new_file, crff)
    char *dat_file;              /* Data file name     */
    char *df_ext;                /* Data file extension    */
    char *df_lun;                /* Data file LUN     */
    int *dfd;                    /* Data file descriptor   */
    char *key_file;              /* Key file name     */
    char *kf_lun;                /* Key file LUN          */
    int *kfd,                    /* Key file descriptor    */
    char *new_file;              /* New file flag     */
    int crff;                    /* Create file flag      */
```

DESCRIPTION
This function is used to open a data file and key access file in order to store the transponder test results. The function requires both the data and key file names and extensions as inputs, and provides file descriptors and logical unit numbers. The create file flag (crff) will tell the function whether or not to create the files

if they do not exist. A crff of "0" means NO and "1" means YES. If the file had already existed, the "new_file" flag will contain a "0," if the file had to be created, it will contain a "1."

rddat()
NAME
    rddat - Read data. Source file - sumdat.cc.

FUNCTION CALL
```
    rddat(lun, rrn, buf, pldt)
    char lun;                 /* LUN of file        */
    long rrn;             /* Record number    */
    struct data_share *buf; /* Data buffer          */
    long *pldt;           /* Amount of data read   */
```

DESCRIPTION
This function will read from the logical unit supplied the current record "rrn" and store it in "buf." The number of bytes read is stored in "pldt." It will return "0" for success, "-1" for end of file, and terminates on VERSAdos errors.

rds_file()
NAME
    rds file - Read DATAS file. Source file - rdsfile.cc.

FUNCTION CALL
```
    rds_file(f_type, e_code)
    int f_type,       /* File type code (obsolete)  */
    int *e_code;          /* Error code if occurred     */
```

GLOBAL VARIABLES
    The following are variables that must be declared external to this function.
    char seq_file[] - Sequence file name.
    struct seqf_head seq_data - Sequence file header.
    int test_array[] - Test sequence array.

DESCRIPTION
    This function was originally intended to read a variety of file types for the DATAS program. It has been reduced to performing the read of test sequence files. It will read and store the sequence file header and the test sequence. If it is successful, it will return a "0," if the file does not exist, it will return a "1," if there is an error in the file open or read, the error code will be stored in e_code and a "2" will be returned.

read_seqf_file
NAME
    read_seqf_file - read test sequence file. Source file-
sfread cc .

FUNCTION CALL
```
    read_seqf_file(s_fil_data, test_array, fp)
    struct seqf_head *s_fil_data;     /* File header   */
    int test_array[];                 /* Test # array  */
    char *fp;                       /* File LUN  */
```

DESCRIPTION

Reads a test sequence file from disk given the logical unit number of the file. The header information and the array of tests is returned.

DETAILED DESCRIPTION

The file is first rewound. If an error occurs, the program will exit with an error message. A lread() is used for all file reads. First, the header is read into a temporary buffer. If an error occurs the error code is returned in negative logic. The data is then transferred from the temporary buffer into the header struct "s_fil_data." The test numbers are then read in until the end of file. The test numbers are stored in the format "struct test_type" which is defined in this function. The test number field from this struct is stored into the test array (test_array[]). The flag field "flg" in the test struct is currently not used.

rep_test()

NAME

rep_test - Reply test. Source file - reptest.cc.

FUNCTION CALL

```
     rep_test(wfm, loc, lead_del)
     struct wave_parm *wfm;       /* Waveform array */
     int loc;                     /* Waveform index for reply */
     long lead_del;               /* Lead reply delay   */
```

DESCRIPTION

This function is used to transmit a reply from channel 2 in order to test transponder tests. It will modify an existing interrogation structure array so precaution must be taken to make sure that the original structure array is large enough to contain the added reply pulse definitions. The reply is stored in channel 2 beginning at the location "loc." The lead reply delay is set to the value contained in "lead_del," which should normally be equal to all the time fields in channel 1 up to the reply window. The function will also set the RF unit to the diagnostic mode and set the channel 2 frequency to 1090 MHz.

rePulse ( )

NAME

repulse - Reply pulse processor. Source file - repulse.cc.

FUNCTION CALL

```
     repulse(pulse, a_code)
     struct RAW_PULSE *pulse; /* Reply pulse information*/
     int a_code;              /* ATCRBS code             */
```

DESCRIPTION

This function will separate the raw reply data for each reply pulse and store it in an array of structs. The struct array is dimensioned as 16 structs, one for each possible reply pulse (al, cl,etc...) including the framing pulses and the Special Position Indicator (SPI). The indexes of the structs correspond directly to the order of the pulses, 0=Fl, l=Al, etc.. The structure definition and information is as follows:

```
     struct RAW PULSE
      {
     int flg;         0=No pulse  l=Pulse present
     int le_time;     time of pulse
     int amp;         amplitude of pulse
     int freq;        frequency of pulse
     int width;       width of pulse
I    nt mp value       mono-pulse value
     }
```

The ATCRBS code in the reply data is used to separate the reply pulses into their respective positions.

If the raw data status word shows an error condition or the ATCRBS code and the number of pulses in the reply data disagree, this function will return an error condition.

This function uses two subfunctions: extrct_data() to extract the reply data for each pulse, and cnt_pulses to count the number of reply pulses present.

setfreq()

NAME

setfreq - Set transmit frequency. Source file - setfreq.cc.

FUNCTION CALL

```
     setfreq(chan, freq)
     unsigned long chan;  /* Channel address    */
     long freq;               /* Frequency in khz   */
```

DESCRIPTION

This function will store the transmit frequency in Binary Coded Decimal (BCD) format in the channel address provided.

set_idle()

NAME

set_idle - Set idle interrogation on or off. Source file-setidle.cc.

FUNCTION CALL

```
    set_idle(s_row, s_status, p_data_loc)
    int s_row;          /* System message row         */
    char s_status;       /* System status            */
    struct data destination *p_data_loc;    / *    Hardware
                                           locations */
```

GLOBAL VARIABLES

The following variables must be declared external to this function:

struct CAL_TYPE calib - Calibration tables.
int i_Power - Interrogation power.
int i_afloss - Airframe loss.

int tx_Port - Active transmitter port.

DESCRIPTION
Based on the current system status, this function will either start or stop the idle interrogation. A message will be displayed on the screen in the row indicated by "s_row" when the idle is turned on or erased when the idle is turned off. The idle interrogation is defined in the include file "idle.h."

setlev()
NAME
setlev - Set power level. Source file - setlev.cc.

FUNCTION CALL
```
    setlev(level, chan, atten, atten_index)
    int *level;          /* Transmit power level       */
    int chan;            /* Transmit channel           */
    int atten;           /* Attenuator selected        */
    int atten_index; /* Attenuator index           */
```

GLOBAL VARIABLES
The following variables must be declared external to this function:
struct CAL_TYPE calib - Calibration tables.
int i afloss - Airframe loss.

DESCRIPTION
This function will attempt to set the transmit level of the channel to the level desired. If the level is too high, the level will be set to the maximum and "P_TOO_HI" will be returned. If the level is too low the level will be set to the minimum and "P_TOO_LO" will be returned. If the level is achievable a "SUCCESS" condition is returned.
setscroll(~
NAME
setscroll - Set scrolling region. Source file - stscrol.cc.

FUNCTION CALL
```
    setscroll(start, stop)
    int start;                /* Starting row  */
    int stop;                 /* Ending row    */
```

DESCRIPTION
This function will define a scrolling region on a WYSE 75 compatible terminal between rows start and stop. To terminate a scrolling region, call setscroll with start at 0, and stop at 24.

spandd()
NAME
spandd - Sensitivity, power, and delay measurement. Source file - spandd.cc.

FUNCTION CALL
```
    spandd(t_data, best_power, best_delay, p_data_loc)
    struct data_share *t_data;    /* Struct to pass back data */
    int *best_power;         /* Best power measured so far  */
    int *best_delay;         /* Best delay measured so far */
    struct data_destination *p_data_loc; /* Load addresses    */
```

GLOBAL VARIABLES
The following variables must be declared external to this function:
- struct CAL_TYPE calib - Calibration tables.
- int i_afloss - Airframe loss.
- int tx_port - Active transmitter port.
- int i power - Interrogation power.

DESCRIPTION
This function will constantly monitor the reply power, sensitivity, and reply delay of the transponder under test. This is used primarily to measure these parameters as the aircraft passes by the main beam in order to find the optimum measurement at center beam. It will transmit a standard Mode A interrogation at 200 PRF. An F9 key entered will stop the function. The successive approximation algorithm "sapx()" is used to measure the sensitivity, and the interrogation loop algorithm "int_loop()" is used to measure reply power and delay.

The current measurements are displayed on the screen along with the best measurements so far, if any parameter improves, it will be displayed in reverse video along with a message "increasing."

stcenter()

NAME
stcenter - String centering function. Source file- stcenter.cc.

FUNCTION CALL
```
     stcenter(str)
     char *str:          /* String      */
```

DESCRIPTION
This function will return a column position where the input string "str" would be in the center of the display screen.

store_ans()

NAME
store_ans - Store parameter answer. Source file-
storeans.cc.

FUNCTION CALL
```
    store_ans(p_inp_data, parm_io, dest)
    char *p_inp_data;   /* Parameter as entered from keyboard*/
    struct PARM_IO_TYPE *parm_io; /* Parameter defines     */   long
dest;          /* Storage destination             */
```

DESCRIPTION
This function will store the character string "p_inp_data" in the location "dest" in the proper format which is defined in "parm_i-o."

store ptr()
NAME
    store_ptr - Store pointer answer. Source file - storeans.cc.

FUNCTION CALL
```
    store_ptr(parm_io, source)
    struct PARM_IO_TYPE *parm_io; /* Parameter defines     */   long
source;                  /* Source address of value*/
```

DESCRIPTION
This function will store from the location "source" into the pointer variable in the "parm_io" struct.

suc_apprx()
NAME
    suc_apprx - Successive approximation algorithm. Source file - sapx.cc.

FUNCTION CALL
```
    suc_apprx(rep_thresh, num_ints, reply, xmit_chan)
    int rep_thresh;         /* Reply threshold (usually 90%)*/
    int num_ints;          /* Number of interrogations   */
    struct reply_format *reply; /* Reply data            */ int
    xmit chan;          /* Transmit channels            */
```

GLOBAL VARIABLES
The following variables must be declared external to this function:
    struct CAL_TYPE calib - Calibration tables.
    int i_afloss - Airframe loss.
    int tx Port - Active transmitter port.

DESCRIPTION
This function is used to rapidly find a certain percent reply point using the successive approximation algorithm. This is the function that will generate the actual interrogations for the algorithm. The intended number of interrogations is passed in the variable "num_ints" and the function will interrogate until either the number of replies are met to insure that the reply threshold is met or the number of misses are encountered to indicate the reply threshold can not be met. The function will run the transmit channels defined in "xmit_chan," and store the reply information in the struct "reply."

DETAILED DESCRIPTION
The reply information supplied by the function is contained in a data structure. The structure format of the "reply" struct is defined in tstlib.h and provides the following information: reply->power.min - The lowest reply power of the **last reply pulse** (F2) measured out of all replies. Is stored in dB's * 10. All calibration offsets are added including airframe loss and receiver coupling loss.

reply->power.mean - The mean reply power of the **last reply pulse** (F2) measured in all replies. Is stored in dB's * 10. All calibration offsets are added including airframe loss and receiver coupling loss.

reply->power.max - The highest reply power of the **last reply pulse** (F2) measured out of all replies. Is stored in dB's * 10. All calibration offsets are added including airframe loss and receiver coupling loss.

reply->frequency.min - The lowest RF transmitter frequency of the first reply pulse measured out of all replies. It is stored in kHz.

reply->frequency.mean - The mean RF transmitter frequency of the first reply pulse measured out of all replies. It is stored in kHz.

reply->frequency.max - The highest RF transmitter frequency of the first reply pulse measured out of all replies. It is stored in kHz

reply->delay.min - The least reply delay measured out of all replies. It is stored in nanoseconds. This is the delay from the lead edge of the reply window. It assumes that the reply window has been adjusted for coupling time.

reply->delav.mean - The mean reply delay measured out of all replies. It is stored in nanoseconds. This is the delay from the lead edge of the reply window. It assumes that the reply window has been adjusted for coupling time.

reply->delav.max - The maximum reply delay measured out of all replies. It is stored in nanoseconds. This is the delay from the lead edge of the reply window. It assumes that the reply window has been adjusted for coupling time.

reply->atcrbs_code - ATCRBS identification code (4096). This code comes from the system reply decoder and is in whatever format the decoder provides. In order for this variable to be stored, it must be the same for at least five consecutive replies. If the code changes during the interrogation loop, this variable will contain whatever the code was for the latest five consecutive replies.

reply->stdev-pls-cnt - Standard deviation of reply pulses. This is provided so that reply reliability can be determined.

reply->mean_pulse_count - Mean number of reply pulses per interrogation.

reply->intfere_ratio - Ratio of replies that were rejected because of interference to accepted replies. This is provided to indicate data reliability.

reply->reply-cnt - Reply count. The number of replies detected.

The variable "xmit_chans" determines from which channels to transmit. Each bit position corresponds to a channel; a "1" means to transmit and a "0" means not to transmit. There are currently three channels in the system, meaning a total of three active bits (e.g., 0x0005 means to transmit on channels 1 and 3). The timing must be set for the channels prior to calling this function. This function will start the system clock, but the PRF must be set prior.

A status (integer) will be returned when the function terminates. The following are return possibilities which are defined in tstlib.h: T_ABORT (-1), PRF_ERROR (2), T_FAIL (1), or T_PASS (0). A status of T_ABORT will be returned if the acceptable level of interference is exceeded. PRF_ERROR will be returned if the system timer interrupt is already set when it is first read at any time during the loop. T_PASS will be returned if the reply threshold is achieved. T_FAIL will be returned if the reply threshold was not met.

sum* ( )

NAME

sum* - Summary function for test *. Source file sum*.cc.

FUNCTION CALL

```
    sum*(t data, io_buf, num_lines)
    struct data_share *t_data;    /* Test data   */
    struct data_share *io_buf;    /* Output buffer    */
    int *num_lines;               /* Number of lines  */
```

DESCRIPTION

This represents a typical call to a test summary function. The test summaries produce a brief description of the results of the transponder for each test. The test data is passed to the function in the structure "t_data." The summary for the test is returned in the structure "io_buf" in ASCII format. The number of lines that the summary message requires for the screen or printer ;s returned in "num lines."

sumdat()

NAME

sumdat - Summarize transponder test data. Source file-sumdat.cc.

FUNCTION CALL

```
    sumdat(start_pnt, hdrinfo, io_dev, dflun, rrn, range,
    line_array, head_rrn, head_sent)
    int *start_pnt;          /* Starting test # index */
    struct HEADER *hdrinfo;  /* Test header info       */
    int io_dev;                /* Output device             */
    char df_lun;         /* Data file LUN            */
    long *rrn;                 /* Current record #      */
    int range;                 /* Display- 0=LINE 1=PAGE    */
    int *line_array;       /* # lines per test      */
    long head_rrn;             /* Record # of header        */
    int head sent;             /* Header sent flag      */
```

DESCRIPTION

This function is the heart of the test summary function. It will retrieve the test data from the data file, call the appropriate data reduction function, and display the results on the screen or printer.

DETAILED DESCRIPTION

The page limit is set based on the output device chosen. The defines for the screen and printer limits are in "sumdef.h."

The test data header information will be displayed if the starting point is at the beginning (*start_pnt = 0) and a full page is being displayed (range = l); or the header has not been sent (head_sent = NO), a single line is being displayed (range = 0), and the output device is the printer. "head_rrn" is used to locate the header record in the file.

The test data will be displayed or printed until either all tests have been displayed or the screen is full. The summary

functions sum* are called to reduce the data. These functions are in a function array defined in "sumdef.h" and there is one for each test. The current record "*rrn," the start location "*start_pnt," and the test line array "*line_array" are maintained to allow for multiple pages. This function will reduce and display one line or one page of test results only. It must be called again to display following pages.

The function returns "0" if successful or "-1" if the end of file is reached.

summenu()

NAME

summenu - Test summary menu. Source file - summenu.cc.

FUNCTION CALL

summenu()

DESCRIPTION

This function drives the initial screen for the transponder test summary program. It prompts the user for the following information: Location Identifier - location code for the site where the data was collected, this is used as the file catalog; Date - date when the data were collected, is used as part of the file name; Data file extension - the extension for the data file; Summary mode - either SINGLE or MULTIPLE (SINGLE will prompt for the aircraft ID to locate a specific test result, MULTIPLE will prompt for an output device to send all the test results collected at the location and date entered).

The function key F5 will exit the function and F6 will start the summary proqram.

sumscrn()

NAME

sumscrn - Test summary screen display. Source file- sumscrn.cc.

FUNCTION CALL

```
    sumscrn(df lun, rrn, s_mode)
    char dflun;         /* Data file LUN         */
    long rrn;           /* Data file record number    */
    int s_mode;         /* Summary mode - SINGLE or MULTIPLE*/
```

DESCRIPTION

his function will call the function "sumdat" to produce the test summaries and headers on the screen. The center portion of the screen will display the test summaries and the bottom will display several function key options. One of the test summaries is highlighted and may be selected using the arrow keys. The function keys perform the following tasks: F5 - EXIT, will return to the previous screen; F6 - PAGE, will display the next page of

results on the screen; F7 - PRINT LINE/ALL, will send the highlighted line test summary and header to the printer; if SHIFT F7 is entered, will send all the test results from that target to the printer; F8 - PLOT LINE/ALL, will send the test data and header data to the PC plot system; if SHIFT F8 is entered, will send all summaries from the target to the PC; F9 - NEXT, will advance to the next aircraft's test result data block.

<u>toupper()</u>
NAME

toupper - convert alphabet characters to upper case. Source file - toupper.cc.

FUNCTION CALL

```
     toupper(sptr)
     char *sptr; /* Character string pointer */
```

DESCRIPTION

Converts all alphabetic characters in the string "sptr" to upper case. Terminates when a null character is reached.

<u>tstdrvr()</u>
NAME

tstdrvr - Test driver function. Source file - tstdrvr.cc.

FUNCTION CALL

```
     tstdrvr(dflun, kflun)
     char dflun; /* Data file LUN */
     char kflun: /* Key file LUN      */
```

GLOBAL VARIABLES

The following variables must be declared external to this function:

<u>int p_mode</u> - Power mode.
<u>int ps_mode</u> - Print summary mode.
<u>int i_power</u> - Interrogation power.
<u>int store</u> - Data store flag.
<u>struct seaf_head seq_data</u> - Test sequence file header.
<u>int test_arrayrl</u> - Test array.
<u>char date[]</u> - System date.
<u>char loc_id[]</u> - Location ID.
<u>int i_afloss</u> - Airframe loss.
<u>struct CAL_TYPE calib</u> - Calibration tables.
<u>int tx_port</u> - Transmitter port.

DESCRIPTION

This function controls the actual transponder field testing process. It sets up the field testing display screen, displays entry prompts for aircraft information, and provides function keys for test control.

DETAILED DESCRIPTION

The entry prompts are initialized to be blanks or 0's before the tests are run. The function will start the idle interrogation and then wait for keyboard entry. The function keys that are active depend on the status of the system, hence, the variable "sys_status." The function "d_fun_keys" is called to define and display the active function keys. Some of the more important function keys are as follows: F5 will exit the function, F9 will start the test sequence if it is not running and stop the sequence if it is running, F10 is used to toggle the idle loop (start-idle, stop-idle), and F7 will increase the interrogation power and F8 will decrease the interrogation power if the interrogation power is under manual control.

The function "tstseq()" is called to loop through the test sequence when the user starts the ATCRBS field tests.

This function is responsible for storing the test data into the data file. The function "wrtdat" is called to do so. The function "sumscrn" is also called to display the test results on the screen.

tstseq()

NAME

tstseq - Test sequence function. Source file - tstseq.cc.

FUNCTION CALL

```
tstseq(p_data_loc, s_msg_row, f_head)
struct data_destination *p_data_loc;    /* Hardware loc*/
int s_msg_row;                /* System message row    */
struct HEADER *f_head;        /* Data file header struct*/
```

GLOBAL VARIABLES

The following variables must be declared external to this function:

char util_buf r l - Utility buffer. A general purpose block of memory used for large data transactions. Used by this function to hold the transponder test results prior to disk storage.

struct data_share r_data - Used to point to the transponder test results data.

int test_array[l - Array of test numbers to run.

int store - Flag to indicate if the data should be stored on disk or not.

DESCRIPTION

This function will loop through the test numbers in the array "test_array" and call their corresponding ATCRBS test functions. "p_data_loc" is passed the transponder tests to provide the hardware locations for hardware control.

DETAILED DESCRIPTION

If the storage flag "store" is TRUE, a temporary file is opened to store the test results as an intermediate step before the user decides to store the test results in the actual data file. The scrolling region on the display, that was defined in previous functions, is cleared in order to display the status of the running tests. The system status messages are displayed in the "s_msg_row" on the screen. The function will then loop through each test stored in the array until complete or stopped by the F9 key entered by the user. Prior to calling the test sequence, the pulse width measurement threshold is set to minimum. Each test number is checked to see if it is valid and if the test exists. Each valid test is run and the results are stored in the temporary file. As each test completes, a status message is displayed on the screen. If the status is valid, the best power and delay measured during the test is displayed. When the test sequence completes, the reply power, sensitivity, and delay are monitored continuously until the user ends the test. The function that monitors these parameters is called "spandd()." The temporary data file is closed when this function is complete.

<u>vfcreat()</u>
NAME

vfcreat - VERSAdos file create. Source file - vfcreat.cc.

FUNCTION CALL

```
vfcreat(vfname)
char *vfname;            /* File name        */
```

DESCRIPTION

Creates a VERSAdos file and returns a logical unit number(char). If there is an error, returns the VERSAdos error code in negative logic. Uses lalloc().

<u>vfopen()</u>
NAME

vfopen - VERSAdos file open. Source file - vfopen.cc.

FUNCTION CALL

```
vfopen(vfname)
char *vfname;            /* File name        */
```

DESCRIPTION

Opens a VERSAdos file and returns a logical unit number(char). If there is an error, returns the VERSAdos error code in negative logic. Uses lassign().

<u>wfmod()</u>
NAME

wfmod - waveform modifier. Source file - wfmod.cc.

FUNCTION CALL

```
     wfmod(pulse_num, width, p_space, p_wfm)
    int pulse_num;            /* Pulse number */
    long width;          /* Pulse width      */
    long p_space;             /* Pulse spacing    */
    struct wave_parm *p_wfm; /* Waveform struct        */
```

DESCRIPTION

This function will modify a pulse while it is stored in the waveform structure. This is primarily used to modify default waveform patterns that have been changed by the user prior to encoding them into the binary format used by the hardware.

The location of the pulse to be modified is found by searching for the "pulse_num," the lead-edge of a PAM action field from the starting address "p_wfm." If the pulse is not found in the struct, a "1" is returned as an error condition.

The pulse spacing refers to the lead-edge of the chosen pulse. If the chosen pulse is the first pulse in the waveform it simply stores the "p_space" value as the lead-edge time of that pulse. If there are other pulses preceding the chosen pulse, the spacing is from the lead-edge of the previous pulse. The function will adjust for the previous pulses width. If the width of the previous pulse is too wide for the spacing selected, a "2" is returned as an error condition.

The pulse width is stored at the trail edge of the chosen pulse, which must be at the next location in the struct or a "3" will be returned as an error condition. The timing of any events following the chosen pulse are adjusted accordingly with the new width of the pulse.

If no error conditions occur, a "0" is returned. If error conditions have occurred, the function "int_err()" can be called to print a message describing the error.

wrtdatf)

NAME

wrtdat - write data file. Source file wrtdat.cc.

FUNCTION CALL

```
     wrtdat(hdrinfo, tfname, dflun, kflun, msgrow)
     struct HEADER *hdrinfo;  /* Header information     */
      char *tfname;             /* Temp file name           */
    char dflun;                 /* Data file LUN            */
    char kflun;                 /* Key file LUN         */
    int msgrow;                 /* Message row          */
```

DESCRIPTION

This function will store the test data in the database file and the key file information in the key-access file.

DETAILED DESCRIPTION

The temporary data file "tfname" is the source of the transponder test data. If the file open of the temp file fails, a fatal error occurs. The data will be stored at the end of the data file. The function uses the function "lposition" to locate the end of the file.

If the position is unsuccessful, a fatal error occurs. The transponder header information from "hdrinfo" is first stored in the data file, then the data is read from the temporary file and stored in the data file. The key information, which is the aircraft ID, etc., is stored in the key file from the header information "hdrinfo."

APPENDIX B

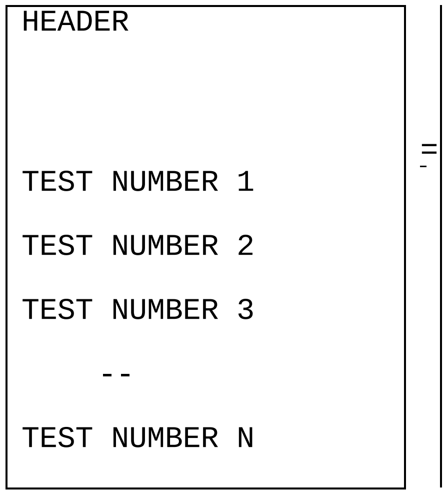
SEQUENCE FILE EDITOR FUNCTIONS

GENERAL DESCRIPTION

The test sequence file editor program seq.lo is used to generate and modify test sequence files. The program is a menu driven screen editor. The heart of the program is comprised of four discrete display functions that perform the basic capabilities of the editor which are: file access, file display, the edit screen, and a list of available tests.

SEQUENCE FILE STRUCTURE.

Test sequence files contain a header record and test number records. The header record currently contains an integer field which contains the number of tests in the file. The remaining 254 bytes are for future expansion if needed. Following the header field is a series of records, one for each test number stored in the file. The test number records contain an integer for the test number and a long integer as a flag field. The flag field is currently not in use. Figure B-1 shows the sequence file format.



```
HEADER - struct seqf_head
         {
         int test num;
         }
         254 bytes free.
 TEST NUMBERS - struct test_type
                  {
                  int num;
                  long flg;
                  }
Test numbers are stored in "num." The
"flg" field is currently not used.
One record is stored for each test
number.
```

FIGURE B-1. TEST SEQUENCE FILE STRUCTURE

SOURCE FILES.

Include files (.h). The following is a list of the include files used by the SEQ program and a brief description of their contents:

displib.h - definitions for WYSE terminal display attributes and escape sequences.

seqflib.h - relevant definitions used by the SEQ editor.

Stdlib.h - standard library, provides common definitions such as TRUE, FALSE, etc.

tstlib.h - common transponder testing library.

Source files (.cc). The seq program may be linked by running the command file seq.cf. A cross-reference between function names and file names follows:

chk_name()	chkname.cc
chk_tst()	chktst.cc
curpos()	curpos.cc
dis_matx()	dismatx.cc
displ()	displ;cc
disp2()	disp2.cc
disp3()	disp3.cc
disp4()	disp4.cc
get_dec()	getdec.cc
get_input()	getinput.cc
getyorn()	getyorn.cc
main()	seq.cc
rd_tests()	rdtests.cc
read_seqf_file()	sfread.cc
toupper()	toupper.cc
vfcreat()	vfcreat.cc
vfdes()	vfdes.cc
vfopen()	vfopen.cc
writ seqf_file()	sfwrite.cc

chk_name

NAME

chk name - Check file name. Source file chkname.cc.

FUNCTION CALL

```
chk_name(filename, count)
char *filename;            /* Filename to check */
int count;                     /* Number of characters   */
```

DESCRIPTION

Tests a character string to see if it is a valid VERSAdos file name.

DETAILED DESCRIPTION

The purpose of this function is to test for a valid filename before it is combined with user number and catalog, etc., for file open. The name is stored in the character string "filename" and the length of the string is passed in the variable "count." The first character is tested to be an alphabetic character and the following characters are tested to be either alphabetic or numeric. If either of these conditions are violated, a "1" is returned to indicate illegal characters. If count is greater than eight, a "2" is returned to indicate the file name is too long. If the file name is valid, the function returns a "O."

<u>chk_tst</u>
NAME
chk_tst - Check transponder test number is within range. Source file - chktst.cc.

FUNCTION CALL
```
     chk_tst(tst_num)
     int tst_num;            /* Test number to check */
```

DESCRIPTION
This function will check that a test number is greater than "O" and less than or equal to T_LIMIT. T_LIMIT is defined in tstlib.h.

<u>curpos</u>
NAME
curpos - Cursor position. Source file - curpos.cc.

FUNCTION CALL
```
     curpos(ro, co)
     int ro;             /* Screen row                */
     int co:             /* Screen column          */
```

DESCRIPTION
This function will position the cursor at the specified row and column on a wyse 7S compatible terminal.

<u>dis_matx</u>
NAME
dis_matx - Display number in 10 x 10 matrix. Source file- dismatx.cc.

FUNCTION CALL
```
     dis_matx(test_array, seq_ans)
     int test_array[];                /* Test # array          */
     int seq ans;                 /* Sequence number       */
```

DESCRIPTION
This function positions and displays the test numbers in the 10 x 10 matrix for the edit function. If there is no test number at the sequence position, a blank box will be displayed. The numbers will be displayed right justified. The location in the matrix area is determined by the sequence number.

<u>displ</u>
NAME
displ - display function #1, file access. Source file- displ.cc.

FUNCTION CALL

```
    displ(s_fil_data, test_array, s_file, access, fp)
    struct seqf_head *s_fil_data;     /* File header       */
    int test_array;                   /* Test # array      */
    char *s_file;                     /* Sequence file */
    int *access;                  /* Access mode R,W   */
    char *fp;                         /* File LUN      */
```

DESCRIPTION

Displ.cc is display function 1, the file access display. This function provides file access for reading and writing test sequence files to the disk.

DETAILED DESCRIPTION

The function creates the display screen and then loops to accept characters from the keyboard. If the default file is accepted with a carriage return, the loop is ended. If a valid file name is entered, the loop is ended; otherwise, a message is displayed as to why the name is invalid. If a space bar is pressed, the file access mode is toggled between read and write, and is displayed as such on the screen. If the function key F5 is pressed, the function returns a "0" as the indication to exit the program.

If the default file name was accepted or a valid file name was entered, a complete filename is constructed using the volume, user number, and catalog defined in stdlib.h as well as the extension defined in seqflib.h.

The file is then opened.

If the access is for read, the file is read with a call to the function read_seqf_file() which is the source file sfread.cc. If the read is successful, the file is closed and the function returns a "2" to indicate that the next display function to call is disp2, the file display function. If the read is not successful, the appropriate error messages are displayed.

If the access is for write, the test array must contain at least one test number before it can be written. If the array is empty an error message is displayed and the user can either exit by entering an F5 key or reenter the program with read access by typing any other key. If the array contains one or more test numbers, the file is checked to see if it currently exists. If it currently exists, the user is warned and is prompted as to whether or not to replace it. If the user wishes to replace the file, the old file is deleted and a new one of the same name is created. If the file did not exist, it is created. The file is then written to the disk with a call to the function writ_seqf_file() which is the source file sfwrite.cc. If the write is successful, the function switches the file access to "read" and returns a "1" to cause the program to return to display 1. If there is an error, it is displayed on the screen.

<u>disp2</u>
NAME:

disp2 - display function #2, file display. Source file-disp2.cc.

FUNCTION CALL

```
    disp2(t_tests, num_strs, s_fil_data, test_array, filename)  char
*t_tests;               /* Array of test names   */
    int num_strs;               /* Number of test names  */
    struct seqf_head *s_fil_data; /* File header           */
    int test_array[];           /* Test # array          */   char
*filename;              /* Sequence file         */
```

DESCRIPTION

Disp2.cc is display function 2, the file display screen. This function will display the contents of a test sequence file on the terminal screen. It will list the sequence number, test number, and the name of each test in the file. The user may page through the list of tests and make a hardcopy of the list on the printer.

DETAILED DESCRIPTION

The function creates the display and then checks if there is a cross-reference test name list available, (num_strs > 0). The variable num_strs contains the value of the return from the function call to rd_tests() in main(). If num_strs is "0" the test list was empty. If it is "-1" the test list file could not be accessed. In either case, an error message is displayed and the user may enter F7 to return to display 1, or enter F8 to go to display 4.

If num_strs indicates that there is a test list available, the first page of the file will be displayed. This display mechanism is contained in a "while loop" since the display can be paged. First the screen is cleared in the area where the tests are displayed. Then, until the screen is full or there are no more tests in the file, the array of tests are indexed by the test numbers stored in the file and printed on the screen. If there is no test name available in the file, only the test number is displayed. The variable "start" is used to keep track of the current position in the file between "pages." The terminal is then polled for keyboard input. There are four active function keys: F5 to page, F6 to print, F7 to exit to display 1, and F8 to exit to display 4 to edit the file.

disp3

NAME

disp3 - display function #3, test list. Source file-disp3.cc.

FUNCTION CALL

```
     disp3(t_tests, num_strs)
     char *t_tests;           /* Array of test names        */
     int num_strs;            /* Number of test names   */
```

DESCRIPTION

Display function 3 is used to produce a list of available transponder tests on the terminal screen. The user may page through the list and make a hardcopy of the list on the printer.

DETAILED DESCRIPTION

The function creates the display and then checks if there is a cross-reference test name list available (num_strs > 0). The variable num_strs contains the value of the return from the _function call to rd_tests() in main(). If num_strs is "0" the test list was empty.

If it is "-1" the test list file could not be accessed. In either case, an error message is displayed and the user may enter F7 to return to display 1.

If num_strs indicates that there is a test list available, the first page of the list will be displayed. This display mechanism is contained in a "while loop" since the display can be paged. First, the screen is cleared in the area where the tests are displayed. Then, the test list file is displayed until the page is full or there are no more tests in the list. The variable "start" is used to keep track of the current position in the file between "pages." The terminal is then polled for keyboard input. There are three active function keys: F5 to page, F6 to print, and F7 to exit to display 4 to edit the file.

disp4

NAME

disp4 - display function #4, edit screen. Source file-disp4.cc.

FUNCTION CALL

```
     disp4(s_fil_data, test_array, access, filename)
     struct seqf_head *s_fil_data; /* File header      */
     int test_array[];             /* Test # array     */
     int *access;                  /* Access mode R,W  */
     char *filename;               /* Sequence file    */
```

DESCRIPTION

Disp4 is display function 4, the file edit screen. This function allows the user to generate or modify test sequence files .

DETAILED DESCRIPTION

The function creates the display and then displays the test numbers in the test matrix area. The function dis_matx() is used to position the tests properly in the matrix. The variable "test_ans" is used to keep track of the test number at the current test sequence position. The current test number and sequence number are displayed at the entry prompts.

The function then waits for keyboard input from the user. If the first character is an escape key, the rest of the input is stored. If the F5 key is entered, insert mode is activated and indicated as such on the display with "INSERT" underlined. If the F6 key is entered, replace mode is activated and indicated with "REPLACE" underlined. If the F7 key is pressed, the test number at the current sequence position is deleted from the array as well as the display matrix. If SHIFT F7 is entered, all the tests are deleted from the array and the display matrix. If F8 is entered, the function returns a "3" to call the test list function. If F9 is entered, the access mode is switched to "write" and a "1" is returned to call the file access function. If an arrow key is entered, the function will position the cursor to the appropriate position, either the "TEST NUMBER" or "SEQUENCE NUMBER" prompt.

If numeric data is entered, the function get_dec() is called to store the input. If the cursor is on the test number question, the function chk_tst() is used to test for a valid test number. If the test number is invalid, an error message is displayed. If there is no more room for tests in the matrix, an error message is displayed. If everything is valid, the test number is entered into the array and displayed in the test matrix. The test sequence number is advanced to the next available position and displayed.

If a sequence number is entered, it is tested to see if is within range. If it is, the sequence number is advanced to that position and the new value is displayed. If the sequence number entered is out of range, an error message is displayed.

get_dec

NAME

get_dec() - Get decimal numbers as input. Source ile-getdec.cc.

FUNCTION CALL

```
    get_dec(p_data, row, col, max_in)
    char *p_data;            /* Pointer to input data  */
    int row,              /* Current row position       */
    int col;              /* Current column position    */
    int max_in;              /* Max digits allowed     */
```

DESCRIPTION

Accepts only numeric characters as input from keyboard.

DETAILED DESCRIPTION

This function is intended to be called only after the first numeric character has been read in. Its purpose is to accept only numeric characters (0 - 9) until a carriage return is entered or the buffer is full. If any other characters are entered they will be ignored except for the backspace and delete keys, which are recognized and will erase characters that were entered. The input characters are stored in the array p_data up to the maximum input length indicated in max_in. Row and Col are used to position the echoed characters on the screen.

get_input

NAME get input() - Get keyboard input. Source file - getinput.cc

FUNCTION CALL

```
    get_input(p_data, count, row, col, mx_in)
    char *p_data;            /* Pointer to input data      */
    int *count;              /* # characters read in           */
    int row;              /* Current row position          */
    int col;              /* Current column position       */
    int mx in;               /* Max character input allowed    */
```

DESCRIPTION

This is a general purpose function that filters only printable characters from keyboard input and echoes them on screen and stores them in a fixed length buffer.

DETAILED DESCRIPTION

This function is intended to be called only after the first character has been read in. Its purpose is to test this character and test and accept any following characters until the buffer is full or a carriage return is entered. If any invalid characters are entered (not between ASCII 0x20 and 0x7f) the function returns a "1" as an error condition. If all characters are valid and a carriage return is entered, the function will return a "0" as a success condition. The backspace and delete keys are recognized and will erase characters that were entered. The input characters are stored in the array p_data up to the maximum input length indicated in mx_in. Row and Col are used to position the echoed characters on the screen. Count will contain the number of characters read in.

getyorn

NAME

getyorn - get "y" or "n" from the keyboard. Source file-getyorn.cc.

FUNCTION CALL

```
getyorn();
```

DESCRIPTION

Waits indefinitely for a "y" or "n" from the keyboard. Lower case and upper case are allowed. Returns a "1" if a "y" is entered, returns a "0" if "n" is entered.

main

NAME

main() - Test sequence file editor main routine. Source file - seq.cc.

FUNCTION CALL

```
main()
```

DESCRIPTION

The main function of the test sequence file editor program controls the calls to the various functions of the program.

DETAILED DESCRIPTION

The main function initializes the appropriate variables to start with display function #1. It then calls the function rd_tests() which stores a list of available transponder tests in the string array "t_tests." The function then enters a "while loop" in which it will remain until the program exits. The "while loop" consists of a screen clear and a switch statement where the program will switch to the appropriate display functions. Each of the display functions returns the value of the next function to ca11. If the value "0" is returned, the program exits.

rd_tests

NAME

rd_tests - Read list of transponder tests. Source file-rdtests.cc.

FUNCTION CALL

```
rd_tests(t_tests)
char t_tests[][SCREEN_WDTH];  /* Array of test names   */
```

DESCRIPTION

Reads and stores a list of transponder tests into the array "t_tests." The list is stored in the file with the name defined by "LISTFILE" which is defined in seqflib.h. Returns the number of strings read.

read_seqf_file

NAME

read_seqf_file - read test sequence file. Source file-sfread cc.

FUNCTION CALL
```
    read_seqf_file(s_fil_data, test_array, fp)
    struct seqf_head *s_fil_data;     /* File header   */
    int test_array[];                 /* Test # array  */
    char *fp,                         /* File LUN      */
```

DESCRIPTION
Reads a test sequence file from disk given the logical unit number of the file. The header information and the array of tests are returned.

DETAILED DESCRIPTION
The file is first rewound. If an error occurs, the program will exit with an error message. An lread() is used for all file reads. First, the header is read into a temporary buffer. If an error occurs, the error code is returned in negative logic. The data are then transferred from the temporary buffer into the header struct "s_fil_data." The test numbers are read in until the end of file. The test numbers are stored in the format "struct test_type" which is defined in this function. The test number field from this struct is stored into the test array (test_array[]). The flag field "flg" in the test struct is currently not used.

toupper
NAME
toupper - convert alphabet characters to upper case. Source file - toupper.cc.

FUNCTION CALL
```
     toupper(sptr)
     char *sptr; /* Character string pointer */
```

DESCRIPTION
Converts all alphabetic characters in the string "sptr" to upper case. Terminates when a null character is reached.

vfcreat
NAME
vfcreat - VERSAdos file create. Source file - vfcreat.cc.

FUNCTION CALL
```
    vfcreat(vfname) char *vfname;     /* File name */
```

DESCRIPTION
Creates a VERSAdos file and returns a logical unit number(char). If there is an error, returns the VERSAdos error code in negative logic. Uses lalloc().

vfdes
NAME
vfdes - VERSAdos file name/file descriptor conversion. Source file - vfdes.cc.

FUNCTION CALL

```
    vfdes(filenl, fdesp)
    char *filenl;            /* File name        */
    struct fdes *fdesp;      /* File descriptor  */
```

DESCRIPTION

Given a complete filename (includes volume, user ID, etc.), will convert to file descriptor. For use where file access requires a file descriptor.

vfopen

NAME

vfopen - VERSAdos file open. Source file - vfopen.cc.

FUNCTION CALL

```
    vfopen(vfname)
    char *vfname;             /* File name */
```

DESCRIPTION

Opens a VERSAdos file and returns a logical number(char). If there is an error, returns the VERSAdos code in negative logic. Uses lassign().

writ_seqf_file

NAME

writ_seqf_file - write test sequence file. Source file sfwrite.cc.

FUNCTION CALL

```
     writ_seqf_filets_fil_data, test_array, fp)
    struct seqf_head *s_fil_data;     /* File header       */
    int test_array[];                 /* Test # array      */
    char *fp;                         /* File LUN          */
```

DESCRIPTION

Writes a test sequence file to the disk given the logical unit number of the file. The header information and the array of tests are written.

DETAILED DESCRIPTION

The header information is first stored in a buffer named "a_buf." It is then written to file using the lwrite function. The function loops for the number of tests indicated in the header and stores the numbers in the test_type format in the file. The file is then closed and the Logical unit is freed.

APPENDIX C

ATCRBS FIELD TESTS

INTRODUCTION

This section provides a detailed description of the DATAS ATCRBS field tests. For each test there is a description section which defines the purpose of the test, a procedure section that briefly describes the more important elements of the test procedure, lists of the interrogation and test parameters, the test data structure definition, and following each test is a sample plot of the test results. Preceding the test by test descriptions is an overview which tells about fundamental items that are common to all tests.

The overview and parts of the individual test descriptions contain information that is primarily important only for software maintenance or development support.

OVERVIEW.

The Air Traffic Control Radar Beacon System (ATCRBS) field test procedures were designed to follow the procedures defined in the Minimum Operational Performance Standards (MOPS) for Air Traffic Control Radar Beacon System/Mode Select (ATCRBS/Mode S) Airborne Equipment (Document No. RTCA/DO-181). Each ATCRBS test is numbered. The test numbers loosely follow the sequential order in which they appear in the MOPS. This is not a strict requirement of the test program design. The reason that the test numbers are not consecutive is that some of the MOPS test procedures are not applicable to a field testing situation, or, they are Mode S transponder test procedures. The tests are referenced by number so that they can be called using an index (test number). They are stored as an array of functions. This function array is defined in the include file "testdef.h."

Each test has a source file named "aftst(X).cc" ((X) is the test number) and an "include file" "tst(X).h." The "include file" primarily contains the structure definition for the test data.

GLOBAL VARIABLES. The calibration tables are used by virtually all of the tests. The calibration tables are stored in the global structure "struct CAL_TYPE calib." The calibration structure types are defined in "calib.h." "tstlib.h" is included since it provides the structure definitions for the interrogation and test parameters as well as some other common definitions. The interrogation parameters are stored in the global structure array "struct INT_TYPE int_cur[]" and the test parameters are stored in the qlobal structure "struct TST_TYPE tst_cur."

Another global variable used by the tests is "int i_afloss" which is the airframe loss. This variable is used to adjust the interrogation power and reply power measured. The airframe loss value originates from the calibration tables and is a function of airframe height.

The global variable "i_power" is the system interrogation power. This is the interrogation level set by either the user or the system, depending on what power mode was selected. This value is used by those tests where interrogation power is not controlled as a function of the test or where a certain interrogation level is not defined by the test.

The global variable "tx_port" defines which transmitter port is currently in use. The value of this variable is stored in the main function (source file afexec.cc). The #defines used to determine the value come from "calib.h."

INTERROGATION WAVEFORMS. The interrogation waveforms are defined within each test function. The process that determines the resulting waveform is quite elaborate because the system was designed to be as flexible as possible.

A default waveform is provided in each test. The waveform is stored in a structure array (struct wave_parm) which was designed to be easily readable to the test programmer. The following is an example section from such a struct:

```
         PULSE      TIME    ATTEN.  CHAN.    ACTION

      {  DELAY,      100L,     1,      1,     DELAY,   },
      {  LE,       10000L,     0,      1,     PAM, },
      {  TE,         800L,     0,      1,     PAM, },
```

There are five variables within each struct (five columns). The pulse field defines the event, i.e. lead edge, trail edge, delay, etc.. The time field stores the time to wait before performing the event. The attenuator field determines which attenuator to select for the event. In order for the attenuators to switch properly in hardware, there must be a transition from one attenuator to another; hence, each test begins with an attenuator switch from 1 to 0. The channel field determines which of the three channels to use. The action field controls what kind of event to perform PAM, DELAY, REPLY (reply window), etc.. These waveforms are defined as static and are initialized when the program is first started. Any modifications to what is stored in these structures will affect the subsequent calls to these tests. The programmer should be careful.

The interrogation waveforms are user modifiable. The interrogation parameters determine certain characteristics about the waveforms (P1 width, P1-P2 spacing etc.). The function "wfmod()" is called for each pulse in the waveform. This function will adjust the values in the struct to set the pulse width and spacing according to what is stored in the interrogation parameters for the test.

As mentioned earlier, the waveform structure is only used to provide a readable way for the programmer to set waveform definitions. The interrogation waveforms must actually be stored in the DATAS transmitter memory locations in the compact binary format used by the hardware. The function "gen_pattern()" is called to translate from the structure format to the hardware format.

TRANSMITTER FREOUENCY. The transmitter frequency should be set by each test, since the frequency set by the previous test may have been some value other than what is required by the current test. The transmitter frequency for most tests (other than those that vary the frequency) is determined by the interrogation parameters. The frequency is set with a call to the function "setfreq()." The frequency should be set for each channel used by the test.

TRANSMITTER POWER. There are several functions that are used to help set the attenuator levels to control the transmitter power. The function "chk_level()" can be used to test if a desired power level is within range of the attenuator. The function "setlev()" will do even more. It will see if the desired level is within range. If it is, it will set it; if it is not, it will set it to the maximum if it is too high, or set it to the minimum if it is too low. The power level is variable by 1/10 dB steps.

TEST NUMBER: 2

FILE: aftst2.cc - ATCRBS field test #2.

TEST: Sensitivity.

DESCRIPTION: Sensitivity refers to how well the transponder is able to receive transmissions. A high sensitivity would mean that a very weak signal could be accepted and processed. A low sensitivity would mean that a very strong signal is required at the receiver before the transponder will respond. Either extreme in the air traffic environment is undesirable since a sensitivity that is too low could cause missed transmissions or too low of an operating range and too high of a sensitivity level could make the transponder too susceptible to fruit. The national standard has determined that the sensitivity must be -73 dBm +/- 4 dBm. This test will determine the minimum RF signal level required to produce a 90 percent reply efficiency for both Modes A and C. This measurement is achieved with the use of a successive approximation algorithm (suc_apprx()). Reply power and delay are measured separately at the end of the test with a short series of ATCRBS interrogations.

A sensitivity measurement relies on having a calibrated transmitter power level at the antenna of the transponder from the DATAS test system. Therefore, a true sensitivity measurement can only be made when the transponder under test's antenna is directly in the nose of the main beam. This test should only be run in the field testing environment when it is assured that the required test conditions are met, otherwise the sensitivity will merely be relative to the quality of transmission coupling. Sensitivity is also measured by the function "spandd()" (Sensitivity, Power, and Delay test) which is run at the end of the series of tests so that the aircraft can be made to roll through the calibrated center beam during these critical measurements. This is the procedure that should be used when the system operators have little or no control over the test environment.

TEST PROCEDURE: The procedure is the same for both Modes A and C. The interrogation power is initially set to -88 dBm and the power variation (the amount the power is changed by) is set to 8 dB. The transponder is interrogated a number of times (initially 20) which will increase as the test progresses. If the transponder replies greater than 90 percent the interrogation power is decreased by the "power variation" amount. If the transponder replies less than 90 percent the interrogation power is increased by the "power variation" amount. Once the transponder has replied 90 percent the variation is divided in half with each iteration. This procedure will continue until the variation has reached minimum. The sensitivity is the transmit power at the end that achieved a 90 percent reply percentage.

For each interrogation mode, the function interrogates to measure reply power and delay.

INTERROGATION PARAMETERS:

Mode A:

P1 width - int cur[2].parm[A_Pl_WDTH] determines P1 width.

P2 width - int cur[2].parm[A_P2_WDTH] determines P2 width.
P3 width - int cur[2].parm[A_P3_WDTH] determines P3 width.
P1-P2 spacing -int cur[2].parm[A_Pl_P2_SP] determines Pl-P2 spacing.
P1-P3 spacing int_cur[2].parm[A_Pl_P3_SP] determines P1-P3 spacing.
Pl P3 power - Determined by test.
P2 power - int_cur[2].parm[A_P2_PO] determines P2 power offset from Pl,P3 power.
PRF - int_cur[2].parm[A_PRF] determines PRF.
Frequency - int_cur[2].parm[A_FREQ] determines frequency.

Mode C:
P1 width - int_cur[2].parm[C_Pl_WDTH] determines P1 width.
P2 width - int_cur[2].parm[C_P2_WDTH] determines P2 width.
P3 width - int_cur[2].parm[C_P3_WDTH] determines P3 width. P1-P2 spacing - int_cur[2].parm[C_Pl_P2_SP] determines P1-P2 spacing.
P1-P3 spacing - int_cur[2].parm[C_Pl_P3_SP] determines P1-P3 spacing.
Pl,P3 Power - Determined by test.
P2 power - int_cur[2].parm[C_P2_PO] determines P2 power offset from Pl,P3 power.
PRF - int_cur[2].parm[C_PRF] determines PRF.
Frequency - int_cur[2].parm[C_FREQ] determines frequency.

TEST PARAMETERS:
tst_cur.tst2.del_ints - determines the number of interrogations to use in the delay and power measurement part of the test.
tst_cur.tst2.meas_acc - determines the measurement accuracy of the sensitivity test, either to 1 dB or 1/10 of a dB.

TEST DATA:
```
    int meas_acc;         /* Measurement accuracy  */
    int a_sens;       /* Mode A sensitivity    */
    int c_sens;       /* Mode C sensitivity    */
    int power;        /* Highest reply power measured    */
    int delay;        /* Lowest reply delay measured     */
    int e code;       /* Test error codes                  */
Data size = 12 bytes.
```

Aircraft ID: N1234   Aircraft Type: CESSNA 170 Test Time:

Transponder Type: KING KT76A

Comment: PILOT REPORTED TRANSPONDER TROUBLE

DATAS TEST #2   ATCRBS SENSITIVITY

SENSITIVITY/VARIATION	FAILURE
MODE A SENSITIVITY - 74.0 dbm	NO
MODE C SENSITIVITY - 74.0 dbm	NO
SENSITIVITY VARIATION - 0.0 dbm	NO

TEST NUMBER: 3

FILE: aftst3.cc - ATCRBS field test #3.

TEST: ATCRBS Dynamic Range.

DESCRIPTION: Dynamic range is how the transponder reacts to interrogations received at various power levels. The transponder should reply greater than 90 percent to Mode A and Mode C interrogations at all power levels between Minimum Transmit Level (MTL) and -21 dBm. MTL refers to the lowest power level that the transponder will reply greater than 90 percent.

The dynamic range test requires a calibrated transmitter power level at the antenna of the transponder from the DATAS test system. Therefore, the dynamic range test should only be run when the transponder under test's antenna is directly in the nose of the main beam. This test should only be run in the field testing environment when it is assured that the required test conditions are met, otherwise, the dynamic range will merely be relative to the quality of transmission coupling.

TEST PROCEDURE: The procedure is the same for both Modes A and C. The DATAS power level is varied from the value stored in the parameter variable "tst_cur.tst3.strt_pow" to the value stored in "tst_cur.tst3.end_pow" in increments of "tst_cur.tst3.pow_inc." The number of replies for each level is stored.

INTERROGATION PARAMETERS:
Mode A:
Pl width - int_cur[3].parm[A_Pl_WDTH] determines P1 width.
P2 width - int_cur[3].parm[A_P2_WDTH] determines P2 width.
P3 width - int_cur[3].parm[A_P3_WDTH] determines P3 width.
P1-P2 spacing - int_cur[3].parm[A_Pl_P2_SP] determines P1-P2 spacing.
P1-P3 spacing - int_cur[3].parm[A_Pl_P3_SP] determines Pl-P3 spacing.
Pl,P3 power - Determined by test.
P2 power - int_cur[3].parm[A_P2_PO] determines P2 power offset from Pl,P3 power.
PRF - int_cur[3].parm[A_PRF] determines PRF.
Frequency - int_cur[3].parm[A_FREQ] determines frequency.

Mode C:
P1 width - int cur[3].parm[C_P1_WDTH] determines P1 width.
P2 width - int cur[3].parm[C_P2_WDTH] determines P2 width.
P3 width - int cur[3].parm[C_P3_WDTH] determines P3 width.
P1-P2 spacing int cur[3].parm[C_Pl_P3_SP] determines P1-P2 spacing.
P1-P3 spacing - int_cur[3].parm[C_P1_P3_SP] determines P1-P3 spacing.
Pl.P3 power - Determined by test.
P2 power - int_cur[3].parm[C_P2_PO] determines P2 power offset from Pl,P3 power.
PRF - int_cur[3].parm[C_PRF] determines PRF.
Frequency - int_cur[3].parm[C_FREQ] determines frequency.

TEST PARAMETERS:

    tst_cur.tst3.num_ints - determines the number of interrogations to send at each power level.
    tst_cur.tst3.strt_pow - Test starting power level.
    tst_cur.tst3.end_pow - Test ending power level.
    tst cur.tst3.pow_inc - Power level increment.

TEST DATA:

```
     int num_ints;   /* Number of interrogations   */
     int strt_pow;   /* Start power                */
     int end_pow;/* End power            */
     int pow_inc;/* Power increment           */
     int rep_cnt[2][801];/* Reply count - [0][n] Mode A
                                          [l][n] Mode C     */
     int power;      /* Highest reply power measured    */
     int delay;      /* Lowest reply delay measured     */
     int e_code;     /* Test error codes                   */
```

Data size = 3218 bytes.

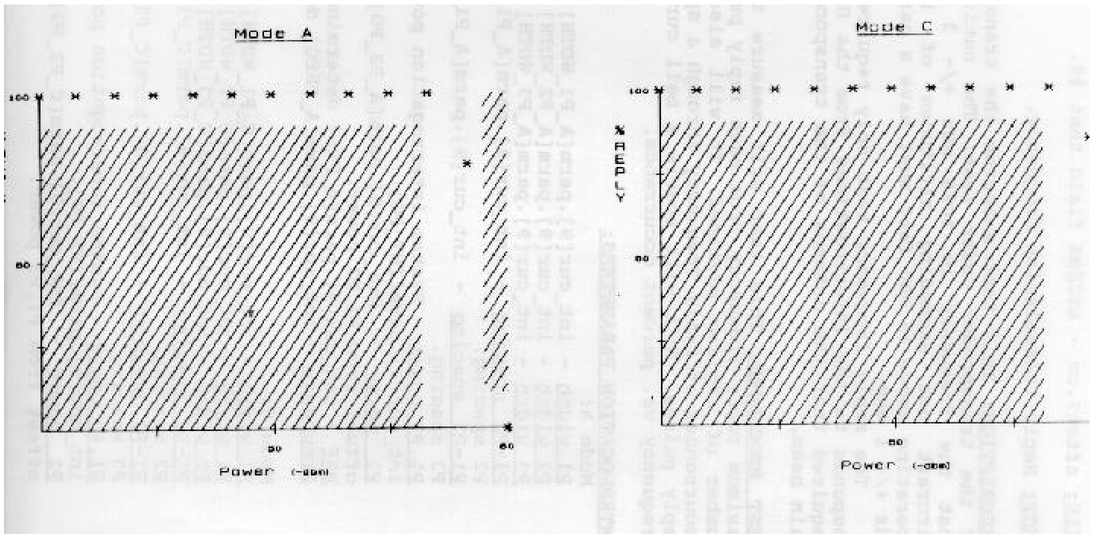
Aircraft ID: N1234   Aircraft Type: CESSNA 170 Test Time:

Transponder Type: KING KT76A   Transponder Serial#

Comment: PILOT REPORTED TRANSPONDER TROUBLE

DATAS TEST #3   DYNAMIC RANGE

100 Interrogations Per Point



TEST NUMBER: 9

FILE: aftst9.cc - ATCRBS field test #9.

TEST: Reply Transmission Frequency.

DESCRIPTION: This test measures the transmitter radio frequency of the transponder under test. The national standard requires that the frequency is 1090 mHz +/- 3 mHz for operation in aircraft not exceeding altitudes of 15,000 feet. Aircraft operating above 15,000 feet must have a carrier frequency of 1090 mHz +/- 1 mHz.

The reply frequency test only requires that the transponder responds to the interrogations from the DATAS system. It is not required that the antenna of the transponder be directly in the main beam.

TEST PROCEDURE: This test will measure the minimum, mean, and maximum reply frequency from each reply pulse from the specified number of replies examined. It will also store the number of occurrences of reply frequency through a specified range from all reply pulses. This will enable a bell curve plot of transmitter frequency vs. percent occurrence.

INTERROGATION PARAMETERS:

Mode A:

Pl width - int_cur[9].parm[A_Pl_WDTH] determines Pl width.
P2 width - int_cur[9].parm[A_P2_WDTH] determines P2 width.
P3 width - int_cur[9].parm[A_P3_WDTH] determines P3 width.
P1-P2 spacing - int_cur[9].parm[A_Pl_P2_SP] determines P1-P2 spacing.
P1-P3 spacing - int_cur[9].parm[A_Pl_P3_SP] determines P1-P3 spacing.
Pl,P3 power - System interrogation power "i_power" offset by int_cur[9].parm[INT_PO].
P2 power - int_cur[9].parm[A_P2_PO] determines P2 power offset from Pl,P3 power.
PRF - int_cur[9].parm[A_PRF] determines PRF.
Frequency - int_cur[9].parm[A_FREQ] determines frequency.

Mode C:

Pl width - int_cur[9].parm[C_P1_WDTH] determines Pl width. P2 width - int_cur[9].parm[C_P2_WDTH] determines P2 width.
P3 width - int_cur[9].parm[C_P3_WDTH] determines P3 width. P1-P2 spacing - int_cur[9].parm[C_Pl_P2_SP] determines P1-P2 spacing.
P1-P3 spacing - int_cur[9].parm[C_Pl_P3_SP] determines Pl-P3 spacing.
Pl,P3 power - System interrogation power "i_power" offset by int_cur[9].parm[INT_PO].
P2 power - int_cur[9].parm[C_P2_PO] determines P2 power offset from Pl,P3 power.
PRF - int_cur[9].parm[C_PRF] determines PRF.
Frequency - int_cur[9].parm[C_FREQ] determines frequency.

TEST PARAMETERS:

    tst_cur.tst9.num_ints determines the number of interrogations
    to send.
    tst_cur.tst9.mode - determines interrogation mode. 0=Mode A.
1=Mode C.

TEST DATA:

```
    struct T9_P_TYPE
    {
       unsigned long min;     /* Min frequency     */
       unsigned long mean;  /* Mean frequency     */
       unsigned long max;     /* Maximum frequency */
    };
    int num_replies; /* Number of replies examined */
    int mode;         /* Interrogation mode 0=A l=C     */
    struct T9_P_TYPE freq[l6]; /* Freq. array      */
    long frq_arr[l01]; /* Freq. array all pulses   */
    int power;         /* Highest reply power measured    */
    int delay;         /* Lowest reply delay measured     */
    int e_code;          /* Test error codes                  */
```

Data size = 606 bytes.

Aircraft ID: N1234 Aircraft Type: CESSNA 170 Test Time:

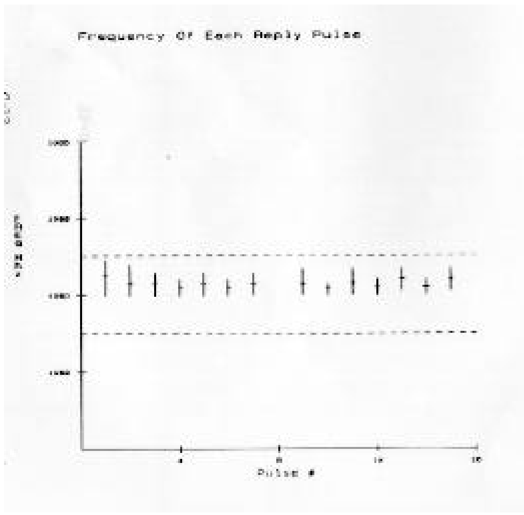
Transponder Type: KING KT76A Transponder Serial#

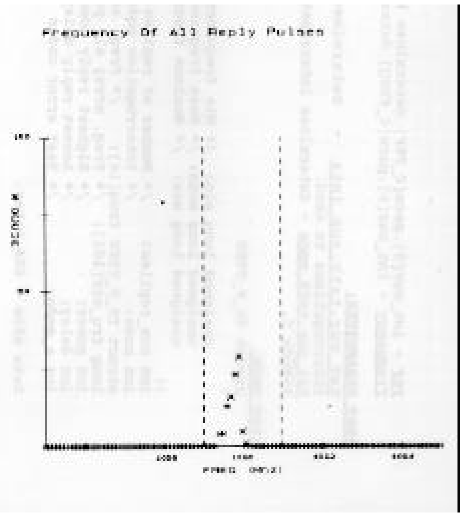
Comment: PILOT REPORT TRANSPONDER TROUBLE

DATAS TEST #9 REPLY TRANSMISSION FREQUENCY

Test Data Based on 100 Replies

Interrogation Mode A





TEST NUMBER: 10

FILE: aftstlO.cc - ATCRBS field test #l0.

TEST: ATCRBS Power Output.

DESCRIPTION: This test measures the transmitter power of the transponder under test by measuring the amplitude of the reply pulses.

A power measurement relies on having a calibrated coupling path between the DATAS system and the aircraft's transponder. Therefore, a true power measurement can only be made when the transponder under test's antenna is directly in the nose of the main beam. This test should only be run in the field testing environment when it is assured that the required test conditions are met, otherwise, the power measured will merely be relative to the quality of transmission coupling. Reply power is also measured by the function "spandd()" (Sensitivity, Power and Delay test) which is run at the end of the series of tests so that the aircraft can be made to roll through the calibrated center beam during these critical measurements. This is the procedure that should be used when the system operators have little or no control over the test environment.

However, this test does contribute some valuable information even if it is run outside of the center beam. Since it stores the reply power of each individual pulse, the relative reply power of each pulse can be compared to the others. It can also be made to run at different Pulse Repetition Frequencies (PRF's) and therefore, provide information relating reply power to transmitter duty cycle.

TEST PROCEDURE: This test has two operating modes: single PRF and multiple PRF modes. The operating mode is determined by the test parameter variable "tst_cur.tstlO.t_mode" (0=single 1=multiple). In "single PRF mode" the test will measure the reply power from a series of interrogations at a single PRF which is determined by the test parameter "tst_cur.tstl0.s_prf." This mode should be selected when a shorter test time is required; reply power vs. PRF information is not desired, or if the test is being conducted in a field environment where a high PRF could interfere with the air traffic control systems. Multiple PRF mode will conduct the same test repeated for 10 different PRF's. The PRF's are determined by the test parameters "tst_cur.tstlO.m_prf_l0" and "tst_cur.tstl0.m_prf_hi." The 10 PRF's are evenly divided between the low and high PRF.

This test will measure the minimum, mean, and maximum reply power of each reply pulse from the specified number of replies examined. It will also store the number of occurrences of reply power through a specified range from all reply pulses. This will enable a bell curve plot of reply power vs. percent occurrence.

INTERROGATION PARAMETERS:

Mode A:

P1 width - int_cur[10].parm[A_P1_WDTH] determines P1 width.

P2 width - int_cur[10].parm[A_P2_WDTH] determines P2 width.

P3 width - int_cur[10].parm[A_P3_WDTH] determines P3 width.

P1-P2 spacing - int_cur[10].parm[A_Pl_P2_SP] determines Pl-P2 spacing.

P1-P3 sPacinq - int_cur[10].parm[A_Pl_P3_SP] determines P1-

P3 spacing.
Pl.P3 Power - System interrogation power "i_power" offset by int_cur[10].parm[INT_PO].
P2 Power - int_cur[10].parm[A_P2_PO] determines P2 power offset from Pl,P3 power.
PRF - int_cur[10].parm[A_ PRF] determines PRF.
Frequency - int_cur[10].parm[A_ FREQ] determines frequency.

Mode C:
P1 width - int_cur[10].parm[C_Pl_WDTH] determines P1 width.
P2 width - int_cur[10].parm[C_P2_WDTH] determines P2 width.
P3 width - int_cur[10].parm[C_P3_WDTH] determines P3 width.
P1-P2 spacing - int_cur[10].parm[C_Pl_P2_SP] determines P1- P2 spacing.
P1-P3 spacing - int_cur[10].parm[C_Pl_P3_SP] determines P1- P3 spacing.
Pl.P3 power - System interrogation power "i_power" offset by int_cur[10].parm[INT_PO].
P2 power - int cur[10].parm[C_P2_PO] determines P2 power offset from Pl,P3 power.
PRF - int_cur[10].parm[C_ PRF] determines PRF .
Frequency - int_cur[10].parm[C_ FREQ] determines frequency.

TEST PARAMETERS:
tst_cur.tstlO.num_ints - determines the number of interrogations to send.
tst_cur.tstl0.i_mode - determines interrogation mode. 0=Mode A. 1=Mode C.
tst_cur.tstlO.t_mode - determines test mode. 0=single PRF mode, 1=Multiple PRF mode.
tst_cur.tstl0.s_prf - PRF for single PRF mode.
tst_cur.tstlO.m_prf_l0 - Starting PRF for multiple PRF mode.
tst cur.tstlO.m_Prf_hi - Ending PRF for multiple PRF mode.

TEST DATA:

```
    struct Tl0_P_TYPE
    {
    int min;     /* Min power          */
    int mean;        /* Mean power             */
    int max;         /* Maximum power      */
};
int num_replies[10];/* Number of replies examined each PRF  */
int i_mode;      /* Interrogation mode 0=A 1=C          */
int t_mode;      /* Test mode 0=single 1=multiple PRF   */
int num_prfs;        /* # PRF's run if multiple             */
int s_prf;       /* Single PRF                          */
int m_prf_lo;        /* Multiple PRF low                */
int m_prf_hi;        /* Multiple PRF high               */
struct T10_P_TYPE pow.[16][10];/* power array for 16 pulses
                                and 10 PRF's                */
int pow_arr[80][10];/* Power array all pulses           */
int power;          /* Highest reply power measured     */
int delay;          /* Lowest reply delay measured      */
int e_code;      /* Test error codes                    */

Data size = 2618 bytes.
```

Aircraft ID: N1234 Aircraft Type: CESSNA 170 Test Time:

Transponder Type: KING KT76A Transponder Serial#

Comment: PILOT REPORTED TRANSPONDER TROUBLE

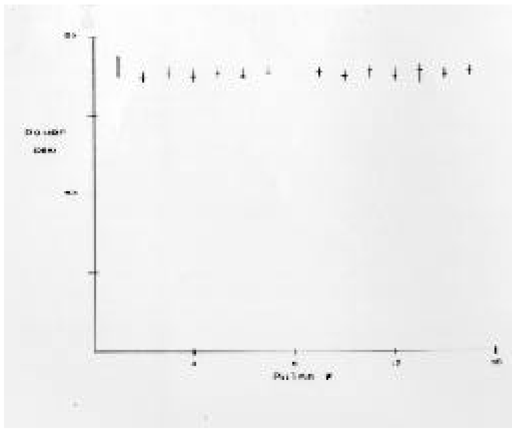
DATAS TEST #10 ATCRBS REPLY POWER

Test Data Based on 100 Replies

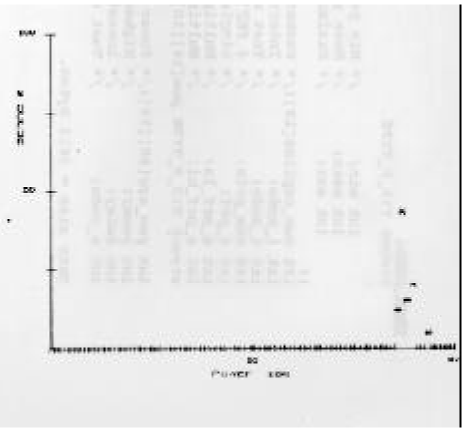
Interrogation Mode A

PRF - 235

Pilot Plot of Each Reply Pulse



Power Plot of All Reply Pulses



TEST NUMBER: 17

FILE: aftstl7 . cc - ATCRBS f ield test # 17 .

TEST: ATCRBS Reply Rate Limit and Sensitivity Reduction.

DESCRIPTION: This test determines the reply rate limit and tests the sensitivity reduction function of the transponder. Reply rate limit is the maximum number of ATCRBS replies the transponder can send in a 1-second interval. The National Standard requires that the limit can be adjusted between 500 continuous replies per second up to the maximum number of which the transponder is capable, or 2000 replies per second, whichever is lesser. Reply rate limit is a function that protects the transponder from over interrogation .

A sensitivity reduction is when the transponder responds to only the stronger interrogation signals when it is being over interrogated .

This test does not require that the transponder be at the center of the main beam; however, for best results the aircraft should at or near a stationary position for the duration of the test. This test attempts to set the interrogation power levels according to the MOPS definitions which are all relative to MTL. This test measures MTL and sets the interrogation power relative to it.

TEST PROCEDURE: This test first performs the reply rate limit test. The interrogation mode used for this test is determined by the test parameter "tst_cur.tstl7.i_mode" (0=Mode A, l=Mode C). The transponder is interrogated for 1-second intervals starting at the PRF determined by the test parameter "tst_cur.tstl7 .prf_l0" and ending at "tst_cur.tstl7 .prf_hi. "There are 16 PRF's measured which are evenly divided within this range. MTL is measured prior to this test and the power is set to 20 dB above MTL. If MTL is not measured successfully or the power can not be achieved, the power is set to the system interrogation power. The number of replies for each PRF is recorded.

The sensitivity test will interrogate for 1 second with a Mode C interrogation, at the reply rate limit determined earlier with the power level at MTL +2 0 dB, along with an asynchronous Mode A interrogation at half the reply rate limit with a power level at MTL +3 dB. The number of replies for each mode is recorded .

INTERROGATION PARAMETERS:

Mode A:

Pl width - int_cur[l7].parm[A_Pl_WDTH] determines Pl width. P2 width - int_cur[l7].parm[A_P2_WDTH] determines P2 width. P3 width - int_cur[l7].parm[A_P3_WDTH] determines P3 width. Pl-P2 spacinq - int_cur[l7].parm[A_Pl_P2_SP] determines Pl-P2 spacing.

Pl-P3 spacinq - int cur[17].parm[A_Pl_P3_SP] determines Pl-P3 spacing.

Pl.P3 power - Determined by the test.

P2 Power - int_cur[l7].parm[A_P2_PO] determines P2 power offset from Pl,P3 power.

PRF - Determined by the test.

Frequency - int_cur[17].parm[A_FREQ] determines frequency.

Mode C:

Pl width - int cur[17].parm[C_Pl_WDTH] determines Pl width.

P2 width - int cur[17].parm[C_P2_WDTH] determines P2 width.

P3 width - int cur[17].parm[C_P3_WDTH] determines P3 width.

Pl-P2 spacing - int_cur[17].parm[C_Pl_P2_SP] determines Pl-P2 spacing.

Pl-P3 spacing - int_cur[17].parm[C_Pl_P3_SP] determines Pl-P3 spacing.

Pl,P3 Power - Determined by the test.

P2 Power - int_cur[17].parm[C_P2_PO] determines P2 ower offset from Pl,P3 power.

PRF - Determined by the test.

Frequency - int_cur[l0].parm[C_FREQ] determines frequency.

TEST PARAMETERS:

tst_cur.tstl7.num_ints - determines the number of interrogations to send in the sensitivity reduction test. tst_cur.tstl7.i_mode - determines interrogation mode. 0=Mode A. l=Mode C for the reply rate limit test. tst_cur.tstl7.Prf_lo - Determines starting PRF. tst cur.tstl7.Prf hi - Determines ending PRF.

TEST DATA:
```
    struct SAMPLE_TYPE /* One for each PRF, RR limit    */
    {
         int code;          /* Mode A or C code         */
         int prf;           /* Pulse repetition frequency    */
         int reply_count; /* Number of replies          */
    };
    struct RATE_TYPE   /* Reply rate limit info         */
    {
         int i_mode;   /* Interrogation mode 0=A 1=C    */
         int prf_lo;   /* Start PRF                     */
         int prf_hi;   /* End PRF                   */
         int num_prfs;     /* Number PRF's run          */
         struct SAMPLE_TYPE sample[l6]; /* Reply info   */
         int r_r_limit,    /* Reply rate limit          */
         int mtl,          /* MTL measured              */
         int int_power;    /* Interrogation power            */
    };
    struct MODE_TYPE   /* Each mode sens. reduc.
    }
        int code;          /* Reply code                     */
        int int_power;     /* Interrogation power       */
        int prf;       /* Pulse Repetition Frequency    */
        int num_ints;      /* Number of interrogations */
        int numreps;   /* Number of replies             */
    };
    stuct SENS_TYPE        /* Sens. reduc. info
    }
        struct MODE_TYPE mode_a; /* Mode A info         */
        struct MODE_TYPE mode_c; /* Mode C info              */
    };
        struct RATE_TYPE rate; /* Reply rate info       */
        struct SENS_TYPE sens; /* Sens reduc info       */
        int power;          /* Highest reply power measured  */
        int delay;          /* Lowest reply delay measured   */
        int e_code;         /* Test error codes                */
```

Data size = 136 bytes.

Aircraft ID: N1234   Aircraft Type: CESSNA 170 Test Time:

Transponder Type: King KT76A

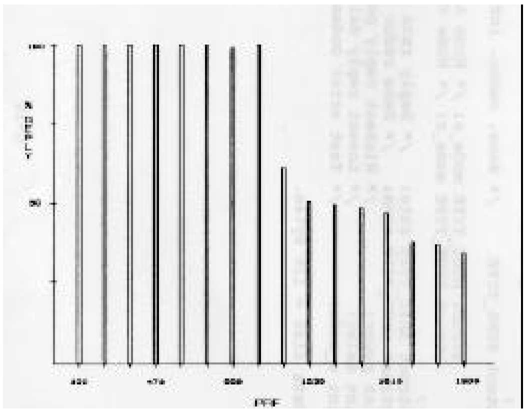
Comment: PILOT REPORTED TRANSPONDER TROUBLE

DATAS TEST #17 REPLY RATE LIMIT AND SENSITIVITY REDUCTION

CODE CHANGED DURING TEST

SENSITIVITY REDUCTION		
MODE	MODE A	MODE C
CODE	7777	0
POWER	-71	-54
PRF	493	986
% REPLY	86	100
FAILURE	YES	NO

Reply Rate Limit
Interrogation Mode A   Code - 7677



REPLY RATE LIMIT - 986

TEST NUMBER: 19

FILE: aftstl9.cc - ATCRBS field test #19.

TEST: ATCRBS Reply Pulse Spacing.

DESCRIPTION: This test measures the reply pulse spacing of each reply pulse with respect to the first reply pulse (F1) and with respect to each previous pulse. Each reply pulse shall be spaced 1.45 microseconds from the previous pulse. The national standard requires that the pulse spacing tolerances for each reply pulse with respect to the first framing pulse shall be +/- 0.10 microsecond. The pulse spacing tolerance of any pulse in the reply group with respect to any other pulse in the reply group except the first framing pulse shall be no more than +/- 0.15 microsecond.

This test does not require that the transponder is at the center of the main beam. However, if the test situation allows control of the aircraft, it would be beneficial if the ATCRBS reply code be set to 7777 or some other code that will produce the maximum number of reply pulses.

TEST PROCEDURE: This test first sets the pulse width threshold with a call to the function "pw_thresh()." This is required so that the pulse spacing measurements are made at the proper level with respect to the rise time of the reply pulses which is affected by the reply power from the transponder being tested.

The test will interrogate the transponder enough times to produce the number of replies determined by the test parameter "tst_cur.tstl9.num_ints." All reply pulses are examined with each reply. The minimum, mean, and maximum position error with respect to F1 and with the previous pulse are recorded.

INTERROGATION PARAMETERS:

Mode A:

P1 width - int_cur[l9].parm[A_Pl_WDTH] determines P1 width.
P2 width - int_cur[l9].parm[A_P2_WDTH] determines P2 width.
P3 width - int_cur[l9].parm[A_P3_WDTH] determines P3 width.
P1-P2 sPacinq - int_cur[l9].parm[A_Pl_P2_SP] determines P1- P2 spacing.
P1-P3 spacinq - int_cur[l9].parm[A_Pl_P3_SP] determines P1- P3 spacing.
Pl.P3 Power - System interrogation power "i_power" offset by int_cur[l9].parm[INT_PO].
P2 power - int_cur[l9].parm[A_P2_PO] determines P2 power offset from Pl,P3 power.
PRF - int_cur[l9].parm[A_PRF] determines PRF.
Frequency - int_cur[l9].parm[A_FREQ] determines frequency.

Mode C:

Pl width - int_cur[l9].parm[C_Pl_WDTH] determines Pl width.
P2 width - int_cur[19].parm[C_P2_WDTH] determines P2 width.
P3 width - int_cur[l9].parm[C_P3_WDTH] determines P3 width.
P1-P2 spacing - int_cur[l9].parm[C_Pl_P2_SP] determines P1- P2 spacing.

P1-P3 spacinq - int_cur[l9].parm[C_Pl_P3_SP] determines P1- P3 spacing.
Pl.P3 power - System interrogation power "i_power" offset by int_cur[l9].parm[INT_PO].
P2 power - int_cur[l9].parm[C_P2_PO] determines P2 power offset from Pl,P3 power.
PRF - int_cur[l9].parm[C_PRF] determines PRF.
Frequency - int_cur[l9].parm[C_FREQ] determines frequency.

TEST PARAMETERS:
tst_cur.tstl9.num_ints - determines the number of interrogations to send.
tst_cur.tstl9.i_mode - determines interrogation mode. 0=Mode A. 1=Mode C.
tst_cur.tstl9.prf_lo - Determines starting PRF.
tst cur.tstl9.prf_hi - Determines ending PRF.

TEST DATA:
```
    struct Tl9_P_TYPE
    {
        int min;         /* Minimum pulse error        */
        int mean;            /* Mean pulse error           */
        int max;         /* Maximum pulse error        */
    };
    int i_mode;  /* Interrogation mode 0=A 1=C */
    int num_replies;     /* Number of replies     */
    int pulse_flg[16];       /* Pulse flags 0=none 1=pulse */
    struct Tl9_P_TYPE perr_fl[16]; /* Position errors F1 ref.*/
    struct T19_P_TYPE perr_pp[16]; /* Position errors pls ref */
    int power;       /* Highest reply power measured   */
    int delay;       /* Lowest reply delay measured    */
    int e_code:      /* Test error codes               */
```

Data size = 234 bytes.

Aircraft ID: N1234 Aircraft Type: CESSNA 170 Test Time:

Transponder Type: KING KT76A

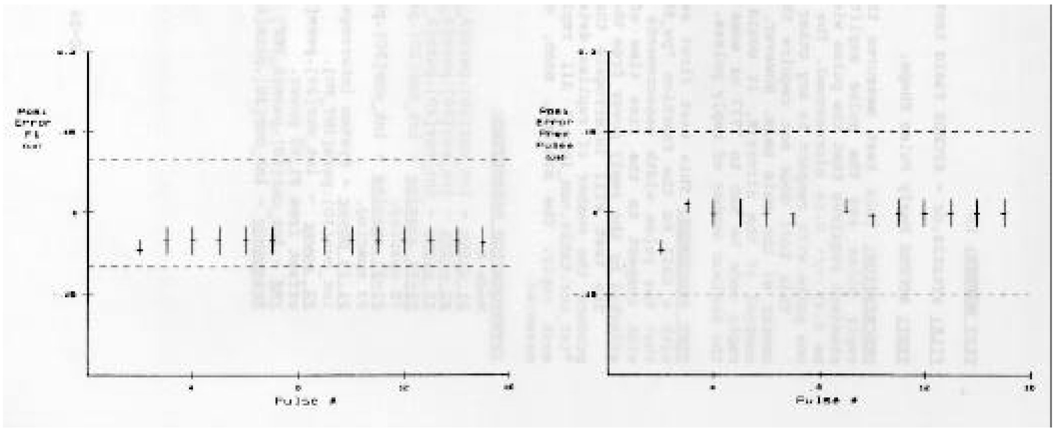
Comment: PILOT REPORTED TRANSPONDER TROUBLE

DATAS TEST #19 ATCRBS REPLY PULSE SPACING

Test Data Based on 100 Replies

Interrogation Mode A

Position Error Referenced From F1 and Previous Pulse



TEST NUMBER: 20

FILE: aftst20.cc - ATCRBS field test #20.

TEST: ATCRBS Reply Pulse Shape.

DESCRIPTION: This test measures the reply pulse width of each reply pulse and the pulse amplitude variation. The national standard requires that the pulse width for all reply pulses shall be 0.45 +/- 0.10 microsecond. The pulse amplitude variation of one pulse with respect to any other pulse shall not exceed 1 dB. This test does not require that the transponder is at the center of the main beam. However, if the test situation allows control of the aircraft, it would be beneficial if the ATCRBS reply code be set to 7777 or some other code that will produce the maximum number of reply pulses.

TEST PROCEDURE: This test first sets the pulse width threshold with a call to the function "pw_thresh()." This is required so that the pulse width measurements are made at the proper level with respect to the rise time of the reply pulses, which is affected by the reply power from the transponder being tested.

The test will interrogate the transponder enough times to produce the number of replies determined by the test parameter "tst_cur.tst20.num_ints." All reply pulses are examined with each reply. The minimum, mean, and maximum pulse widths are measured.

INTERROGATION PARAMETERS:

Mode A:
P1 width - int_cur[20].parm[A_Pl_WDTH] determines P1 width.
P2 width - int_cur[20].parm[A_P2_WDTH] determines P2 width.
P3 width - int_cur[20].parm[A_P3_WDTH] determines P3 width.
P1-P2 spacing - int_cur[20].parm[A_Pl_P2_SP] determines P1-P2 spacing.
P1-P3 sPacina - int_cur[20].parm[A_Pl_P3_SP] determines P1-P3 spacing.
Pl.P3 power - System interrogation power "i_power" offset by int_cur[20].parm[INT_PO].
P2 power - int_cur[20].parm[A_P2_PO] determines P2 power offset from Pl,P3 power.
PRF - int_cur~20].parm[A_PRF] determines PRF.
Frequency - int_cur[20].parm[A_FREQ] determines frequency.

Mode C:
P1 width - int_cur[20].parm[C_Pl_WDTH] determines Pl width.
P2 width - int_cur[20].parm[C_P2_WDTH] determines P2 width.
P3 width - int_cur[20].parm[C_P3_WDTH] determines P3 width.
P1-P2 spacinq - int_cur[20].parm[C_Pl_P2_SP] determines P1- P2 spacing.
P1-P3 sPacinq - int_cur[20].parm[C_Pl_P3_SP] determines Pl- P3 spacing.
Pl,P3 Power - System interrogation power "i_power" offset by int_cur[20].parm[INT_PO].
P2 Power - int_cur[20].parm[C_P2_PO] determines P2 power offset from Pl,P3 power.
PRF - int_cur[20].parm[C_PRF] determines PRF.
Frequency - int_cur[20].parm[C_FREQ] determines frequency.

TEST PARAMETERS:
tst_cur.tst20.num_ints - determines the number of interrogations to send.
tst_cur.tst20.i_mode - determines interrogation mode. 0=Mode A. 1=Mode C.
tst_cur.tst20.prf_lo - Determines starting PRF.
tst cur.tst20.prf_hi - Determines ending PRF.

TEST DATA:
```
    struct T20_P_TYPE
    {
        int min;     /* Minimum pulse width            */
        int mean;        /* Mean pulse width               */
        int max;     /* Maximum pulse width            */
    } ;
    int i mode;          /* Interrogation mode 0=A 1=C */
    int prf;         /* Pulse repetition frequency     */
    int num_replies; /* Number of replies          */
    int pulse_cnt[16];  /* Pulse count each pulse      */
    struct T20_P_TYPE p_width[16]; /* Position widths       */
    int p_var;           /* Power variation in dBm */
    int power;           /* Highest reply power measured*/
    int delay;           /* Lowest reply delay measured*/
    int e_code;      /* Test error codes               */
```

Data size = 142 bytes.

Aircraft ID: N1234   Aircraft Type: CESSNA 170 Test Time:

Transponder Type: KING KT76A

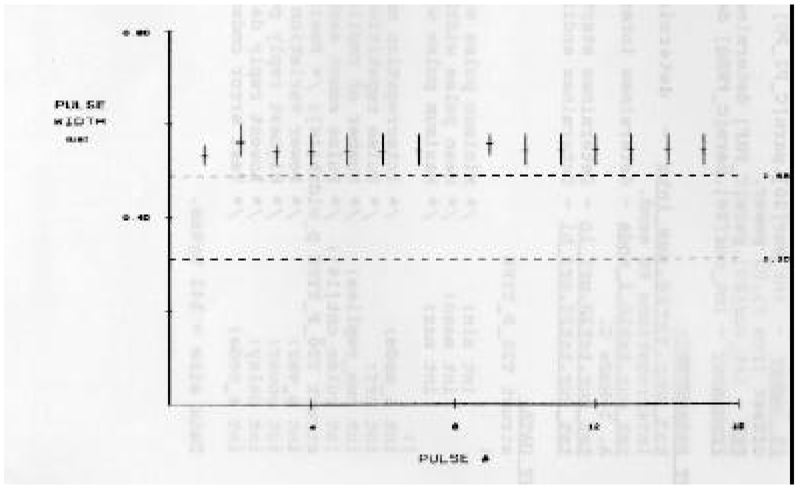
Comment: PILOT REPORTED TRANSPONDER TROUBLE

DATAS TEST #20 ATCRBS REPLY PULSE SHAPE

Test Data Based on 100 Replies

Interrogation Mode A

PRF - 450



Reply Pulse Power Variation = 1.7 dbm   FAILURE

TEST NUMBER: 22

FILE: aftst22.cc - ATCRBS field test #22.
TEST: ATCRBS Reply Delay and Jitter.

DESCRIPTION: This test measures the reply delay and reply jitter. Reply delay is defined as the time between the lead edge of the P3 pulse in the interrogation and the lead edge of the first reply pulse. The national standard requires that the transponder have a reply delay of 3.0 +/- 0.5 microseconds at all RF input levels from MTL to -21 dBm.

Reply jitter is the extreme positions of the leading edge of the reply pulse, or reply delay variation. The National Standard required that the jitter shall not exceed +/- 0.1 microsecond.

This test produces a plot of reply delay and jitter at various interrogation power levels. In order for these power levels to be accurate, the transponder being tested should have the antenna directly at the calibrated point in the main beam. However, if this cannot be guaranteed, these test results can still show the relationship between relative interrogation power and reply delay. An accurate reply delay measurement can be made when the function "spandd()" is run following the series of transponder tests.

TEST PROCEDURE: This test first sets the pulse width threshold with a call to the function "pw_thresh()." This is required so that the delay measurements are made at the proper level with respect to the rise time of the reply pulses, which is affected by the reply power from the transponder being tested.

The test will send the number of interrogations defined in the test parameter "tst_cur.tst22.num_ints" at each interrogation level. The interrogation levels to send are defined by the test parameters "tst_cur.tst22.strt_pow," "tst_cur.tst22.end_pow," and "tst_cur.tst22.pow_inc." The reply delay minimum, mean, and maximum are stored at each point.

INTERROGATION PARAMETERS:

Mode A:
P1 width - int_cur[22].parm[A_Pl_WDTH] determines P1 width.
P2 width - int_cur[22].parm[A_P2_WDTH] determines P2 width.
P3 width - int_cur[22].parm[A_P3_WDTH] determines P3 width.
P1-P2 spacing - int_cur[22].parm[A_Pl_P2_SP] determines Pl- P2 spacing.
P1-P3 spacing - int_cur[22].parm[A_Pl_P3_SP] determines P1- P3 spacing.
Pl,P3 power - Determined by test.
P2 power - int_cur[22].parm[A_P2_PO] determines P2 power offset from Pl,P3 power.
PRF - int_cur[22].parm[A_PRF] determines PRF.
Frequency - int_cur[22].parm[A_FREQ] determines frequency.

Mode C:
P1 width - int_cur[22].parm[C_Pl_WDTH] determines P1 width.
P2 width - int_cur[22].parm[C_P2_WDTH] determines P2 width.
P3 width - int_cur[22].parm[C_P3_WDTH] determines P3 width.
P1-P2 spacing - int_cur~22].parm[C_Pl_P2_SP] determines P1-P2 spacing.
P1-P3 spacing - int_cur[22].parm[C_Pl_P3_SP] determines P1-P3 spacing.
Pl P3 power - Determined by test.
P2 Power - int_cur[22].parm[C_P2_PO] determines P2 power offset from Pl,P3 power.
PRF - int_cur[22].parm[C_PRF] determines PRF.
Frequencv - int_cur[22].parm[C_FREQ] determines frequency.

TEST PARAMETERS:
tst_cur.tst22.num_ints - determines the number of interrogations to send at each power level.
tst cur.tst22.i mode - determines interrogation mode. 0=Mode A. 1=Mode C.
tst_cur.tst22.strt_pow - Starting power level.
tst_cur.tst22.end_pow - Ending power level.
tst_cur.tst22.pow_inc - Power level increment.

TEST DATA:
```
    struct T22_P_TYPE
    {
        int min;         /* Minimum reply delay            */
        int mean;        /* Mean reply delay           */
        int max;         /* Maximum reply delay            */
    } ;
    int i_mode;      /* Interrogation mode 0=A 1=C */
    int prf;             /* Pulse repetition frequency */
    int num_ints;        /* Number of interrogations       */
    int pcnt_rep[80] /* Percent reply array            */
    struct T22_P_TYPE p_delay[80]; /* Reply delays     */
    int power;       /* Highest reply power measured   */
    int delay;       /* Lowest reply delay measured    */
    int e code;  /* Test error codes                   */
```

Data size = 652 bytes.

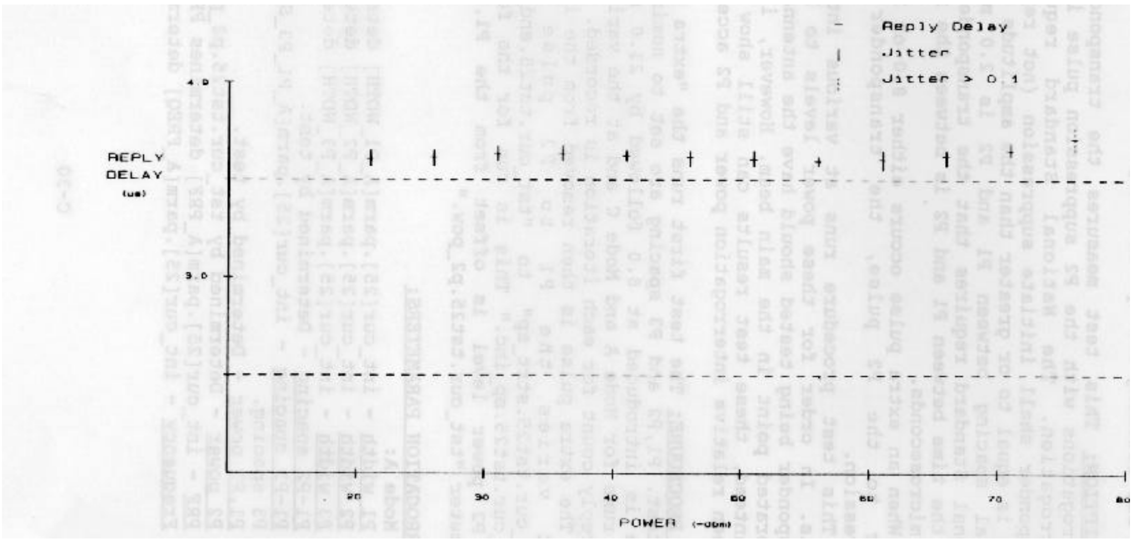
Aircraft ID: N1234 Aircraft Type: CESSNA 170 Test Time:

Transponder Type: KING KT76A Transponder Serial#

Comment: PILOT REPORTED TRANSPONDER TROUBLE

DATAS TEST #22 ATCRBS REPLY DELAY AND JITTER

100 Mode A Interrogations Sent Per Point at 450 PRF



Reply Delay - 3.62 (us)

Jitter - 0.10 (us)

90% Replies Ended at -71 dbm

TEST NUMBER: 25

FILE: aftst25.cc - ATCRBS field test #25.

TEST: SLS Decoding and dynamic range.

DESCRIPTION: This test measures the transponders response to interrogations with the P2 suppression pulse included with the interrogation. The National Standard requires that the transponder shall initiate suppression (not reply) when the P2 pulse is equal to or greater than the amplitude of P1 and P3. The nominal spacing between P1 and P2 is 2.0 microseconds. The National Standard requires that the transponder shall suppress when the time between P1 and P2 is between the range of 1.85 and 2.15 microseconds.

When an extra pulse occurs either 8.0 or 21.0 microseconds prior to the P2 pulse, the transponder shall initiate suppression.

This test procedure runs at various interrogation power levels. In order for these power levels to be accurate, the transponder being tested should have the antenna directly at the calibrated point in the main beam. However, if this cannot be guaranteed, these test results can still show the relationship between relative interrogation power and P2 acceptance.

TEST PROCEDURE: The test first runs the "extra pulse" portion of the test. Pl,P2 and P3 spacing are set to nominal and the extra pulse is introduced at 8.0 followed by 21.0 microseconds. The test runs for Mode A and Mode C and at the various power levels. The reply count for each iteration is recorded.

The extra pulse is then removed from the interrogation. The test varies the P1 to P2 pulse spacing from "tst_cur.tst25.strt_sp" to "tst_cur.tst25.end_sp," varied by "tst_cur.tst25.sp_inc." This is run for the four power levels. The P2 power level is offset from the Pl,P3 level by the parameter "tst_cur.tst25.p2_pow."

INTERROGATION PARAMETERS:

Mode A:
P1 width - int_cur[25].parm[A_P1_WDTH] determines P1 width.
P2 width - int_cur[25].parm[A_P2_WDTH] determines P2 width.
P3 width - int_cur[25].parm[A_P3_WDTH] determines P3 width.
P1-P2 spacinq - Determined by test.
P1-P3 sPacina - int_cur[25].parm[A_Pl_P3_SP] determines P1- P3 spacing.
Pl P3 Power - Determined by test.
P2 power - Determined by tst_cur.tst25.p2 pow.
PRF - int_cur[25].parm[A_PRF] determines PRF.
Frequency - int_cur[25].parm[A_FREQ] determines frequency.

Mode C:
P1 width - int_cur[25].parm[C_Pl_WDTH] determines P1 width.
P2 width - int_cur[25].parm[C_P2_WDTH] determines P2 width.
P3 width - int_cur[25].parm[C_P3_WDTH] determines P3 width.
P1-P2 spacinq - Determined by test.
P1-P3 spacinq - int_cur[25].parm[C_Pl_P3_SP] determines P1-P3 spacing.
Pl P3 power - Determined by test.
P2 Power - Determined by tst_cur.tst25.p2_pow.
PRF - int_cur[25].parm[C_PRF] determines PRF.
Frequency - int_cur[25].parm[C_FREQ] determines frequency.

TEST PARAMETERS:
tst_cur.tst25.num_ints - determines the number of interrogations to send at each power level.
tct cur.tst2s.i mode - determines interrogation mode. 0=Mode A. 1=Mode C.
tst_cur.tst25.strt_sp - Starting P1-P2 spacing.
tst_cur.tst25.end_sp - Ending P1-P2 spacing.
tst_cur.tst25.sp_inc - Spacing increment. tst cur.tst25.p2_Pow - P2 power offset from Pl.

TEST DATA:
```
    int i mode;      /* Interrogation mode 0=A 1=C      */
    int prfi             /* Pulse repetition frequency      */
    int num_ints;        /* Number of interrogations        */
    int strt_sp;         /* Starting P1-P2 spacing      */
    int end_sp;      /* Ending P1-P2 spacing            */
    int sp_inc;      /* P1-P2 spacing increment         */
    int level[4];        /* Power levels                */
    int xl_pulse_sp; /* Extra pulse spacing #l     */
    int x2_pulse_sp; /* Extra pulse spacing #2     */
    int xp_rep_cnt[4][2][2]; /* X-pulse reply count array
                         level x spacing x which reply*/
    int rep_cnt[4][161]; /* Percent reply array             */
    int power;          /* Highest reply power measured     */
    int delay;          /* Lowest reply delay measured      */
    int e_code; /* Test error codes                    */
```

Data size = 1352 bytes.

Aircraft ID: N1234   Aircraft Type: CESSNA 170   Test Time:

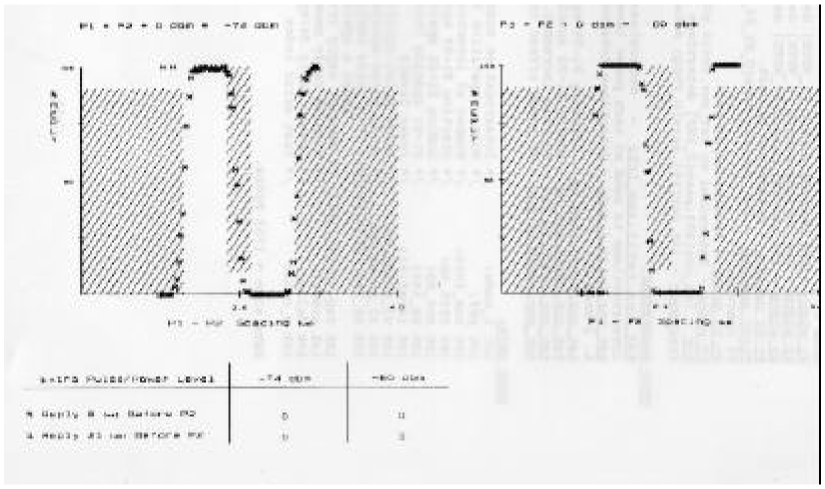
Transponder Type: KING KT76A   Transponder Serial#

Comment: PILOT REPORT TRANSPONDER TROUBLE

DATAS TEST #25 SLS DECODING AND DYNAMIC RANGE

100 Mode A Interrogations Per Point

PRF = 450



Extra Pulse/Power Level	-74 dbm	-60 dbm
% Reply 8 µs Before P2	0	0
% Reply 21 µs Before P2	0	3

Aircraft ID: N1234 Aircraft Type: CESSNA 170 Test Time:

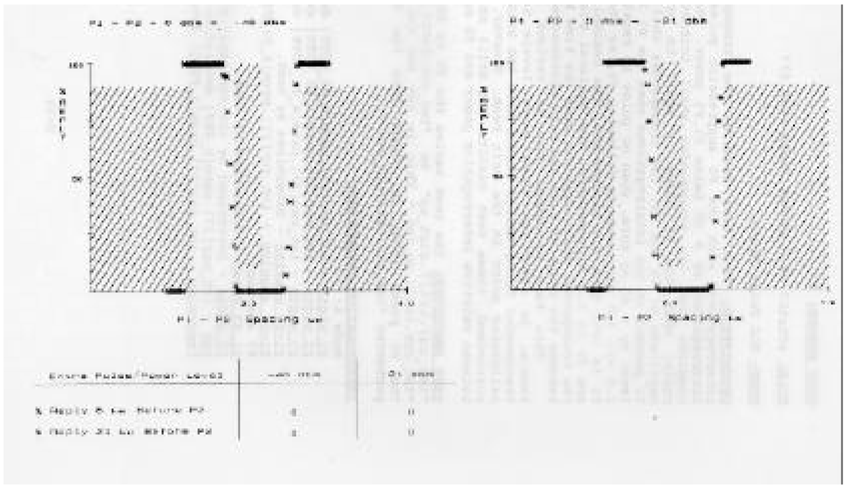
Transponder Type: KING KT76A Transponder Serial#

Comment: PILOT REPORT TRANSPONDER TROUBLE

DATAS TEST #25 SLS DECODING AND DYNAMIC RANGE

100 Mode A Interrogations Per Point

PRF = 450



Extra Pulse/Power Level	-65 dBm	-21 dBm
% Reply 8 us Before P2	0	0
% Reply 21 us Before P2	0	0

TEST NUMBER: 27

FILE: aftst27.cc - ATCRBS field test #27.

TEST: SLS Pulse Ratio.

DESCRIPTION: This test measures the transponders response to interrogations with the P2 suppression pulse included with the interrogation at 9 dB below Pl,P3 power. The National Standard requires that the transponder shall initiate suppression (not reply) when the P2 pulse is equal to or greater than the amplitude of P1 and P3. The transponder shall reply to at least 90 percent of the interrogations when the P1 level exceeds the P2 level by 9 dB or more, when no pulse is received at the position 2.0 +/- 0.7 microseconds following Pl, or when the duration of P2 is less than 0.3 microsecond. The nominal spacing between Pl and P2 is 2.0 microseconds. The National Standard requires that the transponder shall suppress when the time between P1 and P2 is between the range of 1.85 and 2.15 microseconds.

This test procedure runs at various interrogation power levels. In order for these power levels to be accurate, the transponder being tested should have the antenna directly at the calibrated point in the main beam. However, if this cannot be guaranteed, these test results can still show the relationship between relative interrogation power and P2 acceptance.

TEST PROCEDURE: The test varies the P1 to P2 pulse spacing from "tst_cur.tst27.strt_sp" to "tst_cur.tst27.end_sp" varied by "tst_cur.tst27.sp_inc." This is run for the four power levels. The P2 power level is offset from the Pl,P3 level by the parameter "tst_cur.tst27.p2_pow."

INTERROGATION PARAMETERS:

Mode A:
P1 width - int_cur[27].parm[A_Pl_WDTH] determines Pl width.
P2 width - int_cur[27].parm[A_P2_WDTH] determines P2 width.
P3 width - int_cur[27].parm[A_P3_WDTH] determines P3 width.
P1-P2 spacing - Determined by test.
P1-P3 spacing - int_cur[27].parm[A_Pl_P3_SP] determines P1- P3 spacing.
Pl.P3 power - Determined by test.
P2 power - Determined by tst_cur.tst27.p2_pow.
PRF - int_cur[27].parm[A_PRF] determines PRF.
Frequency - int_cur[27].parm[A_FREQ] determins frequency.

Mode C:
P1 width - int_cur[27].parm[C_Pl_WDTH] determines P1 width.
P2 width - int_cur[27].parm[C_P2_WDTH] determines P2 width.
P3 width - int_cur[27].parm[C_P3_WDTH] determines P3 width.
P1-P2 spacing - Determined by test.
P1-P3 spacing - int_cur[27].parm[C_Pl_P3_SP] determines Pl- P3 spacing.
Pl P3 power - Determined by test.
P2 power - Determined by tst_cur.tst27.p2_pow.
PRF - int_cur[27].parm[C_PRF] determines PRF.
Frequency - int_cur[27].parm[C_FREQ] determines frequency.

TEST PARAMETERS:

tst_cur.tst27.num_ints - determines the number of interrogations to send at each power level.
tst cur.tst27.i_mode - determines interrogation mode. 0=Mode A. 1=Mode C.
tst_cur.tst27.strt_sp - Starting P1-P2 spacing.
tst_cur.tst27.end_sp - Ending P1-P2 spacing.
tst_cur.tst27.sp_inc - Spacing increment. tst_cur.tst27.p2_pow - P2 power offset from P1.

TEST DATA:

```
    int i_mode;      /* Interrogation mode 0=A 1=C      */
    int prf;              /* Pulse repetition frequency      */
    int num_ints;         /* Number of interrogations        */
    int strt_sp;          /* Starting P1-P2 spacing     */
    int end_sp;      /* Ending P1-P2 spacing            */
    int sp_inc;      /* P1-P2 spacing increment         */
    int level[4];         /* Power levels               */
    int rep_cnt[4][161];/* Percent reply array               */
    int power;         /* Highest reply power measured  */
    int delay;         /* Lowest reply delay measured   */
    int e_code;        /* Test error codes              */
```

Data size = 1316 bytes.

Aircraft ID: N1234 Aircraft Type: CESSNA 170 Test Time:

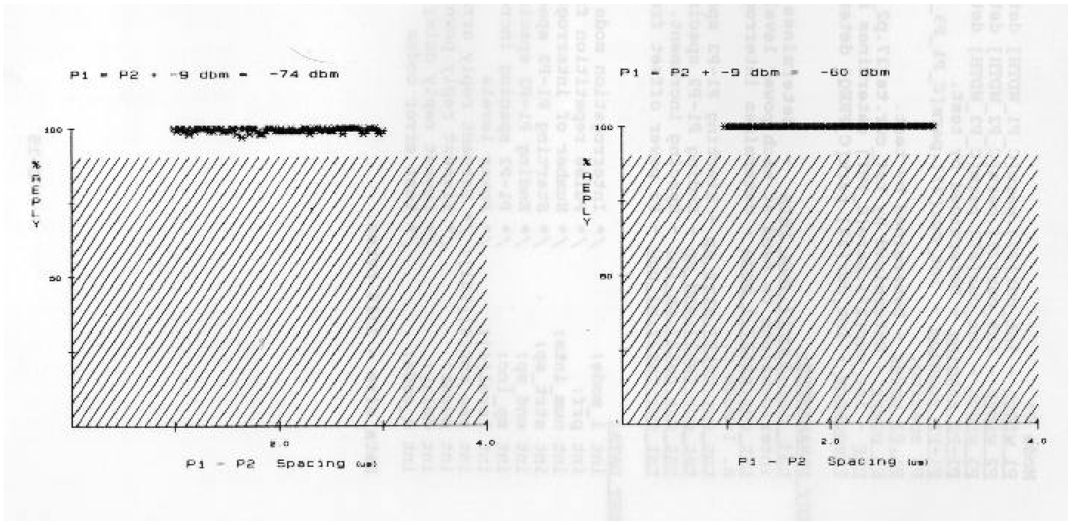
Transponder Type: KING KT76A Transponder Serial#

Comment: PILOT REPORT TRANSPONDER TROUBLE

DATAS TEST #27 SLS PULSE RATIO

100 Mode A Interrogations Per Point

PRF = 450



Aircraft ID: N1234   Aircraft Type: CESSNA 170   Test Time:

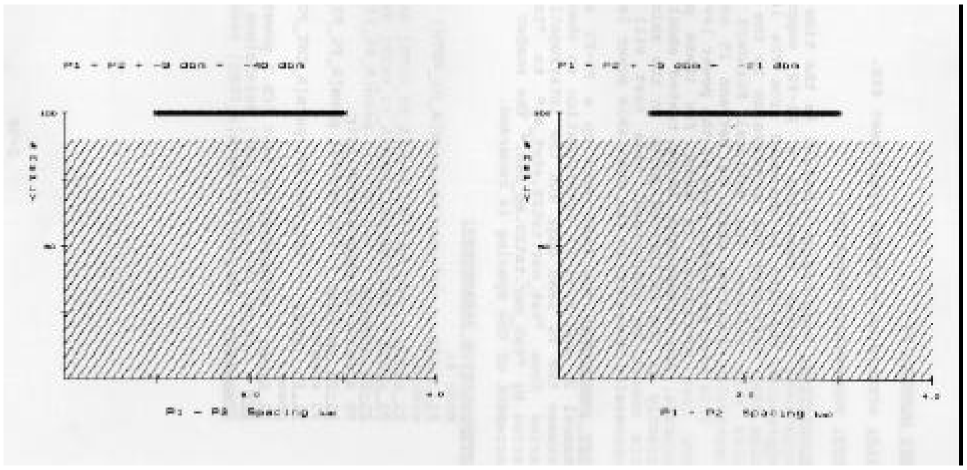
Transponder Type: KING KT76A   Transponder Serial#

Comment: PILOT REPORT TRANSPONDER TROUBLE

DATAS TEST #25 SLS PULSE RATIO

100 Mode A Interrogations Per Point

PRF = 450



TEST NUMBER- 29

FILE: aftst29.cc - ATCRBS field test #29.

TEST: Suppression Duration.

DESCRIPTION: Suppression duration is the time the transponder is suppressed after receiving a Pl-P2 suppression pair. The suppression duration is measured from the lead-edge of the P2 pulse that initiated the suppression to the lead edge of a P1 pulse that follows. The National Standard requires that the suppression duration shall be between 25 and 45 microseconds. This test procedure runs at the power levels required in the MOPS test procedure. In order for these power levels o be accurate, the transponder being tested should have the antenna directly at the calibrated point in the main beam. However, if this cannot be guaranteed, this test will still measure the suppression duration at the available power level.

TEST PROCEDURE: The test sets up a Pl-P2 suppression palr on channel 2, and a Pl-P3 interrogation on channel 1. The spacing between the P2 lead edge and the interrogation Pl lead edge is varied from "tst_cur.tst29.strt_sp" to "tst_cur.tst29.end_sp" varied by "tst_cur.tst29.sp_dec." The number of replies at each increment of the spacing is recorded.

INTERROGATION PARAMETERS:

Mode A:

P1 width - int_cur[29].parm[A_Pl_WDTH] determines both P1 widths.

P2 width - int_cur[29].parm[A_P2_WDTH] determines P2 width.

P3 width - int_cur[29].parm[A_P3_WDTH] determines P3 width.

Pl-P2 spacing - int_cur[29].parm[A_Pl_P2_SP] determines P1- P2 spacing of suppression pair.

Pl-P3 spacing - i-nt_cur[29].parm[A_Pl_P3_SP] determines P1- P3 spacing.

Pl,P3 power - int_cur[29].parm[A_INT_PO] determines P1- P3 power.

P2 Power - Determined by test (Ch.2 power = Ch.1 power).

RF - int_cur[29].parm[A_PRF] determines PRF.

Frequency - int_cur[29].parm[A_FREQ] determines frequency.

Mode C:
<u>P1 width</u> - int_cur[29].parm[C_Pl_WDTH] determines both Pl widths.
<u>P2 width</u> - int_cur[29].parm[C_P2_WDTH] determines P2 width.
<u>P3 width</u> - int_cur[29].parm[C_P3_WDTH] determines P3 width.
<u>Pl-P2 spacing</u> - int_cur[29].parm[C_Pl_P2_SP] determines P1- P2 spacing of suppression pair.
<u>Pl-P3 spacing</u> - int_cur[29].parm[C_Pl_P3_SP] determines Pl- P3 spacing.
<u>Pl.P3 power</u> - int_cur[29].parm[C_INT_PO] determines P1-P3 power.
<u>P2 Power</u> - Determined by test (Ch.2 power = Ch.1 power).
<u>PRF</u> - int_cur[29].parm[C_PRF] determines PRF.
<u>Frequency</u> - int_cur[29].parm[C_FREQ] determines frequency.

<u>TEST PARAMETERS:</u>
<u>tst_cur.tst29.num_ints</u> - determines the number of interrogations to send at each spacing.
<u>tst_cur.tst29.strt_sp</u> - Starting suppression spacing.
<u>tst_cur.tst29.end_sp</u> - Ending suppression spacing.
<u>tst_cur.tst29.sp_dec</u> - Spacing increment.

<u>TEST DATA:</u>
```
     int prf[2];      /* Pulse rep. frequency (each mode)    */
     int num_ints;    /* Number of interrogations            */
     int i_power[2];/* Interrogation power (each mode)  */
     long strt_sp;    /* Starting spacing                    */
     long end_sp;     /* Ending spacing                      */
     long sp_dec;     /* Pl-P2 spacing decrement             */
     int rep_cnt[2][1201]; /* Reply count array              */
     int power;       /* Highest reply power measured   */
     int delay;       /* Lowest reply delay measured    */
    int e code;  /* Test error codes                         */
```

Data size = 4832 bytes.

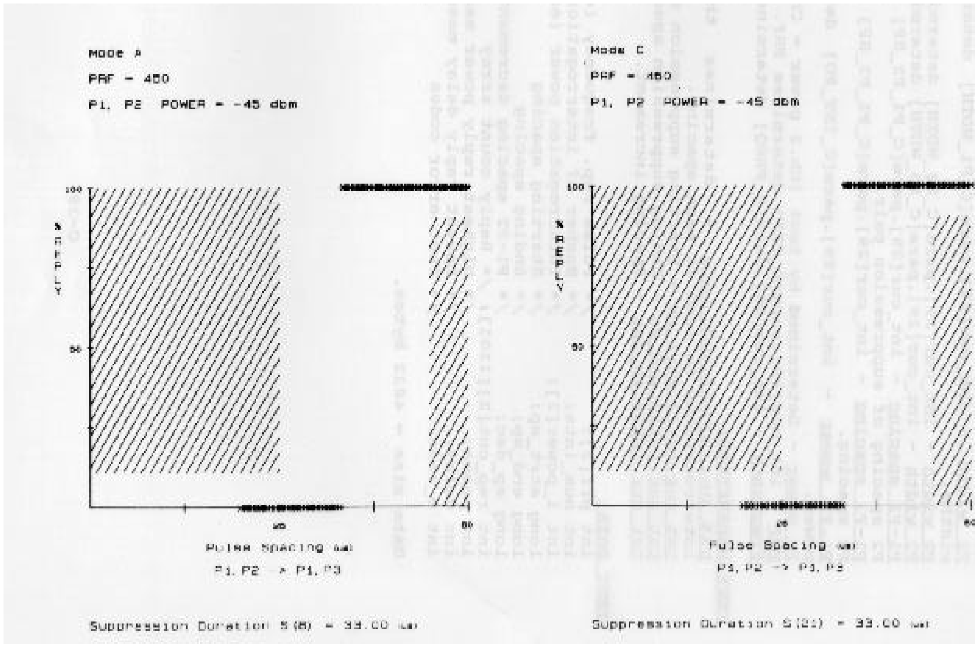
Aircraft ID: N1234 Aircraft Type: CESSNA 170 Test Time:

Transponder Type: KING KT76A Transponder Serial#

Comment: PILOT REPORT TRANSPONDER TROUBLE

DATAS TEST #29 SUPPRESSION DURATION

100 Mode A Interrogations Per Point



TEST NUMBER: 30

FILE: aftst30.cc - ATCRBS field test #30.

TEST: Suppression Reinitiation.

DESCRIPTION: Suppression reinitiation is the ability of the transponder to initiate suppression immediately following a previous suppression. The National Standard requires that the transponder shall be capable of reinitiating suppression within 2 microseconds after a suppression period.

This test procedure runs at the power levels required in the MOPS test procedure. In order for these power levels to be accurate, the transponder being tested should have the antenna directly at the calibrated point in the main beam. However, if this cannot be guaranteed, this test will still measure the suppression duration at the available power level.

TEST PROCEDURE: The test sets up a Pl,P2 suppression pair followed by another Pl,P2 suppression pair on channel 2, and Pl,P3 interrogation on channel 1. The spacing between the first Pl,P2 pair and the second Pl,P2 pair is set to the suppression duration plus 2 microseconds. The suppression duration is measured with a call to the function "getsd()." The spacing between the P2 lead edge of the second Pl,P2 pair and the interrogation Pl lead edge is varied from "tst_cur.tst30.strt_sp" to "tst_cur.tst30.end_sp" varied by "tst_cur.tst30.sp_dec." The number of replies at each increment of the spacing is recorded.

INTERROGATION PARAMETERS:

Mode A:

Pl width - int_cur[30].parm[A_Pl_WDTH] determines all three Pl widths.

P2 width - int_cur[30].parm[A_P2_WDTH] determines both P2 widths.

P3 width - int_cur[30].parm[A_P3_WDTH] determines P3 width.

Pl-P2 spacing - int_cur[30].parm[A_Pl_P2_SP] determines Pl-P2 spacing of both suppression pairs.

Pl-P3 spacinq - int_cur[30].parm[A_Pl_P3_SP] determines Pl- P3 spacing.

Pl.P3 power - int_cur[30].parm[A_INT_PO] determines Pl-P3 power.

P2 power - Determined by test (Ch.2 power = Ch.l power).

PRF - int_cur[30].parm[A_PRF] determines PRF.

Frequency - int_cur[30].parm~A_FREQ] determines frequency.

Mode C:

Pl width - int_cur[30].parm[C_Pl_WDTH] determines all three Pl widths.
P2 width - int_cur[30].parm[C_P2_WDTH] determines both P2 widths.
P3 width - int_cur[30].parm[C_P3_WDTH] determines P3 width.
Pl-P2 spacing - int_cur[30].parm[C_Pl_P2_SP] determines P1-P2 spacing of both suppression pairs.
Pl-P3 spacing - int_cur[30].parm[C_Pl_P3_SP] determines P1-P3 spacing.
Pl,P3 Power - int_cur[30].parm[C_INT_PO] determines P1-P3 power.
P2 Power - Determined by test (Ch.2 power = Ch.1 power).
PRF - int_cur[30].parm[C_PRF] determines PRF.
Frequency - int_cur[30].parm[C_FREQ] determines frequency.

TEST PARAMETERS:
tst_cur.tst30.num_ints - determines the number of interrogations to send at each spacing.
tst_cur.tst30.strt_sp - Starting suppression spacing.
tst_cur.tst30.end_sp - Ending suppression spacing.
tst_cur.tst30.sp_dec - Spacing increment.

TEST DATA:
```
int prf[2];        /* Pulse rep. frequency (each mode)    */
int num_ints;  /* Number of interrogations           */
int i_power[2];    /* Interrogation power (each mode)*/
long strt_sp;  /* Starting spacing                   */
long end_sp;   /* Ending spacing                 */
long sp_dec;   /* P1-P2 spacing decrement            */
long s[2];     /* Suppression durations              */
int rep_cnt[2][1201]; /* Reply count array           */
int power;     /* Highest reply power measured       */
int delay;     /* Lowest reply delay measured        */
int e_code;        /* Test error codes                    */
```

Data size = 4838 bytes.

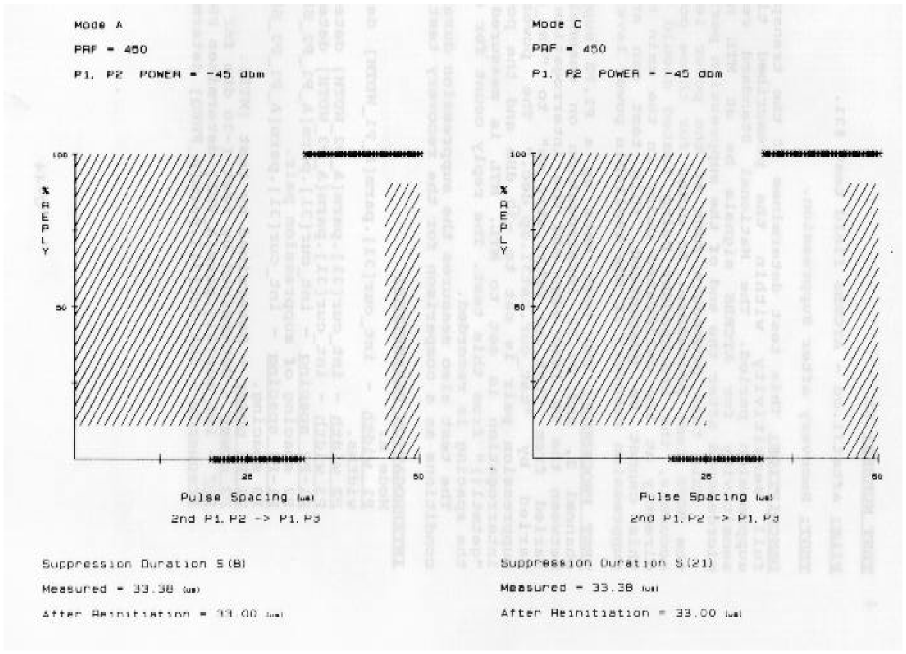
Aircraft ID: N1234 Aircraft Type: CESSNA 170 Test Time:

Transponder Type: KING KT76A Transponder Serial#

Comment: PILOT REPORT TRANSPONDER TROUBLE

DATAS TEST #30 SUPPRESSION REINITIATION

100 Interrogations Per Point



TEST NUMBER: 31

FILE: aftst31.cc - ATCRBS field test #31.

TEST: Recovery after Suppression.

DESCRIPTION: This test determines if the transponder returns to full sensitivity within the prescribed time following a suppression period. The National Standard requires that the sensitivity for ATCRBS signals be at MTL no later than microsecond after the end of the suppression period.

This test procedure runs at the power levels required in the MOPS test procedure. In order for these power levels to be accurate, the transponder being tested should have the antenna directly at the calibrated point in the main beam. However, if this cannot be guaranteed, this test can still measure the suppression recovery if the available power level is adequate.

TEST PROCEDURE: The test sets up a Pl,P2 suppression palr on channel 2f and Pl,P3 interrogation on channel 1. The spacing between the P2 lead edge and the interrogation Pl lead edge is varied from "tst_cur.tst31.strt_sp" to "tst_cur.tst31.end_sp" varied by "tst_cur.tst31.sp_dec." The power level of the suppression pair is set to -30 dBm and the power level of the interrogation is set to MTL. MTL is measured with a call to "getmtl()" from this test. The reply count for each increment of the spacing is recorded.

The test also measures the suppression duration with nominal conditions as a comparison for the recovery test.

INTERROGATION PARAMETERS:

Mode A:

Pl width - int_cur[31].parm[A_Pl_WDTH] determines both P1 widths.
P2 width - int_cur[31].parm[A_P2_WDTH] determines P2 width.
P3 width - int_cur[31].parm[A_P3_WDTH] determines P3 width.
Pl-P2 spacing - int_cur[31].parm[A_Pl_P2_SP] determines P1-P2 spacing of suppression pair.
Pl-P3 spacing - int_cur[31].parm[A_Pl_P3_SP] determines P1-P3 spacing.
Pl.P3 Power - Determined by test (MTL).
P2 power - Determined by test (-30 dBm for Pl,P2).
PRF - int_cur[31].parm[A_PRF] determines PRF.
Frequency - int_cur[31].parm[A_FREQ] determines frequency.

Mode C:
<u>Pl width</u> - int_cur[31].parm[C_Pl_WDTH] determines both Pl widths.
<u>P2 width</u> - int_cur[31].parm[C_P2_WDTH] determines P2 width.
<u>P3 width</u> - int_cur[31].parm[C_P3_WDTH] determines P3 width.
<u>Pl-P2 sPacinq</u> - int_cur[31].parm[C_Pl_P2_SP] determines Pl-P2 spacing of suppression pair.
<u>Pl-P3 spacinq</u> - int_cur[31].parm[C_Pl_P3_SP] determines Pl-P3 spacing.
<u>Pl,P3 power</u> - Determined by test (MTL).
<u>P2 Power</u> - Determined by test (-30 dBm for Pl,P2).
<u>PRF</u> - int_cur[31].parm[C_PRF] determines PRF.
<u>Frequency</u> - int_cur[31].parm[C_FREQ] determines frequency.

<u>TEST PARAMETERS:</u>
<u>tst_cur.tst31.num_ints</u> - determines the number of interrogations to send at each spacing.
<u>tst_cur.tst31.strt_sp</u> - Starting suppression spacing.
<u>tst_cur.tst31.end_sp</u> - Ending suppression spacing.
<u>tst_cur.tst31.sp_dec</u> - Spacing increment.

<u>TEST DATA:</u>
```
     int prf[2];          /* Pulse rep. frequency (each mode)*/
     int mtl[2];          /* MTL measured each mode         */
     int sd[2];           /* Measured Suppression durations */
     int num_ints;        /* Number of interrogations       */
    int sup_power[2];     /* Suppression pair powers        */
    long strt_sp;         /* Starting spacing               */
    long end sp;          /* Ending spacing             */
    long sp=ec;      /* Pl-P2 spacing decrement         */
    int rep_cnt[2][1201]; /* Reply count array          */
    int power;       /* Highest reply power measured     */
    int delay;       /* Lowest reply delay measured      */
    int e code:      /* Test error codes                */
```

Data size = 4846 bytes.

Aircraft ID: N1234   Aircraft Type: CESSNA 170   Test Time:

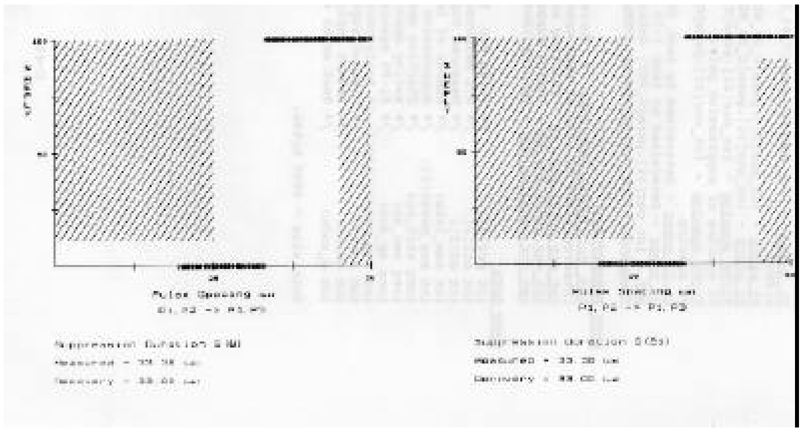
Transponder Type: KING KT76A   Transponder Serial#

Comment: PILOT REPORT TRANSPONDER TROUBLE

DATAS TEST #31 RECOVERING AFTER SUPPRESSION

100 Interrogations Per Point

Mode A	Mode C
PRF = 450	PRF = 450
MTL = -74 dbm	MTL = -74 dbm
P1.P2 SUPPRESSION POWER = -30 dbm	p1. P2 SUPPRESSION POWER= -30 dbm



TEST NUMBER: 34

FILE: aftst34.cc - ATCRBS field test #34.

TEST: Pulse Position Tolerances.

DESCRIPTION: This test determines how the transponder responds to interrogations with the Pl-P3 pulse spacings varied within the required acceptance range as well as beyond the acceptance range. The National Standard requires that the transponder accept the interrogation as valid if the spacing between the Pl and P3 pulse is within +/- 0.2 microsecond of the nominal spacing. The transponder shall not accept the interrogation if the Pl and P3 pulse spacing differs from nominal by 1.0 microsecond or more.
This test procedure runs at the power levels required in the MOPS test procedure. In order for these power levels to be accurate, the transponder being tested should have the antenna directly at the calibrated point in the main beam. However, if this cannot be guaranteed, this test can still measure the pulse position tolerances if the available power level is adequate.

TEST PROCEDURE: The test sets up a standard ATCRBS interrogation. The spacing between the P1 and P3 lead edges is varied from "tst_cur.tst34.strt_sp" to "tst_cur.tst34.end sp" varied by "tst_cur.tst34.sp inc." The power level of the interrogation is set to MTL + 10 dBm. MTL is measured with a call to "getmtl()" from this test. The reply count for each increment of the spacing is recorded.

INTERROGATION PARAMETERS:
Mode A:
Pl width - int_cur[34].parm[A_Pl_WDTH] determines Pl width.
P2 width - int_cur[34].parm[A_P2_WDTH] determines P2 width.
P3 width - int_cur[34].parm[A_P3_WDTH] determines P3 width.
Pl-P2 spacing - int_cur[34].parm[A_Pl_P2_SP] determines Pl- P2 spacing.
Pl-P3 spacing - Determined by test.
Pl.P3 power - Determined by test (MTL + 10 dB).
P2 power - int_cur[34].parm[A_P2_PO] determines P2 power. PRF - int_cur[34].parm[A_PRF] determines PRF.
Frequency - int_cur[34].parm[A_FREQ] determines frequency.

Mode C:
Pl width - int_cur[34].parm[C_Pl_WDTH] determines Pl width.
P2 width - int_cur[34].parm[C_P2_WDTH] determines P2 width.
P3 width - int_cur[34].parm[C_P3_WDTH] determines P3 width.
Pl-P2 spacing - int_cur[34].parm[C_Pl_P2_SP] determines Pl- P2 spacing.
Pl-P3 sPacinq - Determined by test.
Pl.P3 power - Determined by test (MTL + 10 dB).
P2 Power - int_cur[34].parm[C_P2_PO] determines P2 power. PRF - int_cur[34].parm[C_PRF] determines PRF.
Frequency - int_cur[34].parm[C_FREQ] determines frequency.

TEST PARAMETERS:
tst_cur.tst34.num_ints - determines the number of interrogations to send at each spacing. tst_cur.tst34.strt_sp - Starting pulse spacing. tst_cur.tst34.end_sp - Ending pulse spacing. tst_cur.tst34.sP_inc - Spacing increment.

TEST DATA:
```
     int num_ints;        /* Number of interrogations        */
     int mtl[2];      /* MTL measured each mode     */
     int i_power[2];  /* Interrogation powers           */
     long strt_sp;        /* Starting spacing                */
     long end_sp;     /* Ending spacing             */
     long sp_inc;     /* Pl-P2 spacing increment        */
    int rep_cnt[2][401]; /* Reply count array           */
    int power;       /* Highest reply power measured    */
    int delay;       /* Lowest reply delay measured     */
    int e_code;      /* Test error codes               */
```

Data size = 1628 bytes.

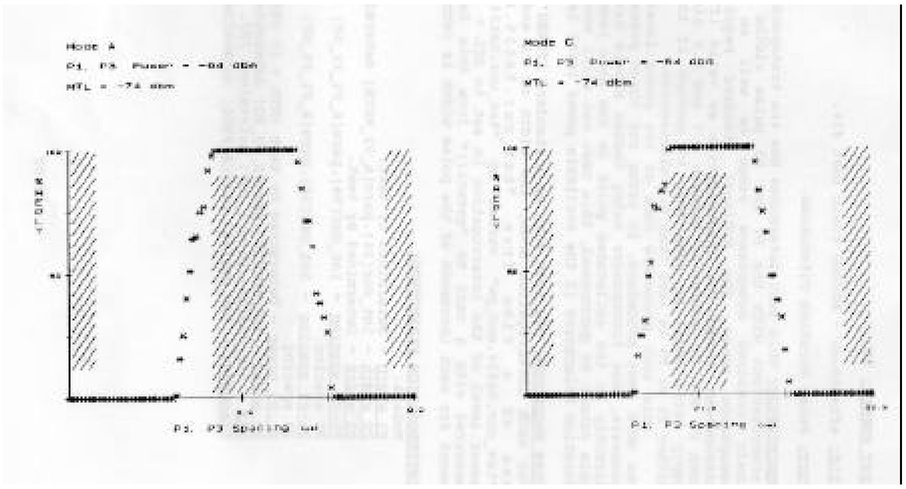
Aircraft ID: N1234   Aircraft Type: CESSNA 170   Test Time:

Transponder Type: KING KT76A   Transponder Serial#

Comment: PILOT REPORT TRANSPONDER TROUBLE

DATAS TEST #34 PULSE POSITION TOLERANCES

100 Interrogations Per Point



TEST NUMBER: 36

FILE: aftst36.cc - ATCRBS field test #36.

TEST: Pulse Duration Tolerances.

DESCRIPTION: This test determines how the transponder responds to interrogations with the P1 and P3 pulse widths varied within their required acceptance range as well as beyond their acceptance range. The National Standard requires that the transponder accept the interrogation as valid if the width of both P1 and P3 pulses is between 0.7 and 0.9 microsecond. The reply ratio shall be less than 10 percent if the duration of either P1 or P3 is less than 0.3 microsecond.

This test procedure runs at the power levels required in the MOPS test procedure. In order for these power levels to be accurate, the transponder being tested should have the antenna directly at the calibrated point in the main beam. However, if this cannot be guaranteed, this test can still measure the pulse duration tolerances if the available power level is adequate.

TEST PROCEDURE: The test sets up a standard ATCRBS interrogation. For each interrogation mode, the width of P1 and P3 is varied one at a time from "tst_cur.tst36.strt_pw" to "tst_cur.tst34.end_pw" varied by "tst_cur.tst34.pw_inc." The power level of the interrogation is set to MTL + 10 dBm. MTL is measured with a call to "getmtl()" from this test. The reply count for each increment of the pulse width is recorded.

INTERROGATION PARAMETERS:

Mode A:
P1 width - Determined by test.
P2 width - int_cur[36].parm[A_P2_WDTH] determines P2 width.
P3 width - Determined by test.
P1-P2 spacing - int_cur[36].parm[A_Pl_P2_SP] determines P1-P2 spacing.
P1-P3 sPacing - int_cur[36].parm[A_Pl_P3_SP] determines P1-P3 spacing.
Pl.P3 Power - Determined by test (MTL + 10 dB).
P2 power - int_cur[36].parm[A_P2_PO] determines P2 power.
PRF - int_cur[36].parm[A_PRF] determines PRF.
Frequency - int cur[36].parm[A_FREQ] determines frequency.

Mode C:
Pl width - Determined by test.
P2 width - int_cur[36].parm[C_P2_WDTH] determines P2 width.
P3 width - Determined by test.
Pl-P2 spacing - int_cur[36].parm[C_Pl_P2_SP] determines Pl-P2 spacing.
Pl-P3 spacing - int_cur[36].parm[C_Pl_P3_SP] determines Pl-P3 spacing.
Pl,P3 power - Determined by test (MTL + 10 dB).
P2 power - int_cur[36].parm[C_P2_PO] determines P2 power.
PRF - int_cur[36].parm[C_PRF] determines PRF.
Frequency - int_cur[36].parm[C_FREQ] determines frequency.

TEST PARAMETERS:
tst_cur.tst36.num_ints - determines the number of interrogations to send at each increment. tst_cur.tst36.strt_pw - Starting pulse width. tst_cur.tst36.end_pw - Ending pulse width.
tst_cur.tst36.pw_inc - Width increment.

TEST DATA:
```
     int num_ints;   /* Number of interrogations       */
     int mtl[2]; /* MTL measured each mode     */
     int i_power[2]; /* Interrogation powers       */
     long strt_pw;   /* Starting width             */
     long end_pw;/* Ending width               */
     long pw_inc;/* Pl-P2 width increment          */
    int rep_cnt[2][2][41]; /* Reply count array        */
    int power;   /* Highest reply power measured    */
    int delay;   /* Lowest reply delay measured     */
    int e_code;  /* Test error codes               */
```

Data size = 352 bytes.

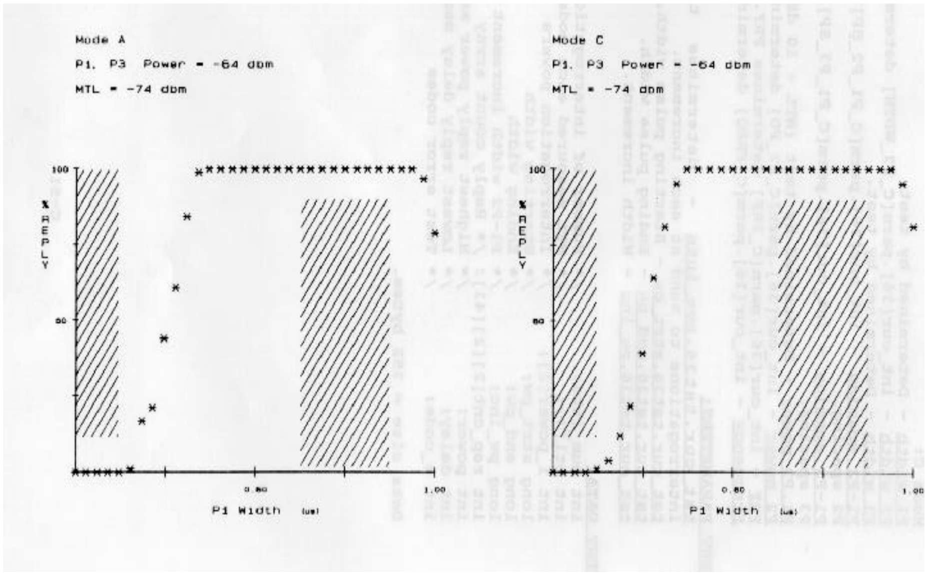
Aircraft ID: N1234   Aircraft Type: CESSNA 170   Test Time:

Transponder Type: KING KT76A   Transponder Serial#

Comment: PILOT REPORT TRANSPONDER TROUBLE

DATAS TEST #36 PULSE DURATION TOLERANCES

100 Interrogations Per Point



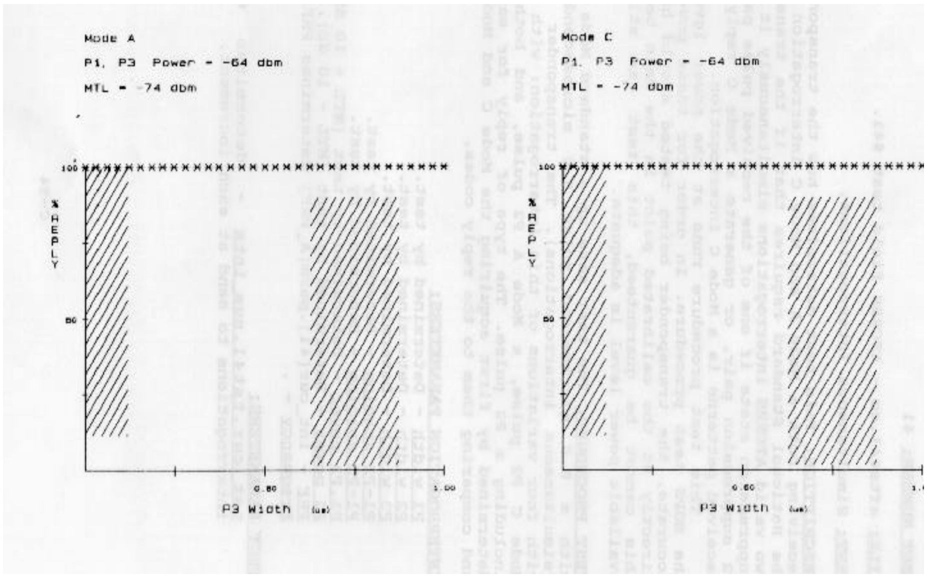
Aircraft ID: N1234   Aircraft Type: CESSNA 170   Test Time:

Transponder Type: KING KT76A   Transponder Serial#

Comment: PILOT REPORT TRANSPONDER TROUBLE

DATAS TEST #36 PULSE DURATION TOLERANCES

100 Interrogations Per Point



TEST NUMBER: 41

FILE: aftst41.cc - ATCRBS field test #41.

TEST: Simultaneous Interrogations.

DESCRIPTION: This test determines how the transponder responds to receiving both a Mode A and Mode C interrogation simultaneously. The National Standard requires that if the transponder receives two valid ATCRBS interrogations simultaneously it shall enter the suppression state if one of the received pulse patterns is a P1P2 suppression pair, or generate a Mode C reply if one of the received patterns is a Mode C interrogation.

This test procedure runs at the power levels required in the MOPS test procedure. In order for these power levels to be accurate, the transponder being tested should have the antenna directly at the calibrated point in the main beam. However, if this cannot be guaranteed, this test can still run if the available power level is adequate.

TEST PROCEDURE: The test sets up a standard Mode C interrogation with a 0.8 microsecond pulse 8.0 microseconds prior to P3 (simultaneous interrogations). The transponder is interrogated with four variations of this interrogation: with no P2 pulses, a Mode C P2 pulse, a Mode A P2 pulse, and both interrogations including a P2 pulse. The type of reply for each situation is determined by first acquiring the Mode C and Mode A reply codes and comparing them to the reply codes.

INTERROGATION PARAMETERS:

- P1 width - Determined by test.
- P2 width - Determined by test.
- P3 width - Determined by test.
- P1-P2 spacinq - Determined by test.
- Pl-P3 sPacinq - Determined by test.
- Pl,P3 Power - Determined by test (MTL + 10 dB).
- P2 power - Determined by test (MTL + 10 dB).
- PRF - int cur[41].parm[A_PRF] determines PRF.
- Frequency -

TEST PARAMETERS:

tst_cur.tst41.num_ints - determines the number of interrogations to send at each increment.

TEST DATA:

```
     int num_ints;    /* Number of interrogations           */
     int code[2];/* Acquired codes O=A l=C         */
     int mtl;         /* Minimum transmit level         */
     int i_power;/* Interrogation power                 */
    int rep_cnt[3][4];    /* Reply count array          */
     int power;  /* Highest reply power measured        */
     int delay;  /* Lowest reply delay measured         */
    int e_code;  /* Test error codes                    */

     Data size = 40 bytes.
```

Aircraft ID: N1234   Aircraft Type: CESSNA 170   Test Time:

Transponder Type: KING KT76A   Transponder Serial#

Comment: PILOT REPORT TRANSPONDER TROUBLE

DATAS TEST #41 SIMULTANEOUS INTERROGATIONS

INTERROGATIONS	% REPLY		
	MODE A	MODE C	OTHER
SI WITH NO P2 PULSES	0	49**	0
SI WITH MODE A P2	0	0	0
SI WITH MODE C P2	0	0	0
SI WITH BOTH P2’S	0	0	0

MODE A CODE = 7777

MODE C CODE = 0

** INDICATES FAILURE

TEST NUMBER: 42

FILE: aftst42.cc - ATCRBS field test #42.

TEST: Single Pulse Desensitization and Recovery.

DESCRIPTION: This test measures the transponders' desensitization from a single interrogation pulse. When the transponder receives any pulse more than 0.7 microsecond in duration it is desensitized temporarily for all received signals by raising the receivers threshold. Immediately after the desensitizing pulse, the receiver shall be between the level of the desensitizing pulse and 9 dB below that. The National Standard requires that the receiver shall recover sensitivity within 3 dB of MTL, within 15 microseconds after reception of the trailing edge of a desensitizing pulse having a signal strength up to 50 dB above MTL. Recovery shall be at an average rate not exceeding 4.0 dB per microsecond.

This test procedure runs at the power levels required in the MOPS test procedure. In order for these power levels to be accurate, the transponder being tested should have the antenna directly at the calibrated point in the main beam. However, if this cannot be guaranteed, this test can still measure the pulse desensitization if the available power level is adequate. The power level required for this test is quite high.

TEST PROCEDURE: The test sets up a standard ATCRBS interrogation preceded by a single pulse. The spacing between the pulse and the interrogation is varied from "tst_cur.tst42.strt_sp" to "tst_cur.tst42.end_sp" varied by "tst_cur.tst42.sp_inc." The power level of the interrogation and whether or not 90 percent replies were receded is recorded for each spacing.

INTERROGATION PARAMETERS:

Mode A:
P1 width - int_cur[42].parm[A_Pl_WDTH] determines P1 width.
P2 width - int_cur[42].parm[A_P2_WDTH] determines P2 width and the width of the desensitizing pulse.
P3 width - int_cur[42].parm[A_P3_WDTH] determines P3 width.
P1-P3 spacing - int_cur[42].parm[A_Pl_P3_SP] determines P-P3 spacing.
Pl,P3 power - Determined by test (MTL).
PRF - int_cur[42].parm[A_PRF] determines PRF.
Frequency - int_cur[42].parm[A_FREQ] determines frequency.

TEST PARAMETERS:

tst_cur.tst42.i_mode - determines the interrogation mode.
tst_cur.tst42.strt_sp - Start spacing.
tst_cur.tst42.end_sp - End spacing.
tst_cur.tst42.sp_inc - Spacing increment.

TEST DATA:

```
     int i_mode;      /* Interrogation mode 0=A 1=C      */
     int mtl;         /* MTL measured                    */
     int dp_power;    /* Desensitizing pulse power       */
     int mx_power;    /* Maximum system power  .       */
     int strt_sp;     /* Start spacing                   */
     int end_sp;      /* End spacing                     */
     int sp_inc;      /* Spacing increment               */
    int rep_90_flg[80]; /* 90 percent reply flags  */
    int int_pow[80]; /* Interrogation powers       */
    int power;   /* Highest reply power measured    */
     int delay;  /* Lowest reply delay measured     */
    int e_code;  /* Test error codes               */
```

Data size = 340 bytes.

TEST NUMBER: 44

FILE: aftst44.cc - ATCRBS field test #44.

TEST: Undesired Replies.

DESCRIPTION: This test is used to monitor the transmitter of the transponder to see if it sends unsolicited replies. If this test is run in an active air traffic environment, replies could be generated by interrogation sources within the area. For a true unsolicited reply test, the transponder should be connected to the test system by coaxial cable.

TEST PROCEDURE: This test simply sets up a constant reply window and monitors for rePlies the allotted time.

INTERROGATION PARAMETERS:
Not applicable.

TEST PARAMETERS:
tst_cur.tst44.time - Number of seconds to monitor replies.

TEST DATA:
```
    int time;        /* Time in seconds for test   */
    int rep_cnt;         /* Reply count                */
```

Data size = 4 bytes.

TEST NUMBER: 55

FILE: aftst55.cc - ATCRBS field test #55.

TEST: Suppression.

DESCRIPTION: This test measures the transponders response to interrogations with the P2 suppression pulse included with the interrogation equal in amplitude to Pl,P3 and at 9 dB below Pl,P3 power. The National Standard requires that the transponder shall initiate suppression (not reply) when the P2 pulse is equal to or greater than the amplitude of P1 and P3. The transponder shall reply to at least 90 percent of the interrogations when the P1 level exceeds the P2 level by 9 dB or more. If multiple PRF mode is selected this test can show the relationship between PRF and suppression acceptance.

This test procedure runs at various interrogation power levels. In order for these power levels to be accurate, the transponder being tested should have the antenna directly at the calibrated point in the main beam. However, if this cannot be guaranteed, these test results can still show the relationship between relative interrogation power and P2 acceptance.

TEST PROCEDURE: This test has two operating modes: single PRF and multiple PRF modes. The operating mode is determined by the test parameter variable "tst_cur.tst55.t_mode" (0=single 1=multiple). In "single PRF mode" the test will run the two suppression ratios at a single PRF which is determined by the test parameter "tst_cur.tst55.s_prf." This mode should be selected when a shorter test time is required, suppression vs. PRF information is not desired, or if the test is being conducted in a field environment where a high PRF could interfere with the air traffic control systems. Multiple PRF mode will conduct the same test repeated for 10 different PRF's. The PRF's are determined by the test parameters "tst_cur.tst55.m_prf_lo" and tst_cur.tst55.m_prf_hi." The 10 PRF's are evenly divided between the low and high PRF.

INTERROGATION PARAMETERS:

Mode A:

P1 width - int_cur[55].parm[A_Pl_WDTH] determines P1 width.

P2 width - int_cur[55].parm[A_P2_WDTH] determines P2 width.

P3 width - int_cur[55].parm[A_P3_WDTH] determines P3 width.

P1-P2 sPacina - int_cur[55].parm[A_Pl_P2_SP] determines P1- P2 spacing.

P1-P3 spacinq - int_cur[55].parm[A_Pl_P3_SP] determines P1- P3 spacing.

Pl.P3 power - int_cur[55].parm[A_INT_PO] determines power. P2 Power - Determined by test.

PRF - Determined by test.

Frequency - int_cur[55].parm[A_FREQ] determines frequency.

Mode C:
<u>P1 width</u> - int cur[55].parm[C_Pl_WDTH] determines P1 width.
<u>P2 width</u> - int cur[55].parm[C_P2_WDTH] determines P2 width.
<u>P3 width</u> - int cur[55].parm[C_P3_WDTH] determines P3 width.
<u>P1-P2 spacing</u> int_cur[55].parm[C_Pl_P2_SP] determines P1-P2 spacing.
<u>Pl-P3 spacing</u> int cur r 551.parm[C_Pl_P3_SP] determines P1-P3 spacing.
<u>Pl.P3 power</u> - int_cur[55].parm[C_INT_PO] determlnes power. <u>P2 Power</u> - Determined by test.
<u>PRF</u> - Determined by test.
<u>Frequency</u> - int_cur[55].parm[C_FREQ] determines frequency.

<u>TEST PARAMETERS:</u>
<u>tst_cur.tst55.num_</u>ints - determines the number of interrogations to send.
<u>tst_cur.tst55.i_mode</u> - determines interrogation mode. 0=Mode A. l=Mode C.
<u>tst_cur.tst55.t_mode</u> - determines test mode 0=single PRF, 1=multiple PRF.
<u>tst_cur.tst55.s_Prf</u> - PRF if single mode..
<u>tst_cur.tst55.m_Prf_lo</u> - Low PRF if multiple mode.
<u>tst_cur.tst55.m_prf_hi</u> - High PRF if multiple mode.

<u>TEST DATA:</u>
```
     int num_ints;    /* Number of interrogations           */
     int i_mode; /* Interrogation mode 0=A 1=C          */
     int t_mode; /* Test mode, 0=lPRF 1=Mul. PRF        */
     int s_prf;  /* Single PRF                          */
     int m~Prf_lo;    /* Multiple PRF low                   */
     int m_prf_hi;    /* Multiple PRF high              */
    int rep_arr[2][10]; /* Reply array                  */
    int power;          /* Highest reply power measured    */
    int delay;          /* Lowest reply delay measured     */
    int e_code;  /* Test error codes                    */
```

Data size = 58 bytes.

Aircraft ID: N1234 Aircraft Type: CESSNA 170 Test Time:

Transponder Type: KING KT76A Transponder Serial#

Comment: PILOT REPORT TRANSPONDER TROUBLE

DATAS TEST #55 SUPPRESSION

Test Data Based on 100 Mode A Interrogations

Single PRF = 235

p2 - p1 DIFF	% REPLY	TEST SPEC
0 DBM	0	10% OR LESS REPLY
-9 DBM	100	90% OR GREATER REPLY