NASA/TM-2008-215552

Guidance and Control Software Project Data

Volume 3: Verification Documents

Edited by
Kelly J. Hayhurst
Langley Research Center, Hampton, Virginia

December 2008

The NASA STI Program Office … in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the lead center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers, but having less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that complement the STI Program Office's diverse offerings include creating custom thesauri, building customized databases, organizing and publishing research results ... even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at ***http://www.sti.nasa.gov***

- E-mail your question via the Internet to help@sti.nasa.gov

- Fax your question to the NASA STI Help Desk at (301) 621-0134

- Phone the NASA STI Help Desk at (301) 621-0390

- Write to:
  NASA STI Help Desk
  NASA Center for AeroSpace Information
  7115 Standard Drive
  Hanover, MD 21076-1320

NASA/TM-2008-215552

Guidance and Control Software Project Data

Volume 3: Verification Documents

Edited by
Kelly J. Hayhurst
Langley Research Center, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

December 2008

Available from:

NASA Center for AeroSpace Information (CASI)
7115 Standard Drive
Hanover, MD 21076-1320
(301) 621-0390

National Technical Information Service (NTIS)
5285 Port Royal Road
Springfield, VA 22161-2171
(703) 605-6000

Table of Contents

Abstract

The Guidance and Control Software (GCS) project was the last in a series of software reliability studies conducted at Langley Research Center between 1977 and 1994. The technical results of the GCS project were recorded after the experiment was completed. Some of the support documentation produced as part of the experiment, however, is serving an unexpected role far beyond its original project context. Some of the software used as part of the GCS project was developed to conform to the RTCA/DO-178B software standard, "Software Considerations in Airborne Systems and Equipment Certification," used in the civil aviation industry. That standard requires extensive documentation throughout the software development life cycle, including plans, software requirements, design and source code, verification cases and results, and configuration management and quality control data. The project documentation that includes this information is open for public scrutiny without the legal or safety implications associated with comparable data from an avionics manufacturer. This public availability has afforded an opportunity to use the GCS project documents for DO-178B training. This report provides a brief overview of the GCS project, describes the 4-volume set of documents and the role they are playing in training, and includes the verification documents from the GCS project.

1 Introduction and Background on Software Error Studies

As the pervasiveness of computer systems has increased, so has the desire and obligation to establish the reliability of these systems. Reliability estimation and prediction are standard activities in many engineering projects. For the software aspects of computer systems, however, reliability estimation and prediction have been topics of dispute, especially for safety-critical systems. A primary challenge is how to accurately model the failure behavior of software such that numerical estimates of reliability have sufficient credibility for systems where the probability of failure needs to be quite small, such as in commercial avionics systems (ref. 1). A second challenge is how to gather sufficient data to make such estimates. Software reliability models are not used in the civil aviation industry, for example, because “currently available methods do not provide results in which confidence can be placed to the level required for this purpose.” (ref. 2)

In an effort to develop methods to credibly assess the reliability of software for safety-critical avionics applications, Langley Research Center initiated a Software Error Studies program in 1977 (ref. 3). A major focus of those studies was on generating significant quantities of software failure data through controlled experimentation to better understand software failure processes. The intent of the Software Error Studies program was to incrementally increase complexity and realism in a series of experiments so that the final study would have statistically valid results, representative of actual software development processes.

The Software Error Studies program started with initial investigations by the Aerospace Corporation to define software reliability measures and data collection requirements (ref. 4-6).

Next, Boeing Computer Services (BCS) and the Research Triangle Institute (RTI) conducted several simple software experiments with aerospace applications including missile tracking, launch interception, spline function interpolation, Earth satellite calculation, and pitch axis control (refs. 7-11). The experiment design used in these studies generally involved a number of programmers (denoted *n*) who independently generated computer code from a given specification of the problem to produce *n* versions of a program. In these experiments, no particular software development standards or life-cycle models were followed. Because the problems were relatively small and simple, the versions were compared to a known error-free version of the program to obtain information on software errors.

Although the initial experiments were small and simplistic compared with real-world avionics development, they yielded some interesting results that have influenced software reliability modeling. The BCS and RTI studies showed widely varying error rates for faults. This finding refuted a common assumption in early software reliability growth models that faults produced errors at equal rates. These studies also provided evidence of fault interaction where one fault could mask potentially erroneous behavior from another fault, or where two or more faults together cause errors when alone they would not. (ref. 12) Additional investigations with *n*-version programs (ref. 13) found that points in the input space that cause an error can cluster and form “error crystals”. Extrapolating this finding to aerospace applications, where input signals tend to be continuous in nature, the error crystals may manifest themselves as clusters of successive faults that could have unintended consequences. (ref. 14)

The last project in the Software Error Studies program was the Guidance and Control Software (GCS) project. It built on the previous experiments in two ways: (1) by requiring that the software specimens for the experiment be developed in compliance with current software development standards, and (2) by increasing the complexity of the application problem (ref. 15). At the time of the GCS project, the RTCA/DO-178B guidelines, "Software Considerations in Airborne Systems and Equipment Certification," (ref. 2) were the primary standard sanctioned by the Federal Aviation Administration (FAA) for developing software to be approved for use in commercial aircraft equipment (ref. 16). The DO-178B document describes objectives and design considerations to be used for the development of software as well as verification, configuration management, and quality assurance activities to be performed throughout the development process. The DO-178B guidelines were selected as the software development standard to be used for the GCS specimens.

The software application selected for the GCS project, as the title indicates, is a guidance and control function for controlling the terminal descent trajectory of a planetary lander vehicle. This terminal descent trajectory is the same fundamental trajectory referred to as the “seven minutes of terror” in the entry, descent, and landing phase of a planetary mission, such as the recent Phoenix Mars Lander (ref. 17). For the GCS project, the software requirements were reverse engineered from a simulation program used to study the probability of success of the original NASA Viking Lander mission to Mars in the 1970s (ref. 18). It is important to emphasize that the software requirements documented for the GCS project, while realistic, are not the actual software requirements used for NASA’s Viking Lander or any other planetary landers.

For the GCS experiment, two[1] teams of software engineers were each tasked to independently design, code, and verify a GCS program, following the software development guidance in DO-178B, as closely as possible. In addition to those teams, another GCS version was produced, without the constraint of compliance with DO-178B, to aid development and verification of the requirements and simulation environment. Once all versions were complete, data on residual

[1] The original plan for the GCS project called for three independent teams. Due to funding constraints, only two teams were able to complete the project.

errors was supposed to be collected by running all the versions simultaneously in a simulation environment, and using any discrepancies among the results of the versions as possible indications of errors.

Results of the operational simulations and data collection are described in (ref. 15). The purpose of this report is not to repeat those results, but to disseminate some of the project documentation that has an unanticipated utility beyond its original project context. The project documentation of interest is the documentation developed by the teams required to comply with the DO-178B standard. That standard requires extensive records of all of the software development life cycle activities. For the GCS project, those records included 18 documents consisting of life cycle plans, development products including requirements and source code, verification cases and results, and configuration management and quality control data. Comparable data from a commercial avionics system would not be available for public review because of proprietary and other legal considerations. The GCS project documentation is not subject to those considerations because it is not data from an actual operational, or even prototype, system. But, the data has sufficient realism to provide a window into the types of activities and data involved in the production of DO-178 compliant software, which makes the GCS documentation desirable from a training perspective.

The remainder of this report provides a brief overview of aspects of the GCS project relevant to using the documentation for training. This information includes a description of the GCS application, a synopsis of the software development processes used to follow the DO-178B guidance, and the data that was generated as a result. Because the complete set of compliance documents is large, the documents have been divided into four sets (planning, development, verification, and other integral process documents) contained in separate volumes of this report. Volume 3 includes in Appendices A-D all of the GCS documents, aside from planning, generated as part of the verification process.

2 Guidance and Control Software Application

The requirements for the GCS application focus on two primary functions: (1) to provide guidance and engine control of the lander vehicle during its terminal phase of descent onto the planet's surface, and (2) to communicate sensory information to an orbiting platform about the vehicle and its descent. Figure 1 shows a sketch of the lander vehicle, taken from (ref. 18), noting the location of the terminal descent propulsion systems.

The guidance package for the lander vehicle contains sensors that obtain information about the vehicle state and environment, a guidance and control computer, and actuators providing the thrust necessary for maintaining a safe descent. The vehicle has three accelerometers (one for each body axis), one Doppler radar with four beams, one altimeter radar, two temperature sensors, three strapped-down gyroscopes, three opposed pairs of roll engines, three axial thrust engines, one parachute release actuator, and a touch down sensor. The vehicle has a hexagonal, box-like shape; three legs and a surface sensing rod protrude from its undersurface.

In general, the requirements for the planetary lander only concern the final descent to the surface. Figure 2 shows a sketch of the phases of the terminal descent trajectory.

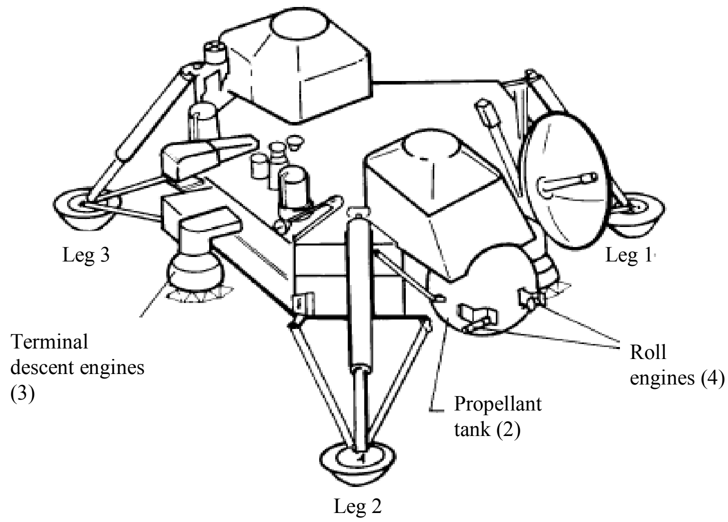


Figure 1. Lander with Terminal Descent Propulsion Systems

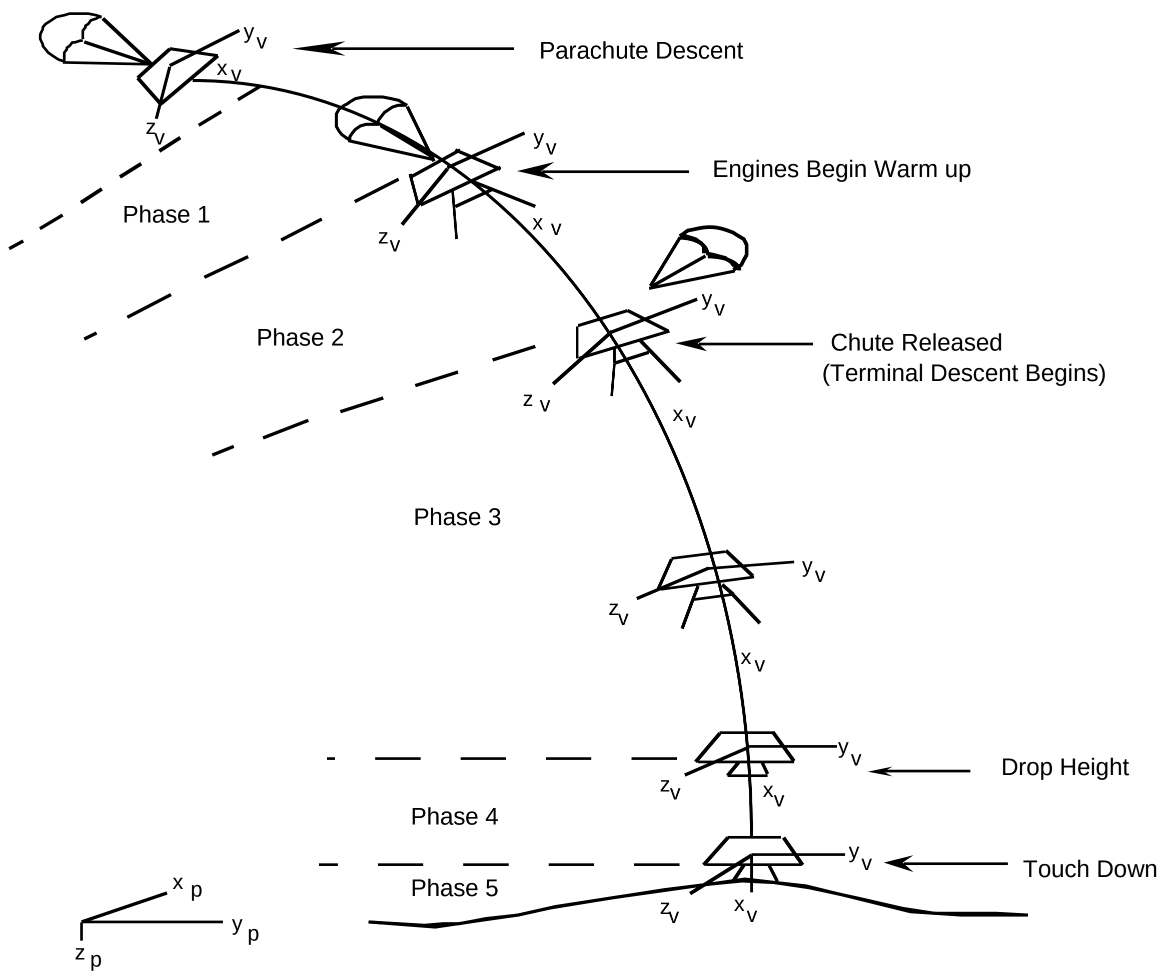


Figure 2. A Typical Terminal Descent Trajectory

After the lander has dropped from orbit, the software controls the engines of the vehicle to the surface of a planet. The initialization of the GCS starts the sensing of vehicle altitude. When a predefined engine ignition altitude is sensed by the altimeter radar, the GCS begins guidance and control of the lander. The axial and roll engines are ignited; while the axial engines are warming up, the parachute remains connected to the vehicle. During this engine warm-up phase, the aerodynamics of the parachute dictate the vehicle's trajectory. Vehicle attitude is maintained by firing the engines in a throttled-down condition. Once the main engines become hot, the parachute is released and the GCS performs an attitude correction maneuver and then follows a controlled acceleration descent until a predetermined velocity-altitude contour is crossed. The GCS then attempts to maintain the descent of the lander along this predetermined velocity-altitude contour. The lander descends along this contour until a predefined engine shut off altitude is reached or touchdown is sensed. After all engines are shut off, the lander free-falls to the surface.

The software requirements for this guidance and control application are contained in a document called the *Guidance and Control Development Specification* (in Volume 2). As mentioned earlier, the initial requirements for this application were reverse engineered from a simulation program used to study the probability of success of the original NASA Viking Lander mission to Mars. Prior to use in the experiment, the requirements were revised to make them suitable for use in an *n*-version software experiment. Each of the GCS programs for the experiment were developed from the same requirements document.

3 Software Life Cycle Processes and Documentation

Having some of the project teams adhere to the DO-178B guidelines as they created a software version for the experiment was a significant element of the GCS project, requiring the development and tracking of numerous software engineering artifacts not normally associated with a software engineering experiment. The purpose of DO-178B is to provide guidelines for the production of software such that the completed implementation performs its intended function with a level of confidence in safety satisfactory for airworthiness. Along with the production of software is the generation of an extensive set of documents recording the production activities.

DO-178B defines software development activities and objectives for the development life cycle of the software, and the evidence that is needed to show compliance. The life-cycle processes are divided into planning, development, and integral processes. The planning process defines and coordinates the software development processes and the integral processes. The software development processes involve identification of software requirements, software design and coding, and integration; that is, the development processes directly result in the software product. Finally, the integral processes function throughout the software development processes to ensure integrity of the software products. The integral processes include software verification, configuration management, and quality assurance processes. Section 11 of DO-178B describes data that should be produced as evidence of performing all of the life cycle process activities (see Table 1).

For the GCS project, some of this data was common for all of the teams, and other data was intended to be specific to each team. For example, each team worked with the same plans, standards, and requirements. Then, each individual team was responsible for independently developing their own design, code, and corresponding verification data. To distinguish the versions, each team was assigned a planetary name: Mercury, Venus, and Pluto[2].

[2] At the time the GCS experiment was conducted, Pluto had not yet been relegated to non-planet status.

Table 1. Life Cycle Data

Planning Process Documents	Development Process Documents	Integral Process Documents
• Plan for Software Aspects of Certification	• Software Requirements Data	• Software Verification Cases and Procedures
• Software Development Plan	• Design Description	• Software Verification Results
• Software Verification Plan	• Source Code	• Software Life Cycle Environment Configuration Index
• Software Configuration Management Plan	• Executable Object Code	• Software Configuration Index
• Software Quality Assurance Plan		• Problem Reports
• Software Requirements Standards		• Software Configuration Management Records
• Software Design Standards		• Software Quality Assurance Records
• Software Code Standards		• Software Accomplishment Summary

The DO-178B data associated with the development of the Pluto version of the GCS was selected for publication. Most of the GCS documents correspond directly with the life cycle data listed in Table 1. All together, the documentation includes over 1000 pages. So, for dissemination purposes, the Pluto data was divided into the following 4 subsets:

Volume 1: Planning Documents

- *Plan for Software Aspects of Certification of the Guidance and Control Software Project*
- *Software Configuration Management Plan for the Guidance and Control Software Project*
- *Software Quality Assurance Plan for the Guidance and Control Software Project*
- *Software Verification Plan for the Guidance and Control Software Project*
- *Software Development Standards for the Guidance and Control Software Project*

Volume 2: Development Documents

- *Guidance and Control Software Development Specification*
- *Design Description for the Pluto Implementation of the Guidance and Control Software*
- *Source Code for the Pluto Implementation of the Guidance and Control Software*

Volume 3: Verification Documents

- *Software Verification Cases and Procedures for the Guidance and Control Software Project*
- *Software Verification Results for the Pluto Implementation of GCS*
- *Review Records for the Pluto Implementation of the Guidance and Control Software*
- *Test Results Logs for the Pluto Implementation of the Guidance and Control Software*

Volume 4: Other Integral Processes Documents

- *Software Accomplishment Summary for the Guidance and Control Software Project*
- *Software Configuration Index for the Guidance and Control Software Project*
- *Problem Reports for the Pluto Implementation of the Guidance and Control Software*
- *Support Documentation Change Reports for the Guidance and Control Software Project*
- *Configuration Management Records for the Guidance and Control Software Project*
- *Software Quality Assurance Records for the Guidance and Control Software Project*

Appendices A-D contain all of the original verification documents, except for verification planning, for the GCS Project. *Software Verification Cases and Procedures for the Guidance and Control Software Project,* in Appendix A, specifies the procedures for conducting reviews, analysis, and testing, and describes the test cases that meet Level A requirements for verification. The results of the review and analysis activities for the requirements and design are recorded in *Review Records for the Pluto Implementation of the Guidance and Control Software,* in Appendix B; and, *Software Verification Results for the Pluto Implementation of the Guidance and Control Software,* in Appendix C, contains the results of all of the testing activities. The *Test Results Logs* in Appendix D records the actual pass/fail results of the testing.

The content of the documents in the appendices has not been altered from the original versions produced during the project.

4 Role in Training

At the time of the GCS project, there was no publicly available information, such as templates, or examples, or training courses, to help a novice developer generate the type of evidence that a certificating authority would expect to see to demonstrate compliance with DO-178B. As mentioned earlier, compliance data from a real avionics system is not typically available for public review because of various legal and safety considerations. For example, an avionics manufacturer would likely consider the design and implementation of a system to be proprietary. Those considerations do not apply to the data from the GCS project, because neither the requirements nor the software versions represent an actual system with safety, liability, or other considerations.

In addition to the availability of data, the GCS requirements and DO-178B compliance data are sufficiently realistic to serve as an example of a DO-178B project: one that is small enough in scale to be studied in a training course. The GCS documentation provides a window into the activities and data produced throughout the development life cycle to comply with DO-178B. Because the Federal Aviation Administration (FAA) was aware of the GCS project, they recognized the potential value of the documentation for training. The FAA has designed software training to include a case study portion that addresses avionics software issues that arise from the application of the DO-178B guidelines. The case study gives students the opportunity to use auditing techniques to identify flaws in lifecycle data. Because the GCS data was produced by novices, there are plenty of flaws to find.

5 Summary

From 1977-1994, NASA Langley Research Center conducted a Software Error Studies program that generated data that provided insights into the software failure process and into conducting software engineering experiments as well. The GCS project was the final experiment in that program. A unique feature of the GCS project was the requirement for some of the software specimens used in the experiment to conform to the RTCA/DO-178B software standard, "Software Considerations in Airborne Systems and Equipment Certification," used in the civil aviation industry. The project documentation produced to meet that requirement has had the unanticipated benefit of serving as case study material in software certification training long after the conclusion of the original experiment. Volume 3 of this report contains all of the verification documents from the GCS project. Other volumes of this report contain the rest of the GCS compliance data including planning, development, and configuration management and quality assurance documents.

6 References

1. Littlewood, Bev, and Strigini, Lorenzo, Software Reliability and Dependability: a Roadmap, 22nd International Conference on Software Engineering, Future of Software Engineering Track, June 4-11, 2000, Limerick Ireland, pp. 175 – 188.
2. Software Considerations in Airborne Systems and Equipment Certification. Doc. No. RTCA/DO-178B, RTCA, Inc., Dec. 1, 1992.
3. Finelli, George B.: NASA Software Failure Characterization Experiments. Reliability Engineering & System Safety, vol. 32, pp. 155–169, 1991.
4. Hecht, H.; Sturm, W. A.; and Tratlner, S.: Reliability Measurement During Software Development. NASA CR-145205, 1977.
5. Hecht, H.: Measurement Estimation and Prediction of Software Reliability. NASA CR-145135, 1977.
6. Maxwell, F. D.: The Determination of Measures of Software Reliability. NASA CR-158960, 1978.
7. Nagel, Phyllis M.; and Skrivan, James A.: Software Reliability: Repetitive Run Experimentation and Modeling. NASA CR-165836, 1982.
8. Nagel, P. M.; Scholz, F. W.; and Skrivan, J. A.: Software Reliability: Additional Investigation Into Modeling With Replicated Experiments. NASA CR-172378, 1984.
9. Dunham, Janet R.: Experiments in Software Reliability: Life-Critical Applications. IEEE Transactions on Software Engineering, vol. SE-12, no. 1, Jan. 1986, pp. 110–123.
10. Dunham, J. R.; and Lauterbach, L. A.: An Experiment in Software Reliability Additional Analyses Using Data From Automated Replications. NASA CR-178395, 1987.
11. Dunham, Janet R.; and Pierce, John L.: An Empirical Study of Flight Control Software Reliability. NASA CR-178058, 1986.
12. Dunham, Janet R.; and Finelli, George B., Real-Time Software Failure Characterization, IEE Aerospace and Electronic Systems Magazine, pp. 38-44, November 1990.
13. Ammann, P. and Knight, J.: "Data Diversity: An Approach To Software Fault Tolerance", Digest of Papers FTCS-17: The 17th Annual International Symposium on Fault Tolerant Computing, Pittsburg, Pennsylvania, July 1987.
14. Finelli, George B, Results of Software Error-Data Experiments, AIAA/AHS/ASEE Aircraft Design, Systems and Operations Conference, September 7-9, 1988, Atlanta, Georgia, AIAA-88-4436.

15. Hayhurst, Kelly J., Framework for Small-Scale Experiments in Software Engineering, Guidance and Control Software Project: Software Engineering Case Study, NASA/TM-1998-207666, May 1998.

16. Federal Aviation Administration, Advisory Circular, 20-115B, January 11, 1993.

17. Tobin, Kate, NASA Preps for '7 Minutes of Terror' on Mars, May 23, 2008, http://www.cnn.com/2008/TECH/space/05/23/mars.lander/index.html.

18. Holmberg, Neil A.; Faust, Robert P.; and Holt, H. Milton: Viking '75 Spacecraft Design and Test Summary. Volume I—Lander Design. NASA RP-1027, 1980.

Appendix A: Software Verification Cases and Procedures for the Guidance and Control Software Project

Authors: Cuong C. Quach, NASA Langley Research Center, and Debbie Taylor, Computer Sciences Corp.

This document was produced as part of Guidance and Control Software (GCS) Project conducted at NASA Langley Research Center. Although some of the requirements for the Guidance and Control Software application were derived from the NASA Viking Mission to Mars, this document does not contain data from an actual NASA mission.

A. Contents

A. List of Tables

A.1 Introduction

The purpose of this document, as described in Section 11.13 of Requirements and Technical Concepts for Aviation RTCA/DO-178B, "Software Considerations in Airborne Systems and Equipment Certification" (ref. A.2), is to provide details about how software verification process activities are to be implemented for the Guidance and Control Software (GCS) project. As stated in the preface, the development and verification of this software strictly follows guidelines described in DO-178B. This document focuses on review and analysis as well as testing methods. In particular, this document will provide details on procedures for conducting reviews and analysis, describe the test cases that meet Level A requirements, and test procedures to use for verification. Methods adopted for tracking test cases as well as accounting for test coverage will also be discussed.

As stated in the *Software Verification Plan,* GCS verification activities are independent from the development process. The development process produces artifacts that must undergo some level of verification as described in DO-178B. Figure A.1 gives an overview of verification activities for the GCS project and how they are related to the development processes. The procedures for conducting the verification activities given in Figure A.1 are described in the sections below.

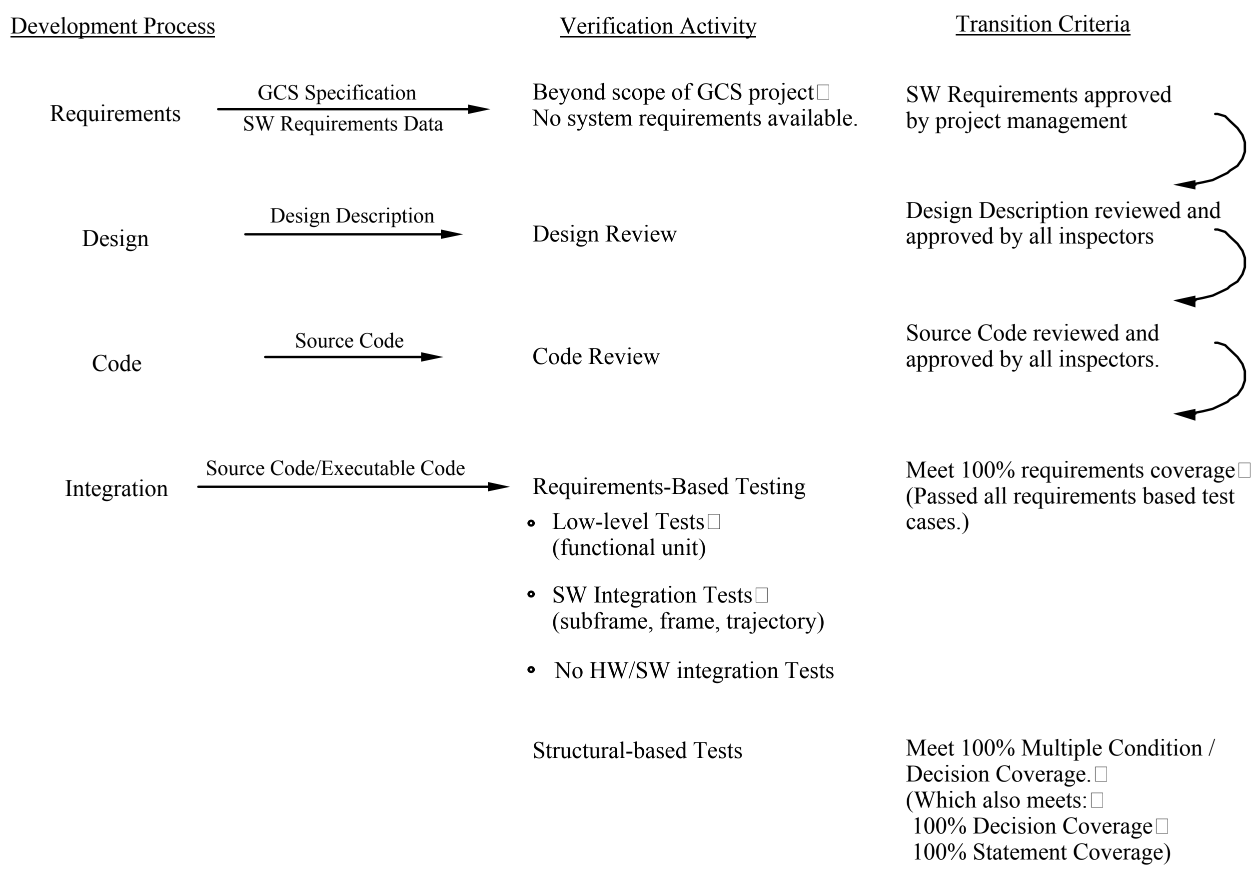


Figure A.1: Overview of verification activities.

The GCS project includes the development of two GCS implementations, Mercury and Pluto. Both implementations are developed based on the same requirements and subject to the same review and test procedures. Similarly, both are tested with the same set of requirements-based test cases. Since the methods for reviewing design and code and developing test cases and accounting for coverage are the same for both the Mercury and Pluto implementations, this document will treat those topics generically. Additionally, since requirements-based test cases will be identical for both implementations, there will only be one set of requirements test cases for both Pluto and Mercury and one set of procedures for executing those test cases.

A.2 Review and Analysis Procedures

As stated in sections 6.1 to 6.3 of DO-178B, one of the general objectives of the software verification process is to verify that "the high-level requirements have been developed into software architecture and low-level requirements that satisfy the high-level requirements." Additionally, the results of the coding process must be verified to ensure correctness and accuracy with respect to the low-level requirements. During the Transitional Design process of the GCS project, the programmers create a detailed software design that meets the requirements defined in Version 2.3 (including formal modifications) of the GCS Specification.

For the GCS project, the review of the detailed design and the source code for each implementation will consist of a series of inspections that are executed by a structured, team approach. This inspection approach is based on the Design Review and Assessment Technical Assessment Procedures (DRATAP) used by the U. S. Army Missile Command (ref. A.3) and has been tailored to fulfill the requirements of DO-178B and the GCS project. The DRATAP itself is a version of the Fagan Inspection methodology (ref. A.4) which has been tailored to meet the needs of the Missile Command. Though the procedure for both the design and the code review will be basically identical, the objectives in each are slightly different with respect to the product being reviewed.

The inspection methodology is based on a team approach where all members of the review team have specific roles to perform. For the GCS project, there is a unique review team for each implementation. Each review team consists of the Programmer and Verification Analyst assigned to the implementation under review, the System Analyst, and the Software Quality Assurance representative. Prior to the start of the actual inspection sessions, an overview meeting will be held to review the procedures and roles for the inspections and distribute all materials that are needed to perform the inspections. During the Inspection Sessions, the review team will discuss and identify defects, clarity problems, and concerns about the product under review.

This Review Procedure identifies the tools used during the inspection, the roles of the review team members during the inspections, the completion criteria, and the data that result from the completed process. The verification tools needed for the inspections include the Review Procedures (section A.5), the Design Review Checklist (section A.6) or the Code Review Checklist (section A.9), the Traceability Matrix (section A.7) and supplemental data, and Individual Inspection Preparation Logs (section A.8). The Inspection Logs can be produced electronically and do not have to exactly follow the format given in section A.8, but all pertinent information from section A.8 should be included.

The Review Checklist will be utilized by each member of the review team as a guide during the inspection process to aid in finding defects and problems. The checklist is composed of a series of questions about the detailed design with a yes/no column to be completed with the questions. The questions are phrased such that a "no" response may indicate a defect or a problem that requires further investigation and results in the generation of a Problem Report.

The GCS Requirements Traceability Matrix is also used during the inspection process. The Traceability Matrix provides an organized list of the requirements, derived from the GCS Specification. Each inspector with the exception of the programmer will use the Traceability Matrix and supplemental data during individual inspections; however, only one Traceability Matrix will result from a complete review. It is the responsibility of the Moderator of the inspection team to complete the Traceability Matrix for each implementation's review and to add low-level as well as any derived requirements to the matrix as necessary. The traceability data document is a supplement to the matrix, and provides clarification of requirements and verification criteria. The Traceability Matrix will be completed when the entire review process is finished. There will be a Traceability Matrix for each implementation. The Traceability Matrix will be the same for each implementation at the start of the Design Reviews. According to the DO-178B guidelines, it is also necessary to trace the derived requirements through the verification activities. As the Design Reviews progress, the Traceability Matrix for each implementation may be modified as low-level and/or derived requirements are identified. The Moderator will ensure that all derived requirements are added to the Traceability Matrix.

The Traceability Matrix will be used during the verification activities to track the requirements through each implementation's design, source code, and testing of its executable image. In the Traceability Matrix, columns are provided for each verification activity: design review, code review, and all phases of testing. Consequently, one of the outputs of a review should be a Traceability Matrix that has been modified to include any low-level and/or derived requirements that are identified and justified, and the P-Spec number or module name from the artifact where each requirement is addressed.

A Problem Report is generated when it is determined that a product (Design, Code, Executable) contains a defect. The project's Problem and Action Reporting Procedures are used to track errors and the changes made to the design and any other software development artifacts as a result of errors. A Problem Report generated during a review includes detailed information about the defect; a description of the problem including a reference to the document and document section that justifies the problem report, the location in the design (P-Spec#) or source code (Module name), the implementation's name, and other critical information. An example of a Problem Report and instructions for completing it can be found in the *Software Development Standards*.

The Traceability Matrix is given in the *Software Verification Plan* and will be under configuration management. Any changes made to these documents must conform to the *Configuration Management Plan.*

The following section describes the role of all the participants in the inspection sessions.

A.2.1 Review Team

As stated above, a review of the detailed design or source code for each implementation will be conducted by a team through a series of inspections. Except for the Moderator, all members of the review team will be Inspectors. In addition, the following members of the review team will have an additional role in the inspection sessions: the Software Quality Assurance (SQA) representative will be the Moderator, the Programmer will be the Reader, and the Verification Analyst will be the recorder. Each of these roles is described below.

A.2.2 Inspector

Each Inspector performs a critical reading of the product under review with the intent of identifying defects (as described above) in the product. The Review Procedures, checklist, and

Inspection Log will be supplied to each Inspector at the overview meeting to aid in the review. The Traceability Matrix will also be supplied to the Verification Analyst and the System Analyst. The critical reading of the assigned portion of the product to be inspected must be completed before the first inspection session. Each Inspector should bring the completed checklist and a list of any problems noted during the review (recorded on the Inspection Logs) to the inspection sessions. The specific activities of an Inspector are:

1. Attend the Overview Meeting and all Inspection Sessions.
2. Review the verification procedures and tools (checklist, Inspection Logs, etc.) assigned by the Moderator.
3. Review the product description and complete the checklist.
4. Record suspected defects on the Inspection Log.
5. Submit the completed Inspection Log to the Moderator at least four hours prior to the Inspection Session.

A.2.3 Moderator

The Moderator provides the leadership for the inspection sessions. The Moderator performs the following activities:

1. Chairs the Inspection Sessions and the Overview Meeting.
2. Schedules the Inspection Sessions and the Overview Meeting.
3. Collects all materials necessary for the Inspection Sessions and distributes these to the review team. These materials include the product description, appropriate Review Checklist, Review Procedures, appropriate Standards, blank Inspection Logs, and any other documentation deemed necessary. Note that there is only one "official" Traceability Matrix that is produced by the review, and this is will become part of the *Software Verification Results*.
4. Ensures that all time guidelines are followed.
5. Ensures that all issues are resolved and/or recorded to the satisfaction of the team.
6. Ensures that the appropriate column of the Traceability Matrix is completed with the design P-Spec or code module number that satisfies the requirement or a Problem Report number, adding low-level and/or derived requirements to the Traceability Matrix as necessary.
7. Ensures that any follow-up actions are documented, assigned for action, and resolved; and schedules any necessary follow-up sessions.

A.2.4 Reader

During the Overview meeting, the Reader will give a brief description of the product under review and the supporting documents. At the Inspection Session, the Reader guides the team through each part of the product and must answer questions that arise about the product from the other members of the review team. The parts of the product that are identified in the Inspections Logs as suspect will be examined in detail. The Reader also performs the function of an Inspector.

A.2.5 Recorder

The Recorder documents problems noted in an Inspection Session and initiates the necessary Problem Reports. At the conclusion of the review, the Recorder will produce an electronic copy of the Review Minutes. The Recorder also performs the function of an Inspector.

A.2.6 Overview Meeting

The purpose of this meeting is to ensure that the material to be reviewed and the associated requirements are understood by all members of the review team. During this meeting, the Moderator will discuss the scope of and procedures and tools for the inspections and will discuss the role of each of the participants. The Moderator will also distribute the materials necessary to inspect the product. These materials include the Design or Code Description, Review Procedures, Review Checklist, Design or Code Standards, and blank Inspection Logs. All members of the review team are required to attend this meeting. The Overview Meeting should be held at least twenty-four man hours (which may be as many as 6 days due to the part-time schedules of some of the GCS participants) before the scheduled time for the first inspection session.

A.2.7 Procedures for the Inspection Sessions

Prior to the Inspection Sessions, there is a period of time devoted to preparation for the inspections. This preparation specifically consists of the review and assessment of the product by each Inspector. Inspectors should review the product in detail, using the appropriate checklists. Any suspected defects should be noted on the Inspection Log, and this form should be returned to the Moderator at least four hours prior to the Inspection Session. The log should cite specific requirements, Design Standards, or Code Standards for each suspected defect. The review team is also responsible for identifying derived requirements in the product. All inspectors should be allotted at least twenty-four man hours for preparation for the inspections.

During the Inspection Sessions, the Reader guides the team through the product and answers questions about the product from the members of the review team. All problems noted by the Inspectors and logged on the Inspection Logs should be discussed. The Programmer should provide sufficient justification for all derived requirements, and the derived requirements should be added to the Traceability Matrix to track their implementation throughout the development process. The Recorder will initiate all necessary Problem Reports.

The inspection sessions should be limited to two hours per session, and no more than three sessions should be scheduled during any given week. The inspection sessions should be repeated until all of the product has been inspected. The following guidelines will be followed during each inspection session:

1. Inspectors should bring all documents and notes, including a copy of the Inspection Log, to the session.
2. Inspectors should avoid suggesting solutions to defects.
3. If no resolution to an issue is achieved after a reasonable discussion, the issue should be logged for later action and continue to the next problem.
4. If a session lasts over two hours, the session should be stopped and a continuation scheduled (within one or two days).

5. After the session, the Recorder should prepare Problem Reports for all items determined to be problems by the Inspection Team.
6. Each implementation's Review notes, compiled by the Recorder, will be put into an informal document, called Review Minutes.

The following data will result from the completed Design or Code Review process: a copy of the Review minutes, the Traceability Matrix with the appropriate portion completed including the addition of any derived requirements, and Problem Reports. The SQA Representative is responsible for completing a report on the Design or Code Review Process. The SQA Representative is also responsible for ensuring that all Problem Reports are addressed, tracked, and satisfactorily closed (see the *Software Quality Assurance Plan* for details). The review process is complete when the product has been completely reviewed according to the inspection procedures and all reported problems are resolved.

A.3 Test Case Overview

This section describes the requirements-based test cases developed for GCS testing as required by DO-178B section 11.13b. Requirements-based test cases are developed for the functional unit, subframe, frame, and trajectory testing. In this section, test cases are organized by the functional units, subframe, frame and trajectory. Traceability of requirements to test cases is established in Table A.10-1 in section A.10. As stated in the *Software Verification Plan*, there are two categories of requirements-based test cases at the functional unit level. These are the Normal Range cases and the Robustness cases. Each functional unit test case name will contain the “NR” or “RO” differentiate cases from each group. Test cases have been devised to provide the coverage as described in the *Software Verification Plan* .

Equivalence class coverage is the first coverage requirement in DO-178B. Equivalence class partitioning, as described in the *Software Verification Plan*, has been applied to GCS data elements and the equivalence classes given in Table A.9-1. Cases that test each equivalence class are given in Table A.9-2. For GCS purposes, variables from the RUN_PARAMETERS data store are considered not to change. Even though these variables are listed in the input list of functional units in the GCS Specification, they will not be tested as part of the input space of the functional units. Another exception to creating equivalence class for GCS variables is that some variables, while defined as integers in the actual code, are used as enumerated types. These variables are tested as state transitions.

Data for each test case originates in its respective data files as given in the tables below. These data files are used in the procedure given in section A.5 to generate the test-input and expected-results files (these are also given in tables below). Each file is written in FORTRAN namelist format and contain the values of variables in all four data stores, and are the actual files used to actually test the code. The test-input file contains the input values of variables before the functional unit is executed. The expected-results file contains the value of what the variables should be after the functional unit has executed.

Test stubs (or test drivers) have been written to insure that the integrity of the four data stores are maintained. When each test case is executed, using the execution procedure described in the test case execution section below, the expected-result file is compared to the values generated by the tested code. All four data stores are compared even though the tested code may only effect several variables in a single data store. This ensures that the remaining data elements not inadvertently overwritten during execution of a functional unit test case.

There is a general problem of verifying history variable rotations when all the variables are the same values. It is not possible to verify that a rotation has occurred. This problem is particularly acute for the status variables and computation indicators that require histories. In testing the history rotation of these variables, it is necessary to introduce alternating patterns so that the rotation can be verified. For variables that are matrices, this alternating pattern introduces values into matrix elements that would otherwise be zeros. Unfortunately, this is necessary to be sure that even those elements are rotated.

The sections below give a comprehensive listing of requirements-based test cases for each functional unit, subframe, frame, and trajectory. Each functional units section gives a list of variables being tested, any special conditions that test case has to cover, and a table of all the test cases in for that functional unit. Only files specific to each test case are given in tables in this section. Other files needed for generating and executing the test cases are given in Table A.11-1 and Table A.11-2 along with test case generation procedures in section A.11.

The first column of each Test Case Table, “Test Case Data File”, gives the name of the data file used to generate the test case. A description follows in the second column. The last two columns labeled “Test-Input File” and “Expected-Results File” give the files generated by using the procedure in section A.11.

A.3.1 ARSP Functional Unit Test Cases

Table A.1 gives a listing of all requirements-based test cases for the ARSP functional unit. All test cases manipulate the variables:

AR_COUNTER
AR_ALTITUDE
AR_STATUS
K_ALT

K_ALT only needs to be tested for rotation in this functional unit. For this case, the K_ALT rotation can be tested at the same time as testing AR_STATUS = FAIL. These two variables are independent of each other. To verify upper and lower bounds checking for AR_ALTITUDE, the various histories of AR_ALTITUDE are set beyond the bounds while their corresponding AR_STATUS histories are set to healthy. This is unrealistic but its the only way to force the bounds checks. AR_FREQUENCY is also listed in the GCS Specification as an input variable to this functional unit but is not tested because it is from the RUN_PARAMETERS data store. The values assigned to the tested variables are given in the Test Case Data File.

Table A.1: Test cases for ARSP functional unit.

Test Case Data File	Description	Test-Input File	Expected-Results File
arsp_ro_001.m	Test AR_COUNTER out of UPPER bound	arsp_ro_001.tc	arsp_ro_001.ex
arsp_ro_002.m	Test AR_COUNTER out of LOWER bound	arsp_ro_002.tc	arsp_ro_002.ex
arsp_ro_003.m	Force extrapolation with AR_ALTITUDE[0] out of LOWER bound to see if bounds checking messages are executed.	arsp_ro_003.tc	arsp_ro_003.ex
arsp_ro_004.m	Force extrapolation with AR_ALTITUDE[1] out of LOWER bound to see if bounds checking messages are executed.	arsp_ro_004.tc	arsp_ro_004.ex
arsp_ro_005.m	Force extrapolation with AR_ALTITUDE[2] out of LOWER bound to see if bounds checking messages are executed.	arsp_ro_005.tc	arsp_ro_005.ex
arsp_ro_006.m	Force extrapolation with AR_ALTITUDE[3] out of LOWER bound to see if bounds checking messages are executed.	arsp_ro_006.tc	arsp_ro_006.ex
arsp_ro_007.m	Force extrapolation with AR_ALTITUDE[0] out of UPPER bound to see if bounds checking messages are executed	arsp_ro_007.tc	arsp_ro_007.ex
arsp_ro_008.m	Force extrapolation with AR_ALTITUDE[1] out of UPPER bound to see if bounds checking messages are executed	arsp_ro_008.tc	arsp_ro_008.ex
arsp_ro_009.m	Force extrapolation with AR_ALTITUDE[2] out of UPPER bound to see if bounds checking messages are executed	arsp_ro_009.tc	arsp_ro_009.ex
arsp_ro_010.m	Force extrapolation with AR_ALTITUDE[3] out of UPPER bound to see if bounds checking messages are executed	arsp_ro_010.tc	arsp_ro_010.ex
arsp_nr_011.m	Test normal extrapolation & test setting AR_STATUS =1 & K_ALT = 1 (row 2 of table 5.4 in Spec.)	arsp_nr_011.tc	arsp_nr_011.ex
arsp_nr_012.m	Test for proper setting of AR_STATUS[0] and K_ALT[0] according to row 3 of table 5.4 with no echo returned & AR_STATUS[0] = Failed	arsp_nr_012.tc	arsp_nr_012.ex
arsp_nr_013.m	Test for proper setting of AR_STATUS[0] and K_ALT[0] according to row 3 of table 5.4 with no echo returned & AR_STATUS[1] = Failed	arsp_nr_013.tc	arsp_nr_013.ex
arsp_nr_014.m	Test for proper setting of AR_STATUS[0] and K_ALT[0] according to row 3 of table 5.4 with no echo returned & AR_STATUS[2] = Failed	arsp_nr_014.tc	arsp_nr_014.ex
arsp_nr_015.m	Test for proper setting of AR_STATUS[0] and K_ALT[0] according to row 3 of table 5.4 with no echo returned & AR_STATUS[3] = Failed	arsp_nr_015.tc	arsp_nr_015.ex
arsp_nr_016.m	Test Zero - AR_COUNTER and setting of AR_STATUS[0] to healthy	arsp_nr_016.tc	arsp_nr_016.ex
arsp_nr_017.m	Test upper bound - AR_COUNTER	arsp_nr_017.tc	arsp_nr_017.ex
arsp_ro_018.m	Test INVALID status - AR_STATUS[0]	arsp_ro_018.tc	arsp_ro_018.ex
arsp_ro_019.m	Test INVALID status - AR_STATUS[1]	arsp_ro_019.tc	arsp_ro_019.ex
arsp_ro_020.m	Test INVALID status - AR_STATUS[2]	arsp_ro_020.tc	arsp_ro_020.ex
arsp_ro_021.m	Test INVALID status - AR_STATUS[3]	arsp_ro_021.tc	arsp_ro_021.ex
arsp_nr_022.m	Test AR_ALTITUDE calculation based on AR_COUNTER and setting of AR_STATUS[0] and K_ALT[0] according to row 1 of table 5.4 in the Spec. Also test history rotations for AR_ALTITUDE, AR_STATUS[0,2], & K_ALT[0,2]	arsp_nr_022.tc	arsp_nr_022.ex
arsp_nr_023.m	Test AR_ALTITUDE calculation based on AR_COUNTER and test history rotations for AR_ALTITUDE, AR_STATUS[1,3], & K_ALT[1,3]	arsp_nr_023.tc	arsp_nr_023.ex

A.3.2 ASP Functional Unit Test Cases

Table A.2 is a listing of all requirements-based test cases for the ASP functional unit. Variables involved in the test cases are:

A_ACCELERATION
A_COUNTER
ATMOSPHERIC_TEMP
A_STATUS

Note that A_ACCELERATION and A_STATUS are variables with a history dimensions. The oldest elements in these variables will not require testing since it is discarded after the history rotation.

Table A.2: Test cases for ASP functional unit.

Test Case Data File	Description	Test-Input File	Expected-Results File
asp_nr_001.m	Test A_ACCELERATION calculated from A_COUNTER & A_STATUS set to HEALTHY	asp_nr_001.tc	asp_nr_001.ex
asp_nr_002.m	Test A_ACCELERATION calculated from average & A_STATUS set to UNHEALTHY	asp_nr_002.tc	asp_nr_002.ex
asp_nr_003.m	Test UNHEALTHY A_STATUS. A_ACCELERATION calculated from A_COUNTER but previous A_STATUS[1] was UNHEALTHY	asp_nr_003.tc	asp_nr_003.ex
asp_nr_004.m	Test UNHEALTHY A_STATUS. A_ACCELERATION calculated from A_COUNTER but previous A_STATUS[2] was UNHEALTHY	asp_nr_004.tc	asp_nr_004.ex
asp_nr_005.m	Test UNHEALTHY A_STATUS. A_ACCELERATION calculated from A_COUNTER but previous A_STATUS[3] was UNHEALTHY	asp_nr_005.tc	asp_nr_005.ex
asp_nr_006.m	Test History variable rotation for A_ACCELERATION[0-4] & A_STATUS[0,2]	asp_nr_006.tc	asp_nr_006.ex
asp_nr_007.m	Test History variable rotation A_STATUS[1]	asp_nr_007.tc	asp_nr_007.ex
asp_ro_008.m	Test LOW out of bound for ATMOSPHERIC_TEMP	asp_ro_008.tc	asp_ro_008.ex
asp_ro_009.m	Test HIGH out of bound for ATMOSPHERIC_TEMP	asp_ro_009.tc	asp_ro_009.ex
asp_ro_010.m	Test LOW out of bound for A_COUNTER[1]	asp_ro_010.tc	asp_ro_010.ex
asp_ro_011.m	Test LOW out of bound for A_COUNTER[2]	asp_ro_011.tc	asp_ro_011.ex
asp_ro_012.m	Test LOW out of bound for A_COUNTER[3]	asp_ro_012.tc	asp_ro_012.ex
asp_ro_013.m	Test HIGH out of bound for A_COUNTER[1]	asp_ro_013.tc	asp_ro_013.ex
asp_ro_014.m	Test HIGH out of bound for A_COUNTER[2]	asp_ro_014.tc	asp_ro_014.ex
asp_ro_015.m	Test HIGH out of bound for A_COUNTER[3]	asp_ro_015.tc	asp_ro_015.ex
asp_nr_016.m	Test A_COUNTER at zero - based on hueristic!!	asp_nr_016.tc	asp_nr_016.ex
asp_ro_017.m	Test A_ACCELERATION[0,x] out of LOWER bound	asp_ro_017.tc	asp_ro_017.ex
asp_ro_018.m	Test A_ACCELERATION[0,x] out of UPPER bound	asp_ro_018.tc	asp_ro_018.ex
asp_ro_019.m	Test A_ACCELERATION[0,y] out of LOWER bound	asp_ro_019.tc	asp_ro_019.ex
asp_ro_020.m	Test A_ACCELERATION[0,y] out of UPPER bound	asp_ro_020.tc	asp_ro_020.ex
asp_ro_021.m	Test A_ACCELERATION[0,z] out of LOWER bound	asp_ro_021.tc	asp_ro_021.ex
asp_ro_022.m	Test A_ACCELERATION[0,z] out of UPPER bound	asp_ro_022.tc	asp_ro_022.ex
asp_ro_023.m	Test A_ACCELERATION[1,x] out of LOWER bound	asp_ro_023.tc	asp_ro_023.ex
asp_ro_024.m	Test A_ACCELERATION[1,x] out of UPPER bound	asp_ro_024.tc	asp_ro_024.ex
asp_ro_025.m	Test A_ACCELERATION[1,y] out of LOWER bound	asp_ro_025.tc	asp_ro_025.ex
asp_ro_026.m	Test A_ACCELERATION[1,y] out of UPPER bound	asp_ro_026.tc	asp_ro_026.ex
asp_ro_027.m	Test A_ACCELERATION[1,z] out of LOWER bound	asp_ro_027.tc	asp_ro_027.ex
asp_ro_028.m	Test A_ACCELERATION[1,z] out of UPPER bound	asp_ro_028.tc	asp_ro_028.ex
asp_ro_029.m	Test A_ACCELERATION[2,x] out of LOWER bound	asp_ro_029.tc	asp_ro_029.ex
asp_ro_030.m	Test A_ACCELERATION[2,x] out of UPPER bound	asp_ro_030.tc	asp_ro_030.ex
asp_ro_031.m	Test A_ACCELERATION[2,y] out of LOWER bound	asp_ro_031.tc	asp_ro_031.ex
asp_ro_032.m	Test A_ACCELERATION[2,y] out of UPPER bound	asp_ro_032.tc	asp_ro_032.ex
asp_ro_033.m	Test A_ACCELERATION[2,z] out of LOWER bound	asp_ro_033.tc	asp_ro_033.ex
asp_ro_034.m	Test A_ACCELERATION[2,z] out of UPPER bound	asp_ro_034.tc	asp_ro_034.ex
asp_ro_035.m	Test A_ACCELERATION[3,x] out of LOWER bound	asp_ro_035.tc	asp_ro_035.ex
asp_ro_036.m	Test A_ACCELERATION[3,x] out of UPPER bound	asp_ro_036.tc	asp_ro_036.ex
asp_ro_037.m	Test A_ACCELERATION[3,y] out of LOWER bound	asp_ro_037.tc	asp_ro_037.ex
asp_ro_038.m	Test A_ACCELERATION[3,y] out of UPPER bound	asp_ro_038.tc	asp_ro_038.ex
asp_ro_039.m	Test A_ACCELERATION[3,z] out of LOWER bound	asp_ro_039.tc	asp_ro_039.ex
asp_ro_040.m	Test A_ACCELERATION[3,z] out of UPPER bound	asp_ro_040.tc	asp_ro_040.ex
asp_ro_041.m	Test UNHEALTHY A_STATUS. A_ACCELERATION calculated from A_COUNTER but previous A_STATUS[1,1] was INVALID	asp_ro_041.tc	asp_ro_041.ex
asp_ro_042.m	Test UNHEALTHY A_STATUS. A_ACCELERATION calculated from A_COUNTER but previous A_STATUS[1,1] was INVALID	asp_ro_042.tc	asp_ro_042.ex
asp_ro_043.m	Test UNHEALTHY A_STATUS. A_ACCELERATION calculated from A_COUNTER but previous A_STATUS[3,1] was INVALID	asp_ro_043.tc	asp_ro_043.ex
asp_ro_044.m	Test UNHEALTHY A_STATUS. A_ACCELERATION calculated from A_COUNTER but previous A_STATUS[3,2] was INVALID	asp_ro_044.tc	asp_ro_044.ex

A.3.3 GSP Functional Unit Test Cases

Table A.3 gives a listing of all requirements-based test cases for the GSP functional unit. Three variables tested by in test cases are:

ATMOSPHERIC_TEMP
G_COUNTER
G_ROTATION

Note that G_ROTATION is in the input list only because it has to be accessed for history rotations.

Table A.3: Test cases for GSP functional unit.

Test Case Data File	Description	Test-Input File	Expected-Results File
gsp_nr_001.m	Test History rotation for G_ROTATION	gsp_nr_001.tc	gsp_nr_001.ex
gsp_ro_002.m	Test out of LOWER bound for ATMOSPHERIC_TEMP	gsp_ro_002.tc	gsp_ro_002.ex
gsp_ro_003.m	Test out of UPPER bound for ATMOSPHERIC_TEMP	gsp_ro_003.tc	gsp_ro_003.ex
gsp_ro_004.m	Test out of LOWER bound for G_COUNTER[1]	gsp_ro_004.tc	gsp_ro_004.ex
gsp_ro_005.m	Test out of LOWER bound for G_COUNTER[2]	gsp_ro_005.tc	gsp_ro_005.ex
gsp_ro_006.m	Test out of LOWER bound for G_COUNTER[3]	gsp_ro_006.tc	gsp_ro_006.ex
gsp_ro_007.m	Test out of UPPER bound for G_COUNTER[1]	gsp_ro_007.tc	gsp_ro_007.ex
gsp_ro_008.m	Test out of UPPER bound for G_COUNTER[2]	gsp_ro_008.tc	gsp_ro_008.ex
gsp_ro_009.m	Test out of UPPER bound for G_COUNTER[3]	gsp_ro_009.tc	gsp_ro_009.ex

A.3.4 TSP Functional Unit Test Cases

Table A.4 is a listing of all requirements-based test cases for the TSP functional unit. All test cases manipulate the variables:

SS_TEMP
THERMO_TEMP

Table A.4: Test cases for TSP functional unit.

Test Case Data File	Description	Test-Input File	Expected-Results File
tsp_nr_001.m	Test normal range of Both SS_TEMP & THERMO_TEMP - outputs THERMO_TEMP calculation for equivalence class THERMO_TEMP.1 and SS_TEMP.1	tsp_nr_001.tc	tsp_nr_001.ex
tsp_nr_002.m	Test normal range of SS_TEMP - outputs SS_TEMP calculation for equivalence class SS_TEMP.2	tsp_nr_002.tc	tsp_nr_002.ex
tsp_nr_003.m	Test normal range of SS_TEMP - outputs SS_TEMP calculation for equivalence class SS_TEMP.3	tsp_nr_003.tc	tsp_nr_003.ex
tsp_ro_004.m	Test SS_TEMP out of upper range - outputs SS_TEMP calculation for equivalence class SS_TEMP.4	tsp_ro_004.tc	tsp_ro_004.ex
tsp_ro_005.m	Test SS_TEMP out of lower range - outputs SS_TEMP calculation for equivalence class SS_TEMP.5	tsp_ro_005.tc	tsp_ro_005.ex
tsp_nr_006.m	Test THERMO_TEMP - outputs THERMO_TEMP calculation for equivalence class THERMO_TEMP.2	tsp_nr_006.tc	tsp_nr_006.ex
tsp_nr_007.m	Test THERMO_TEMP - outputs THERMO_TEMP calculation for equivalence class THERMO_TEMP.3	tsp_nr_007.tc	tsp_nr_007.ex
tsp_ro_008.m	Test THERMO_TEMP - outputs THERMO_TEMP calculation for equivalence class THERMO_TEMP.4	tsp_ro_008.tc	tsp_ro_008.ex
tsp_ro_009.m	Test THERMO_TEMP - outputs THERMO_TEMP calculation for equivalence class THERMO_TEMP.5	tsp_ro_009.tc	tsp_ro_009.ex
tsp_ro_010.m	Force use of THERMO_TEMP to test out of LOWER bound for THERMO_TEMP - Equivalence class THERMO_TEMP.7	tsp_ro_010.tc	tsp_ro_010.ex
tsp_ro_011.m	Force use of THERMO_TEMP to test out of UPPER bound for THERMO_TEMP - Equivalence class THERMO_TEMP.6	tsp_ro_011.tc	tsp_ro_011.ex

A.3.5 TDSP Functional Unit Test Cases

Table A.6 gives a listing of all requirements-based test cases for the TDSP functional unit. All test cases manipulate the variables:

TDS_STATUS
TD_COUNTER

Table 5.13 of the GCS Specification does not define the processing that is to occur if the TDS_STATUS is failed. Furthermore, there are no provisions to prevent this functional unit from executing when that occurs. To ensure robustness, it will be necessary to test the behavior of the functional unit when TDS_STATUS is failed. Table A.5 below lists the missing conditions from Table 5.13 of the GCS Specification and gives their respective test case. These cases are also given in Table A.6.

Table A.5: Conditions not given in Table 5.13 of the GCS Specification

Input		**Output**		**Test Case**
TDS_ STATUS	TD_ COUNTER	TD_ SENSED	TDS_ STATUS	Names
failed	all zeroes	unchanged	failed	TDSP_RO_004.TC
failed	all ones	unchanged	failed	TDSP_RO_005.TC
failed	mixture of ones & zeroes	unchanged	failed	TDSP_RO_006.TC

Table A.6: Test cases for TDSP functional unit.

Test Case Data File	**Description**	**Test-Input File**	**Expected-Results File**
tdsp_nr_001.m	Test healthy status & all counter bits off	tdsp_nr_001.tc	tdsp_nr_001.ex
tdsp_nr_002.m	Test healthy status & all counter bits on	tdsp_nr_002.tc	tdsp_nr_002.ex
tdsp_nr_003.m	Test healthy status & mixed counter bits	tdsp_nr_003.tc	tdsp_nr_003.ex
tdsp_ro_004.m	Test unhealthy status & zero counter	tdsp_ro_004.tc	tdsp_ro_004.ex
tdsp_ro_005.m	Test unhealthy status & all counter bits on	tdsp_ro_005.tc	tdsp_ro_005.ex
tdsp_ro_006.m	unhealthy status & mixed counter bits	tdsp_ro_006.tc	tdsp_ro_006.ex
tdsp_ro_007.m	Tests INVALID TDS_STATUS	tdsp_ro_007.tc	tdsp_ro_007.ex

A.3.6 TDLRSP Functional Unit Test Cases

Table A.8 is a listing of all test cases for the TDLRSP functional unit. All test cases manipulate the variables:

FRAME_COUNTER	TDLR_COUNTER
FRAME_BEAM_UNLOCKED	TDLR_STATE
K_MATRIX	TDLR_VELOCITY

For robustness testing purposes, Table 5.11 of the GCS Specification is missing several cases that should be tested. These conditions are given in Table A.7 below. Note that the *Beam_lock_time* calculated by:

Beam_lock_time = DELTA_T *(FRAME_COUNTER - FRAME_BEAM_UNLOCKED)

Table A.7 also identifies the test cases for each of those conditions. These cases are also repeated in Table A.8.

Table A.7: Conditions not given in Table 5.11 of the GCS Specification.

Input			**Output**		**Test Case**
TDLR_ STATE	TDLR_ COUNTER	*Beam_lock_time* ≥ TDLR_LOCK_TIME	TDLR_ STATE	FRAME_BEAM_ UNLOCKED	Names
locked	≠ 0	d	locked	Unchanged	TDLRSP_RO_006.TC
unlocked	≠ 0	no	unlocked	Unchanged	TDLRSP_RO_002.TC
unlocked	= 0	no	unlocked	Unchanged	TDLRSP_RO_004.TC

Table A.8: Test cases for TDLRSP functional unit.

Test Case Data File	Description	Test-Input File	Expected-Results File
tdlrsp_nr_001.m	Test: 1) TDLR_STATE = 0 & TDLR_COUNTER != 0 (line 2 of table 5.11) 2) line 16 of table 5.12 2) history rotation for TDLR_VELOCITY & K_MATRIX	tdlrsp_nr_001.tc	tdlrsp_nr_001.ex
tdlrsp_ro_002.m	Test: 1) TDLR_STATE = 0 & TDLR_COUNTER != 0 but elapsed time < TDLR_LOCK_TIME (not listed in table 5.11)	tdlrsp_ro_002.tc	tdlrsp_ro_002.ex
tdlrsp_nr_003.m	Test: TDLR_STATE = 0 & TDLR_COUNTER = 0 (line 3 of table 5.11)	tdlrsp_nr_003.tc	tdlrsp_nr_003.ex
tdlrsp_ro_004.m	Test: TDLR_STATE = 0 & TDLR_COUNTER = 0 but elapsed time < TDLR_LOCK_TIME (not listed in table 5.11)	tdlrsp_ro_004.tc	tdlrsp_ro_004.ex
tdlrsp_nr_005.m	Test: 1) TDLR_STATE = 1 & TDLR_COUNTER = 0 (line 1 of table 5.11) 2) line 1 of table 5.12 (no beams in lock)	tdlrsp_nr_005.tc	tdlrsp_nr_005.ex
tdlrsp_ro_006.m	Test: 1) TDLR_STATE = 1 & TDLR_COUNTER != 0 (not listed in table 5.11) 2) line 1 of table 5.12 (no beams in lock)	tdlrsp_ro_006.tc	tdlrsp_ro_006.ex
tdlrsp_nr_007.m	Test: Beam 1 in lock (line 2 of table 5.12)	tdlrsp_nr_007.tc	tdlrsp_nr_007.ex
tdlrsp_nr_008.m	Test: Beam 2 in lock (line 3 of table 5.12)	tdlrsp_nr_008.tc	tdlrsp_nr_008.ex
tdlrsp_nr_009.m	Test: Beam 3 in lock (line 4 of table 5.12)	tdlrsp_nr_009.tc	tdlrsp_nr_009.ex
tdlrsp_nr_010.m	Test: Beam 4 in lock (line 5 of table 5.12)	tdlrsp_nr_010.tc	tdlrsp_nr_010.ex
tdlrsp_nr_011.m	Test: Beams 1 & 2 in lock (line 6 of table 5.12)	tdlrsp_nr_011.tc	tdlrsp_nr_011.ex
tdlrsp_nr_012.m	Test: Beams 1 & 3 in lock (line 7 of table 5.12)	tdlrsp_nr_012.tc	tdlrsp_nr_012.ex
tdlrsp_nr_013.m	Test: Beams 1 & 4 in lock (line 8 of table 5.12)	tdlrsp_nr_013.tc	tdlrsp_nr_013.ex
tdlrsp_nr_014.m	Test: Beams 2 & 3 in lock (line 9 of table 5.12)	tdlrsp_nr_014.tc	tdlrsp_nr_014.ex
tdlrsp_nr_015.m	Test: Beams 2 & 4 in lock (line 10 of table 5.12)	tdlrsp_nr_015.tc	tdlrsp_nr_015.ex
tdlrsp_nr_016.m	Test: Beams 3 & 4 in lock (line 11 of table 5.12)	tdlrsp_nr_016.tc	tdlrsp_nr_016.ex
tdlrsp_nr_017.m	Test: Beams 1, 2, & 3 in lock (line 12 of table 5.12)	tdlrsp_nr_017.tc	tdlrsp_nr_017.ex
tdlrsp_nr_018.m	Test: Beams 1, 2, & 4 in lock (line 13 of table 5.12)	tdlrsp_nr_018.tc	tdlrsp_nr_018.ex
tdlrsp_nr_019.m	Test: Beams 1, 3, & 4 in lock (line 14 of table 5.12)	tdlrsp_nr_019.tc	tdlrsp_nr_019.ex
tdlrsp_nr_020.m	Test: Beams 2, 3, & 4 in lock (line 15 of table 5.12)	tdlrsp_nr_020.tc	tdlrsp_nr_020.ex
tdlrsp_nr_021.m	Test: ALL Beams in lock (line 16 of table 5.12)	tdlrsp_nr_021.tc	tdlrsp_nr_021.ex
tdlrsp_ro_022.m	Test FRAME_BEAM_UNLOCKED out of UPPER bound	tdlrsp_ro_022.tc	tdlrsp_ro_022.ex
tdlrsp_ro_023.m	Test FRAME_BEAM_UNLOCKED out of LOWER bound	tdlrsp_ro_023.tc	tdlrsp_ro_023.ex
tdlrsp_ro_024.m	Test FRAME_COUNTER out of UPPER bound	tdlrsp_ro_024.tc	tdlrsp_ro_024.ex
tdlrsp_ro_025.m	Test FRAME_COUNTER out of LOWER bound	tdlrsp_ro_025.tc	tdlrsp_ro_025.ex
tdlrsp_ro_026.m	Test TDLR_STATE INVALID value	tdlrsp_ro_026.tc	tdlrsp_ro_026.ex
tdlrsp_ro_027.m	Test TDLR_COUNTER out of LOWER bound	tdlrsp_ro_027.tc	tdlrsp_ro_027.ex
tdlrsp_ro_028.m	Test TDLR_COUNTER out of UPPER bound	tdlrsp_ro_028.tc	tdlrsp_ro_028.ex

A.3.7 GP Functional Unit Test Cases

Table 9 is a listing of all requirements-based test cases for the GP functional unit. All test cases manipulate the variables:

AE_SWITCH	GP_PHASE
AE_TEMP	GP_VELOCITY
AR_ALTITUDE	G_ROTATION
A_ACCELERATION	K_ALT
CHUTE_RELEASED	K_MATRIX
CL	RE_SWITCH
CONTOUR_CROSSED	TDLR_VELOCITY
FRAME_COUNTER	TDS_STATUS
GP_ALTITUDE	TD_SENSED
GP_ATTITUDE	

GP robustness test cases # 60 - 65 are supposed to provide out-of-bounds testing for GP_VELOCITY(1...3,0) which is both computed and then used in GP. The computation for this is impossible to reverse engineer to get starting values. Currently the best way to do this is to make other time histories (specifically GP_VELOCITY(1...3,2)) out of bounds, thereby forcing GP_VELOCITY(1...3,0) out of bounds.

Table A.9: Test cases for GP functional unit.

Test Case Data File	Description	Test-Input File	Expected-Results File
gp_tc.1	Initial GP Frame with All valid inputs. Tests Equivalence Classes: A_ACCELERATION.1 GP_ALTITUDE.1 GP_ATTITUDE.1 GP_VELOCITY.1 G_ROTATION.6 TDLR_VELOCITY.1 G_ROTATION.8 G_ROTATION.10	gp_nr_001.tc	gp_nr_001.ex
gp_tc.2	Transition Frame, Frame 246 with all valid inputs Tests Equivalence Classes: A_ACCELERATION.1 GP_ALTITUDE.1 GP_ATTITUDE.1 GP_VELOCITY.1 G_ROTATION.6 TDLR_VELOCITY.1 G_ROTATION.8 G_ROTATION.9	gp_nr_002.tc	gp_nr_002.ex
gp_tc.3	FRAME = 251 with CHUTE_RELEASED set to 1. All valid data tested. Also tests Equivalence Classes: A_ACCELERATION.1 GP_ALTITUDE.1 GP_ATTITUDE.1 GP_VELOCITY.1 G_ROTATION.6 TDLR_VELOCITY.1 G_ROTATION.8 G_ROTATION.9	gp_nr_003.tc	gp_nr_003.ex
gp_tc.4	FRAME = 252 with CHUTE_RELEASED = 1 where GP_PHASE goes to 3. All valid data tested. Also tests Equivalence Classes: A_ACCELERATION.1 GP_ALTITUDE.1 GP_ATTITUDE.1 GP_VELOCITY.1 G_ROTATION.6 TDLR_VELOCITY.1 G_ROTATION.8 G_ROTATION.9	gp_nr_004.tc	gp_nr_004.ex
gp_tc.5	FRAME 950 when CONTOUR_CROSSED will be set to 1 by the end of the frame. All valid data tested. Also tests Equivalence Classes: A_ACCELERATION.1 GP_ALTITUDE.1 GP_ATTITUDE.1 GP_VELOCITY.1 G_ROTATION.5 TDLR_VELOCITY.1 G_ROTATION.6	gp_nr_005.tc	gp_nr_005.ex
gp_tc.6	FRAME 951 with CONTOUR_CROSSED = 1. Tests all valid data and equivalence Classes: A_ACCELERATION.1 GP_ALTITUDE.1 GP_ATTITUDE.1 GP_VELOCITY.1 G_ROTATION.5 TDLR_VELOCITY.1 G_ROTATION.6	gp_nr_006.tc	gp_nr_006.ex
gp_tc.7	FRAME = 2073 when CL = 2. Tests valid data and Equivalence Classes: A_ACCELERATION.1 GP_ALTITUDE.1 GP_ATTITUDE.1 GP_VELOCITY.1 G_ROTATION.5 TDLR_VELOCITY.1 G_ROTATION.6	gp_nr_007.tc	gp_nr_007.ex
gp_tc.8	FRAME = 2078 where CL = 2 and GP_PHASE changes to 4. Tests valid data and Equivalence Classes: A_ACCELERATION.1 GP_ALTITUDE.1 GP_ATTITUDE.1 GP_VELOCITY.1 G_ROTATION.5 TDLR_VELOCITY.1 G_ROTATION.6	gp_nr_008.tc	gp_nr_008.ex
gp_tc.9	FRAME = 2073 where CL = goes to 2. Tests valid data & Equivalence Classes: A_ACCELERATION.1 GP_ALTITUDE.4 GP_ATTITUDE.1 GP_VELOCITY.1 G_ROTATION.5 TDLR_VELOCITY.1	gp_nr_053.tc	gp_nr_053.ex

gp_tc.10	FRAME = 2078 CL = 2, GP_PHASE changes to 5 (TD_SENSED = 1, GP_PHASE = 2, and engines are not HOT, Chute is attached) Tests valid data and Equivalence Classes: A_ACCELERATION.1 GP_ALTITUDE.3 GP_ATTITUDE.1 GP_VELOCITY.1 G_ROTATION.5 TDLR_VELOCITY.1	gp_nr_102.tc	gp_nr_102.ex
gp_tc.11	FRAME = 2078 CL = 2, GP_PHASE changes to 5 (ALT <= DROP_HEIGHT, TDS_STATUS = failed, GP_PHASE = 3). Tests valid data and Equivalence Classes: A_ACCELERATION.1 GP_ALTITUDE.1 GP_ATTITUDE.1 GP_VELOCITY.1 G_ROTATION.5 TDLR_VELOCITY.1	gp_nr_103.tc	gp_nr_103.ex
gp_tc.12	FRAME = 2078 CL = 2, GP_PHASE changes to 5 (Chute released, Engines Hot, Touchdown sensed). Tests valid data and Equivalence Classes: A_ACCELERATION.1 GP_ALTITUDE.1 GP_ATTITUDE.1 GP_VELOCITY.1 G_ROTATION.5 TDLR_VELOCITY.1	gp_nr_104.tc	gp_nr_104.ex
gp_tc.13	FRAME = 2078 CL = 2, GP_PHASE changes to 5 (Chute released, Engines off, Touchdown sensed). Tests valid data & Equivalence Classes: A_ACCELERATION.1 GP_ALTITUDE.1 GP_ATTITUDE.1 GP_VELOCITY.1 G_ROTATION.5 TDLR_VELOCITY.1	gp_nr_105.tc	gp_nr_105.ex
gp_tc.14	FRAME = 2078 CL = 2, GP_PHASE changes to 5 (Chute released, Engines off, TDS_STATUS = failed) Tests valid data and Equivalence Classes: A_ACCELERATION.1 GP_ALTITUDE.1 GP_ATTITUDE.1 GP_VELOCITY.1 G_ROTATION.5 TDLR_VELOCITY.1	gp_nr_106.tc	gp_nr_106.ex
gp_tc.15	Based on FRAME = 951 GPALT2 is < 0. (after rotation) Tests Equivalence Classes: GP_ALTITUDE.3	gp_ro_009.tc	gp_ro_009.ex
gp_tc.16	Based on FRAME = 951 GPALT2 is > 2000. (after rotation) Tests Equivalence Classes: GP_ALTITUDE.2	gp_ro_010.tc	gp_ro_010.ex
gp_tc.17	Based on FRAME = 951 A_ACCELERATION(1,0) < -20. Tests Equivalence Classes: A_ACCELERATION.3	gp_ro_011.tc	gp_ro_011.ex
gp_tc.18	Based on FRAME = 951 A_ACCELERATION(1,0) > 5. Tests Equivalence Classes: A_ACCELERATION.2	gp_ro_012.tc	gp_ro_012.ex
gp_tc.19	Based on FRAME = 951 A_ACCELERATION(2,0) < -20. Tests Equivalence Classes: A_ACCELERATION.3	gp_ro_013.tc	gp_ro_013.ex
gp_tc.20	Based on FRAME = 951 A_ACCELERATION(2,0) > 5. Tests Equivalence Classes: A_ACCELERATION.2	gp_ro_014.tc	gp_ro_014.ex
gp_tc.21	Based on FRAME = 951 A_ACCELERATION(3,0) < -20. Tests Equivalence Classes: A_ACCELERATION.3	gp_ro_015.tc	gp_ro_015.ex
gp_tc.22	Based on FRAME = 951 A_ACCELERATION(3,0) > 5. Tests Equivalence Classes: A_ACCELERATION.2	gp_ro_016.tc	gp_ro_016.ex
gp_tc.23	Based on FRAME = 951 A_ACCELERATION(1,1) < -20. Tests Equivalence Classes: A_ACCELERATION.3	gp_ro_017.tc	gp_ro_017.ex
gp_tc.24	Based on FRAME = 951 A_ACCELERATION(1,1) > 5. Tests Equivalence Classes: A_ACCELERATION.2	gp_ro_018.tc	gp_ro_018.ex
gp_tc.25	Based on FRAME = 951 A_ACCELERATION(2,1) < -20. Tests Equivalence Classes: A_ACCELERATION.3	gp_ro_019.tc	gp_ro_019.ex
gp_tc.26	Based on FRAME = 951 A_ACCELERATION(2,1) > 5. Tests Equivalence Classes: A_ACCELERATION.2	gp_ro_020.tc	gp_ro_020.ex
gp_tc.27	Based on FRAME = 951 A_ACCELERATION(3,1) < -20. Tests Equivalence Classes: A_ACCELERATION.3	gp_ro_021.tc	gp_ro_021.ex
gp_tc.28	Based on FRAME = 951 A_ACCELERATION(3,1) > 5. Tests Equivalence Classes: A_ACCELERATION.2	gp_ro_022.tc	gp_ro_022.ex
gp_tc.29	Based on FRAME = 951 A_ACCELERATION(1,2) < -20. Tests Equivalence Classes: A_ACCELERATION.3	gp_ro_023.tc	gp_ro_023.ex
gp_tc.30	Based on FRAME = 951 A_ACCELERATION(1,2) < 5. Tests Equivalence Classes: A_ACCELERATION.2	gp_ro_024.tc	gp_ro_024.ex

gp_tc.31	Based on FRAME = 951 A_ACCELERATION(2,2) < -20. Tests Equivalence Classes: A_ACCELERATION.3	gp_ro_025.tc	gp_ro_025.ex
gp_tc.32	Based on FRAME = 951 A_ACCELERATION(2,2) >5. Tests Equivalence Classes: A_ACCELERATION.2	gp_ro_026.tc	gp_ro_026.ex
gp_tc.33	Based on FRAME = 951 A_ACCELERATION(3,2) < -20. Tests Equivalence Classes: A_ACCELERATION.3	gp_ro_027.tc	gp_ro_027.ex
gp_tc.34	Based on FRAME = 951 A_ACCELERATION(3,2) > 5. Tests Equivalence Classes: A_ACCELERATION.2	gp_ro_028.tc	gp_ro_028.ex
gp_tc.35	Based on FRAME = 951 GP_ATTITUDE(1,1,2) > 1. (after rotation) Tests Equivalence Classes: GP_ATTITUDE.2	gp_ro_029.tc	gp_ro_029.ex
gp_tc.36	Based on FRAME = 951 GP_ATTITUDE(1,1,2) < -1. (after rotation) Tests Equivalence Classes: GP_ATTITUDE.3	gp_ro_030.tc	gp_ro_030.ex
gp_tc.37	Based on FRAME = 951 GP_ATTITUDE(1,2,2) > 1. (after rotation) Tests Equivalence Classes: GP_ATTITUDE.2	gp_ro_031.tc	gp_ro_031.ex
gp_tc.38	Based on FRAME = 951 GP_ATTITUDE(1,2,2) < -1. (after rotation) Tests Equivalence Classes: GP_ATTITUDE.3	gp_ro_032.tc	gp_ro_032.ex
gp_tc.39	Based on FRAME = 951 GP_ATTITUDE(1,3,2) > 1. (after rotation) Tests Equivalence Classes: GP_ATTITUDE.2	gp_ro_033.tc	gp_ro_033.ex
gp_tc.40	Based on FRAME = 951 GP_ATTITUDE(1,3,2) < -1. (after rotation) Tests Equivalence Classes: GP_ATTITUDE.3	gp_ro_034.tc	gp_ro_034.ex
gp_tc.41	Based on FRAME = 951 GP_ATTITUDE(2,1,2) > 1. (after rotation) Tests Equivalence Classes: GP_ATTITUDE.2	gp_ro_035.tc	gp_ro_035.ex
gp_tc.42	Based on FRAME = 951 GP_ATTITUDE(2,1,2) < -1. (after rotation) Tests Equivalence Classes: GP_ATTITUDE.3	gp_ro_036.tc	gp_ro_036.ex
gp_tc.43	Based on FRAME = 951 GP_ATTITUDE(2,2,2) > 1. (after rotation) Tests Equivalence Classes: GP_ATTITUDE.2	gp_ro_037.tc	gp_ro_037.ex
gp_tc.44	Based on FRAME = 951 GP_ATTITUDE(2,2,2) < -1. (after rotation) Tests Equivalence Classes: GP_ATTITUDE.3	gp_ro_038.tc	gp_ro_038.ex
gp_tc.45	Based on FRAME = 951 GP_ATTITUDE(2,3,2) > 1. (after rotation) Tests Equivalence Classes: GP_ATTITUDE.2	gp_ro_039.tc	gp_ro_039.ex
gp_tc.46	Based on FRAME = 951 GP_ATTITUDE(2,3,2) < -1. (after rotation) Tests Equivalence Classes: GP_ATTITUDE.3	gp_ro_040.tc	gp_ro_040.ex
gp_tc.47	Based on FRAME = 951 GP_ATTITUDE(3,1,2) > 1. (after rotation) Tests Equivalence Class GP_ATTITUDE.2	gp_ro_041.tc	gp_ro_041.ex
gp_tc.48	Based on FRAME = 951 GP_ATTITUDE(3,1,2) < -1. (after rotation) Tests Equivalence Classes: GP_ATTITUDE.3	gp_ro_042.tc	gp_ro_042.ex
gp_tc.49	Based on FRAME = 951 GP_ATTITUDE(3,2,2) > 1. (after rotation) Tests Equivalence Classes: GP_ATTITUDE.2	gp_ro_043.tc	gp_ro_043.ex
gp_tc.50	Based on FRAME = 951 GP_ATTITUDE(3,2,2) < -1. (after rotation) Tests Equivalence Classes: GP_ATTITUDE.3	gp_ro_044.tc	gp_ro_044.ex
gp_tc.51	Based on FRAME = 951 GP_ATTITUDE(3,3,2) > 1. (after rotation) Tests Equivalence Classes: GP_ATTITUDE.2	gp_ro_045.tc	gp_ro_045.ex
gp_tc.52	Based on FRAME = 951 GP_ATTITUDE(3,3,2) < -1. (after rotation) Tests Equivalence Classes: GP_ATTITUDE.3	gp_ro_046.tc	gp_ro_046.ex
gp_tc.53	FRAME = 951 ARALT0 is < 0. Tests Equivalence Classes: AR_ALTITUDE.3	gp_ro_047.tc	gp_ro_047.ex
gp_tc.54	FRAME = 951 ARALT0 is > 2000. Tests Equivalence Classes: AR_ALTITUDE.2	gp_ro_048.tc	gp_ro_048.ex
gp_tc.55	FRAME = 951 ARALT1 is < 0. Tests Equivalence Classes: AR_ALTITUDE.3	gp_ro_049.tc	gp_ro_049.ex
gp_tc.56	FRAME = 951 ARALT1 is > 2000. Tests Equivalence Classes: AR_ALTITUDE.2	gp_ro_050.tc	gp_ro_050.ex
gp_tc.57	FRAME = 951 ARALT2 is < 0. Tests Equivalence Classes: AR_ALTITUDE.3	gp_ro_051.tc	gp_ro_051.ex
gp_tc.58	FRAME = 951 ARALT2 is > 2000. Tests Equivalence Classes: AR_ALTITUDE.2	gp_ro_052.tc	gp_ro_052.ex
gp_tc.59	Based on FRAME = 951 GPVEL2(1) is < -100. (after rotation) Tests Equivalence Classes: GP_VELOCITY.3	gp_ro_054.tc	gp_ro_054.ex
gp_tc.60	Based on FRAME = 951 GPVEL2(1) is > 100. (after rotation) Tests Equivalence Classes: GP_VELOCITY.2	gp_ro_055.tc	gp_ro_055.ex
gp_tc.61	Based on FRAME = 951 GPVEL2(2) is < -100. (after rotation) Tests Equivalence Classes: GP_VELOCITY.3	gp_ro_056.tc	gp_ro_056.ex

gp_tc.62	Based on FRAME = 951 GPVEL2(2) is > 100. (after rotation) Tests Equivalence Classes: GP_VELOCITY.2	gp_ro_057.tc	gp_ro_057.ex
gp_tc.63	Based on FRAME = 951 GPVEL2(3) is < -100. (after rotation) Tests Equivalence Classes: GP_VELOCITY.3	gp_ro_058.tc	gp_ro_058.ex
gp_tc.64	Based on FRAME = 951 GPVEL2(3) is > 100. (after rotation) Tests Equivalence Classes: GP_VELOCITY.2	gp_ro_059.tc	gp_ro_059.ex
gp_tc.65	Based on FRAME = 951 GPVEL2(1) is > 100. (after rotation) forcing GPVEL0(1) to be out-of-bounds Tests Equivalence Classes: GP_VELOCITY.2	gp_ro_060.tc	gp_ro_060.ex
gp_tc.66	Based on FRAME = 951 GPVEL2(1) is < -100. (after rotation) forcing GPVEL0(1) to be out-of-bounds. Tests Equivalence Classes: GP_VELOCITY.3	gp_ro_061.tc	gp_ro_061.ex
gp_tc.67	Based on FRAME = 951 GPVEL2(2) is > 100. (after rotation) forcing GPVEL0(2) to be out-of-bounds Tests Equivalence Classes: GP_VELOCITY.2	gp_ro_062.tc	gp_ro_062.ex
gp_tc.68	Based on FRAME = 951 GPVEL2(2) is < -100. (after rotation) forcing GPVEL0(2) to be out-of-bounds Tests Equivalence Classes: GP_VELOCITY.3	gp_ro_063.tc	gp_ro_063.ex
gp_tc.69	Based on FRAME = 951 GPVEL2(3) is > 100. (after rotation) forcing GPVEL0(3) to be out-of-bounds Tests Equivalence Classes: GP_VELOCITY.2	gp_ro_064.tc	gp_ro_064.ex
gp_tc.70	Based on FRAME = 951 GPVEL2(3) is < -100. (after rotation) forcing GPVEL0(3) to be out-of-bounds Tests Equivalence Classes: GP_VELOCITY.3	gp_ro_065.tc	gp_ro_065.ex
gp_tc.71	Based on FRAME = 951 P0 = G_ROTATION(1, 0) < -1 (as used by the program in GP_ROTATION) Tests Equivalence Classes: G_ROTATION.12	gp_ro_066.tc	gp_ro_066.ex
gp_tc.72	Based on FRAME = 951 Q0 = G_ROTATION(2, 0) < -1 (as used by the program in GP_ROTATION) Tests Equivalence Classes: G_ROTATION.12	gp_ro_067.tc	gp_ro_067.ex
gp_tc.73	Based on FRAME = 951 R0 = G_ROTATION(3, 0) < -1 (as used by the program in GP_ROTATION) Tests Equivalence Classes: G_ROTATION.12	gp_ro_068.tc	gp_ro_068.ex
gp_tc.74	Based on FRAME = 951 p0 = G_ROTATION(1, 0) > 1 (as used by the program in GP_ROTATION) Tests Equivalence Classes: G_ROTATION.11	gp_ro_069.tc	gp_ro_069.ex
gp_tc.75	Based on FRAME = 951 q0 = G_ROTATION(2, 0) > 1 (as used by the program in GP_ROTATION) Tests Equivalence Classes: G_ROTATION.11	gp_ro_070.tc	gp_ro_070.ex
gp_tc.76	Based on FRAME = 951 r0 = G_ROTATION(2, 0) > 1 (as used by the program in GP_ROTATION) Tests Equivalence Classes: G_ROTATION.11	gp_ro_071.tc	gp_ro_071.ex
gp_tc.77	Based on FRAME = 951 p1 = G_ROTATION(1, 1) < -1 (as used by the program in GP_ROTATION) Tests Equivalence Classes: G_ROTATION.12	gp_ro_072.tc	gp_ro_072.ex
gp_tc.78	Based on FRAME = 951 q1 = G_ROTATION(2, 1) < -1 (as used by the program in GP_ROTATION) Tests Equivalence Classes: G_ROTATION.12	gp_ro_073.tc	gp_ro_073.ex
gp_tc.79	Based on FRAME = 951 r1 = G_ROTATION(3,1) < -1 (as used by the program in GP_ROTATION) Tests Equivalence Classes: G_ROTATION.12	gp_ro_074.tc	gp_ro_074.ex
gp_tc.80	Based on FRAME = 951 p1 = G_ROTATION(1, 1) > 1 (as used by the program in GP_ROTATION) Tests Equivalence Classes: G_ROTATION.11	gp_ro_075.tc	gp_ro_075.ex
gp_tc.81	Based on FRAME = 951 q1 = G_ROTATION(2, 1) > 1 (as used by the program in GP_ROTATION) Tests Equivalence Classes: G_ROTATION.11	gp_ro_076.tc	gp_ro_076.ex
gp_tc.82	Based on FRAME = 951 r1 = G_ROTATION(3, 1) < -1 (as used by the program in GP_ROTATION) Tests Equivalence Classes: G_ROTATION.11	gp_ro_077.tc	gp_ro_077.ex
gp_tc.83	Based on FRAME = 951 p2 = G_ROTATION(1, 2) < -1 (as used by the program in GP_ROTATION) Tests Equivalence Classes: G_ROTATION.12	gp_ro_078.tc	gp_ro_078.ex

gp_tc.84	Based on FRAME = 951 q2 = G_ROTATION(2, 2) < -1 (as used by the program in GP_ROTATION) Tests Equivalence Classes: G_ROTATION.12	gp_ro_079.tc	gp_ro_079.ex
gp_tc.85	Based on FRAME = 951 r2 = G_ROTATION(3, 2) < -1 (as used by the program in GP_ROTATION) Tests Equivalence Classes: G_ROTATION.12	gp_ro_080.tc	gp_ro_080.ex
gp_tc.86	Based on FRAME = 951 p0 = G_ROTATION(1, 2) > 1 (as used by the program in GP_ROTATION) Tests Equivalence Classes: G_ROTATION.11	gp_ro_081.tc	gp_ro_081.ex
gp_tc.87	Based on FRAME = 951 q2 = G_ROTATION(2,2) > 1 (as used by the program in GP_ROTATION) Tests Equivalence Classes: G_ROTATION.11	gp_ro_082.tc	gp_ro_082.ex
gp_tc.88	Based on FRAME = 951 r2 = G_ROTATION(3, 2) > 1 (as used by the program in GP_ROTATION) Tests Equivalence Classes: G_ROTATION.11	gp_ro_083.tc	gp_ro_083.ex
gp_tc.89	FRAME = 951 TDLVEL0 (1) < -100 Tests Equivalence Classes: TDLR_VELOCITY.3	gp_ro_084.tc	gp_ro_084.ex
gp_tc.90	FRAME = 951 TDLVEL0 (1) > 100 Tests Equivalence Classes: TDLR_VELOCITY.2	gp_ro_085.tc	gp_ro_085.ex
gp_tc.91	FRAME = 951 TDLVEL0 (2) < -100 Tests Equivalence Classes: TDLR_VELOCITY.3	gp_ro_086.tc	gp_ro_086.ex
gp_tc.92	FRAME = 951 TDLVEL0 (2) > 100 Tests Equivalence Classes: TDLR_VELOCITY.2	gp_ro_087.tc	gp_ro_087.ex
gp_tc.93	FRAME = 951 TDLVEL0 (3) < -100 Tests Equivalence Classes: TDLR_VELOCITY.3	gp_ro_088.tc	gp_ro_088.ex
gp_tc.94	FRAME = 951 TDLVEL0 (3) > 100 Tests Equivalence Classes: TDLR_VELOCITY.2	gp_ro_089.tc	gp_ro_089.ex
gp_tc.95	FRAME = 951 TDLVEL1 (1) < -100 Tests Equivalence Classes: TDLR_VELOCITY.3	gp_ro_090.tc	gp_ro_090.ex
gp_tc.96	FRAME = 951 TDLVEL1 (1) > 100 Tests Equivalence Classes: TDLR_VELOCITY.2	gp_ro_091.tc	gp_ro_091.ex
gp_tc.97	FRAME = 951 TDLVEL1 (2) < -100 Tests Equivalence Classes: TDLR_VELOCITY.3	gp_ro_092.tc	gp_ro_092.ex
gp_tc.98	FRAME = 951 TDLVEL1 (2) > 100 Tests Equivalence Classes: TDLR_VELOCITY.2	gp_ro_093.tc	gp_ro_093.ex
gp_tc.99	FRAME = 951 TDLVEL1 (3) < -100 Tests Equivalence Classes: TDLR_VELOCITY.3	gp_ro_094.tc	gp_ro_094.ex
gp_tc.100	FRAME = 951 TDLVEL1 (3) > 100 Tests Equivalence Classes: TDLR_VELOCITY.2	gp_ro_095.tc	gp_ro_095.ex
gp_tc.101	FRAME = 951 TDLVEL2 (1) < -100 Tests Equivalence Classes: TDLR_VELOCITY.3	gp_ro_096.tc	gp_ro_096.ex
gp_tc.102	FRAME = 951 TDLVEL2 (1) > 100 Tests Equivalence Classes: TDLR_VELOCITY.2	gp_ro_097.tc	gp_ro_097.ex
gp_tc.103	FRAME = 951 TDLVEL2 (2) < -100 Tests Equivalence Classes: TDLR_VELOCITY.3	gp_ro_098.tc	gp_ro_098.ex
gp_tc.104	FRAME = 951 TDLVEL2 (2) > 100 Tests Equivalence Classes: TDLR_VELOCITY.2	gp_ro_099.tc	gp_ro_099.ex
gp_tc.105	FRAME = 951 TDLVEL2 (3) < -100 Tests Equivalence Classes: TDLR_VELOCITY.3	gp_ro_100.tc	gp_ro_100.ex
gp_tc.106	FRAME = 951 TDLVEL2 (3) > 100 Tests Equivalence Classes: TDLR_VELOCITY.2	gp_ro_101.tc	gp_ro_101.ex
gp_tc.107	This is a special robustness test that tests the valid inputs not accounted for in the Spec table 5.10 In this test GP_PHASE = 1 and alt > ENGINES_ON_ALTITUDE Tests Equivalence Classes: A_ACCELERATION.1 GP_ALTITUDE.2 GP_ATTITUDE.1 GP_VELOCITY.1 G_ROTATION.10 TDLR_VELOCITY.1 G_ROTATION.6	gp_ro_107.tc	gp_ro_107.ex

gp_tc.108	This is a special robustness test that tests the valid inputs not accounted for in the Spec table 5.10 In this test GP_PHASE = 2, AE_TEMP = 0, CHUTE_RELEASED = 1 Tests Equivalence Classes: A_ACCELERATION.1 GP_ALTITUDE.2 GP_ATTITUDE.1 GP_VELOCITY.1 G_ROTATION.6 TDLR_VELOCITY.1 G_ROTATION.8 G_ROTATION.9	gp_ro_108.tc	gp_ro_108.ex
gp_tc.109	This is a special robustness test that tests the valid inputs not accounted for in the Spec table 5.10 In this test GP_PHASE = 2, AE_TEMP = 1, CHUTE_RELEASED = 1 Tests Equivalence Classes: A_ACCELERATION.1 GP_ALTITUDE.1 GP_ATTITUDE.1 GP_VELOCITY.1 G_ROTATION.6 TDLR_VELOCITY.1 G_ROTATION.8 G_ROTATION.9	gp_ro_109.tc	gp_ro_109.ex
gp_tc.110	This is a special robustness test that tests the valid inputs not accounted for in the Spec table 5.10 In this test GP_PHASE = 2, AE_TEMP = 2, CHUTE_RELEASED = 0 Tests Equivalence Classes: A_ACCELERATION.1 GP_ALTITUDE.1 GP_ATTITUDE.1 GP_VELOCITY.1 G_ROTATION.6 TDLR_VELOCITY.1 G_ROTATION.8 G_ROTATION.9	gp_ro_110.tc	gp_ro_110.ex
gp_tc.111	This is a special robustness test that tests the valid inputs not accounted for in the Spec table 5.10 In this test GP_PHASE = 3, AE_TEMP = 2, CHUTE_RELEASED = 1, TD_SENSED=0 alt > DROP_HEIGHT, TDS_STATUS = healthy, EQ <= MAX_NORMAL_VELOCITY. Tests Equivalence Classes: A_ACCELERATION.1 GP_ALTITUDE.1 GP_ATTITUDE.1 GP_VELOCITY.1 G_ROTATION.6 TDLR_VELOCITY.1 G_ROTATION.8 G_ROTATION.9	gp_ro_111.tc	gp_ro_111.ex
gp_tc.112	This is a special robustness test that tests the valid inputs not accounted for in the Spec table 5.10 In this test GP_PHASE = 3, AE_TEMP = 2, CHUTE_RELEASED = 1, TD_SENSED=0 alt <= DROP_HEIGHT, TDS_STATUS = failed, EQ <= MAX_NORMAL_VELOCITY Tests Equivalence Classes: A_ACCELERATION.1 GP_ALTITUDE.1 GP_ATTITUDE.1 GP_VELOCITY.1 G_ROTATION.6 TDLR_VELOCITY.1 G_ROTATION.8 G_ROTATION.9	gp_ro_112.tc	gp_ro_112.ex
gp_tc.113	This is a special robustness test that tests the valid inputs not accounted for in the Spec table 5.10 In this test GP_PHASE = 3, AE_TEMP = 2, CHUTE_RELEASED = 1, TD_SENSED=0 alt <= DROP_HEIGHT, TDS_STATUS = healthy, EQ > MAX_NORMAL_VELOCITY Tests Equivalence Classes: A_ACCELERATION.1 GP_ALTITUDE.1 GP_ATTITUDE.1 GP_VELOCITY.1 G_ROTATION.6 TDLR_VELOCITY.1 G_ROTATION.8 G_ROTATION.9	gp_ro_113.tc	gp_ro_113.ex
gp_tc.114	This is a special robustness test that tests the valid inputs not accounted for in the Spec table 5.10 In this test GP_PHASE = 3, AE_TEMP = 2, CHUTE_RELEASED = 1, TD_SENSED=0 alt > DROP_HEIGHT, TDS_STATUS = failed, EQ <= MAX_NORMAL_VELOCITY Tests Equivalence Classes: A_ACCELERATION.1 GP_ALTITUDE.1 GP_ATTITUDE.1 GP_VELOCITY.1 G_ROTATION.6 TDLR_VELOCITY.1 G_ROTATION.8 G_ROTATION.9	gp_ro_114.tc	gp_ro_114.ex

gp_tc.115	FRAME_COUNTER = 0, which is out-of-bounds, making FRAME_ENGINES_IGNITED out-of-bounds. FRAME_COUNTER is an input from the simulator, so this is an unusual case (an invalid case), but the only way it can be tested Tests Equivalence Classes: FRAME_ENGINES_IGNITED.3	gp_ro_115.tc	gp_ro_115.ex
gp_tc.116	FRAME_COUNTER = -32768, which is out-of-bounds, making FRAME_ENGINES_IGNITED out-of-bounds. FRAME_COUNTER is an input from the simulator, so this is an unusual case (an invalid case), but the only way it can be tested Tests Equivalence Classes: FRAME_ENGINES_IGNITED.2	gp_ro_116.tc	gp_ro_116.ex
gp_tc.117	This is a special robustness test that tests the valid inputs not accounted for in the Spec table 5.9 In this test AE_SWITCH = on, GP_ALTITUDE > DROP_HEIGHT, SQRT(2*GRAVITY*GP_ALTITUDE) + GP_VELOCITY(x) <= MAX_NORMAL_VELOCITY Tests Equivalence Classes: A_ACCELERATION.1 GP_ALTITUDE.1 GP_ATTITUDE.1 GP_VELOCITY.1 G_ROTATION.5 TDLR_VELOCITY.1 G_ROTATION.6	gp_ro_117.tc	gp_ro_117.ex

Tables 10a and 10b below provide more information about the robustness test cases that test table 5.10 of the GCS Specification. These table cover GP_PHASE transitions resulting from variable combinations that are possible but not specified. The information is divided into two tables to avoid confusion resulting from the heterogeneous mix of variables used in determining the value of GP_PHASE as given in Table 5.10 of the GCS Specification. Table A.10a covers transitions for GP_PHASE equal 1 and 2; while Table A.10b covers transitions for GP_PHASE equal 3

Table A.10a: Valid data not accounted for in Table 5.10 of the GCS specification

Input					**Output**	**Test Case**
GP_ PHASE	TD_SENSED	AE_ TEMP	CHUTE_ RELEASED	GP_ALTITUDE	GP_ PHASE	Names
1	Not Sensed	Cold	Not Released	> ENGINES_ON_ ALTITUDE	1	GP_RO_107.TC
2	Not Sensed	Cold	Released	< ENGINES_ON_ ALTITUDE	2	GP_RO_108.TC
2	Not Sensed	Warm	Released	< ENGINES_ON_ ALTITUDE	2	GP_RO_109.TC
2	Not Sensed	Hot	Not Released	< ENGINES_ON_ ALTITUDE	2	GP_RO_110.TC

Table A.10a: Valid data not accounted for in Table 5.10 (Part B) of the GCS specification

Input							**Output**	**Test Case**
GP_ PHASE	TD_ SENSED	AE_ TEMP	CHUTE_ RELEASED	Altitude	$\sqrt{2 \bullet \mathrm{Gravity} \bullet \mathrm{GP\_ALTITUDE}} + \mathrm{GP\_VELOCITY}(x)$	TDS_ STATUS	GP_ PHASE	Names
3	Not Sensed	Hot	Released	>DROP_ HEIGHT	≤ MAX_NORMAL_ VELOCITY	healthy	3	GP_RO_111.TC
3	Not Sensed	Hot	Released	≤DROP_ HEIGHT	≤ MAX_NORMAL_ VELOCITY	failed	3	GP_RO_112.TC
3	Not Sensed	Hot	Released	≤DROP_ HEIGHT	>MAX_NORMAL_ VELOCITY	healthy	3	GP_RO_113.TC
3	Not Sensed	Hot	Released	>DROP_ HEIGHT	≤ MAX_NORMAL_ VELOCITY	failed	3	GP_RO_114.TC

A.3.8 AECLP Functional Unit Test Cases

Table A.11 gives a listing of all requirements-based test cases for the AECLP functional unit. Table A.12 gives additional AE_TEMP transitions for robustness test cases that test Table 5.1 of the GCS Specification. It covers conditions not given in Table 5.1 of the GCS Specification. All test cases manipulate the variables:

A_ACCELERATION	GP_ROTATION
AE_SWITCH	GP_VELOCITY
AE_TEMP	INTERNAL_CMD
CHUTE_RELEASED	PE_INTEGRAL
CL	TE_DROP
CONTOUR_CROSSED	TE_INTEGRAL
FRAME_COUNTER	TE_LIMIT
FRAME_ENGINES_IGNITED	VELOCITY_ERROR
GP_ALTITUDE	YE_INTEGRAL
GP_ATTITUDE	

Table A.11: Test cases for AECLP functional unit.

Test Case Data File	Description	Test-Input File	Expected-Results File
aeclp_tc.1	Initial AECLP Frame. Tests valid inputs and Equivalence Classes: A_ACCELERATION.1 PE_INTEGRAL.1 YE_INTEGRAL.1 TE_INTEGRAL.1 TE_LIMIT.1 INTERNAL.CMD.1 AE_CMD.1 GP_ALTITUDE.1 VELOCITY_ERROR.1 G_ROTATION.6	aeclp_nr_001.tc	aeclp_nr_001.ex
aeclp_tc.2	Frame 2, tests valid inputs and Equivalence Classes: A_ACCELERATION.1 PE_INTEGRAL.1 YE_INTEGRAL.1 TE_INTEGRAL.1 TE_LIMIT.1 INTERNAL.CMD.1 AE_CMD.1 GP_ALTITUDE.1 VELOCITY_ERROR.1 G_ROTATION.6	aeclp_nr_002.tc	aeclp_nr_002.ex
aeclp_tc.3	Frame 251, tests valid inputs and Equivalence Classes: A_ACCELERATION.1 PE_INTEGRAL.1 YE_INTEGRAL.1 TE_INTEGRAL.1 TE_LIMIT.1 INTERNAL.CMD.1 AE_CMD.1 GP_ALTITUDE.1 VELOCITY_ERROR.1 G_ROTATION.6	aeclp_nr_003.tc	aeclp_nr_003.ex
aeclp_tc.4	Frame 252, tests valid inputs and Equivalence Classes: A_ACCELERATION.1 PE_INTEGRAL.1 YE_INTEGRAL.1 TE_INTEGRAL.1 TE_LIMIT.1 INTERNAL.CMD.1 AE_CMD.1 GP_ALTITUDE.1 VELOCITY_ERROR.1 G_ROTATION.8 G_ROTATION.6	aeclp_nr_004.tc	aeclp_nr_004.ex
aeclp_tc.5	Frame 950, tests valid inputs and Equivalence Classes: A_ACCELERATION.1 PE_INTEGRAL.1 YE_INTEGRAL.1 TE_INTEGRAL.1 TE_LIMIT.1 INTERNAL.CMD.1 AE_CMD.1 GP_ALTITUDE.1 VELOCITY_ERROR.1 G_ROTATION.10 G_ROTATION.5	aeclp_nr_005.tc	aeclp_nr_005.ex
aeclp_tc.6	Frame 951, tests valid inputs and Equivalence Classes: A_ACCELERATION.1 PE_INTEGRAL.1 YE_INTEGRAL.1 TE_INTEGRAL.1 TE_LIMIT.2 INTERNAL.CMD.1 AE_CMD.1 GP_ALTITUDE.1 VELOCITY_ERROR.1 G_ROTATION.10 G_ROTATION.5	aeclp_nr_006.tc	aeclp_nr_006.ex
aeclp_tc.7	Frame 2077 tests valid inputs and Equivalence Classes: A_ACCELERATION.1 PE_INTEGRAL.1 YE_INTEGRAL.1 TE_INTEGRAL.1 TE_LIMIT.2 INTERNAL.CMD.1 AE_CMD.1 GP_ALTITUDE.1 VELOCITY_ERROR.1 G_ROTATION.10 G_ROTATION.5	aeclp_nr_007.tc	aeclp_nr_007.ex
aeclp_tc.8	Frame 2078 tests valid inputs and Equivalence Classes: A_ACCELERATION.1 PE_INTEGRAL.1 YE_INTEGRAL.1 TE_INTEGRAL.1 TE_LIMIT.2 INTERNAL.CMD.1 AE_CMD.1 GP_ALTITUDE.1 VELOCITY_ERROR.1 G_ROTATION.6	aeclp_nr_008.tc	aeclp_nr_008.ex

aeclp_tc.9	Frame 2083 tests valid inputs and Equivalence Classes: A_ACCELERATION.1 PE_INTEGRAL.1 YE_INTEGRAL.1 TE_INTEGRAL.1 TE_LIMIT.2 INTERNAL.CMD.1 AE_CMD.1 GP_ALTITUDE.1 VELOCITY_ERROR.1	aeclp_nr_009.tc	aeclp_nr_009.ex
aeclp_tc.10	Frame 250 tests valid inputs and Equivalence Classes: A_ACCELERATION.1 PE_INTEGRAL.1 YE_INTEGRAL.1 TE_INTEGRAL.1 TE_LIMIT.1 INTERNAL.CMD.1 AE_CMD.1 GP_ALTITUDE.1 VELOCITY_ERROR.1 G_ROTATION.8 G_ROTATION.6	aeclp_nr_010.tc	aeclp_nr_010.ex
aeclp_tc.11	Frame 949 tests valid inputs and Equivalence Classes: A_ACCELERATION.1 PE_INTEGRAL.1 YE_INTEGRAL.1 TE_INTEGRAL.1 TE_LIMIT.1 INTERNAL.CMD.1 AE_CMD.1 GP_ALTITUDE.1 VELOCITY_ERROR.1 G_ROTATION.5	aeclp_nr_011.tc	aeclp_nr_011.ex
aeclp_tc.12	Frame 955 tests valid inputs and Equivalence Classes: A_ACCELERATION.1 PE_INTEGRAL.1 YE_INTEGRAL.1 TE_INTEGRAL.1 TE_LIMIT.2 INTERNAL.CMD.1 AE_CMD.1 GP_ALTITUDE.1 VELOCITY_ERROR.1 G_ROTATION.5	aeclp_nr_012.tc	aeclp_nr_012.ex
aeclp_tc.13	Tests Frame 955 with all valid inputs and Equivalence Classes: GP_ALTITUDE.4	aeclp_ro_013.tc	aeclp_ro_013.ex
aeclp_tc.14	Tests Frame 955 with all valid inputs and Equivalence Classes: GP_ALTITUDE.3	aeclp_ro_014.tc	aeclp_ro_014.ex
aeclp_tc.15	Tests Frame 955 with all valid inputs and Equivalence Classes: GP_ATTITUDE.3	aeclp_ro_015.tc	aeclp_ro_015.ex
aeclp_tc.16	Tests Frame 955 with all valid inputs and Equivalence Classes: GP_ATTITUDE.2	aeclp_ro_016.tc	aeclp_ro_016.ex
aeclp_tc.17	Tests Frame 955 with all valid inputs and Equivalence Classes: GP_ROTATION.3	aeclp_ro_017.tc	aeclp_ro_017.ex
aeclp_tc.18	Tests Frame 955 with all valid inputs and Equivalence Classes: GP_ROTATION.2	aeclp_ro_018.tc	aeclp_ro_018.ex
aeclp_tc.19	Tests Frame 955 with all valid inputs and Equivalence Classes: GP_ROTATION.3	aeclp_ro_019.tc	aeclp_ro_019.ex
aeclp_tc.20	Tests Frame 955 with all valid inputs and Equivalence Classes: GP_ROTATION.2	aeclp_ro_020.tc	aeclp_ro_020.ex
aeclp_tc.21	Tests Frame 955 with all valid inputs and Equivalence Classes: GP_VELOCITY.3	aeclp_ro_021.tc	aeclp_ro_021.ex
aeclp_tc.22	Tests Frame 955 with all valid inputs and Equivalence Classes: GP_VELOCITY.2	aeclp_ro_022.tc	aeclp_ro_022.ex
aeclp_tc.23	Tests Frame 955 with all valid inputs and Equivalence Classes: GP_VELOCITY.3	aeclp_ro_023.tc	aeclp_ro_023.ex
aeclp_tc.24	Tests Frame 955 with all valid inputs and Equivalence Classes: GP_VELOCITY.2	aeclp_ro_024.tc	aeclp_ro_024.ex
aeclp_tc.25	Tests Frame 955 with all valid inputs and Equivalence Classes: GP_VELOCITY.3	aeclp_ro_025.tc	aeclp_ro_025.ex
aeclp_tc.26	Tests Frame 955 with all valid inputs and Equivalence Classes: GP_VELOCITY.2	aeclp_ro_026.tc	aeclp_ro_026.ex
aeclp_tc.27	Tests Frame 955 with all valid inputs and Equivalence Classes: PE_INTEGRAL.3	aeclp_ro_027.tc	aeclp_ro_027.ex
aeclp_tc.28	Tests Frame 955 with all valid inputs and Equivalence Classes: PE_INTEGRAL.2	aeclp_ro_028.tc	aeclp_ro_028.ex
aeclp_tc.29	Tests Frame 955 with all valid inputs and Equivalence Classes: TE_INTEGRAL.3	aeclp_ro_029.tc	aeclp_ro_029.ex

aeclp_tc.30	Tests Frame 955 with all valid inputs and Equivalence Classes: TE_INTEGRAL.2 TE_LIMIT.3	aeclp_ro_030.tc	aeclp_ro_030.ex
aeclp_tc.31	Tests Frame 955 with all valid inputs and Equivalence Classes: TE_LIMIT.5	aeclp_ro_031.tc	aeclp_ro_031.ex
aeclp_tc.32	Tests Frame 955 with all valid inputs and Equivalence Classes: TE_LIMIT.4	aeclp_ro_032.tc	aeclp_ro_032.ex
aeclp_tc.33	Tests Frame 955 with all valid inputs and Equivalence Classes: VELOCITY_ERROR.3	aeclp_ro_033.tc	aeclp_ro_033.ex
aeclp_tc.34	Tests Frame 955 with all valid inputs and Equivalence Classes: VELOCITY_ERROR.3 TE_LIMIT.3	aeclp_ro_034.tc	aeclp_ro_034.ex
aeclp_tc.35	Tests Frame 955 with all valid inputs and Equivalence Classes: YE_INTEGRAL.3	aeclp_ro_035.tc	aeclp_ro_035.ex
aeclp_tc.36	Tests Frame 955 with all valid inputs and Equivalence Classes: YE_INTEGRAL.2	aeclp_ro_036.tc	aeclp_ro_036.ex
aeclp_tc.37	Tests Frame 955 with all valid inputs and Equivalence Classes: A_ACCELERATION.3	aeclp_ro_037.tc	aeclp_ro_037.ex
aeclp_tc.38	Tests Frame 955 with all valid inputs and Equivalence Classes: A_ACCELERATION.2	aeclp_ro_038.tc	aeclp_ro_038.ex
aeclp_tc.39	This robustness case tests a condition not listed in table 5.1 of the Spec. The combination of these values may cause invalid state transitions. Also Tests Equivalence Classes: A_ACCELERATION.1 PE_INTEGRAL.1 YE_INTEGRAL.1 TE_INTEGRAL.1 TE_LIMIT.1 INTERNAL.CMD.1 AE_CMD.1 GP_ALTITUDE.2 VELOCITY_ERROR.1	aeclp_ro_039.tc	aeclp_ro_039.ex
aeclp_tc.40	This robustness case tests a condition not listed in table 5.1 of the Spec. The combination of these values may cause invalid state transitions. Also Tests Equivalence Classes: A_ACCELERATION.1 PE_INTEGRAL.1 YE_INTEGRAL.1 TE_INTEGRAL.1 TE_LIMIT.1 INTERNAL.CMD.1 AE_CMD.1 GP_ALTITUDE.2 VELOCITY_ERROR.1	aeclp_ro_040.tc	aeclp_ro_040.ex
aeclp_tc.41	This robustness case tests a condition not listed in table 5.1 of the Spec. The combination of these values may cause invalid state transitions. Also Tests Equivalence Classes: A_ACCELERATION.1 PE_INTEGRAL.1 YE_INTEGRAL.1 TE_INTEGRAL.1 TE_LIMIT.1 INTERNAL.CMD.1 AE_CMD.1 GP_ALTITUDE.1 VELOCITY_ERROR.1	aeclp_ro_041.tc	aeclp_ro_041.ex
aeclp_tc.42	This robustness case tests a condition not listed in table 5.1 of the Spec. The combination of these values may cause invalid state transitions. Also Tests Equivalence Classes: A_ACCELERATION.1 PE_INTEGRAL.1 YE_INTEGRAL.1 TE_INTEGRAL.1 TE_LIMIT.1 INTERNAL.CMD.1 AE_CMD.1 GP_ALTITUDE.2 VELOCITY_ERROR.1	aeclp_ro_042.tc	aeclp_ro_042.ex
aeclp_tc.43	This robustness case tests a condition not listed in table 5.1 of the Spec. The combination of these values may cause invalid state transitions. Also Tests Equivalence Classes: A_ACCELERATION.1 PE_INTEGRAL.1 YE_INTEGRAL.1 TE_INTEGRAL.1 TE_LIMIT.1 INTERNAL.CMD.1 AE_CMD.1 GP_ALTITUDE.1 VELOCITY_ERROR.1	aeclp_ro_043.tc	aeclp_ro_043.ex

aeclp_tc.44	This robustness case tests a condition not listed in table 5.1 of the Spec. The combination of these values may cause invalid state transitions. Also Tests Equivalence Classes: A_ACCELERATION.1 PE_INTEGRAL.1 YE_INTEGRAL.1 TE_INTEGRAL.1 TE_LIMIT.1 INTERNAL.CMD.1 AE_CMD.1 GP_ALTITUDE.2 VELOCITY_ERROR.1	aeclp_ro_044.tc	aeclp_ro_044.ex
aeclp_tc.45	This robustness case tests a condition not listed in table 5.1 of the Spec. The combination of these values may cause invalid state transitions. Also Tests Equivalence Classes: A_ACCELERATION.1 PE_INTEGRAL.1 YE_INTEGRAL.1 TE_INTEGRAL.1 TE_LIMIT.1 INTERNAL.CMD.1 AE_CMD.1 GP_ALTITUDE.2 VELOCITY_ERROR.1	aeclp_ro_045.tc	aeclp_ro_045.ex
aeclp_tc.46	This robustness case tests a condition not listed in table 5.1 of the Spec. The combination of these values may cause invalid state transitions. Also Tests Equivalence Classes: A_ACCELERATION.1 PE_INTEGRAL.1 YE_INTEGRAL.1 TE_INTEGRAL.1 TE_LIMIT.1 INTERNAL.CMD.1 AE_CMD.1 GP_ALTITUDE.1 VELOCITY_ERROR.1	aeclp_ro_046.tc	aeclp_ro_046.ex
aeclp_tc.47	This robustness case tests a condition not listed in table 5.1 of the Spec. The combination of these values may cause invalid state transitions. Also Tests Equivalence Classes: A_ACCELERATION.1 PE_INTEGRAL.1 YE_INTEGRAL.1 TE_INTEGRAL.1 TE_LIMIT.1 INTERNAL.CMD.1 AE_CMD.1 GP_ALTITUDE.2 VELOCITY_ERROR.1	aeclp_ro_047.tc	aeclp_ro_047.ex
aeclp_tc.48	This case uses all valid inputs, but the value for G_ROTATION(2) has been computed to give a specific result in INTERNAL_CMD. INTERNAL_CMD(1) = -.701 (which is out of bounds) Tests Equivalence Classes: INTERNAL.CMD.3	aeclp_ro_048.tc	aeclp_ro_048.ex
aeclp_tc.49	This case uses all valid inputs, but the value for G_ROTATION(2) has been computed to give a specific result in INTERNAL_CMD. INTERNAL_CMD(1) = 1.701 (which is out of bounds) Tests Equivalence Classes: INTERNAL.CMD.2	aeclp_ro_049.tc	aeclp_ro_049.ex
aeclp_tc.50	This case uses all valid inputs, but the value for G_ROTATION(3) has been computed to give a specific result in INTERNAL_CMD. INTERNAL_CMD(2) = -.701 (which is out of bounds) Tests Equivalence Classes: INTERNAL.CMD.3	aeclp_ro_050.tc	aeclp_ro_050.ex
aeclp_tc.51	This case uses all valid inputs, but the value for G_ROTATION(3) has been computed to give a specific result in INTERNAL_CMD. INTERNAL_CMD(2) = 1.701 (which is out of bounds) Tests Equivalence Classes: INTERNAL.CMD.2	aeclp_ro_051.tc	aeclp_ro_051.ex
aeclp_tc.52	This case uses all valid inputs, but the value for TE_INIT has been computed to give a specific result in INTERNAL_CMD. INTERNAL_CMD(3) = -.701 (which is out of bounds) Tests Equivalence Classes: INTERNAL.CMD.3	aeclp_ro_052.tc	aeclp_ro_052.ex
aeclp_tc.53	This case uses all valid inputs, but the value for TE_INIT has been computed to give out of bound results in INTERNAL_CMD. INTERNAL_CMD(3) = 1.701 Tests Equivalence Classes: INTERNAL.CMD.2	aeclp_ro_053.tc	aeclp_ro_053.ex

aeclp_tc.54	AE_SWITCH is still off at end of frame, giving AE_CMD = 0 FRAME_ENGINES_IGNITED > 1 All valid inputs. Tests Equivalence Classes: A_ACCELERATION.1 PE_INTEGRAL.1 YE_INTEGRAL.1 TE_INTEGRAL.1 TE_LIMIT.1 INTERNAL.CMD.1 AE_CMD.1 GP_ALTITUDE.1 VELOCITY_ERROR.1	aeclp_nr_054.tc	aeclp_nr_054.ex
aeclp_tc.55	This tests INTERNAL_CMD > 1.0 Tests Equivalence Classes: A_ACCELERATION.1 PE_INTEGRAL.1 YE_INTEGRAL.1 TE_INTEGRAL.1 TE_LIMIT.1 INTERNAL.CMD.1 AE_CMD.1 GP_ALTITUDE.1 VELOCITY_ERROR.1	aeclp_nr_055.tc	aeclp_nr_055.ex
aeclp_tc.56	Tests Equivalence Classes: FRAME_ENGINES_IGNITED.2	aeclp_ro_056.tc	aeclp_ro_056.ex
aeclp_tc.57	Tests Equivalence Classes: FRAME_ENGINES_IGNITED.3	aeclp_ro_057.tc	aeclp_ro_057.ex

Table A.12: AE_TEMP transitions not covered in Table 5.1 of GCS Specification.

Input			Output	Test Case
AE_TEMP	GP_ALTITUDE	(FRAME_COUNTER - FRAME_ENGINES_IGNITED) * DELTA_T	AE_TEMP	Names
COLD	> ENGINES_ON_ALTITUDE	< FULL_UP_TIME	COLD	AECLP_RO_39.TC
COLD	> ENGINES_ON_ALTITUDE	≥ FULL_UP_TIME	COLD	AECLP_RO_40.TC
COLD	≤ ENGINES_ON_ALTITUDE	≥ FULL_UP_TIME	COLD	AECLP_RO_41.TC
WARM	> ENGINES_ON_ALTITUDE	< FULL_UP_TIME	WARM	AECLP_RO_42.TC
WARM	≤ ENGINES_ON_ALTITUDE	< FULL_UP_TIME	WARM	AECLP_RO_43.TC
WARM	> ENGINES_ON_ALTITUDE	≥ FULL_UP_TIME	WARM	AECLP_RO_44.TC
HOT	> ENGINES_ON_ALTITUDE	< FULL_UP_TIME	HOT	AECLP_RO_45.TC
HOT	≤ ENGINES_ON_ALTITUDE	< FULL_UP_TIME	HOT	AECLP_RO_46.TC
HOT	> ENGINES_ON_ALTITUDE	≥ FULL_UP_TIME	HOT	AECLP_RO_47.TC

A.3.9 RECLP Functional Unit Test Cases

The requirements-based test cases for the RECLP functional unit are given in Table A.13. This test suite involves three test variables, RE_SWITCH, G_ROTATION, and THETA. RE_SWITCH is 1 for all test cases the values for the other two variables are given in the Description column. The majority of the testing for this functional unit involves determination of RE_CMD based on the values of G_ROTATION and THETA. RE_CMD is determined by plotting G_ROTATION and THETA on Figure A.5.2 of the GCS Specification.

Table A.13: Test cases for RECLP functional unit.

Test Case Data File	Description	Test-Input File	Expected-Results File
reclp_tc.1	This case tests THETA = 0.002569999999999999, G_ROTATION = 0.00157 RE_CMD = 1. Tests valid inputs and Equivalence Classes: G_ROTATION.1 THETA.2	reclp_nr_001.tc	reclp_nr_001.ex
reclp_tc.2	This case tests THETA = -0.002569999999999999, G_ROTATION = 0.00157 RE_CMD = 1. Tests valid inputs and Equivalence Classes: G_ROTATION.1 THETA.3	reclp_nr_002.tc	reclp_nr_002.ex
reclp_tc.3	This case tests THETA = -0.002569999999999999, G_ROTATION = -0.00157 RE_CMD = 1. Tests valid inputs and Equivalence Classes: G_ROTATION.1 THETA.3	reclp_nr_003.tc	reclp_nr_003.ex
reclp_tc.4	This case tests THETA = 0.002569999999999999, G_ROTATION = -0.00157 & RE_CMD = 1. Tests valid inputs and Equivalence Classes: G_ROTATION.1 THETA.4	reclp_nr_004.tc	reclp_nr_004.ex
reclp_tc.5	This case tests THETA = 0.00478, G_ROTATION = -0.00157 RE_CMD = 1. Tests valid inputs and Equivalence Classes: G_ROTATION.1 THETA.5	reclp_nr_005.tc	reclp_nr_005.ex
reclp_tc.6	This case tests THETA = -0.00478, G_ROTATION = -0.00157 RE_CMD should be 2. Tests valid inputs and Equivalence Classes: G_ROTATION.1 THETA.2	reclp_nr_006.tc	reclp_nr_006.ex

reclp_tc.7	This case tests THETA = -0.00478, G_ROTATION = 0.00157 RE_CMD should be 1. Tests valid inputs and Equivalence Classes: G_ROTATION.1 THETA.2	reclp_nr_007.tc	reclp_nr_007.ex
reclp_tc.8	This case tests THETA = 0.00478, G_ROTATION = 0.00157 RE_CMD should be 2. Tests valid inputs and Equivalence Classes: G_ROTATION.1 THETA.5	reclp_nr_008.tc	reclp_nr_008.ex
reclp_tc.9	This case tests THETA = 0.00634, G_ROTATION = 0.00157 RE_CMD should be 7. Tests valid inputs and Equivalence Classes: G_ROTATION.1 THETA.6	reclp_nr_009.tc	reclp_nr_009.ex
reclp_tc.10	This case tests THETA = -0.00634, G_ROTATION = 0.00157 RE_CMD should be 6. Tests valid inputs and Equivalence Classes: G_ROTATION.1 THETA.1	reclp_nr_010.tc	reclp_nr_010.ex
reclp_tc.11	This case tests THETA = -0.00634, G_ROTATION = -0.00157 RE_CMD should be 6. Tests valid inputs and Equivalence Classes: G_ROTATION.1 THETA.1	reclp_nr_011.tc	reclp_nr_011.ex
reclp_tc.12	This case tests THETA = 0.00634, G_ROTATION = -0.00157 RE_CMD should be 7. Tests valid inputs and Equivalence Classes: G_ROTATION.1 THETA.6	reclp_nr_012.tc	reclp_nr_012.ex
reclp_tc.13	This case tests THETA = 0.002569999999999999, G_ROTATION = 0.00828 RE_CMD should be 7. Tests valid inputs and Equivalence Classes: G_ROTATION.1 THETA.4	reclp_nr_013.tc	reclp_nr_013.ex
reclp_tc.14	This case tests THETA = -0.002569999999999999, G_ROTATION = 0.00828 RE_CMD should be 1 Tests valid inputs and Equivalence Classes: G_ROTATION.1 THETA.3	reclp_nr_014.tc	reclp_nr_014.ex
reclp_tc.15	This case tests THETA = -0.002569999999999999, G_ROTATION = -0.00828 RE_CMD should be 6 Tests valid inputs and Equivalence Classes: G_ROTATION.1 THETA.3	reclp_nr_015.tc	reclp_nr_015.ex

reclp_tc.16	This case tests THETA = 0.002569999999999999, G_ROTATION = -0.00828 RE_CMD should be 1 Tests valid inputs and Equivalence Classes: G_ROTATION.1 THETA.4	reclp_nr_016.tc	reclp_nr_016.ex
reclp_tc.17	This case tests THETA = 0.00634 & G_ROTATION = -0.00828 RE_CMD should be 7 Tests valid inputs and Equivalence Classes: G_ROTATION.1 THETA.6	reclp_nr_017.tc	reclp_nr_017.ex
reclp_tc.18	This case test following: THETA = 0.00634 & G_ROTATION = 0.00828 RE_CMD should be 7 Tests valid inputs and Equivalence Classes: G_ROTATION.1 THETA.6	reclp_nr_018.tc	reclp_nr_018.ex
reclp_tc.19	This case test following: THETA = -0.00634 & G_ROTATION = 0.00828 RE_CMD should be 6 Tests valid inputs and Equivalence Classes: G_ROTATION.1 THETA.1	reclp_nr_019.tc	reclp_nr_019.ex
reclp_tc.20	This case test following: THETA = -0.00634 & G_ROTATION = -0.00828 RE_CMD should be 6 Tests valid inputs and Equivalence Classes: G_ROTATION.1 THETA.1	reclp_nr_020.tc	reclp_nr_020.ex
reclp_tc.21	This case test following: THETA = 0.0042 & G_ROTATION = 0.00826 RE_CMD should be 5 Tests valid inputs and Equivalence Classes: G_ROTATION.1 THETA.4	reclp_nr_021.tc	reclp_nr_021.ex
reclp_tc.22	This case test following: THETA = -0.0042 & G_ROTATION = 0.00826 RE_CMD should be 1 Tests valid inputs and Equivalence Classes: G_ROTATION.1 THETA.3	reclp_nr_022.tc	reclp_nr_022.ex
reclp_tc.23	This case test following: THETA = -0.0042 & G_ROTATION = -0.00826 RE_CMD should be 4 Tests valid inputs and Equivalence Classes: G_ROTATION.1 THETA.3	reclp_nr_023.tc	reclp_nr_023.ex
reclp_tc.24	This case test following: THETA = 0.0042 & G_ROTATION = -0.00826 RE_CMD should be 1 Tests valid inputs and Equivalence Classes: G_ROTATION.1 THETA.4	reclp_nr_024.tc	reclp_nr_024.ex

reclp_tc.25	This case test following: THETA = 0.0065 & G_ROTATION = -0.00826 RE_CMD should be 7 Tests valid inputs and Equivalence Classes: G_ROTATION.1 THETA.6	reclp_nr_025.tc	reclp_nr_025.ex
reclp_tc.26	This case tests with: THETA = -0.0061, G_ROTATION = -0.00826 RE_CMD should be 6	reclp_nr_026.tc	reclp_nr_026.ex
reclp_tc.27	This case tests with: THETA = -0.0065, G_ROTATION = 0.00826 RE_CMD should be 6	reclp_nr_027.tc	reclp_nr_027.ex
reclp_tc.28	This case tests with: THETA = 0.0061, G_ROTATION = 0.00826 RE_CMD should be 7	reclp_nr_028.tc	reclp_nr_028.ex
reclp_tc.29	This case tests with: THETA = 0.0061, G_ROTATION = 0.009999 RE_CMD should be 7	reclp_nr_029.tc	reclp_nr_029.ex
reclp_tc.30	This case tests with: THETA = -0.0061, G_ROTATION = 0.009999 RE_CMD should be 6	reclp_nr_030.tc	reclp_nr_030.ex
reclp_tc.31	This case tests with: THETA = -0.0061, G_ROTATION = -0.009999 RE_CMD should be 6	reclp_nr_031.tc	reclp_nr_031.ex
reclp_tc.32	This case tests with: THETA = 0.0065, G_ROTATION = -0.009999 RE_CMD should be 7	reclp_nr_032.tc	reclp_nr_032.ex
reclp_tc.33	This case tests with: THETA = 0.0063, G_ROTATION = -0.00826 RE_CMD should be 7	reclp_nr_033.tc	reclp_nr_033.ex
reclp_tc.34	This case tests with: THETA = -0.0063, G_ROTATION = 0.00826 RE_CMD should be 6	reclp_nr_034.tc	reclp_nr_034.ex
reclp_tc.35	This case tests with: THETA = -0.0063, G_ROTATION = 0.009999 RE_CMD should be 1	reclp_nr_035.tc	reclp_nr_035.ex
reclp_tc.36	This case tests with: THETA = 0.0063, G_ROTATION = -0.009999 RE_CMD should be 1	reclp_nr_036.tc	reclp_nr_036.ex
reclp_tc.37	This case tests with: THETA = -0.006400000000000001, G_ROTATION = 0.009999 RE_CMD should be 6	reclp_nr_037.tc	reclp_nr_037.ex
reclp_tc.38	This case tests with: THETA = 0.006400000000000001, G_ROTATION = -0.009999 RE_CMD should be 5	reclp_nr_038.tc	reclp_nr_038.ex
reclp_tc.39	This case tests with: THETA = 0.006400000000000001, G_ROTATION = -0.0100001 RE_CMD should be 1	reclp_nr_039.tc	reclp_nr_039.ex
reclp_tc.40	This case tests with: THETA = -0.006400000000000001, G_ROTATION = -0.0100001 RE_CMD should be 6	reclp_nr_040.tc	reclp_nr_040.ex

reclp_tc.41	This case tests with: THETA = -0.006400000000000001, G_ROTATION = 0.0100001 RE_CMD should be 1	reclp_nr_041.tc	reclp_nr_041.ex
reclp_tc.42	This case tests with: THETA = 0.006400000000000001, G_ROTATION = 0.0100001 RE_CMD should be 7	reclp_nr_042.tc	reclp_nr_042.ex
reclp_tc.43	This case tests with: THETA = 0.006400000000000001, G_ROTATION = -0.015709 RE_CMD should be 6	reclp_nr_043.tc	reclp_nr_043.ex
reclp_tc.44	This case tests with: THETA = 0.006400000000000001, G_ROTATION = 0.015709 RE_CMD should be 7	reclp_nr_044.tc	reclp_nr_044.ex
reclp_tc.45	This case tests the +P2 boundary with Theta > 0. These numbers are valid but not necessarily realistic for GCS. THETA = 0.038, G_ROTATION = 0.02 == P2 RE_CMD should be 5	reclp_nr_045.tc	reclp_nr_045.ex
reclp_tc.46	This case tests the -P2 boundary with Theta < 0. These numbers are valid but not necessarily realistic for GCS. THETA = -0.038, G_ROTATION = -0.02 == -P2 RE_CMD should be 5	reclp_nr_046.tc	reclp_nr_046.ex
reclp_tc.47	Boundary test with THETA = 0.039, G_ROTATION = 0.01 == P1 RE_CMD should be 3	reclp_nr_047.tc	reclp_nr_047.ex
reclp_tc.48	Boundary test with THETA = -0.039, G_ROTATION = -0.01 == -P1 RE_CMD should be 2	reclp_nr_048.tc	reclp_nr_048.ex
reclp_tc.49	Boundary test with THETA = 0.019, G_ROTATION = 0.01 == P1 RE_CMD should be 1	reclp_nr_049.tc	reclp_nr_049.ex
reclp_tc.50	Boundary test with THETA = -0.019, G_ROTATION = -0.01 == -P1 RE_CMD should be 1	reclp_nr_050.tc	reclp_nr_050.ex
reclp_tc.51	Boundary test for -THETA2 with THETA = -0.04 == -THETA2, G_ROTATION = 0.01 == P1 RE_CMD should be 1	reclp_nr_051.tc	reclp_nr_051.ex
reclp_tc.52	Boundary test with THETA = -0.042, G_ROTATION = 0.02 == P2 RE_CMD should be 1	reclp_nr_052.tc	reclp_nr_052.ex
reclp_tc.53	Boundary test with THETA = -0.04299999999999999, G_ROTATION = 0.03 == P3 RE_CMD should be 1	reclp_nr_053.tc	reclp_nr_053.ex
reclp_tc.54	Boundary test with THETA = -0.044, G_ROTATION = 0.04 == P4 RE_CMD should be 1	reclp_nr_054.tc	reclp_nr_054.ex
reclp_tc.55	Boundary test with THETA = 0.04 == THETA2, G_ROTATION = -0.01== P1 RE_CMD should be 1	reclp_nr_055.tc	reclp_nr_055.ex
reclp_tc.56	Boundary test with THETA = 0.042, G_ROTATION = -0.02 == -P2 RE_CMD should be 1	reclp_nr_056.tc	reclp_nr_056.ex

reclp_tc.57	Boundary test with THETA = 0.0429999999999999, G_ROTATION = -0.03 == -P3 RE_CMD should be 1	reclp_nr_057.tc	reclp_nr_057.ex
reclp_tc.58	Boundary test with THETA = 0.044, G_ROTATION = -0.04 == -P4 RE_CMD should be 1	reclp_nr_058.tc	reclp_nr_058.ex
reclp_tc.59	Boundary test with THETA = -0.004, G_ROTATION = 0.04 == P4 RE_CMD should be 1	reclp_nr_059.tc	reclp_nr_059.ex
reclp_tc.60	This case tests with: THETA = 0.0157079632679441, G_ROTATION = 1.01 RE_CMD should be 7	reclp_ro_060.tc	reclp_ro_060.ex
reclp_tc.61	This case tests with: THETA = 0.0157079632679441, G_ROTATION = -1.01 RE_CMD should be 7	reclp_ro_061.tc	reclp_ro_061.ex
reclp_tc.62	This case tests with: THETA = 3.1476718651402, G_ROTATION = 0.5 RE_CMD should be 7	reclp_ro_062.tc	reclp_ro_062.ex
reclp_tc.63	This case tests with: THETA = -3.1476718651402, G_ROTATION = 0.5 RE_CMD should be 7	reclp_ro_063.tc	reclp_ro_063.ex
reclp_tc.64	This case tests with: THETA = -0.05, G_ROTATION = 0.5 RE_CMD should be 7	reclp_nr_064.tc	reclp_nr_064.ex
reclp_tc.65	Test origin: THETA = 0., G_ROTATION = 0 RE_CMD should be 1	reclp_nr_065.tc	reclp_nr_065.ex
reclp_tc.66	Test THETA at -Pi: THETA = -3.1476718651402, G_ROTATION = 0 RE_CMD should be 6	reclp_nr_066.tc	reclp_nr_066.ex
reclp_tc.67	Test THETA at Pi: THETA = 3.1476718651402, G_ROTATION = 0 RE_CMD should be 7	reclp_nr_067.tc	reclp_nr_067.ex
reclp_tc.68	This case tests with: THETA = 0.05, G_ROTATION = -0.5 RE_CMD should be 6	reclp_nr_068.tc	reclp_nr_068.ex

A.3.10 CRCP Functional Unit Test Cases

Table A.14 gives a listing of all requirements-based test cases for the CRCP functional unit. Since only two variables are involved in the testing, their values are also given for each test case. All test cases manipulate the variables:

AE_TEMP
CHUTE_RELEASED

Table A.14: Test cases for CRCP functional unit.

Test Case Data File	Description	Test-Input File	Expected-Results File
crcp_tc.1	Test initial frame with: AE_TEMP = 0, CHUTE_RELEASE = 0	crcp_nr_001.tc	crcp_nr_001.ex
crcp_tc.2	AE_TEMP = 0, CHUTE_RELEASE = 1 This is a valid, but unlikely case.	crcp_nr_002.tc	crcp_nr_002.ex
crcp_tc.3	Frame 251: AE_TEMP = 1, CHUTE_RELEASE = 0	crcp_nr_003.tc	crcp_nr_003.ex
crcp_tc.4	Frame 251: AE_TEMP = 1, CHUTE_RELEASE = 1 This is a valid, but unlikely case.	crcp_nr_004.tc	crcp_nr_004.ex
crcp_tc.5	Frame 252: AE_TEMP = 2, CHUTE_RELEASE = 0	crcp_nr_005.tc	crcp_nr_005.ex
crcp_tc.6	Frame 252: AE_TEMP = 2, CHUTE_RELEASE = 1	crcp_nr_006.tc	crcp_nr_006.ex
crcp_tc.7	Frame 252: AE_TEMP = 0, CHUTE_RELEASE = -1 This is a valid, but unlikely case.	crcp_ro_007.tc	crcp_ro_007.ex
crcp_tc.8	Frame 252: AE_TEMP = 0, CHUTE_RELEASE = 2 This is a valid, but unlikely case.	crcp_ro_008.tc	crcp_ro_008.ex
crcp_tc.9	AE_TEMP = 3, CHUTE_RELEASE = 0 This is a valid, but unlikely case.	crcp_ro_009.tc	crcp_ro_009.ex
crcp_tc.10	AE_TEMP = -1, CHUTE_RELEASE = 0 This is a valid, but unlikely case.	crcp_ro_010.tc	crcp_ro_010.ex

A.3.11 CP Functional Unit Test Cases

CP requirements-based functional unit test cases are given in Table A.15. All test cases manipulate the variables:

FRAME_COUNTER
SUBFRAME_COUNTER

Even though the GCS Specification lists many more variables as inputs for CP, the specific value of the variables do not effect the operation of CP. The CP functional unit only copies these values to the PACKET array. The variables are not used for decision in CP. Therefore, it is unnecessary to test specific values of those variables. The only two variables that influence CP operation are the ones listed above.

Table A.15: Test cases for CP functional unit.

Test Case Data File	Description	Test-Input File	Expected-Results File
cp_nr_001.m	Test Packet and CRC generation for subframe 1 variables	cp_nr_001.tc	cp_nr_001.ex
cp_nr_002.m	Test Packet and CRC generation for subframe 2 variables	cp_nr_002.tc	cp_nr_002.ex
cp_nr_003.m	Test Packet and CRC generation for subframe 3 variables	cp_nr_003.tc	cp_nr_003.ex
cp_nr_004.m	Test Packet and CRC generation for subframe 1 variables when frame number is greater than 1	cp_nr_004.tc	cp_nr_004.ex
cp_nr_005.m	Test Packet and CRC generation for subframe 1 variables when sequence number is greater than 255	cp_nr_005.tc	cp_nr_005.ex

A.3.12 SP Subframe Test Cases

All four of the requirements of SP subframe as listed in the traceability matrix, Table A.10-1, are tested by test case SP_001. It tests to see if the TSP calculations are performed before other functional units, verifies that all other functional units execute including CP. The data file sp_001.m is used to generate the test-input file sp_001.tc and the expected results file sp_001.ex.

A.3.13 GP Subframe Test Cases

Table A.16 gives the test cases for the GP subframe. Since the GP subframe has only the GP functional unit, tests of this subframe will be similar to test of the GP functional unit. The difference is that subframe test also include calling CP to create the communications packet for the GP subframe.

Table A.16: Test cases for GP Subframe.

Test Case Data File	Description	Test-Input File	Expected-Results File
gpsf_tc.1	Initial frame, tests all valid inputs	gpsf_001.tc	gpsf_001.ex
gpsf_tc.2	Transition frame 246.	gpsf_002.tc	gpsf_002.ex
gpsf_tc.3	FRAME = 251; CHUTE_RELEASED set to 1 this frame. All valid inputs tested.	gpsf_003.tc	gpsf_003.es
gpsf_tc.4	FRAME = 252; CHUTE_RELEASED = 1 & GP_PHASE goes to 3	gpsf_004.tc	gpsf_004.ex
gpsf_tc.5	FRAME = 950 CONTOUR_CROSSED will be set to 1 by the end of the frame. All valid data tested.	gpsf_005.tc	gpsf_005.ex
gpsf_tc.6	FRAME = 951 with CONTOUR_CROSSED = 1	gpsf_006.tc	gpsf_006.ex
gpsf_tc.7	FRAME = 2073 where CL = 2. All valid data tested.	gpsf_007.tc	gpsf_007.ex
gpsf_tc.8	FRAME = 2078 with CL = 2, and GP_PHASE changes to 4. All valid data tested.	gpsf_008.tc	gpsf_008.ex

A.3.14 CLP Subframe Test Cases

CLP subframe test cases are given in Table A.17. Since the AECLP functional unit must be executed first in this subframe, CLP subframe test cases data is depends heavily on the AECLP inputs. As can be seen from the traceability matrix (Table A.10-1), each CLP test case will test all four of the CLP subframe requirements.

Table A.17: Test cases for CLP Subframe.

Test Case Data File	Description	Test-Input File	Expected-Results File
clp_tc.1	Test initial frame using data from aeclp_tc.1	clp_001.tc	clp_001.ex
clp_tc.2	Test frame 2 using data from aeclp_tc.2	clp_002.tc	clp_002.ex
clp_tc.3	Test frame 251 using data from aeclp_tc.3	clp_003.tc	clp_003.es
clp_tc.4	Test frame 252 using data from aeclp_tc.4	clp_004.tc	clp_004.ex
clp_tc.5	Test frame 950 using data from aeclp_tc.5	clp_005.tc	clp_005.ex
clp_tc.6	Test frame 951 using data from aeclp_tc.6	clp_006.tc	clp_006.ex
clp_tc.7	Test frame 2077 using data from aeclp_tc.7	clp_007.tc	clp_007.ex
clp_tc.8	Test frame 2078 using data from aeclp_tc.8	clp_008.tc	clp_008.ex
clp_tc.9	Test frame 2083 using data from aeclp_tc.9	clp_009.tc	clp_009.es
clp_tc.10	Test frame 250 using data from aeclp_tc.10	clp_010.tc	clp_010.ex
clp_tc.11	Test frame 949 using data from aeclp_tc.11	clp_011.tc	clp_011.ex
clp_tc.12	Test frame 955 using data from aeclp_tc.12	clp_012.tc	clp_012.ex
clp_tc.13	Test using aeclp_tc.54 data where AE_SWITCH is still off at end of frame, giving AE_CMD = 0 FRAME_ENGINES_IGNITED > 1	clp_013.tc	clp_013.ex
clp_tc.14	Test using aeclp_tc.55 data where INTERNAL_CMD > 1.0	clp_014.tc	clp_014.ex

A.3.15 Frame Test Cases

Frame test cases are given in Table A.18. They exercise all functional units for frames with significant transitions during the terminal descent. These transition include changes in GP_PHASE or other trajectory status variables and are given in the Table A.10-1.

Table A.18: Frame test cases.

Test Case Data File	Description	Test-Input File	Expected-Results File
frame_tc.1	Test initial frame with frame counter set to 1. All valid data used.	frame_001.tc	frame_001.ex
frame_tc.2	Test frame 246 where GP_PHASE = 2. This is the frame that occurs just before AE_TEMP transitions from 1 to 2 and CHUTE_RELEASED transitions from 0 to 1.	frame_002.tc	frame_002.ex
frame_tc.3	Test frame 251 where GP_PHASE = 2. This is the frame that occurs just before AE_TEMP transitions from 1 to 2 and CHUTE_RELEASED transitions from 0 to 1.	frame_003.tc	frame_003.es
frame_tc.4	Test frame 252 where GP_PHASE transitions from 2 to 3.	frame_004.tc	frame_004.ex
frame_tc.5	Test frame where CONTOUR_CROSSED transitions from 0 to 1.	frame_005.tc	frame_005.ex
frame_tc.6	Test the frame just after CONTOUR_CROSSED transitions to 1. This case added for completeness.	frame_006.tc	frame_006.ex
frame_tc.7	Test the frame when CL = 2.	frame_007.tc	frame_007.ex
frame_tc.8	Test frame when GP_PHASE transitions from 3 to 4.	frame_008.tc	frame_008.ex
frame_tc.9	Test frame when GP_PHASE starts as 5; no execution should occur.	frame_009.tc	frame_009.ex

A.3.16 Trajectory Test Cases

The ultimate goal of each GCS implementation is to land the spacecraft safely given some initial set of parameters. These parameters reflect environmental conditions, the spacecraft, and the flight conditions at the beginning of the terminal descent. In full trajectory testing, each implementation's code is linked and run in the simulator's environment. Unlike previous tests which exercise the implementation as a stand-alone process, trajectory testing requires the implementation to run as a subprocess of the simulator program. This is part of the high level requirements. Additionally, the GCS Specification requires the implementation to be able to execute multiple consecutive frames until the termination condition is reached. Since a landing is not specifically stated as a high level requirement of the GCS software, trajectory testing will encompass both successful landing cases and expected crash cases and will cover the part of the simulator's input space that directly effects the implementation. Keep in mind that the objective of trajectory testing is to verify each implementation's ability to run consecutive and multiple frames. Whether the final result is a landing or a crash is inconsequential.

It is assumed for testing purposes that the GCS Simulator provides a stable model of the flight and atmospheric dynamics when given a set of initial conditions. This is significant because test case inputs for trajectory testing are parameters for the simulator, not the implementation. There are nominally four sets of input parameters for the simulator. They are physical parameters of the Viking Lander, aerodynamic response of the Lander, the atmospheric conditions during descent, and the terminal descent conditions of the vehicle. Of these four sets, the atmospheric and initial entry conditions have been identified to most directly effect the implementations and hence will be considered as the input space for the implementation running under the simulator. The physical parameters for the Lander will not be considered because modifying these parameters could constitute testing various configurations of the vehicle and are beyond the scope of testing GCS implementations. The aerodynamic responses of the vehicle are also not considered to be part of the input space because they are used by the simulator. Section 2.1.2.2 of the GCS_SIM User's Guide (ref. A.5) even gives staunch warning about modifications to this data set.

The specific parameters to be considered for trajectory tests are given below for the two categories.

Atmospheric Conditions parameters:

- Initial Wind Velocity
- wind_gradiant
- Initial Temperature
- temperature_gradiant

Terminal Descent parameters:

- Initial Altitude
- Rotation Rates(x,y,z)
- Velocity(x,y,z)
- Rotational Angle around (-y,-z, x)

All parameters are in the USAGE_DISTRIBUTION.DAT input file for the GCS simulator except for wind_gradiant, and temperature_gradiant which are found in the INITIAL_CONSTANTS.DAT file. Hence, trajectory test cases inputs will consist of versions of these two files with carefully selected values for the above variables. Special instructions for modifying values in the USAGE_DISTRIBUTIONS.DAT and INITIAL_CONSTANTS.DAT files are given in Section 2.1.2.1 of the GCS_SIM User's Guide.

The GCS simulator is capable of selecting its own initial conditions based on the values in the USAGE_DISTRIBUTION.DAT file if those values are given in the form of a distribution. It does so based on a seed for a random number generator which it also selects if one is not specified. For trajectory tests, a seed will be specified although it is understood that the seed will not effect the specific values being tested because the values in the USAGE_DISTRIBUTION.DAT file will be specified in a manner that forces those specific values to be the specified ones. The seed is used to select values for other variables not being tested. To be consistent, the same seed will be used for all trajectory test cases. This seed will be 114291523 and it is set in the RUN_TRAJ.COM file.

Specific values used for atmospheric test cases are given in Table A.19 along with the test case names. The limiting and optimal values for the parameters are derived from the GCS subsystem description in the Viking '75 (ref. A.6) and from the GCS Specification. The optimal wind velocity is given as 51m/s while the maximum is given as 90m/s; the minimum is obviously 0m/s. The simulator allows wind gradient to vary from -1.10×10^{-2} to 1.10×10^{-2}. The units are derived based on analysis of the GCS simulator. The limiting values of -200 to 25 degrees are based on the GCS Specification for the range of the ATMOSPHERIC_TEMP variable. A linear temperature gradient is calculated based on a 1.5 km drop mentioned in the GCS Specification. Note in the table below that nominal(N) values are used for all elements other than the variable of specific interest for a test case. The nominal value is the value picked by the simulator software given the above seed value. This is a consistent value for all test cases because all test cases will be run using the same seed. A sample of the nominal value range is in the GCS_SIM User's Guide.

Table A.19: Atmospheric Test Cases

Initial Wind Velocity	**Wind_Gradient (/sec)**	**Initial Temp**	**Temp gradient (degree/km)**	**Test Case Number**
90	N	N	N	001
0	N	N	N	002
51	N	N	N	003
N	$-1.10\text{x}10^{-2}$	N	N	004
N	0	N	N	005
N	$1.10\text{x}10^{-2}$	N	N	006
N	N	-200	N	007
N	N	0	N	008
N	N	25	N	009
N	N	N	150	010
N	N	N	0	011
N	N	N	-150	012

Specific values for terminal descent condition test cases are given in Table A.4. Limiting values for initial altitude are 2000 meters (maximum value for altitude variables given in the Specification) and 1400 meters (optimal altitude given in (ref. A.6)). A value of 0 is a legal value for altitude but would not be applicable for an initial altitude. Rotation rates of -1.0 rad/sec to 1.0 rad/sec are permitted by the simulator software. No information is available on the velocity range. Hence the range given by the usage distribution in the GCS_SIM User's Guide (p.12) was used. It allows the X and Y velocity to vary between ±20 m/s and the Z velocity to vary between 0 and 200 m/s. The Rotational Angles are used by the simulator to calculate the initial attitude cosines. No limits for the rotation angles were found while reviewing the simulator code for calculating the attitude cosines. However, tests of the simulator software show that the -Y and -Z rotation angles can vary between ±0.83 rads and the X rotation angle can vary from 0 to 2π.

Table A.20: Terminal Descent test cases

Initial Altitude	**Rotation Rates**			**Velocity**			**Vehicle Orientation**			**Test Case Number**
(m)	x (rad/s)	y (rad/s)	z (rad/s)	x (m/s)	y (m/s)	z (m/s)	x (rad)	y (rad)	z (rad)	
2000	N	N	N	N	N	N	N	N	N	013
1400	N	N	N	N	N	N	N	N	N	014
700	N	N	N	N	N	N	N	N	N	015
N	N	N	N	-20	N	N	N	N	N	016
N	N	N	N	20	N	N	N	N	N	017
N	N	N	N	N	-20	N	N	N	N	018
N	N	N	N	N	20	N	N	N	N	019
N	N	N	N	N	N	0	N	N	N	020
N	N	N	N	N	N	200	N	N	N	021
N	N	N	N	0	0	N	N	N	N	022
N	N	N	N	N	N	N	0	N	N	023
N	N	N	N	N	N	N	6.28	N	N	024
N	N	N	N	N	N	N	N	-0.83	N	025
N	N	N	N	N	N	N	N	0.83	N	026
N	N	N	N	N	N	N	N	N	-0.83	027
N	N	N	N	N	N	N	N	N	0.83	028
N	1	N	N	N	N	N	N	N	N	029
N	-1	N	N	N	N	N	N	N	N	030
N	N	1	N	N	N	N	N	N	N	031
N	N	-1	N	N	N	N	N	N	N	032
N	N	N	1	N	N	N	N	N	N	033
N	N	N	-1	N	N	N	N	N	N	034

The nominal values selected by using the standard seed for the above test cases are as follows:

Initial altitude:	1498.24 meters
Rotation Rates: (rad/sec)	
about x	-6.14×10^{-2}
about y	-8.80×10^{-2}
about z	-9.92×10^{-2}
Velocity (meters/sec)	
x	-1.58
y	20
z	57.03
Initial wind velocity	24.71 m/sec
Initial temperature	-140.56º C
Orientation Angles (radian)	
about x	0.20
about -y	-0.17
about -z	-1.17

Tables A.19 and A.20 give specific input values for all trajectory test cases. To be consistent with other sections above, Table A.21 is included to summarize all trajectory test cases for use with test case generation and execution procedures.

Table A.21: Trajectory test case summary.

Test Case Data File	Description	Test-Input File	Expected-Results File
traj_atm_ic_001.tc traj_atm_ud_001.tc	Test high wind velocity	traj_atm_ic_001.tc traj_atm_ud_001.tc	traj_atm_001.seed
traj_atm_ic_002.tc traj_atm_ud_002.tc	Test no wind	traj_atm_ic_002.tc traj_atm_ud_002.tc	traj_atm_002.seed
traj_atm_ic_003.tc traj_atm_ud_003.tc	Test optimal velocity	traj_atm_ic_003.tc traj_atm_ud_003.tc	traj_atm_003.seed
traj_atm_ic_004.tc traj_atm_ud_004.tc	Test low wind gradient	traj_atm_ic_004.tc traj_atm_ud_004.tc	traj_atm_004.seed
traj_atm_ic_005.tc traj_atm_ud_005.tc	Test 0 wind gradient	traj_atm_ic_005.tc traj_atm_ud_005.tc	traj_atm_005.seed
traj_atm_ic_006.tc traj_atm_ud_006.tc	Test high wind gradient	traj_atm_ic_006.tc traj_atm_ud_006.tc	traj_atm_006.seed
traj_atm_ic_007.tc traj_atm_ud_007.tc	Test low initial temp.	traj_atm_ic_007.tc traj_atm_ud_007.tc	traj_atm_007.seed
traj_atm_ic_008.tc traj_atm_ud_008.tc	Test 0 initial temp.	traj_atm_ic_008.tc traj_atm_ud_008.tc	traj_atm_008.seed
traj_atm_ic_009.tc traj_atm_ud_009.tc	Test high initial temp.	traj_atm_ic_009.tc traj_atm_ud_009.tc	traj_atm_009.seed
traj_atm_ic_010.tc traj_atm_ud_010.tc	Test high temp. gradient	traj_atm_ic_010.tc traj_atm_ud_010.tc	traj_atm_010.seed
traj_atm_ic_011.tc traj_atm_ud_011.tc	Test 0 temp. gradient	traj_atm_ic_011.tc traj_atm_ud_011.tc	traj_atm_011.seed
traj_atm_ic_012.tc traj_atm_ud_012.tc	Test low temp gradient	traj_atm_ic_012.tc traj_atm_ud_012.tc	traj_atm_012.seed
traj_td_ic_013.tc traj_td_ud_013.tc	Test highest initial altitude	traj_td_ic_013.tc traj_td_ud_013.tc	traj_td_013.seed
traj_td_ic_014.tc traj_td_ud_014.tc	Test optimal altitude	traj_td_ic_014.tc traj_td_ud_014.tc	traj_td_014.seed
traj_td_ic_015.tc traj_td_ud_015.tc	Test lowest altitude	traj_td_ic_015.tc traj_td_ud_015.tc	traj_td_015.seed
traj_td_ic_016.tc traj_td_ud_016.tc	Test X velocity min. value	traj_td_ic_016.tc traj_td_ud_016.tc	traj_td_016.seed
traj_td_ic_017.tc traj_td_ud_017.tc	Test X velocity max. value.	traj_td_ic_017.tc traj_td_ud_017.tc	traj_td_017.seed
traj_td_ic_018.tc traj_td_ud_018.tc	Test Y velocity min. value	traj_td_ic_018.tc traj_td_ud_018.tc	traj_td_018.seed
traj_td_ic_019.tc traj_td_ud_019.tc	Test Y velocity max. value.	traj_td_ic_019.tc traj_td_ud_019.tc	traj_td_019.seed
traj_td_ic_020.tc traj_td_ud_020.tc	Test Z velocity min. value	traj_td_ic_020.tc traj_td_ud_020.tc	traj_td_020.seed
traj_td_ic_021.tc traj_td_ud_021.tc	Test Z velocity max. value.	traj_td_ic_021.tc traj_td_ud_021.tc	traj_td_021.seed

traj_td_ic_022.tc traj_td_ud_022.tc	Test no X & Y velocity	traj_td_ic_022.tc traj_td_ud_022.tc	traj_td_022.seed
traj_td_ic_023.tc traj_td_ud_023.tc	Test min. X entry angle	traj_td_ic_023.tc traj_td_ud_023.tc	traj_td_023.seed
traj_td_ic_024.tc traj_td_ud_024.tc	Test max. X entry angle	traj_td_ic_024.tc traj_td_ud_024.tc	traj_td_024.seed
traj_td_ic_025.tc traj_td_ud_025.tc	Test min. Y entry angle	traj_td_ic_025.tc traj_td_ud_025.tc	traj_td_025.seed
traj_td_ic_026.tc traj_td_ud_026.tc	Test max. Y entry angle	traj_td_ic_026.tc traj_td_ud_026.tc	traj_td_026.seed
traj_td_ic_027.tc traj_td_ud_027.tc	Test min. Z entry angle	traj_td_ic_027.tc traj_td_ud_027.tc	traj_td_027.seed
traj_td_ic_028.tc traj_td_ud_028.tc	Test max. Z entry angle	traj_td_ic_028.tc traj_td_ud_028.tc	traj_td_028.seed
traj_td_ic_029.tc traj_td_ud_029.tc	Test positive X rotation rate	traj_td_ic_029.tc traj_td_ud_029.tc	traj_td_029.seed
traj_td_ic_030.tc traj_td_ud_030.tc	Test negative X rotation rate	traj_td_ic_030.tc traj_td_ud_030.tc	traj_td_030.seed
traj_td_ic_031.tc traj_td_ud_031.tc	Test positive Y rotation rate	traj_td_ic_031.tc traj_td_ud_031.tc	traj_td_031.seed
traj_td_ic_032.tc traj_td_ud_032.tc	Test negative Y rotation rate	traj_td_ic_032.tc traj_td_ud_032.tc	traj_td_032.seed
traj_td_ic_033.tc traj_td_ud_033.tc	Test positive Z rotation rate	traj_td_ic_033.tc traj_td_ud_033.tc	traj_td_033.seed
traj_td_ic_034.tc traj_td_ud_034.tc	Test negative Z rotation rate	traj_td_ic_034.tc traj_td_ud_034.tc	traj_td_034.seed

A.3.17 Pass/Fail Criteria

This section focuses on the strategy used to determine pass or failure of a test case. Two techniques are used to determine whether a test case passes or fails. The first is applicable to functional unit, subframe, frame, and structural test cases. Trajectory cases use a different technique because those cases are run with the GCS simulator.

The GCS Specification requires all data flow, into and out of the software, to go through the four data stores. Hence, results of any functional unit can be checked by examining the data in the stores after the functional unit has executed. To determine whether a test case passes or fails, test drivers compare the values of the data store with values from the expected-results file. If all variables match their expected results, the test case passes. To further simplify the matching process, the expected-results file uses the same NAMELIST format as the data stores.

The criteria for what constitutes a correct match varies for different variables. For variables defined as INTEGERS and LOGICALS an exact match between the expected and actual results is required. For real variables, the value generated during the test run must match to within a tolerance of the expected value. The tolerances are determined based on empirical experience with the GCS simulator and vary for different variables. The tolerance calculation is different for the two sets of variables as given in Table A.22 and A.23. For variables listed in Table A.22, the tolerance is either the absolute error (ε) or the relative error (δ) depending on whether the values is less than the threshold (β), which is also empirically determined. For variables listed in Table A.23, the tolerance is the absolute error (ε).

Table A.22: Accuracy tolerances for variables in set 1.

Data Element Name	β	ε	δ
A_ACCELERATION(1,0)	1.0	.001	5.0D-9
AR_ALTITUDE	1.0	.001	5.0D-10
ATMOSPHERIC_TEMP	1.0	.001	5.0D-10
GP_ALTITUDE	5.0	.01	5.0D-5
GP_VELOCITY	1.0	.001	5.0D-6
TDLR_VELOCITY	1.0	.001	5.0D-10
TE_LIMIT	1.0	.001	5.0D-2

Table A.23: Accuracy tolerances for variables in set 2.

Data Element Name	ε
A_ACCELERATION(2,0)	.001
A_ACCELERATION(3,0)	.001
G_ROTATION	.001
GP_ATTITUDE	.001
GP_ROTATION	.001
INTERNAL_CMD	.001
PE_INTEGRAL	.001
TE_INTEGRAL	.001
THETA	.01
VELOCITY_ERROR	.001
YE_INTEGRAL	.001

It should be noted that the GCS simulator performs accuracy checks on only the current values of the variables. For testing purposes, all history values of all variables must be checked to ensure that data store integrity is maintained. To reduce the complexity of the comparison, test drivers are set up to generate an output file if a mismatch is found for any variable. An absolute error more than 1.0 x 10^{-8} is considered a mismatch for testing purposes

When a trajectory is run, one of the output files gives the starting seed, ending seed, whether the space craft landed, the number of frames executed, and the final value of GP_PHASE. This information is sufficient to determine whether the trajectory is run to completion and whether a landing or a crash has occurred. More importantly, the number of frames and the final value of GP_PHASE can be used to determine whether the high level requirements have been met, since the highest level requirement is for each implementation to run consecutive and multiple frames until the GP_PHASE is 5. If the ".SEED" output file for any trajectory test cases shows that multiple frames were executed and that the final value of GP_PHASE is 5, then the implementation can be considered to have met the high-level requirements. In addition to visual survey of the ".SEED" files, the trajectory test case execution procedure includes an additional step to compare the output seed files with those from the VENUS prototype. This is an additional check to match the implementation to the VENUS prototype.

A.4 Test Case Execution Procedures

Once test cases and drivers have been developed and submitted to configuration management, the verification analysts are ready to initiate testing of the GCS implementations. Given that the GCS development activities follow the water-fall model, all the code is available when testing commences. This is consistent with the requirements in DO-178B. Further, DO-178B requires GCS testing to be conducted in the following order:

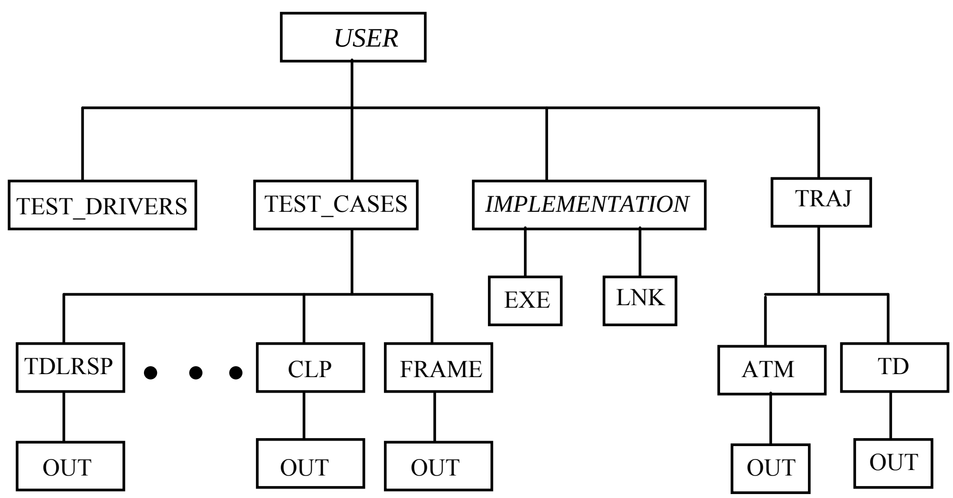
1) requirements-based functional unit testing.
2) requirements-based subframe testing
3) requirements-based frame testing
4) requirements-based trajectory testing

5) structural analysis and testing of functional units

The general procedure for any of the above categories of testing is to first request the configuration items necessary for the test, place them in the appropriate directories, build the test case with the test subjects, run the command file that executes the test suite, check for any analysis files, determine if the items in the analysis file warrant problem reporting. If code modifications are made, then all test cases for the changed code are re-executed. The sections below will describe the directory structure that must be created to execute the test cases. The specific configuration items and execution procedures necessary for the test case are also be described.

A.4.1 Environment and Directory Structure for Test Case Execution

GCS implementations are written in VAX FORTRAN and are intended for execution on DEC machines. To perform functional unit testing of GCS implementations, it is necessary to have a directory structure that matches the DCL commands in the test support files. Otherwise, those path names need to be edited to reflect the directory structure of the specific user. The figure below illustrates the directory structure that a Verification Analyst must have for testing to avoid excessive editing to support files.



Note that the top level directory, "USER", is the user's home directory. The TEST_DRIVER directory is used for storing the test drivers and support files. The TEST_CASES directory has a series of subdirectories. There should be a subdirectory for each functional unit although not all are shown. There should also be a subdirectory for each subframe and one for frame test cases as shown. The IMPLEMENTATION directory is for storing the code to be tested. The name should be changed to the name of the appropriate implementation. The test case execution procedures will reference these directories for storing items Fetched from CMS. Again, it is important that the naming of the directories be adhered to as the DCL command files will reference those specific names.

For trajectory testing, a separate [TRAJ] directory is needed for trajectory tests with two sub directories. The [TRAJ] directory will hold the simulator, the implementation to be tested, and the data and support files required for trajectory testing. The [ATM] subdirectory will hold the tests that vary the atmospheric conditions; the [TD] sub directory will hold the test cases that vary the initial terminal descent conditions. Each sub directory will also contain an [OUT] directory for simulator output. The [TRAJ] directory will contain the ".COM" files and executables for the implementation and the simulator.

A.4.2 Functional Unit Test Case Execution Procedure

The following describes specific steps that must be followed for executing functional unit test cases. Because this procedure is written for both the Pluto and Mercury implementations, some file renaming will be necessary to account for the way files are stored in CMS. Additionally, directory references must be changed to account for the tester's top level directory name.

1) Fetch all the code belonging to an implementation and place the code files in the [*implementation*] directory:

   Because specific file names will vary between implementations, all source files related to an implementation should be fetched. Extra files are of no consequence since the link command files will not use them. The link command files(obtained in the next step) will link only the necessary files for a functional unit test concerned and disregard the extra files.

2) Fetch the following data, support and utility files from the CMS library. Place them in their respective directories. Refer to Table A.1 for specific names.

Data files [TEST_CASES.functional-unit]	Support Files (directory below)	Utility Files [TEST_DRIVERS]
*functional-unit*_NR_*xx*.TC, .EX *functional-unit*_RO_*xx*.TC, .EX	*i*_LNK*functional-unit*.COM -> [IMPLEMENTATION.LNK] *i*_TEST_*functional-unit*.FOR -> [TEST_DRIVERS]	COMPARE_EXTERNAL.FOR COMPARE_GUIDANCE.FOR COMPARE_RUNPRAM.FOR READ_TC.FOR READ_EX.FOR STRUCT.FOR_INC COMMONS.FOR_INC *i*_TC_DRIVER.COM

   Note that the "*i*_" represents the initial of the implementation. That is "*M*_" for Mercury and "*P*_" for Pluto; the *xxx* represents the three digits identifying the test case; and *functional-unit* is replaced by the name of the functional unit (e.g. ASP, GP). Two files must also be renamed so that the implementation initial is removed. That is:

*i*_TC_DRIVER.COM	renames to	TC_DRIVER.COM
*i*_TEST_ *functional-unit*.FOR	renames to	TEST_*functional-unit*.FOR

3) The files in step 1, the implementation source code to be tested, should be compiled using the VAX FORTRAN compiler. No special compile switches are necessary. (e.g. FOR ASP.FOR) All object files should be placed in the same directory.

4) All files with ".FOR" extension fetched in step 2 should be compiled and object files placed in the same directory. Again no switches should be used.

5) The i_LNK*functional-unit*.COM DCL command file fetched in step 2 above will link all the object files for a functional unit. The resulting executable will be placed in the [EXE] directory. Before using this link file, the file should be checked to ensure that the correct directory reference are used The following command should be initiated from the [LNK] directory:

   @*i*_LNK*functional-unit*.COM

6) The test cases can actually be executed in this step by the entering the command given below. The command should be issued from the [TEST_DRIVERS] directory. The command should be repeated for the number of test cases in the test suite for the functional unit.

   @TC_DRIVER *functional-unit tt xxx*

   where: *functional-unit* is replaced by the name of the functional unit
   tt is replaced by the test type (NR or RO)
   xxx is replaced by the test case number

7) Once execution completes, the tester should look in the [USER.TEST_CASE.*functional-unit*.OUT] directory to see if any analysis files have been generated. If there are any, the tester should review them to see if a PR or SDCR should be initiated.

8) The tester should maintain a record of test cases executed for each test subject. An example of the test log to be used is in section A.10. A test log should be completed for tests on each functional unit. All test cases executed for a functional unit, structural or requirements-based, can be recorded on the same log. This is because any errors requiring code modification will require re-execution of all test cases for that functional unit. Listing all test cases in the same log will reduce the burden of identifying which test cases were re-executed The logs should be maintained for each implementation as the test history will vary depending on the errors discovered in the specific implementation.

A.4.3 Subframe and Frame Test Case Execution Procedure

After all functional units have been tested, the Verification Analyst can begin Integration Testing. This section describes the procedure for executing subframe and frame test cases on the VAX. According to the GCS Specification, each subframe must issue a call to the subroutine Sim_Rendezvous, This will not be done in the test driver because Sim-Rendezvous is not in the scope of the GCS implementation. The order of operations for the subframe test stub is as follows:

Load in the test data

Execute all functional units for the subframe

Generate the expected value for the PACKET data element based on current test case values

Compare all output with the expected results

In order to run the Subframe tests new drivers and command files are created. The CLP_DRIVER.COM runs the Control Law Processing Subframe and SP_DRIVER.COM runs the Sensor Processing Subframe. The test execution procedure for frame and subframe cases is similar to the procedure described in the functional unit. The difference is in the specific files that must be Fetched. Table A.1 should be referenced for specific file names for the subframe or frame test cases. Therefore, steps similar to the previous procedure will be condensed in the description below.

1) Fetch the implementation's source code and place them in the [IMPLEMENTATION] directory.
2) Fetch the necessary test cases and place them in the appropriate subframe directories under the [TEST_CASES] directory. Place frame test cases in the [FRAME] directory.
3) Fetch the support files as listed in Table A.11-1 for the desired subframe. Note that these files should be renamed to remove the implementation's prefix.
3) Compile and link all FORTRAN files as before
4) The newly renamed LNK*subframe*.COM file should be used to build the executable for the test case. This step is identical to that for functional unit execution except the command is:

   @LNK*subframe*
5) The test cases can then be executed using the following command syntax

   @*subframe*_DRIVER *xx*

   for example:

   @CLP_DRIVER 001

   for frame test cases:

   @FRAME_DRIVER *xx*

   will run the CLP driver program using the CLP_001.TC test-input and the expected-results file: CLP_001.EX. The output (if any) will be in the analysis file. Note that if no analysis file is generated, then no error was found while executing the test suite.
6) Again, the tester should check the OUT directory under the specific test case directory for any analysis files and determine if a PR or SDCR is necessary.
7) Again, a test log should be completed with an entry for each test case run. The test log should show the disposition of each test case if errors are found.

A.4.4 Trajectory Test Case Execution Procedure

The procedure for trajectory testing will differ from previous procedures because tests must be run with the simulator. Trajectory test procedure is divided into two parts. The first part builds an executable to run with the simulator. The second part actually runs the test cases.

Procedure for linking the implementation to the simulator

1) Fetch all files related to a specific implementation and place them in the [IMPLEMENTATION] directory along with the command file to build the implementation:

    *i*_BUILD.COM (Note that the *i*_ should be the initial of the implementation.)

2) Also fetch the following simulator utility files and place them in the [IMPLEMENTATION] directory:

    GCS_SIM_RENDEZVOUS.OBJ
    GCS_SETUP.OBJ
    GCS_WHO_AM_I.OBJ
    PAGE_ALIGN.OPT

3) Build the implementation executable with the following command:

    @*i*_BUILD

4) The implementation executable should be in the [.EXE] directory upon completion. The executable should be copied into the [TRAJ] directory for trajectory tests.

Procedure for running trajectory test cases

1) Fetch the trajectory data, support and utility files from CMS and place them in the respective directories.

    (Files also listed in Table A.11-1)

Data files (given below)	Support Files [TRAJ]	Utility Files [TRAJ]
[TRAJ.ATM]	i_TRAJ.COM	ACCURACY.DAT
TRAJ_ATM_IC_*xx*.TC	i_RUN_TRAJ.COM	ALTERNATE_ACCURAC Y.DAT
TRAJ_ATM_IC_*xx*.TC		GCS_LIST.DAT
TRAJ_ATM_*xx*.SEED		GCS_SIM_SWITCHES.DA T
		LIMITS.DAT
[TRAJ.TD]		TABULAR_DATA.DAT
TRAJ_TD_IC_*xx*.TC		TRAJ_SIM.EXE
TRAJ_TD_IC_*xx*.TC		
TRAJ_TD_*xx*.SEED		

2) Edit the GCS_LIST.DAT file by replacing the second line in the file with the name of the implementation's executable to be tested.

4) Execute test cases from the [TRAJ] directory with the command:

   @RUN_TRAJ

   The output files for trajectory test cases will be placed in the respective [OUT] directory.

5) After executing all test cases, the VMS DIFFERENCE command should be used to compare the seed files in the respective [OUT] directories with those fetched from CMS.

A.4.5 Structural Test Case Execution Procedure

Structural test case execution procedure for both implementations are identical to functional unit test case execution. The file naming pattern is slightly different for each implementation. The specific names are given in Table A.2. The general procedure is as follows:

1) If the executable for a functional unit has not been built at this point, then follow the procedure in functional unit test execution procedure steps 1 to 4 to build an executable.

2) Fetch the structural test case from CMS. Refer to Table A.2 for specific names. Note that structural test case files should be placed in the same directory as those for functional unit tests.

   The Pluto implementation structural test cases have the following naming pattern:

   *functional-unit*_PST_*xxx*.TC   for test case input files
   *functional-unit*_PST_*xxx*.EX   for expected results files

   The naming pattern for the Mercury structural test cases are:

   M_*functional-unit*_ST_*xxx*.TC   for test case input files
   M_*functional-unit*_ST_*xxx*.EX   for expected results files

3) Fetch the support and utility files from CMS for the desired functional unit

4) Execute structural test:

   For Pluto, the command to run the test cases is:

   "@TC_DRIVER *functional-unit* PST *xxx*"

   For Mercury, the command to run the test cases is:

   "@M_ST_DRIVER *functional-unit* *xxx*"

   where the driver is given the name of the functional unit and the test case number.

A.5 DESIGN REVIEW CHECKLIST

Process Structure

1. Does the design clearly follow the data flow and control flow described in the specification?	yes/no
2. Does integration with the simulator follow the sequencing and implementation described in the specification?	yes/no
3. Does process sequencing comply with the functional unit scheduling as presented in Table 4.3 in the specification?	yes/no
4. Do modules have high internal cohesion?[3]	yes/no
5. Do modules have low external coupling?[4]	yes/no
6. Are all module interfaces described?	yes/no

Data Usage

1. Do the design data stores comply with those described in the specification's Data Requirements Dictionary Part II?	yes/no
2. Do the specified data in the design data dictionary conform to the specification's Data Requirements Dictionary Part I?	yes/no
3. If the design includes variables in addition to the global data store variables defined in the GCS specification, and these variables represent flows between processes, are they included in the design data dictionary?	yes/no
4. Do process inputs and outputs comply with the functional unit inputs and outputs in the specification?	yes/no
5. Are all inputs to processes used?	yes/no
6. Does each process modify only those global variables that are specified outputs for that process?	yes/no
7. Are all the input/output variables of a process defined in the INPUT/OUTPUT section of the design P-Spec for that process?	yes/no

Detail Requirements

1. Are sufficient algorithmic details given (including those not provided by the specification)?	yes/no
2. Are all specified logical conditions included in the design?	yes/no
3. Do logical conditions correctly use logical and relational operators?	yes/no
4. Are exceptional conditions anticipated and handled as described in the specification?	yes/no

Traceability

1. Does the design satisfy all the functional requirements described in the specification?	yes/no
2. Can all parts of the design be traced back to the requirements?	yes/no

[3]Cohesion refers to the degree to which the internal elements of a module are bound to a related task.
[4]Coupling refers to the degree of interconnectedness between modules.

Clarity

1. Is the overall function of each process described?	yes/no
2. Are assumptions documented?	yes/no

Compliance with Standards

1. Are all derived requirements identified and justified?	yes/no
2. Was a successful balance check performed on the team*work* model of the design?	yes/no
3. Do the software design and the design documentation comply with the approved methodology and the design standards?	yes/no

A.6 CODE REVIEW CHECKLIST

Data Usage

1. Are COMMON BLOCKS labeled with the same names as the global data stores defined in GCS Data Requirements Dictionary Part II?	yes/no
2. Do the variables in the COMMON BLOCKS use the same names and order as the variables in the global data stores defined in the GCS Data Requirements Dictionary Part II?	yes/no
3. Do the variables in the COMMON BLOCKS have the same data types, number of dimensions, and size of each dimension as specified in the GCS Data Requirements Dictionary Part I?	yes/no
4. If the code includes variables in addition to those defined in the global data stores in the GCS Data Requirements Dictionary Part II, are they defined, initialized, and used only within the scope of a subframe?	yes/no
5. Are references to array subscripts expressed in column, row order?	yes/no
6. Is array subscript usage within array bounds?	yes/no
7. Are constant values used only as constants and not as variables?	yes/no
8. Are DO loop index variables used only within the loop?	yes/no
9. Does the code maintain that same loop index within a loop?	yes/no

Structure

1. Does the code comply with the software architecture from the design?	yes/no
2. Does the code avoid the use of GOTO statements?	yes/no
3. Do all the code's statements perform a clear function?	yes/no
4. Is the code void of any isolated or dead code segments?	yes/no

Functions and Subroutines

1. Does each unit have a single function, and is it clearly described?	yes/no
2. Do actual and formal parameters agree in number, order, dimension, and data type?	yes/no
3. Are the functions of subroutine input and output parameters described?	yes/no
4. Are all the parameters passed to a subroutine used?	yes/no
5. Do the functions and subroutines return data of the correct type?	yes/no
6. Is there a call to GCS_SIM_RENDEZVOUS before each subframe?	yes/no
7. Are calls to GCS_SIM_RENDEZVOUS void of all parameters?	yes/no
8. Does the code avoid using system calls?	yes/no

Traceability

1. Does the code satisfy all the requirements in the Requirements Traceability Matrix including all derived requirements?	yes/no
2. Do units map to a well-defined section in the Design?	yes/no

Logic

1. Do logical conditions correctly use logical operators (.AND., .OR., .NOT.)?	yes/no
2. Do logical conditions correctly use relational operators (.GT., .GE., .LT., .LE., .EQ., .NE.)?	yes/no
3. Are all logical conditions included?	yes/no
4. Are comparisons of real variables to exact values avoided?	yes/no
5. Is loop nesting correct?	yes/no
6. Do loops have single exit and single entry points?	yes/no

Exceptional Conditions

1. Is there code to detect the exceptional conditions listed in the Non-Functional section of the Requirements Traceability Matrix?	yes/no
2. If an exceptional condition is detected, does the code print the appropriate message to FORTRAN logical unit 6?	yes/no

Computations

1. Are mixed type mathematical expressions avoided?	yes/no
2. Do computations contain values with the same unit dimensions?	yes/no
3. Does the code avoid assigning real expressions to integers causing truncation?	yes/no
4. Are bit manipulations done correctly?	yes/no

Compliance with Standards

1. Does the code follow basic structured programming techniques?	yes/no
2. Does the software code and documentation comply with the approved code standards?	yes/no

A.7 REQUIREMENTS TRACEABILITY MATRIX

Functional Requirements	DESIGN	CODE	TESTCASES
0-1 Specify four separate, globally accessible data stores: EXTERNAL, GUIDANCE_STATE, RUN_PARAMETERS, and SENSOR_OUTPUT.			
2-1 Control flow of the frame processing.			
2-1.1 The appropriate control flow for a frame is: call to GCS_SIM_RENDEZVOUS. Satisfy the Sensor Processing subframe requirements (2-2). call to GCS_SIM_RENDEZVOUS. Satisfy Guidance Processing subframe requirements (2-3). call to GCS_SIM_RENDEZVOUS fulfill Control Law Processing subframe requirements (2-4) or terminate (2-1.2).			
2-1.2 The implementation is to terminate immediately upon completion of the Control Law Processing subframe requirements during the frame in which GP_PHASE is set to 5.			
2-2 Sensor Processing subframe requirements.			
2-2.1 Satisfy the TSP requirements (2.1.5) prior to fulfilling any of the other requirements in (2.1.1 and 2.1.4).			
2-2.2 Satisfy all requirements in the sensor processing requirements hierarchy (2.1).			
2-2.3 Satisfy all requirements in the communications processing requirements (2.4) upon satisfying 2-2.1.			
2-2.4 Adhere to the functional unit scheduling in Table 4.3 of the GCS specification.			
2-3 The Guidance Processing subframe requirements.			
2-3.1 Satisfy all requirements in the guidance processing requirements (2.2).			
2-3.2 Satisfy all requirements in the communications processing requirements (2.4) upon satisfying 2-3.1.			
2-4 The Control Law Processing subframe requirements.			
2-4.1 Satisfy the AECLP requirements (2.3.1) prior to fulfilling any of the CRCP requirements (2.3.3).			
2-4.2 Satisfy all requirements in the control law processing requirements hierarchy (2.3).			
2-4.3 Satisfy all requirements in the communications processing requirements (2.4) upon satisfying 2-4.1.			
2-4.4 Adhere to the functional unit scheduling in Table 4.3 of the GCS specification.			
2.1 SP -- Sensor Processing			
2.1.1 ASP -- Accelerometer Sensor Processing			
2.1.1-1 Rotate variables.			
2.1.1-2 Adjust gain for temperature.			
2.1.1-3 Remove characteristic bias.			
2.1.1-4 Correct for misalignment.			
2.1.1-5 Determine Accelerations.			
2.1.1-5.1 Acceleration based on current A_COUNTER.			
2.1.1-5.2 Acceleration based on mean of previous accelerations.			
2.1.1-6 Determine Accelerometer Status			
2.1.1-6.1 A_STATUS = healthy			
2.1.1-6.2 A_STATUS = unhealthy			
2.1.2 ARSP -- Altimeter Radar Sensor Processing			
2.1.2-1 Rotate variables.			
2.1.2-2 Determine altitude when echo is received. (based on AR_COUNTER)			
2.1.2-3 Determine altitude when echo is not received			
2.1.2-3.1 Determine altitude based on third-order polynomial.			
2.1.2-3.2 Determine altitude based on previous calculation.			
2.1.2-4 Set altimeter radar status.			
2.1.2-4.1 AR_STATUS = healthy			
2.1.2-4.2 AR_STATUS = failed			
2.1.2-5 Set values of K_ALT.			

2.1.2-5.1	K_ALT = 1			
2.1.2-5.2	K_ALT = 0			
2.1.3	**TDLRSP -- Touch Down Landing Radar Sensor Processing**			
2.1.3-1	Rotate variables			
2.1.3-2	Determine state for each radar beam.			
2.1.3-2.1	TDLR_STATE = unlocked.			
2.1.3-2.2	TDLR_STATE = locked.			
2.1.3-3	Determine Whether to set FRAME_BEAM_UNLOCKED			
2.1.3-3.1	Set FRAME_BEAM_UNLOCKED to FRAME_COUNTER			
2.1.3-3.2	Leave FRAME_BEAM_UNLOCKED unchanged			
2.1.3-4	Calculate the beam velocities			
2.1.3-5	Process beam velocities based on which beam(s) locked.			
2.1.3-5.1	no beams locked			
2.1.3-5.2	Beam1 locked			
2.1.3-5.3	Beam2 locked			
2.1.3-5.4	Beam3 locked			
2.1.3-5.5	Beam4 locked			
2.1.3-5.6	Beam1 & Beam2 locked			
2.1.3-5.7	Beam1 & Beam3 locked			
2.1.3-5.8	Beam1 & Beam4 locked			
2.1.3-5.9	Beam2 & Beam3 locked			
2.1.3-5.10	Beam2 & Beam4 locked			
2.1.3-5.11	Beam3 & Beam4 locked			
2.1.3-5.12	Beam1, Beam2, & Beam3 locked			
2.1.3-5.13	Beam1, Beam2, & Beam4 locked			
2.1.3-5.14	Beam1, Beam3, & Beam4 locked			
2.1.3-5.15	Beam2, Beam3, & Beam4 locked			
2.1.3-5.16	Beam1, Beam2, Beam3, & Beam4 locked			
2.1.3-6	Convert to body velocities.			
2.1.3-7	Set values in K_MATRIX.			
2.1.3-7.1	Kx = 0			
2.1.3-7.2	Kx = 1			
2.1.3-7.3	Ky = 0			
2.1.3-7.4	Ky = 1			
2.1.3-7.5	Kz = 0			
2.1.3-7.6	Kz = 1			
2.1.3-8	Set TDLR_STATUS.			
2.1.4	**GSP -- Gyroscope Sensor Processing**			
2.1.4-1	Rotate variables.			
2.1.4-2	Determine the vehicle rotation rates along each of the vehicle's three axes.			
2.1.4-2.1	Adjust gain.			
2.1.4-2.2	Convert G_COUNTER.			
2.1.4-3	Set gyroscope status to healthy.			
2.1.5	**TSP -- Temperature Sensor Processing**			
2.1.5-1	Calculate solid state temperature			
2.1.5-2	Calculate Thermal Temperature			
2.1.5-3	Determine which Temperature to use (SS or Thermocouple)			
2.1.5-3.1	Calculate the Thermo sensor upper limit			
2.1.5-3.2	Calculate the Thermo sensor lower limit			
2.1.5-4	Determine Atmospheric Temperature			
2.1.5-5	Set status to healthy.			
2.1.6	**TDSP -- Touch Down Sensor Processing**			
2.1.6-1	Determine status of touch down sensor.			
2.1.6-2	Determine whether touch down has been sensed.			
2.2	**GP -- Guidance Processing**			
2.2-1	Rotate variables.			
2.2-2	Determine the attitude, velocities, and altitude.			
2.2-2.1	Set up the GP_ROTATION matrix.			
2.2-2.2	Calculate new values of attitude, velocity, and altitude.			
2.2-3	Determine if the engines should be on or off.			
2.2-3.1	Engines on			
2.2-3.2	Engines off			
2.2-4	Set FRAME_ENGINES_IGNITED			
2.2-5	Determine velocity error.			
2.2-6	Determine optimal velocity			
2.2-7	Determine if contour has been crossed.			
2.2-8	Determine guidance phase.			
2.2-8.1	GP_PHASE = 1			
2.2-8.2	GP_PHASE = 2			
2.2-8.3	GP_PHASE = 3			
2.2-8.4	GP_PHASE = 4			

2.2-8.5	GP_PHASE = 5			
2.2-9	Determine which set of control law parameters to use.			
2.2-9.1	CL = 1			
2.2-9.2	CL = 2			
2.3	**CLP -- Control Law Processing**			
2.3.1	**AECLP -- Axial Engine Control Law Processing**			
2.3.1-1	Generate the appropriate axial engine commands when AE_CMD=ON.			
2.3.1-1.1	Determine engine temperature			
2.3.1-1.1.1	AE_TEMP = COLD			
2.3.1-1.1.2	AE_TEMP = WARM			
2.3.1-1.1.3	AE_TEMP = HOT			
2.3.1-1.2	Compute limiting errors for pitch			
2.3.1-1.3	Compute limiting error for yaw			
2.3.1-1.4	Compute limiting error for thrust			
2.3.1-1.5	Compute pitch, yaw, and thrust errors.			
2.3.1-1.5.1	CHUTE_RELEASED = 1			
2.3.1-1.5.2	CHUTE_RELEASRD = 0			
2.3.1-1.5.3	CONTOUR_CROSSED = 1			
2.3.1-1.5.4	CONTOUR_CROSSED = 0			
2.3.1-1.6	Compute INTERNAL_CMD			
2.3.1-1.7	Compute axial engine valve settings (AE_CMD).			
2.3.1-1.7.1	when INTERNAL_CMD < 0.0			
2.3.1-1.7.2	when 0.0 ≤ INTERNAL_CMD ≥ 1.0			
2.3.1-1.7.3	when 1.0 < INTERNAL_CMD			
2.3.1-2	Generate the appropriate axial engine commands when AE_CMD=OFF.			
2.3.1-2.1	Set AE_CMD = 0			
2.3.1-3	Set axial engine status to healthy.			
2.3.2	**RECLP -- Roll Engine Control Law Processing**			
2.3.2-1	Generate the appropriate roll engine command.			
2.3.2-2	Set roll engine status to healthy.			
2.3.3	**CRCP -- Chute Release Control Processing**			
2.3.3-1	Determine appropriate parachute release command.			
2.3.3-1.1	AE_TEMP = COLD			
2.3.3-1.2	AE_TEMP = WARM			
2.3.3-1.3	AE_TEMP = HOT			
2.3.3-1.4	CHUTE_RELEASED = 0			
2.3.3-1.5	CHUTE_RELEASED = 1			
2.4	CP -- Communications Processing			
2.4-1	Set communicator status to healthy.			
2.4-2	Get synchronization pattern.			
2.4-3	Determine sequence number.			
2.4-4	Prepare sample mask.			
2.4-4.1	Subframe 1 mask			
2.4-4.2	Subframe 2 mask			
2.4-4.3	Subframe 3 mask			
2.4-5	Prepare data section.			
2.4-5.1	Use subframe 1 data			
2.4-5.2	Use subframe 2 data			
2.4-5.3	Use subframe 3 data			
2.4-2.5	Calculate checksum.			

A.8 SAMPLE REVIEW LOG FORM

Individual Inspection Preparation Log

Page 1 of __

Name:____________________________ Date Log Submitted:______

Implementation:____________________ Date of Inspection:________

Role: o Reader o Recorder

o Moderator o Inspector

Defects/Clarity Problems/Concerns

Location of Concern (i.e. module name/P-Spec #, page number, etc.)	Description of the problem

The Moderator needs to have this completed form at least 4 hours before the scheduled Inspection session.

Individual Inspection Preparation Log

Page__ of __

Defects/Clarity Problems/Concerns

Location	Description of the problem

A.9 GCS Equivalence Classes

This section gives the equivalence classes used for testing GCS implementations. Two tables are include to allow cross referencing of equivalence class to test cases. Table A.9-1, GCS Equivalence Classes, lists all the variables that have equivalent classes. These do not include variables in the RUN_PARAMETER data store. Also, as stated previously, variables defined as integers but used as enumerated types are not considered testable with equivalence classes. Finally, variables that are outputs to the GCS simulator are not tested. The GCS Equivalence Class Table gives the variable name, its type, its limits as defined in the GCS Specification, and the equivalence classes defined for that variable. Table A.9-1 also associates an equivalence class name with each class definition. This name is used in Table A.9-2 to test cases with the equivalence class. The equivalence class names are derived from the variable name with an arbitrary number extension. The names do not imply whether the class is valid or invalid. That can be readily determined by reviewing the variable's limits; typically, the first one listed is the valid equivalence class. The following abbreviations are used in Table A.9-1 to represent FORTRAN Type definition:

R8 -	REAL*8	I2 -	INTEGER*2
Ll -	LOGICAL*1	I4 -	INTEGER*4

Table A.9-1 : GCS Equivalence Classes

VARIABLE NAME	TYPE	Limits	Equivalence Class Definitions	Equivalence Class Name
A_ACCELERATION	R8	[-20., 5.]	-20. ≤ A_ACCELERATION ≤ 5 A_ACCELERATION > 5. A_ACCELERATION < -20.	A_ACCELERATION.1 A_ACCELERATION.2 A_ACCELERATION.3
A_COUNTER	I2	[0, (2^{15})-1]	0 ≤ A_COUNTER ≤ (2^{15})-1 A_COUNTER > (2^{15})-1 A_COUNTER < 0	A_COUNTER.1 A_COUNTER.2 A_COUNTER.3
A_STATUS	Ll	0=HEALTHY 1=UNHEALTH Y	HEALTHY UNHEALTHY INVALID	A_STATUS.1 A_STATUS.2 A_STATUS.3
AR_ALTITUDE	R8	[0., 2000.]	0. ≤ AR_ALTITUDE ≤ 2000. AR_ALTITUDE > 2000. AR_ALTITUDE < 0	AR_ALTITUDE.1 AR_ALTITUDE.2 AR_ALTITUDE.3
AR_COUNTER	I2	[-1, (2^{15})-1]	-1 < AR_COUNTER ≤ (2^{15})-1 AR_COUNTER > (2^{15})-1 AR_COUNTER ≤ -1	AR COUNTER.1 AR_COUNTER.2 AR_COUNTER.3
AR_STATUS	L1	0=HEALTHY 1=FAILED	HEALTHY FAILED INVALID	AR_STATUS.1 AR_STATUS.2 AR_STATUS.3
ATMOSPHERIC_TE MP	R8	[-200., 25.]	-200. ≤ ATMOSPHERIC_TEMP ≤ 25. ATMOSPHERIC_TEMP > 25. ATMOSPHERIC_TEMP < -200.	ATMOSPHERIC_TEMP.1 ATMOSPHERIC_TEMP.2 ATMOSPHERIC_TEMP.3
FRAME_BEAM_ UNLOCKED	I4	[0, (2^{31})-1]	0 ≤ FRAME_BEAM_UNLOCKED ≤ (2□□^{31})-1 FRAME_BEAM_UNLOCKED > (2^{31})-1 FRAME_BEAM_UNLOCKED < 0	FRAME_BEAM_UNLOCKED.1 FRAME_BEAM_UNLOCKED.2 FRAME_BEAM_UNLOCKED.3
FRAME COUNTER	I4	[1, (2^{31})-1]	1 ≤ FRAME COUNTER £ (2^{31})-1 FRAME COUNTER > (2^{31})-1 FRAME COUNTER < 1	FRAME COUNTER.1 FRAME COUNTER.2 FRAME COUNTER.3

Table A.9-1 : (Continued)

VARIABLE NAME	TYPE	Limits	Equivalence Class Definitions	Equivalence Class Name
FRAME_ENGINES_ IGNITED	I4	[1, (2^{31})-1]	1 ≤ FRAME_ENGINES_IGNITED ≤ (2^{31})-1 FRAME_ENGINES_IGNITED > (2^{31})-1 FRAME_ENGINES_IGNITED < 1	FRAME_ENGINES_IGNITED.1 FRAME_ENGINES_IGNITED.2 FRAME_ENGINES_IGNITED.3
G_COUNTER	I2	[-$(2^{14}$-l), 2^{14}-1]	-((2^{14}) -1) ≤ G_COUNTER ≤ (2^{14})-1 G_COUNTER > (2^{14})-1 G_COUNTER < -((2^{14}) -1)	G_COUNTER.1 G_COUNTER.2 G_COUNTER.3
G_ROTATION	R8	[-1.0, 1.0]	-1.≤ G_ROTATION < -P4 -P4 ≤ G_ROTATION < -P3 -P3 ≤ G_ROTATION < -P2 -P2 ≤ G_ROTATION < -P1 -P1 ≤ G_ROTATION < 0 0 ≤ G_ROTATION < P1 P1 ≤ G_ROTATION < P2 P2 ≤ G_ROTATION < P3 P3 ≤ G_ROTATION < P4 P4 ≤ G_ROTATION < 1 G_ROTATION > 1 G_ROTATION < -1	G_ROTATION.1 G_ROTATION.2 G_ROTATION.3 G_ROTATION.4 G_ROTATION.5 G_ROTATION.6 G_ROTATION.7 G_ROTATION.8 G_ROTATION.9 G_ROTATION.10 G_ROTATION.11 G_ROTATION.12
GP_ALTITUDE	R8	[0., 2000.]	0.≤ GP_ALTITUDE ≤ ENGINES_ON_ALTITUDE ENGINES_ON_ALTITUDE.< GP_ALTITUDE ≤ 2000 GP_ALTITUDE > 2000 GP_ALTITUDE < 0	GP_ALTITUDE.1 GP_ALTITUDE.2 GP_ALTITUDE.3 GP_ALTITUDE.4
GP_ATTITUDE	R8	[-l.,1.]	-1. ≤ GP_ATTITUDE ≤ 1. GP_ATTITUDE > 1. GP_ATTITUDE < -1.	GP_ATTITUDE.1 GP_ATTITUDE.2 GP_ATTITUDE.3
GP_ROTATION	R8	[-l.,1.]	-1. ≤ GP_ROTATION(i,j) ≤ 1. GP_ROTATION(i,j) > 1. GP_ROTATION(i,j) > -1.	GP_ROTATION.1 GP_ROTATION.2 GP_ROTATION.3
GP_VELOCITY	R8	[-100., 100.]	-100. ≤ GP_VELOCITY(I) ≤ 100. GP_VELOCITY > 100. GP_VELOCITY < -100.	GP_VELOCITY.1 GP_VELOCITY.2 GP_VELOCITY.3
INTERNAL_CMD	R8	[-.7, 1.7]	-.7 ≤ INTERNAL_CMD < 0 0 £ INTERNAL_CMD ≤ 1.0 1.0 < INTERNAL_CMD ≤ 1.7 INTERNAL_CMD > 1.7 INTERNAL_CMD < -.7	INTERNAL_CMD.1 INTERNAL_CMD.2 INTERNAL_CMD.3 INTERNAL_CMD.4 INTERNAL_CMD.5
PE_INTEGRAL	R8	[-100., 100.]	-100 ≤ PE_INTEGRAL ≤ 100. PE_INTEGRAL > l00. PE_INTEGRAL < -100.	PE_INTEGRAL.1 PE_INTEGRAL.2 PE_INTEGRAL.3
SS_TEMP	I2	[0, (2^{15})-1]	M3 - 0.15L ≤ SS_TEMP ≤ M4 + 0.15L 0 ≤ SS_TEMP < M3 - 0.15L M4 + 0.15L < SS_TEMP ≤ (2^{15})-1 SS_TEMP > (2^{15})-1 SS_TEMP < 0	SS_TEMP.1 SS_TEMP.2 SS_TEMP.3 SS_TEMP.4 SS_TEMP.5
TD_COUNTER	I2	[-2^{15}, (2^{15})-1]	TD_COUNTER = 0 TD_COUNTER = -1 -2^{15} ≤ TD_COUNTER ≤ (2^{15})-1	TD_COUNTER.1 TD_COUNTER.2 TD_COUNTER.3
TDLR_COUNTER	I2	[0, (2^{15})-1]	0 ≤ TDLR_COUNTER ≤ (2^{15})-1 TDLR_COUNTER > (2^{15})-1 TDLR_COUNTER < 0	TDLR_COUNTER.1 TDLR_COUNTER.2 TDLR_COUNTER.3
TDLR_STATE	Ll	0=UNLOCKED 1=LOCKED	BEAM UNLOCKED BEAM LOCKED INVALID	TDLR_STATE.1 TDLR_STATE.2 TDLR_STATE.3
TDLR_VELOCITY	R8	[-100., 100.]	-100.≤ TDLR VELOCITY ≤ 100. TDLR_VELOCITY > 100. TDLR_VELOCITY < -100.	TDLR_VELOCITY.1 TDLR_VELOCITY.2 TDLR_VELOCITY.3

TDS_STATUS	L1	0=HEALTHY 1=FAILED	HEALTHY FAILED INVALID	TDS_STATUS.1 TDS_STATUS.2 TDS_STATUS.3
TE_INTEGRAL	R8	[-100., 100.]	-100. ≤ TE_INTEGRAL ≤ 100. TE_INTEGRAL > 100. TE_INTEGRAL < -100.	TE_INTEGRAL.1 TE_INTEGRAL.2 TE_INTEGRAL.3
TE_LIMIT	R8	[-100., 100.]	-100. ≤ TE_LIMIT < TE_MIN TE_MIN ≤ TE_LIMIT ≤ TE_MAX TE_MAX < TE_LIMIT ≤ 100. TE_LIMIT > 100. TE_LIMIT < -100.	TE_LIMIT.1 TE_LIMIT.2 TE_LIMIT.3 TE_LIMIT.4 TE_LIMIT.5
THERMO_TEMP	I2	[0, (2^{15})-1]	M3 ≤ THERMO_TEMP ≤ M4 M3 - 0.15L ≤ THERMO_TEMP < M3 M4 < THERMO_TEMP ≤ M4 + 0.15L 0 ≤ THERMO_TEMP < M3 - 0.15L M4 + 0.15L < THERMO_TEMP ≤ (2^{15})-1 THERMO_TEMP > (2^{15})-1 THERMO_TEMP < 0	THERMO_TEMP.1 THERMO_TEMP.2 THERMO_TEMP.3 THERMO_TEMP.4 THERMO_TEMP.5 THERMO_TEMP.6 THERMO_TEMP.7
THETA	R8	[-p,p]	-p ≤ THETA < -THETA2 -THETA2 ≤ THETA < -THETA1 -THETA1 ≤ THETA < 0 0 ≤ THETA < THETA1 THETA1 ≤ THETA < THETA2 THETA2 ≤ THETA ≤ p THETA > p THETA < -p	THETA.1 THETA.2 THETA.3 THETA.4 THETA.5 THETA.6 THETA.7 THETA.8
VELOCITY_ERROR	R8	[-300., 20.]	-300. ≤ VELOCITY_ERROR ≤ 20. VELOCITY_ERROR > 20. VELOCITY_ERROR < -300.	VELOCITY_ERROR.1 VELOCITY_ERROR.2 VELOCITY_ERROR.3
YE_INTEGRAL	R8	[-100., 100.]	-100. ≤ YE_INTEGRAL ≤ 100. YE_INTEGRAL > 100. YE_INTEGRAL < -100.	YE_INTEGRAL.1 YE_INTEGRAL.2 YE_INTEGRAL.3

Table A.9-2 : List of Test Cases by Equivalence Class Name

Equivalence Class Name	Test case(s)
A_ACCELERATION.1	AECLP_NR_001--012.TC, GP_NR_001--008.TC, ASP_NR_001.TC, ASP_NR_002.TC
A_ACCELERATION.2	AECLP_RO_038.TC, GP_RO_012.TC, GP_RO_014.TC, GP_RO_016.TC, GP_RO_018.TC, GP_RO_020.TC, GP_RO_022.TC, GP_RO_024.TC, GP_RO_026.TC, GP_RO_028.TC, ASP_RO_018.TC, ASP_RO_020.TC, ASP_RO_022.TC, ASP_RO_024.TC, ASP_RO_026.TC, ASP_RO_028.TC, ASP_RO_030.TC, ASP_RO_032.TC, ASP_RO_034.TC, ASP_RO_036.TC, ASP_RO_038.TC, ASP_RO_040.TC
A_ACCELERATION.3	AECLP_RO_037.TC, GP_RO_011.TC, GP_RO_013.TC, GP_RO_015.TC, GP_RO_017.TC, GP_RO_019.TC, GP_RO_021.TC, GP_RO_023.TC, GP_RO_025.TC, GP_RO_027.TC, ASP_RO_017.TC, ASP_RO_019.TC, ASP_RO_021.TC, ASP_RO_023.TC, ASP_RO_025.TC, ASP_RO_027.TC, ASP_RO_029.TC, ASP_RO_031.TC, ASP_RO_033.TC, ASP_RO_035.TC, ASP_RO_037.TC, ASP_RO_039.TC
A_COUNTER.1	ASP_NR_001.TC, ASP_NR_003.TC, ASP_NR_016.TC
A_COUNTER.2	ASP_RO_013 -- 015.TC
A_COUNTER.3	ASP_RO_010 -- 012.TC
A_STATUS.1	ASP_NR_001.TC
A_STATUS.2	ASP_NR_003 -- 005.TC
A_STATUS.3	ASP_RO_041 -- 049.TC
AR_ALTITUDE.1	GP_NR_001--008.TC ARSP_NR_011.TC, ARSP_NR_016.TC, ARSP_NR_017.TC
AR_ALTITUDE.2	GP_RO_048.TC, GP_RO_050.TC, GP_RO_052.TC, ARSP_RO_007 -- 010.TC
AR_ALTITUDE.3	GP_RO_047.TC, GP_RO_049.TC, GP_RO_051.TC ARSP_RO_003 -- 006.TC
AR COUNTER.1	ARSP_NR_011.TC, ARSP_NR_016.TC, ARSP_NR_017.TC, ARSP_NR_022.TC, ARSP_NR_023.TC,
AR_COUNTER.2	ARSP_RO_001.TC
AR_COUNTER.3	ARSP_RO_002.TC
AR_STATUS.1	ARSP_NR_011.TC, ARSP_NR_016.TC, ARSP_NR_017.TC
AR_STATUS.2	ARSP_NR_012 -- 015.TC
AR_STATUS.3	ARSP_RO_018 -- 021.TC
ATMOSPHERIC_TEMP.1	ASP_NR_001.TC GSP_NR_001.TC
ATMOSPHERIC_TEMP.2	ASP_RO_009.TC GSP_RO_003.TC
ATMOSPHERIC_TEMP.3	ASP_RO_008.TC GSP_RO_002.TC
FRAME_BEAM_UNLOCKED.1	TDLRSP_NR_001.TC, TDLRSP_NR_003.TC, TDLRSP_NR_005.TC, TDLRSP_NR_007 -- 021.TC,
FRAME_BEAM_UNLOCKED.2	TDLRSP_RO_022.TC
FRAME_BEAM_UNLOCKED.3	TDLRSP_RO_023.TC
FRAME COUNTER.1	TDLRSP_NR_001.TC, TDLRSP_NR_003.TC, TDLRSP_NR_005.TC, TDLRSP_NR_007 -- 021.TC
FRAME COUNTER.2	TDLRSP_RO_024.TC
FRAME COUNTER.3	TDLRSP_RO_025.TC
FRAME_ENGINES_IGNITED.1	GP_NR_001-008.TC, GP_NR_053.TC, AECLP_NR_001-012.TC
FRAME_ENGINES_IGNITED.2	AECLP_RO_056.TC
FRAME_ENGINES_IGNITED.3	AECLP_RO_057.TC
G_COUNTER.1	GSP_NR_001.TC
G_COUNTER.2	GSP_RO_007 -- 009.TC
G_COUNTER.3	GSP_RO_004 -- 006.TC
G_ROTATION.1	RECLP_NR_046,056--058,068.TC
G_ROTATION.2	RECLP_NR_039,040,048.TC

Table A.9-2 : (Continued)

Equivalence Class Name	Test case(s)
G_ROTATION.3	RECLP_NR_015--017.TC RECLP_NR_020,031,032,036,038,050,055.TC
G_ROTATION.4	RECLP_NR_023--026,033.TC
G_ROTATION.5	AECLP_NR_005--007.TC, AECLP_NR_011--012.TC GP_NR_005--008.TC RECLP_NR_003--006,011,012.TC
G_ROTATION.6	AECLP_NR_001--004.TC, AECLP_NR_008,010.TC GP_NR_001--008.TC RECLP_NR_001--059.TC, RECLP_NR_064-067.TC
G_ROTATION.7	RECLP_NR_022,027,028,034.TC
G_ROTATION.8	AECLP_NR_004,010.TC GP_NR_001--004.TC RECLP_NR_013,014,018,019.TC RECLP_NR_021,029,030,035,037.TC
G_ROTATION.9	GP_NR_002--004.TC RECLP_NR_010.TC, RECLP_NR_041--044.TC, RECLP_NR_047,049,051.TC
G_ROTATION.10	AECLP_NR_005--007.TC GP_NR_001.TC RECLP_NR_001--059.TC, RECLP_NR_064-067.TC
G_ROTATION.11	RECLP_RO_060.TC, GP_RO_069--071.TC, GP_RO_075--077.TC, GP_RO_081-083.TC
G_ROTATION.12	RECLP_RO_061.TC, GP_RO_066--068.TC, GP_RO_072--074.TC, GP_RO_078--080.TC
GP_ALTITUDE.1	AECLP_NR_001--012.TC, GP_NR_001--008.TC
GP_ALTITUDE.2	AECLP_RO_039,040,042,044,045,047.TC
GP_ALTITUDE.3	AECLP_RO_014.TC, GP_RO_010.TC
GP_ALTITUDE.4	AECLP_RO_013.TC, GP_RO_009.TC
GP_ATTITUDE.1	AECLP_NR_001--012.TC, GP_NR_001--008.TC
GP_ATTITUDE.2	AECLP_RO_016.TC, GP_RO_029.TC, GP_RO_031.TC, GP_RO_033.TC, GP_RO_035.TC, GP_RO_037.TC, GP_RO_039.TC, GP_RO_041.TC, GP_RO_043.TC, GP_RO_045.TC
GP_ATTITUDE.3	AECLP_RO_015.TC, GP_RO_030.TC, GP_RO_032.TC, GP_RO.034_TC, GP_RO_036.TC, GP_RO_038.TC, GP_RO_040.TC, GP_RO_042.TC, GP_RO_044.TC, GP_RO_046.TC
GP_ROTATION.1	AECLP_NR_001--012.TC, GP_NR_001--008.TC
GP_ROTATION.2	AECLP_RO_018.TC, AECLP_RO_020.TC
GP_ROTATION.3	AECLP_RO_017.TC, AECLP_RO_019.TC
GP_VELOCITY.1	AECLP_NR_001--012.TC, GP_NR_001--008.TC
GP_VELOCITY.2	AECLP_R0_022.TC, AECLP_RO_024.TC, AECLP_RO_026.TC, GP_RO_055.TC, GP_RO_057.TC, GP_RO_059.TC, GP_RO_60.TC, GP_RO_062.TC, GP_RO_064.TC
GP_VELOCITY.3	AECLP_RO_021.TC, AECLP_RO_023.TC, AECLP_R0_025.TC, GP_RO_O54.TC, GP_RO_056.TC, GP_RO_058.TC, GP_RO_061.TC, GP_RO_63.TC, GP_RO_65.TC
INTERNAL_CMD.1	AECLP_NR_004,008,009.TC
INTERNAL_CMD.2	AECLP_NR_001--007.TC
INTERNAL_CMD.3	AECLP_NR_055.TC
INTERNAL_CMD.4	AECLP_RO_049.TC, AECLP_RO_051.TC, AECLP_RO_053.TC
INTERNAL_CMD.5	AECLP_RO_048.TC, AECLP_RO_050.TC, AECLP_RO_052.TC
PE_INTEGRAL.1	AECLP_NR_001--012.TC
PE_INTEGRAL.2	AECLP_RO_028.TC

Table A.9-2 : (Continued)

Equivalence Class Name	**Test case(s)**
PE_INTEGRAL.3	AECLP_RO_027.TC
SS_TEMP.1	TSP_NR_001.TC
SS_TEMP.2	TSP_NR_002.TC
SS_TEMP.3	TSP_NR_003.TC
SS_TEMP.4	TSP_RO_004.TC
SS_TEMP.5	TSP_RO_005.TC
TD_COUNTER.1	TDSP_NR_001.TC
TD_COUNTER.2	TDSP_NR_002.TC
TD_COUNTER.3	TDSP_NR_003.TC
TDLR_COUNTER.1	TDLRSP_NR_001.TC, TDLRSP_NR_003.TC, TDLRSP_NR_005.TC, TDLRSP_NR_007 -- 021.TC,
TDLR_COUNTER.2	TDLRSP_RO_028.TC
TDLR_COUNTER.3	TDLRSP_RO_027.TC
TDLR_STATE.1	TDLRSP_NR_001.TC, TDLRSP_NR_007 -- 021.TC,
TDLR_STATE.2	TDLRSP_NR_005.TC, TDLRSP_NR_007 -- 021.TC,
TDLR_STATE.3	TDLRSP_RO_026.TC
TDLR_VELOCITY.1	GP_NR_001--008.TC
TDLR_VELOCITY.2	GP_RO_085.TC, GP_RO_087.TC, GP_RO_089.TC, GP_RO_091.TC, GP_RO_093.TC, GP_RO_095.TC, GP_RO_097.TC, GP_RO_099.TC, GP_RO_101.TC
TDLR_VELOCITY.3	GP_RO_084.TC, GP_RO_086.TC, GP_RO_088.TC, GP_RO_090.TC, GP_RO_092.TC, GP_RO_094.TC, GP_RO_096.TC, GP_RO_098.TC, GP_RO_100.TC
TDS_STATUS.1	TDSP_NR_001 -- 003.TC
TDS_STATUS.2	TDSP_NR_004 -- 006.TC
TDS_STATUS.3	TDSP_RO_007.TC
TE_INTEGRAL.1	AECLP_NR_001--012.TC
TE_INTEGRAL.2	AECLP_RO_30.TC
TE_INTEGRAL.3	AECLP_RO_029.TC
TE_LIMIT.1	AECLP_NR_005.TC, AECLP_RO_029,033.TC
TE_LIMIT.2	AECLP_NR_006-009.TC
TE_LIMIT.3	AECLP_RO_030,034.TC
TE_LIMIT.4	AECLP_RO_032.TC
TE_LIMIT.5	AECLP_RO_031.TC
THERMO_TEMP.1	TSP_NR_001.TC
THERMO_TEMP.2	TSP_NR_006.TC
THERMO_TEMP.3	TSP_NR_007.TC
THERMO_TEMP.4	TSP_NR_008.TC
THERMO_TEMP.5	TSP_NR_009.TC
THERMO_TEMP.6	TSP_RO_010.TC
THERMO_TEMP.7	TSP_RO_011.TC
THETA.1	RECLP_NR_010,011,019,020,027,034,035,037.TC RECLP_NR_040,041,046,048,050-054,066.TC
THETA.2	RECLP_NR_006,007,026,030,031.TC
THETA.3	RECLP_NR_002,003,014,015,022,023,054,059.TC RECLP_NR_064.TC
THETA.4	RECLP_NR_001,004,013,016,021,024,065,068.TC
THETA.5	RECLP_NR_005,008,028,029.TC
THETA.6	RECLP_NR_009,012,017,018,025,032,033,036.TC RECLP_NR_038,039,042-045,047,049,055-058.TC RECLP_NR_067.TC
THETA.7	RECLP_RO_062.TC
THETA.8	RECLP_RO.063.TC
VELOCITY_ERROR.1	AECLP_NR_001--012.TC
VELOCITY_ERROR.2	AECLP_RO_034.TC
VELOCITY_ERROR.3	AECLP_RO_033.TC
YE_INTEGRAL.1	AECLP_NR_001--012.TC
YE_INTEGRAL.2	AECLP_RO_036.TC
YE_INTEGRAL.3	AECLP_RO_035.TC

A.10 Traceability Matrix For Requirements-based Test Cases

Table A.10-1 is the Traceability Matrix with Requirements test cases filled in. It gives a detailed listing of the GCS requirements and gives the test cases that test those requirements. Note that cases listed fall into the normal range category as defined by DO-178B because they verify that the software functions according to the GCS Specification. Since these cases are requirements-based, this table is identical for both the Mercury and Pluto implementations. Hence the information is placed here instead of in the results document.

Table A.10-1 : Traceability Matrix with Requirements -based Test cases

Functional Requirements	**TESTCASE NAME**
0-1 Specify four separate, globally accessible data stores: EXTERNAL, GUIDANCE_STATE, RUN_PARAMETERS, and SENSOR_OUTPUT.	All Test Cases
2-1 Control flow of the frame processing.	
2-1.1 The appropriate control flow for a frame is: 1) call to GCS_SIM_RENDEZVOUS. 2) Satisfy the Sensor Processing subframe requirements (2-2). 3) Call to GCS_SIM_RENDEZVOUS. 4) Satisfy Guidance Processing subframe requirements (2-3). 5) Call to GCS_SIM_RENDEZVOUS 6) Satisfy Control Law Processing subframe requirements (2-4) 7) Terminate if GP_PHASE = 5 (2-1.2).	FRAME_001-009.TC
2-1.1-1 GP_PHASE transition from 1 to 2	FRAME_001.TC
2-1.1-2 GP_PHASE = 2, just before AE_TEMP transition	FRAME_002.TC
2-1.1-3 AE_TEMP transitions from WARM to HOT and CHUTE_RELEASED transitions from 0 to 1	FRAME_003.TC
2-1.1-4 GP_PHASE transitions from 2 to 3	FRAME_004.TC
2-1.1-5 CONTOUR_CROSSED transitions from 0 to 1	FRAME_005.TC
2-1.1-6 Frame after CONTOUR_CROSSED transitions	FRAME_006.TC
2-1.1-7 CL = 2	FRAME_007.TC
2-1.1-8 GP_PHASE transitions from 3 to 4	FRAME_008.TC
2-1.2 The implementation is to terminate immediately upon completion of the Control Law Processing subframe requirements during the frame in which GP_PHASE is set to 5.	FRAME_009.TC
2-2 Sensor Processing subframe requirements.	
2-2.1 Satisfy the TSP requirements (2.1.5) prior to fulfilling any of the other requirements in (2.1.1 and 2.1.4).	SP_001.TC
2-2.2 Satisfy all requirements in the sensor processing requirements hierarchy (2.1).	SP_001.TC
2-2.3 Satisfy all requirements in the communications processing requirements (2.4) upon satisfying 2-2.1.	SP_001.TC
2-2.4 Adhere to the functional unit scheduling in Table 4.3 of the GCS specification.	SP_001.TC
2-3 The Guidance Processing subframe requirements.	
2-3.1 Satisfy all requirements in the guidance processing requirements (2.2).	GPSF_001-008.TC

2-3.2	Satisfy all requirements in the communications processing requirements (2.4) upon satisfying 2-3.1.	GPSF_001-008.TC
2-4	**The Control Law Processing subframe requirements.**	
2-4.1	Satisfy the AECLP requirements (2.3.1) prior to fulfilling any of the CRCP requirements (2.3.3).	CLP_001-014.TC
2-4.2	Satisfy all requirements in the control law processing requirements hierarchy (2.3).	CLP_001-014.TC
2-4.3	Satisfy all requirements in the communications processing requirements (2.4) upon satisfying 2-4.1.	CLP_001-014.TC
2-4.4	Adhere to the functional unit scheduling in Table 4.3 of the GCS specification.	CLP_001-014.TC
2.1	**SP -- Sensor Processing**	
2.1.1	**ASP -- Accelerometer Sensor Processing**	
2.1.1-1	Rotate variables.	ASP_NR_006-007.TC
2.1.1-2	Adjust gain for temperature.	ASP_NR_001-005.TC
2.1.1-3	Remove characteristic bias.	ASP_NR_001-005.TC
2.1.1-4	Correct for misalignment.	ASP_NR_001-005.TC
2.1.1-5	Determine Accelerations.	
2.1.1-5.1	Acceleration based on current A_COUNTER.	ASP_NR_001.TC, ASP_NR_003-005.TC
2.1.1-5.2	Acceleration based on mean of previous accelerations.	ASP_NR_002.TC
2.1.1-6	Determine Accelerometer Status	
2.1.1-6.1	A_STATUS = healthy	ASP_NR_001.TC
2.1.1-6.2	A_STATUS = unhealthy	ASP_NR_002.TC
2.1.2	**ARSP -- Altimeter Radar Sensor Processing**	
2.1.2-1	Rotate variables.	ARSP_NR_022-023.TC
2.1.2-2	Determine altitude when echo is received. (based on AR_COUNTER)	ARSP_NR_016.TC, ARSP_NR_017.TC, ARSP_NR_022.TC
2.1.2-3	Determine altitude when echo is not received	
2.1.2-3.1	Determine altitude based on third-order polynomial.	ARSP_NR_011.TC
2.1.2-3.2	Determine altitude based on previous calculation.	ARSP_NR_012-015.TC
2.1.2-4	Set altimeter radar status.	
2.1.2-4.1	AR_STATUS = healthy	ARSP_NR_022.TC
2.1.2-4.2	AR_STATUS = failed	ARSP_NR_011.TC, ARSP_NR_012.TC
2.1.2-5	Set values of K_ALT.	
2.1.2-5.1	K_ALT = 1	ARSP_NR_011.TC
2.1.2-5.2	K_ALT = 0	ARSP_NR_012-015.TC
2.1.3	**TDLRSP -- Touch Down Landing Radar Sensor Processing**	
2.1.3-1	Rotate variables	TDLRSP_NR_001.TC
2.1.3-2	Determine state for each radar beam.	
2.1.3-2.1	TDLR_STATE = unlocked.	TDLRSP_NR_005.TC
2.1.3-2.2	TDLR_STATE = locked.	TDLRSP_NR_001.TC
2.1.3-3	Determine Whether to set FRAME_BEAM_UNLOCKED	
2.1.3-3.1	Set FRAME_BEAM_UNLOCKED to FRAME_COUNTER	TDLRSP_NR_003.TC, TDLRSP_NR_005.TC
2.1.3-3.2	Leave FRAME_BEAM_UNLOCKED unchanged	TDLRSP_NR_001.TC
2.1.3-4	Calculate the beam velocities	TDLRSP_NR_001.TC
2.1.3-5	Process beam velocities based on which beam(s) locked.	

2.1.3-5.1 no beams locked	TDLRSP_NR_005.TC
2.1.3-5.2 Beam1 locked	TDLRSP_NR_007.TC
2.1.3-5.3 Beam2 locked	TDLRSP_NR_008.TC
2.1.3-5.4 Beam3 locked	TDLRSP_NR_009.TC
2.1.3-5.5 Beam4 locked	TDLRSP_NR_010.TC
2.1.3-5.6 Beam1 & Beam2 locked	TDLRSP_NR_011.TC
2.1.3-5.7 Beam1 & Beam3 locked	TDLRSP_NR_012.TC
2.1.3-5.8 Beam1 & Beam4 locked	TDLRSP_NR_013.TC
2.1.3-5.9 Beam2 & Beam3 locked	TDLRSP_NR_014.TC
2.1.3-5.10 Beam2 & Beam4 locked	TDLRSP_NR_015.TC
2.1.3-5.11 Beam3 & Beam4 locked	TDLRSP_NR_016.TC
2.1.3-5.12 Beam1, Beam2, & Beam3 locked	TDLRSP_NR_017.TC
2.1.3-5.13 Beam1, Beam2, & Beam4 locked	TDLRSP_NR_018.TC
2.1.3-5.14 Beam1, Beam3, & Beam4 locked	TDLRSP_NR_019.TC
2.1.3-5.15 Beam2, Beam3, & Beam4 locked	TDLRSP_NR_020.TC
2.1.3-5.16 Beam1, Beam2, Beam3, & Beam4 locked	TDLRSP_NR_001.TC, TDLRSP_NR_021.TC
2.1.3-6 Convert to body velocities.	TDLRSP_NR_001.TC
2.1.3-7 Set values in K_MATRIX.	
2.1.3-7.1 Kx = 0	TDLRSP_NR_007-010.TC
2.1.3-7.2 Kx = 1	TDLRSP_NR_001.TC
2.1.3-7.3 Ky = 0	TDLRSP_NR_007-010.TC
2.1.3-7.4 Ky = 1	TDLRSP_NR_001.TC
2.1.3-7.5 Kz = 0	TDLRSP_NR_007-010.TC
2.1.3-7.6 Kz = 1	TDLRSP_NR_001.TC
2.1.3-8 Set TDLR_STATUS.	TDLRSP_NR_001.TC
2.1.4 GSP -- Gyroscope Sensor Processing	
2.1.4-1 Rotate variables.	GSP_NR_001.TC
2.1.4-2 Determine the vehicle rotation rates along each of the vehicle's three axes.	
2.1.4-2.1 Adjust gain.	GSP_NR_001.TC
2.1.4-2.2 Convert G_COUNTER.	GSP_NR_001.TC
2.1.4-3 Set gyroscope status to healthy.	GSP_NR_001.TC
2.1.5 TSP -- Temperature Sensor Processing	
2.1.5-1 Calculate solid state temperature	TSP_NR_001-002.TC
2.1.5-2 Calculate Thermal Temperature	TSP_NR_001.TC
2.1.5-3 Determine which Temperature to use (SS or Thermocouple)	
2.1.5-3.1 Calculate the Thermo sensor upper limit	TSP_NR_001-002.TC
2.1.5-3.2 Calculate the Thermo sensor lower limit	TSP_NR_001-002.TC
2.1.5-4 Determine Atmospheric Temperature	TSP_NR_001-002.TC
2.1.5-5 Set status to healthy.	TSP_NR_001.TC
2.1.6 TDSP -- Touch Down Sensor Processing	
2.1.6-1 Determine status of touch down sensor.	TDSP_NR_001-003.TC
2.1.6-2 Determine whether touch down has been sensed.	TDSP_NR_001-003.TC
2.2 GP -- Guidance Processing	
2.2-1 Rotate variables.	GP_NR_001-008.TC
2.2-2 Determine the attitude, velocities, and altitude.	
2.2-2.1 Set up the GP_ROTATION matrix.	GP_NR_001-008.TC
2.2-2.2 Calculate new values of attitude, velocity, and altitude.	GP_NR_001-008.TC

2.2-3	Determine if the engines should be on or off.	
2.2-3.1	Engines on	GP_NR_003-008.TC
2.2-3.2	Engines off	GP_NR_001-002.TC
2.2-4	Set FRAME_ENGINES_IGNITED	GP_NR_002.TC
2.2-5	Determine velocity error.	GP_NR_001-008.TC
2.2-6	Determine optimal velocity	GP_NR_001-008.TC
2.2-7	Determine if contour has been crossed.	GP_NR_001-008.TC
2.2-8	Determine guidance phase.	
2.2-8.1	GP_PHASE = 1	GP_NR_001.TC
2.2-8.2	GP_PHASE = 2	GP_NR_001-004.TC
2.2-8.3	GP_PHASE = 3	GP_NR_004-008.TC
2.2-8.4	GP_PHASE = 4	GP_NR_008.TC
2.2-8.5	GP_PHASE = 5	GP_NR_102-106.TC
2.2-9	Determine which set of control law parameters to use.	
2.2-9.1	CL = 1	GP_NR_001-008.TC
2.2-9.2	CL = 2	GP_NR_053.TC
2.3	**CLP -- Control Law Processing**	
2.3.1	**AECLP -- Axial Engine Control Law Processing**	
2.3.1-1	Generate the appropriate axial engine commands when AE_CMD=ON.	
2.3.1-1.1	Determine engine temperature	
2.3.1-1.1.1	AE_TEMP = COLD	AECLP_NR_001-002.TC
2.3.1-1.1.2	AE_TEMP = WARM	AECLP_NR_003.TC, AECLP_NR_010.TC
2.3.1-1.1.3	AE_TEMP = HOT	AECLP_NR_004-009.TC, AECLP_NR_011-012.TC
2.3.1-1.2	Compute limiting errors for pitch	AECLP_NR_001-012.TC
2.3.1-1.3	Compute limiting error for yaw	AECLP_NR_001-012.TC
2.3.1-1.4	Compute limiting error for thrust	AECLP_NR_001-012.TC
2.3.1-1.5	Compute pitch, yaw, and thrust errors.	
2.3.1-1.5.1	CHUTE_RELEASED = 1	AECLP_NR_004-009.TC, AECLP_NR_011-012.TC
2.3.1-1.5.2	CHUTE_RELEASRD = 0	AECLP_NR_001-003.TC, AECLP_NR_010.TC
2.3.1-1.5.3	CONTOUR_CROSSED = 1	AECLP_NR_005-009.TC, AECLP_NR_012.TC
2.3.1-1.5.4	CONTOUR_CROSSED = 0	AECLP_NR_001-004.TC, AECLP_NR_010-011.TC
2.3.1-1.6	Compute INTERNAL_CMD	AECLP_NR_001-0012.TC, AECLP_NR_055.TC
2.3.1-1.7	Compute axial engine valve settings (AE_CMD).	
2.3.1-1.7.1	when INTERNAL_CMD < 0.0	AECLP_NR_004.TC
2.3.1-1.7.2	when 0.0 ≤ INTERNAL_CMD ≥ 1.0	AECLP_NR_001-003.TC, AECLP_NR_006-012.TC
2.3.1-1.7.3	when 1.0 < INTERNAL_CMD	AECLP_NR_055.TC
2.3.1-2	Generate the appropriate axial engine commands when AE_CMD=OFF.	
2.3.1-2.1	Set AE_CMD = 0	AECLP_NR_054.TC
2.3.1-3	Set axial engine status to healthy.	AECLP_NR_001-012.TC
2.3.2	**RECLP -- Roll Engine Control Law Processing**	
2.3.2-1	Generate the appropriate roll engine command.	RECLP_NR_001-059.TC, RECLP_NR_064-068.TC

2.3.2-2	Set roll engine status to healthy.	RECLP_NR_001-059.TC, RECLP_NR_064-068.TC
2.3.3	**CRCP -- Chute Release Control Processing**	
2.3.3-1	Determine appropriate parachute release command.	
2.3.3-1.1	AE_TEMP = COLD	CRCP_NR_001-002.TC
2.3.3-1.2	AE_TEMP = WARM	CRCP_NR_003-004.TC
2.3.3-1.3	AE_TEMP = HOT	CRCP_NR_005-006.TC
2.3.3-1.4	CHUTE_RELEASED = 0	CRCP_NR_001.TC, CRCP_NR_003.TC, CRCP_NR_005.TC
2.3.3-1.5	CHUTE_RELEASED = 1	CRCP_NR_002.TC, CRCP_NR_004.TC, CRCP_NR_006.TC
2.4	CP -- Communications Processing	
2.4-1	Set communicator status to healthy.	CP_NR_001-005.TC
2.4-2	Get synchronization pattern.	CP_NR_001-005.TC
2.4-3	Determine sequence number.	CP_NR_001-005.TC
2.4-4	Prepare sample mask.	
2.4-4.1	Subframe 1 mask	CP_NR_001.TC
2.4-4.2	Subframe 2 mask	CP_NR_002.TC
2.4-4.3	Subframe 3 mask	CP_NR_003.TC
2.4-5	Prepare data section.	
2.4-5.1	Use subframe 1 data	CP_NR_001.TC
2.4-5.2	Use subframe 2 data	CP_NR_002.TC
2.4-5.3	Use subframe 3 data	CP_NR_003.TC
2.4-2.5	Calculate checksum.	CP_NR_001-005.TC

A.11 Test Case Summary

This section summarizes all the files used in GCS testing into 2 tables and is created for quick referencing when carrying out procedures for generating test cases (section A.12) or executing test cases (Test Case Execution Procedures). Files used for requirements-based testing are given in Table A.11-1 and those for structural testing are given in Table A.11-2. Table A.11-1 organizes the files for requirements-based test cases by the four phases as described in the *Software Verification Plan* . Structural test cases in Table A.11-2 are divided by the two implementations because they are implementation specific. Both tables divide all files into 2 general groups:

1. the files used for **generating** the test-inputs and expected-values files on the SUN platform
2. the files used for **executing** the test cases on the VAX platform.

Files used for **generating** test cases are in a separate group because they are generated using *Mathematica* (ref. A.7). For the GCS project, *Mathematica* is supplied for the SUN platform only hence these files are effectively used only on that platform. These files are further divided into three groups separated by vertical doted lines. Files specific to a functional unit are listed in a row. Files containing actual test data are in the Data Files sub-column; files used in test case generation are in the Support Files sub-column; finally, utility files are in their own sub-column. The utility files are used throughout the test case generating process and are not specific to any functional unit subframe or frame. Although some utility files are used by only a small subset of cases, fetching them for other test cases will not hurt if they are not used. As stated in the Test Cases Overview, the Data Files are used to generate the Test-Input and Expected-Values files using the procedure in section A.12. It is the TC and EX files that are used in testing the implementations. It should be noted that files for generating CP functional unit and trajectory test cases are not list in the **generating** column but given in a special blocks in the **executing** column. This is because trajectory test cases are generated on the VAX running the Venus prototype with the GCS Simulator. CP test cases are generated on the VAX because CP is sensitive to the bit representation of numerical values. CP_NR_xx.EX files are generated on the VAX from CP_NR_xx.TC files. This is more apparent after reviewing the procedures for generating trajectory and CP functional unit test cases in section A.12.

Files used for **executing** test cases on the VAX platform are also divided into 3 groups. The first group are the Test-Inputs files (with ".tc" extension) and Expected-Values files (with ".ex" or ".seed" extension). These ASCII files are the outputs of the test case generating process and are transferred from the SUN. Samples of these files are given in section A.14. The second group consists of files that facilitate test case execution. These files are sometime refered to as test stubs or test drivers and an example is given in section A.15. They are, in general, VMS FORTRAN files and VMS DCL files. The third group are again utility files used for different phases of testing. This section of the table is referenced when carrying out test case execution procedures.

Table A.11-1: File list for requirements-based test suites.

<table>
<tr><th rowspan="2">Test Phase</th><th colspan="3">Generating
Test Case Input and Expected Values files
(Using Mathematica on the SUN)</th><th colspan="3">Executing
Test Cases with both implementations
(Using VAX Fortran Programming Environment)</th></tr>
<tr><th>Data files (Implementation independent)</th><th>Support files (Implementation independent)</th><th>Utility Files for generating test case input and expected values files</th><th>Output files from Mathematica serve as input files for test case execution</th><th>Test execution support files (implementation specific for all test phases except for trajectory tests)</th><th>Utility Files for executing test cases</th></tr>
<tr><td>Functional Units
NR = Normal Range Test Case
RO = Robustness Test Case</td><td>arsp_nr_xxx.m
arsp_ro_xxx.m

asp_nr_xxx.m
asp_ro_xxx.m

gsp_nr_xxx.m
gsp_ro_xxx.m

tdlrsp_nr_xxx.m
tdlrsp_ro_xxx.m

tdsp_nr_xxx.m
tdsp_ro_xxx.m

tsp_nr_xxx.m
tsp_ro_xxx.m

gp_tc.xx

aeclp_tc.xx

crcp_tc.xx

reclp_tc.xx</td><td>arsp.m
run_arsp_tc.m

asp.m
run_asp_tc.m

gsp.m
run_gsp_tc.m

tdlrsp.m
run_tdlrsp_tc.m

tdsp.m
run_tdsp_tc.m

tsp.m
run_tsp_tc.m

gp.m
run_gp.xx

aeclp.m
run_aeclp.xx

crcp.m
run_crcp.xx

reclp.m
run_reclp.xx</td><td>input
namelist1
namelist_ex
write_nml.m
write_exnml.m</td><td>arsp_nr_xxx.tc, ex
arsp_ro_xxx.tc, ex

asp_nr_xxx.tc, ex
asp_ro_xxx.tc, ex

gsp_nr_xxx.tc, ex
gsp_ro_xxx.tc, ex

tdlrsp_nr_xxx.tc, ex
tdlrsp_ro_xxx.tc, ex

tdsp_nr_xxx.tc, ex
tdsp_ro_xxx.tc, ex

tsp_ro_xxx.tc, ex
tsp_nr_xxx.tc, ex

gp_nr_xxx.tc, ex
gp_ro_xxx.tc, ex

aeclp_nr_xxx.tc, ex
aeclp_ro_xxx.tc, ex

reclp_nr_xxx.tc, ex
reclp_ro_xxx.tc, ex

crcp_nr_xxx.tc, ex
crcp_ro_xxx.tc, ex

cp_nr_xx.tc, ex</td><td>i_lnkarsp.com
i_test_arsp.for

i_lnkasp.com
i_test_asp.for

i_lnkgsp.com
i_test_gsp.for

i_lnktdlrsp.com
i_test_tdlrsp.for

i_lnktdsp.com
i_test_tdsp.for

i_lnktsp.com
i_test_tsp.for

i_lnkgp.com
i_test_gp.for

i_lnkaeclp.com
i_test_aeclp.for

i_lnkreclp.com
i_test_reclp.for

i_lnkcrcp.com
i_test_crcp.for

i_lnkcp.com
i_test_cp.for

(files for generating CP expected values)
common.inc
cp.for
cp.com</td><td>struct.for_inc
commons.for_inc
compare_external.for
compare_guidance.for
compare_runpram.for
compare_sensor.for
read_tc.for
read_ex.for
i_tc_driver.com

name_list.inc
exname_list.inc</td></tr>
</table>

Subframe	sp_001.m	arsp.m, asp.m gsp.m, tsp.m tdsp.m, tdlrsp.m run_gpsf.xx gp.m		sp_001.tc, ex	i_lnksp.com i_test_sp.for i_sp_driver.com	cp_ex.for
	gpsf_tc.xx	run_clp.xx aeclp.m reclp.m crcp.m		gpsf_xx.tc, ex	i_lnkgpsf.com i_test_gpsf.for i_gpsf_driver.co m	
	clp_tc.xx			clp_xx.tc, ex	i_lnkclp.com i_test_clp.for i_clp_driver,.com	
Frame	frame_xx.m	frame.m run_frame_tc.m arsp.m, asp.m, gsp.m, tsp.m, tdsp.m, tdlrsp.m gp.m, aeclp.m, reclp.m,, crcp.m		frame_xx.tc, ex	i_lnkframe.com i_test_frame.for i_frame_driver.co m	

Table A.11-1 (continued): File list for requirements-based test suites.

Test Phase	Generating Test Case Input and Expected Values files (Using Mathematica on the SUN)	Executing Test Cases with both implementation.s (Using VAX Fortran Programming Environment)		
		Output files from Mathematica serve as input files for test case execution	Test execution support files (implementation specific for all test phases except for trajectory tests)	Utility Files for executing test cases
Trajectory	*Simulator Input and expected-values files generated on the VAX*	traj_atm_ic_xx.tc traj_atm_ud_xx.tc traj_atm_xx.seed traj_td_ic_xx.tc traj_td_ud_xx.tc traj_td_xx.seed	i_traj.com i_run_traj.com i_build.com (files for creating trajectory expected values) venusrs.exe venus_runges _switches.dat runsimi.com do_assign.com venus_traj.com run_venus_traj.com	traj_sim.exe page_align.opt gcs_sim_rendezvous.obj gcs_setup.obj gcs_who_am_i.obj gcs_list.dat gcs_sim_switches.dat tabular_data.dat accuracy.dat alternate_accuracy.dat limits.dat

Table A.11-1 and A.11-2 are a handy quick references of all the files involved for any specific test suite. For example, to regenerate test-input and expected-values files for the GP functional unit, the tester may survey Table A.11-1 and see that gp_tc.xx data files are needed; gp.m, and run_gp.xx support files are needed; and the utility files are needed.

File names in both Table A.11-1 and A.11-2 have been abbreviated to show only the group names. Names given in the tables with “xx” and “xxx” in the name denote a group of files where the specific file name can be derived by substituting the “x” with two or three digits. The full unabbreviated list is given in the Test Case Overview. File names with “i_” in the execution support file sub-column are implementation specific where the “i” is the initial of the implementation. “P” for Pluto files and “M” for Mercury files.

Table A.11-2: File list for structural testing of Mercury and Pluto.

Test Phase (Structural)	**Generating** Test Case Input and Expected Values files (Using Mathematica on the SUN)			**Executing** Test Cases with both implementations. (Using VAX Fortran Programming Environment)		
	Data files	Support files	Utility Files for generating test case input and expected values files	Output files from Mathematica serve as input files for test case execution	Test execution support files (implementation specific for all test phases except for trajectory tests)	Utility Files for executing test cases
Mercury	m_aeclp_st.xx m_gp_st.xx m_reclp_st.xx m_asp_st_xx.m m_arsp_st_xx.m m_tdlrsp_st_xx.m m_tsp_st_001.m	m_run_aeclp_st.xx m_run_gp_st.xx m_run_reclp_st.xx	input namelist1 namelist_ex write_nml.m write_exnml.m	m_aeclp_st.xx.tc, ex m_gp_st_xx.tc, ex m_reclp_st_xx.tc, ex m_asp_st_xx.tc, ex m_arsp_st_xx.tc, ex m_tdlrsp_st_xx.tc, ex m_tsp_st_001.tc, ex	m_lnkaeclp.com m_test_aeclp.for m_lnkgp.com m_test_gp.for m_lnkreclp.com m_test_reclp.for m_lnkasp.com m_test_asp.for m_lnkarsp.com m_test_arsp.for m_lnktdlrsp.com m_test_tdlrsp.for m_lnktsp.com m_test_tsp.for st_driver.com	struct.for_inc commons.for_inc compare_external.for or compare_guidance.for or compare_runpram.for or compare_sensor.for read_tc.for read_ex.for
Pluto	aeclp_pst_xx.m asp_pst_xx.m gp_pst_xx.m reclp_pst_xx.m	run_aeclp_pst.m run_asp_pst.m run_gp_pst.m run_reclp_pst.m	gp_pst_st7_code.m write_nml_st7.m write_exnml_st7.m	aeclp_pst_xx. tc, ex asp_pst_xx. tc, ex gp_pst_xx. tc, ex reclp_pst_xx. tc, ex	p_lnkaeclp.com p_test_aeclp.for p_lnkasp.com p_test_asp.for p_lnkgp.com p_test_gp.for p_lnkreclp.com p_test_reclp.for p_tc_driver.com	

A.12 Procedure To Generate Test Cases

All test cases, except trajectory and CP are generated on the SUN platform due to available licensing for *Mathematica*. The files generated on the SUN (the ".tc" and ".ex" files) are then transferred to the VAX platform for use in executing the test case for each implementation. The file naming convention for each step of the process is given in Tables G-1 and will be described in the procedures where the files are used. For all functional units other than CP, a model is created using *Mathematica*, before test cases are developed. *Mathematica* is a programming tool that allows complex computations to be easily modeled.

Generating Functional Unit Requirements-based Test Cases

All test cases, except trajectory and CP are generated on the SUN platform due to available licensing for *Mathematica*. The files generated on the SUN (files with ".tc" and ".ex" extensions) are then transferred to the VAX platform for use in executing the test case for each implementation. The file naming convention for each step of the process is given in Tables G-1 and will be described in the procedures where the files are used. For all functional units other than CP, a model is created using *Mathematica*, before test cases are developed. *Mathematica* is a programming tool that allows complex computations to be easily modeled. Then, based on the input list given for a functional unit in the GCS Specification, relevant parameters are identified for the test suite for each functional unit. The relevant parameters are all the variables in the input list that are a part of the EXTERNAL, SENSOR_OUTPUT and GUIDANCE_STATE data stores. Each test case is created by assigning relevant values to the selected parameters in a file to be read by *Mathematica* -- the data files. These values are judiciously chosen based on the coverage requirement that the test case is to fulfill. As stated in the *Software Verification Plan* , the number of cases in the test suite for each functional unit is minimized by selecting values that can satisfy multiple coverage requirements. That is, selecting a particular value for a variable may satisfy its valid equivalence class coverage and also satisfy a low-level requirement in the traceability matrix. Additionally, Myers (ref. A.8) states that the valid equivalence class of several variables can be combined in a single test. These two guidelines serve to significantly reduce the number of requirements-based test cases needed to satisfy the coverage requirements given in DO-178B.

The procedures for generating functional unit test-input and expected-values files are given below for the functional unit ARSP. The procedure is the same for all functional units except the file naming convention changes slightly. This procedure was used to generate the existing test cases and only needs to be used if there is a change in the GCS requirements that necessitates a change in the test data files or *Mathematica* models. The procedure presupposes that the user is using a UNIX system which has *Mathematica*. Since the host system for development and testing is a VAX system, it is assumed that the user has the capability to transfer files between the two hardware platforms.

1) Create a working directory. All files fetched from CMS should be placed in this directory.

2) Reserve the ARSP data and support files from CMS

   ARSP_NR_xxx.M

   ARSP_RO_xxx.M

   ARSP.M

   RUN_ARSP_TC.M

Note that files for functional units in the second and third subframe use a slightly different naming convention. This procedure uses ARSP as an example, see Table A.1 for specific file names of other functional units.

3) Fetch all the utility files:

The specific file names are given in Table A.1 in the last column under the **GENERATING** group. These files give background data for each test case and write the actual test-input and expected-values files.

4) Apply any necessary changes to data and support files:

ARSP.M models the calculations in ARSP. Should the GCS Specification change for ARSP, then this file should be updated.

ARSP_NR_xxx.M and ARSP_RO_xxx.M contain the data needed for Mathematica to generate the respective test cases. If the test input data need to be changed, these are the files to update or new files should be created with the same format as those that currently exist. The easiest way to create new data files is to duplicate one of the existing files and change the data. When new data files are added to the test suite, the file that loads ARSP data files into *Mathematica*, RUN_ARSP_TC.M, must also be updated. This is done by adding an entry into RUN_ARSP_TC.M specifying the name of the new data file. Note that this example gives file names using the naming pattern for functional units in the first subframe. Functional units in the second and third subframe use the form *functional_unit*_TC.xx. If a new test data file is create for this case, a corresponding support file must also be created. These support files have the naming pattern: RUN_*functional_unit*.xx.

5) Run *Mathematica*:

Mathematica should be run in the same directory where all the files are placed. As currently installed, this is done by the command:

"math"

6) Run the data through the model to generate test-input and expected-values:

For ARSP or any functional unit in the first subframe, use the command:

<<run_arsp_tc.m

For functional units in the second and third subframe, each test case must be individually executed with:

<<run_*functional_unit*.xx

7) This procedure will create a test-input (".tc") and expected-results(".ex") file for each data file. These files following the same naming convention for all subframes and is as follows:

arsp_nr_xxx.tc	or	arsp_ro_xxx.tc
arsp_nr_xxx.ex	or	arsp_ro_xxx.ex

8) Now the files can be replaced into CMS and fetched for test case execution.

Generating Functional Unit Structure Based Test Cases

As stated in the GCS Verification Plan, structural testing is performed only at the functional unit level. These test cases are derived with the use of McCabe's ACT software. ACT is used to generate a decision tree for each functional unit. These trees are in the Verification Results document for each implementation. The decision trees are accompanied by tables indicating the decision at each node and the test cases that test the true and false branch of the decision.

MC/DC is satisfied by performing the following. The verification analyst for each implementation compares the test paths to the requirements-based test cases and list, in the tables, all the requirements test cases that exercise the test paths in the tables. If there are any decisions not exercised by the requirements-based test cases, then test cases are devised to exercise those decisions. These cases will be documented in the same table. The process for regenerating the test-input and expected-results for structure based cases will be identical to the process for the requirements-based cases. The naming conventions for the test cases differ for each implementation but the procedure will be the same for both implementations.

1) Create a working directory. All files fetched from CMS should be placed in this directory.

2) Reserve data and support files:

   Table A.2 gives the file names that need to be reserved for both implementations. Only those functional units that currently need structural test cases are given. For any functional unit, data and support files as given in Table A.2 should be reserved and placed in the same directory.

3) Fetch all the utility files:

   The specific file names are given in Table A.2 in the last group under the **GENERATING** column. These files give background data for each test case and write the actual test-input and expected-values files.

4) Applying any necessary changes to the data and support files:

   Any changes to the structure of the code in a functional unit will require examining the new structure to see if new test cases are necessary. For both implementations, this entails regenerating the decision tree graph and modifying the decision tables to add or delete decisions. Data files for the test suites (reserved in step 2) must also be updated to agree with the new decision table.

5) Regenerating test-input and expected-values files:

   This step requires launching *Mathematica* and running the appropriate support files. The specific support files for a given functional unit of any implementation is given in Table A.2. The procedure for launching *Mathematica* and running the support file is identical to steps 5 and 6 of the functional unit procedure. The difference is the filenames used.

6) Now the files can be replaced into CMS and fetched for test case execution.

Generating CP Test Cases

The functional unit, CP, warrants some special considerations due to the nature of its task. Unlike other functional units, CP's function is to create a transmission packet, containing packed data and CRC-16 based checksum. Since the result of the checksum is dependent on the bit ordering of the data in the packet, it is necessary to generate the expected-results file using the same VMS platform that is to run the implementation. This is the only practical way to ensure that the checksum generated for each expected-result file is valid. Additionally, since the VMS platform provides an algorithm for calculating the checksum based on the CRC-16, a comparative algorithm would not have to be devised. It is assumed, for the purposes of the GCS project, that the CRC-16 checksum generator supplied by the VMS operating system is flight qualified. So to generate the expected-results files for CP test cases, a VAX FORTRAN model is written to build the packet and execute the VMS CRC algorithm.

CP expected-results are generated on the VAX that also runs the implementations. The following procedure is used to generate the expected-results files for CP:

1) Create a working directory on the VAX. All files fetched from CMS should be placed in this directory.

2) Reserve data files & fetching support and utility files:

   Data files for CP are the same as the CP test-input files given in Table A.1.

      CP_NR_xxx.TC

   Support files are also given in Table A.1 under the VAX support files. This is the group at the bottom specially noted.

      COMMON.INC

      CP.FOR

      CP.COM

   Utility files are listed in Table A.2 and also given below:

      NAME_LIST.INC

      EXNAME_LIST.INC

3) Apply any necessary changes to files:

   Any changes to specific data items should be applied to the CP_NR_xxx.TC

   Any changes to the functional specification for CP should be applied to CP.FOR

   If new data files are added to the test suite, CP.COM should be changed to add the command for generating expected-results files for the new test case.

4) Regenerating expected-results files:

      "@CP"

5) Now the files can be replaced into CMS and fetched for test case execution.

Generating Integration Level Test Cases

As stated above, integration testing takes place in three phases: subframe, frame, and trajectory testing. In both the subframe and frame tests the expected values will be computed using the same *Mathematica* models used in functional unit testing. Instead of operating individually, these units are linked together to produce models of the subframe or frame. The linked models will be used to generate the appropriate expected values for the subframes and frames in a similar manner as for the functional units. The same comparison and pass/fail criteria used in the functional unit testing will be used in the subframe and frame testing. As mentioned in the Software Verification Plan, only requirements-based software integration testing will be performed for GCS implementations. Hence integration test cases will be requirements-based only. And, because the GCS Specification requires each implementation to use global data stores, the DO-178B requirement to test for parameter passing errors is eliminated. Additionally, since the software is not required to perform any kind of data initializations, the DO-178B requirement to test for incorrect initializations is also eliminated. This greatly reduces the number of test cases necessary during subframe and frame integration testing. The sections below describe development of subframe, frame, and trajectory test cases.

Generating Subframe Test Cases

The overall purpose of integration testing at the subframe level is to ensure that functional units within each subframe will inter-operate and that linking these units does not introduce errors. Additional objectives are given in the test plan found in the Software Verification Plan. For subframe integration tests, test cases are created to test subframe requirements as listed in the Traceability Matrix. Additionally, cases will be selected from the functional unit tests that exercise critical state transitions within a subframe. These cases are documented in the Traceability Matrix in section A.6 and also listed in Table A.1. The procedure for generating subframe test-inputs and expected-results are as follows:

1) Create a working directory on the UNIX environment. All files fetched from CMS should be placed in this directory.

2) Depending on the subframe to regenerate, reserve the data and output files as given below. Also fetch support and utility files as indicated below (These files are also given in Table A.1) :

	Data files	Support Files	Utility Files	Output Files
Subframe 1	SP_001.M	ARSP.M ASP.M GSP.M TSP.M TDSP.M TDLRSP.M	INPUT WRITE_NML.M WRITE_EXNML.M	SP_001.TC SP_001.EX
Subframe 2	GPSF_TC.xx	RUN_GPSF.xx GP.M	INPUT NAMELIST1 NAMELIST_EX	GPSF_xxx.TC GPSF_xxx.EX
Subframe 3	CLP_TC.xx	RUN_CLP.xx AECLP.M RECLP.M CRCP.M	INPUT NAMELIST1 NAMELIST_EX	CLP_xxx.TC CLP_xxx.EX

3) Apply any necessary changes to files:

As in the previous procedures, If test data needs to be modified, the data files should be changed. If any of the functional unit models are modified, then this procedure must be carried out to regenerate the subframe test cases.

4) Regenerating test-input and expected-values files:

Run *Mathematica* from the directory where the files are located. Then

for Subframe 1:

“<<sp_001.m”

for Subframe 2:

“<<run_gpsf.xx” (Where the command has to be repeated for each test case number xx.)

for Subframe 3:

“<<run_clp.xx” (Where the command has to be repeated for each test case number xx.)

5) Now the files can be replaced into CMS and fetched for testing with the GCS implementations.

Generating Frame Test Cases

The integration testing at the frame level is to ensure that the three subframes are independent and that linking these subframes does not introduce errors. Enough cases will be selected so that all state transitions will be tested as well as some random single frames. Multiple frame tests will be covered in the Trajectory Testing.

The frame test case development process will closely follow the subframe process. That is, the Mathematica models of all functional units are linked together to create the frame model FRAME.M. This includes all functional units except for CP and Sim_Rendezvous. Like subframe testing, test cases will be input into the model to generate the test case input files and expected results files. The procedure for generating frame test-input and expected-values is identical to that for the subframe except for the files involved:

1) Create a working directory on the UNIX environment. All files fetched from CMS should be placed in this directory.

2) Reserve the output files and fetch the data, support, and utility files listed below:

Data files	Support Files	Utility Files	Output Files
FRAME_XXX.M	FRAME.M	INPUT	FRAME_XXX.TC
	RUN_FRAME_TC.M	WRITE_NML.M	FRAME_XXX.EX
	ARSP.M	WRITE_EXNML.M	
	ASP.M		
	GSP.M		
	TSP.M		
	TDSP.M		
	TDLRSP.M		
	GP.M		
	AECLP.M		
	RECLP.M		
	CRCP.M		

3) Apply any necessary changes to files:

As in the previous procedures, If test data need to be modified, the data files (FRAME_xxx.M) should be changed. If any of the functional unit models are modified, then this procedure must be carried out to regenerate the frame test cases.

4) Regenerating test-input and expected-values files:

Run *Mathematica* from the directory where the files are located. Then within *Mathematica* enter the command:

"<<run_frame_tc.m" (This command will regenerate all test frame test cases.)

5) Now the files can be replaced into CMS and fetched for testing with the GCS implementations.

Generating Trajectory Test Cases

As indicated in Table A.21, there are two input files for each trajectory test case. All files with "IC" in the name correspond to the INITIAL_CONSTANTS.DAT file required by the simulator. All files with the "UD" in the name correspond to the USAGE_DISTRIBUTIONS.DAT file used by the simulator. The "ATM" and "TD" stand for the names of the respective groups. The files are renamed by the test drivers to the appropriate names as required by the GCS Simulator prior to simulator execution. These data files are created on the VAX system that will run the simulator. As Table A.11-1 and Table A.21 indicates, there is also a ".SEED" file for every trajectory test case. This is the expected-values file for each trajectory test. The ".SEED" files are generated by running the simulator with the VENUS prototype of GCS. Hence the procedure for generating trajectory expected-values files is similar to those for executing trajectory test cases:

1) A directory structure similar to the one for trajectory test case execution must first be created on the VAX.

2) Fetch the following from CMS and placed into the [TRAJ] directory

   a) Trajectory testing utility files as listed in Table A.1 except object files and PAGE_ALIGN.OPT:

      ACCURACY.DAT

      ALTERNATE_ACCURACY.DAT

      GCS_LIST.DAT

      GCS_SIM_SWITCHES.DAT

      LIMITS.DAT

      TABULAR_DATA.DAT

      TRAJ_SIM.EXE

   b) The following files for the VENUS prototype:

      VENUSRS.EXE

      VENUS_RUNGES_SWITCHES.DAT

      RUNSIMI.COM

      DO_ASSIGN.COM

      VENUS_TRAJ.COM

      RUN_VENUS_TRAJ.COM

3) Fetch the following from CMS and place in the [ATM] directory:

      TRAJ_ATM_IC_xxx.TC

      TRAJ_ATM_UD_xxx.TC

      TRAJ_ATM_ xxx.SEED

4) Fetch the following from CMS and place in the [TD] directory:

      TRAJ_TD_IC_xxx.TC

      TRAJ_TD_UD_xxx.TC

      TRAJ_TD_ xxx.SEED

5) The ".SEED" files can then be generated by executing the following command from the [TRAJ] directory at the operating system prompt:

      "@run_venus_traj"

6) Now the files can be replaced into CMS and fetched for testing with the GCS implementations. Note that the ".SEED" files are spread between the [ATM] and [TD] directories.

A.13 *Mathematica* Models

The following are the *Mathematica* models used to generate the expected results for each test case. There is a model for each functional unit except for CP. CP test cases were created using special procedures in place of the *Mathematica* model expected. These procedure are described above in the section Special Procedures for Developing CP Test Cases. Not included in this section are models for subframe and frame testing. Those models simply call the respective functional unit models appropriate subframe and all the models for the frame.

All attempts have been made to keep the copies in this document current. If there are any discrepancies, the version of the model in CMS should supersede the version of the model in this document.

AECLP

```
(***************************************************************************)
(* Filename : aeclp.tc.code                                                *)
(*                                                                         *)
(* Description:                                                            *)
(*                                                                         *)
(* This file contains the Mathematica code to calculate the expected values *)
(* for AECLP.                                                              *)
(* The following assumptions are made:                                     *)
(*     1) the data related to the 4 GCS data stores are pre-loaded         *)
(*     2) the specific data for a test case is also loaded                 *)
(***************************************************************************)

(*  rotate the variables *)

GPALT0 = GPALT[[1]]
GPALT1 = GPALT[[2]]
GPALT2 = GPALT[[3]]
GPALT3 = GPALT[[4]]
GPALT4 = GPALT[[5]]

(* set up the GROT and GPROT arrays *)

q0 = QV
r0 = RV

Array [GPROT0, {3,3}]

GROT0 = N[{p0, q0, r0}, 30]
GPROT0 = N[{{0, r0, -q0}, {-r0, 0, p0}, {q0, -p0, 0}}, 30]
GROT1 = N[{p1, q1, r1}, 30]
GPROT1 = N[{{0, r1, -q1}, {-r1, 0, p1}, {q1, -p1, 0}}, 30]
GROT2 = N[{p2, q2, r2}, 30]
GPROT2 = N[{{0, r2, -q2}, {-r2, 0, p2}, {q2, -p2, 0}}, 30]
GROT3 = N[{p3, q3, r3}, 30]
GPROT3 = N[{{0, r3, -q3}, {-r3, 0, p3}, {q3, -p3, 0}}, 30]
GROT4 = N[{p4, q4, r4}, 30]
GPROT4 = N[{{0, r4, -q4}, {-r4, 0, p4}, {q4, -p4, 0}}, 30]

(* set up the matrix needed for INTERNAL_CMD calcualtion *)

MM1 = {{GP1, 0., 1.}, {GP2, -GPY, 1.}, {GP2, GPY, 1}}

(* set the local variables in the test case equal to the namelist variables *)

CHUTREL = CHUTR
CONTCROSSED = CONTC
ENGONALT = EOALT
FRAMECOUNTER = FRAMEC
FRMENGIGN = FRMEI
GPVEL0[[1,1]] = XDOT
GPVEL0[[2,1]] = YDOT
GPVEL0[[3,1]] = ZDOT
```

```
AACC0[[1,1]] =  XDDOT
GROT0[[2]] = QV
GROT0[[3]] = RV

Print [ALPHA]


(* Compute Limiting error for Pitch and Yaw *)

If[AESWITCH != 0,
   FTIME = N[(FRAMEC - FRMEI)*DELT, 20];
   If[GPALT0 <= EOALT && AETEMP == 0 && FTIME < FULLUPT, AETEMP = 1];
   If[GPALT0 <= EOALT && AETEMP == 1 && FTIME >= FULLUPT, AETEMP = 2];

   PEI = N[PEI + DELT * ZDOT/Abs[XDOT], 20];
   YEI = N[YEI + DELT * YDOT/Abs[XDOT], 20];

   PEL = N[GQ[[CL]]*QV + GW[[CL]]*ZDOT/Abs[XDOT] + GWI[[CL]]*PEI];
   If[PEL < PEMIN[[CL]], PEL = PEMIN[[CL]]];
   If[PEL > PEMAX[[CL]], PEL = PEMAX[[CL]]];

   YEL = N[-GR[[CL]]*RV + GV[[CL]]*YDOT/Abs[XDOT] + GVI[[CL]]*YEI];
   If[YEL < YEMIN[[CL]], YEL = YEMIN[[CL]]];
   If[YEL > YEMAX[[CL]], YEL = YEMAX[[CL]]];

]

(* Compute Liniting Error for Thurst *)

If[CONTC != 0 && AESWITCH != 0,
   TEI = N[TEI + DELT * VELERR, 20];
   X = N[-GAX * (XDDOT + GRAVITY*GPATT0[[1,3]]) + GVE*VELERR + GVEI[[CL]]*TEI, 30];
   X1 = N[(X*GA)/OMEGA, 30];
   EOMEG = N[E^-(OMEGA*DELT), 30];
   TEL = N[X1 + (TEL - X1)*EOMEG, 30];
   If[TEL < TEMIN[[CL]], TEL = TEMIN[[CL]]];
   If[TEL > TEMAX[[CL]], TEL = TEMAX[[CL]]]
]

(* Compute Pitch, Yaw and Thrust errors *)

IA = {0., 0., 0.}

If[AESWITCH !=0,
   If[CHUTR == 1 && CONTC == 1, PE = N[PEL,30]];
   If[CHUTR == 1 && CONTC == 1, YE = N[YEL,30]];
   If[CHUTR == 1 && CONTC == 1, TE = N[TEL,30]];

   If[CHUTR == 1 && CONTC == 0, PE = N[PEL,30]];
   If[CHUTR == 1 && CONTC == 0, YE = N[YEL,30]];
   If[CHUTR == 1 && CONTC == 0, TE = N[TEDROP,30]];

   P1 = N[GQ[[CL]]*QV, 20];
   Y1 = N[-GR[[CL]]*RV, 20];

   If[CHUTR == 0, PE = P1];
```

```
   If[CHUTR == 0, YE = Y1];
   If[CHUTR == 0, TE = TEINIT];

   MM2 = {PE, YE, TE};

(* Print [StringForm["PE = ``", PE]]; *)
(* Print [StringForm["YE = ``", YE]]; *)
(* Print [StringForm["TE = ``", TE]]; *)

(* compute INTERNAL_COMMAND *)

   INTERCMD = N[MM1 . MM2, 20];

(* Print [StringForm["MM1 = ``", MM1]]; *)
(* Print [StringForm["MM2 = ``", MM2]]; *)
(* Print [StringForm["INTERCMD = ``", INTERCMD]];  *)

(* compute AE_CMD *)

IA = 127 INTERCMD;
   If[INTERCMD[[1]] < 0., IA[[1]] = 0.];
   If[INTERCMD[[1]] > 1., IA[[1]] = 127.];
   If[INTERCMD[[2]] < 0., IA[[2]] = 0.];
   If[INTERCMD[[2]] > 1., IA[[2]] = 127.];
   If[INTERCMD[[3]] < 0., IA[[3]] = 0.];
   If[INTERCMD[[3]] > 1., IA[[3]] = 127.]
]
AECMD = Round[IA]

(* ALPHA = "\nAECLP Test Outputs:\n"                  *)
(* Print [ALPHA]                                      *)
(* Print [StringForm["AE_TEMP = ``", AETEMP]]         *)
(* Print [StringForm["AE_STATUS = ``", AESTATUS]]     *)
(* Print [StringForm["PE_INTEGRAL = ``", PEI]]        *)
(* Print [StringForm["TE_INTEGRAL = ``", TEI]]        *)
(* Print [StringForm["TE_LIMIT = ``", TEL]]           *)
(* Print [StringForm["YE_INTEGRAL = ``", YEI]]        *)
(* Print [StringForm["INTERNAL_CMD = ``", INTERCMD]]  *)
(* Print [StringForm["AE_CMD = ``", Round[IA]]]       *)
```

ARSP

```
(*************************************************************************************)
(* Filename :     arsp.m                                                            *)
(* Create Date :  6-27-94                                                           *)
(* Description:                                                                     *)
(*    This file contains the Mathematica code to calculate expected values          *)
(*    for ARSP functional unit.  The following assumotions are made:                *)
(*        1) data related to the 4 GCS data stores are pre-loaded.                  *)
(*        2) the specific data for a test case is also loaded                       *)
(*************************************************************************************)

(* Local variables added for readability *)
healthy = 0       (* used for AR_STATUS *)
failed  = 1       (* used for AR_STATUS *)

(**** Rotate history variables ****)
(* AR_ALTITUDE *)
ARALT4 = ARALT3
ARALT3 = ARALT2
ARALT2 = ARALT1
ARALT1 = ARALT0

(* AR_STATUS *)
ARSTATUS4 = ARSTATUS3
ARSTATUS3 = ARSTATUS2
ARSTATUS2 = ARSTATUS1
ARSTATUS1 = ARSTATUS0

(* K_ALT *)
KALT4 = KALT3
KALT3 = KALT2
KALT2 = KALT1
KALT1 = KALT0

(**** Calculate AR_ALTITUDE ****)

If [ ARCOUNTER > 0
    , ARALT0 = (ARCOUNTER 3 10^8) / (ARFREQ 2)   (* echo received *)
    , IF [    (ARSTATUS4 == healthy)            (* echo not received *)
        && (ARSTATUS3 == healthy)
        && (ARSTATUS2 == healthy)
        && (ARSTATUS1 == healthy)
        , ARALT0 = 4 ARALT1 - 6 ARALT2     (* extimate w/ 3rd order poly *)
        + 4 ARALT3 - ARALT4
        , ARALT0 = ARALT1                 (* set to previous value *)
        ]
    ]

(**** Set AR_STATUS and K_ALT ****)

If [ ARCOUNTER > 0
    , ARSTATUS0 = healthy;
    KALT0 = 1
```

```
    , ARSTATUS0 = failed;
    IF [    (ARSTATUS4 == healthy)
         && (ARSTATUS3 == healthy)
         && (ARSTATUS2 == healthy)
         && (ARSTATUS1 == healthy)
         , KALT = 1
         , KALT = 0 ]
]
```

ASP

```
(****************************************************************************
 Filename :     asp.m
 Create Date :  6-27-94
 Description:
    This file contains the Mathematica code to calculate expected values
    for ASP functional unit.  The following assumptions are made:
        1) data related to the 4 GCS data stores are pre-loaded.
        2) the specific data for a test case is also loaded
 History:
    V0: created (CCQ)
    V1: update to reflect GCS Spec Mod2.3-7 (CCQ)
       new IF block for determining ASTATUS
****************************************************************************)
debug = 1   (* set to 1 for status and debug messages *)

(* Local variables added for readability *)
  healthy = 0       (* used for A_STATUS *)
  failed  = 1       (* used for A_STATUS *)

If [ debug==1, Print["Rotate history..."] ]
(**** Rotate history variables ****)
  (* A_ACCELERATION *)
  AACC4 = AACC3
  AACC3 = AACC2
  AACC2 = AACC1
  AACC1 = AACC0

  (* A_STATUS *)
  ASTATUS4 = ASTATUS3
  ASTATUS3 = ASTATUS2
  ASTATUS2 = ASTATUS1
  ASTATUS1 = ASTATUS0

If [ debug==1, Print["Adjust G_GAIN..."] ]
(**** Adjust A_GAIN() for temperature ****)
again = { N[ AGAIN0[[1]] + G1 ATMTEMP + G2 ATMTEMP^2, 30]
        ,N[ AGAIN0[[2]] + G1 ATMTEMP + G2 ATMTEMP^2, 30]
        ,N[ AGAIN0[[3]] + G1 ATMTEMP + G2 ATMTEMP^2, 30]
             }

If [ debug==1, Print["Remove Bias..."] ]
(**** REMOVE CHARACTERISTIC BIAS ****)
aaccelm = { N[ ABIAS[[1]] + again[[1]] ACOUNTER[[1]], 30]
               ,N[ ABIAS[[2]] + again[[2]] ACOUNTER[[2]], 30]
               ,N[ ABIAS[[3]] + again[[3]] ACOUNTER[[3]], 30]
          }

If [ debug==1, Print["Correct misallignment..."] ]
(**** Correct for Misalignment ****)
  (** NOTE: matrix multiply **)
aacc = N[ ALPMAT . aaccelm, 30]
  (** NOTE: this is a reassignment to correct for the way AACC is declared in the INPUT file.   **)
```

```
AACC0[[1,1]] = aacc[[1]]
AACC0[[2,1]] = aacc[[2]]
AACC0[[3,1]] = aacc[[3]]


If [debug==1, Print["determine acceleration..."] ]
(**** Determine accelerations and accelerometer status ****)
(************** V1 changes **************)
(*-------- old code from V0 ---------*)
(* comments had to be removed from old code,
  Mathematica will not allow embeded comments.

For[ axis=1, axis<=3, axis++,
    If [   (ASTATUS3[[axis]] == healthy)
            && (ASTATUS2[[axis]] == healthy)
            && (ASTATUS1[[axis]] == healthy)
    ,
        mean = N[ ( AACC1[[axis,1]]
                        +AACC2[[axis,1]]
                        +AACC3[[axis,1]]) / 3, 30];
             std = N[ Sqrt[ (( (AACC1[[axis,1]] - mean)^2
                             +(AACC2[[axis,1]] - mean)^2
                             +(AACC3[[axis,1]] - mean)^2
                            ) / 3)
                       ], 30];
             If [debug==1, Print["ax[[",axis,",1]]:: mean=",mean," & std=",std]];
             If [debug==1, Print["ax[[",axis,",1]]=",aacc[[axis]] ]];
             If [debug==1, Print["Perform std-compare..."]];
             If [ Abs[ mean - AACC0[[axis,1]] ] > (ASCALE std)
        ,   AACC0[[axis,1]] = N[mean,30];
                 ASTATUS0[[axis]] = failed;
                 If [debug==1, Print["axis[[",axis,",1]] = mean value"]]
        ,   ASTATUS0[[axis]] = healthy;
                 If [debug==1, Print["axis[[",axis,",1]] = sensor value"]]
        ]
    ,
             If [debug==1, Print["In ELSE branch..."]];
        ASTATUS0[[axis]] = healthy
    ]
]

*)
(*-------- new code for V1 ---------*)
If [debug==1, Print["Start new code"]]
If [debug==1, Print[ASTATUS0] ]
If [debug==1, Print[ASTATUS1] ]
If [debug==1, Print[ASTATUS2] ]
If [debug==1, Print[ASTATUS3] ]
If [debug==1, Print[ASTATUS4] ]
For[ axis=1, axis<=3, axis++,
    ASTATUS0[[axis]] = healthy;
    If [debug==1, Print["ASTATUS0[[",axis,"]]",ASTATUS0[[axis]] ]];
    If [debug==1, Print["a_status ="
                    ,ASTATUS3[[axis]]
                    ,ASTATUS2[[axis]]
                    ,ASTATUS1[[axis]]
```

```
                        ]];
    If [   (ASTATUS3[[axis]] == healthy)
            && (ASTATUS2[[axis]] == healthy)
            && (ASTATUS1[[axis]] == healthy)
    ,(* check extreme values and set A_STATUS, & A_ACCELERATION *)
       If [  (AACC1[[axis,1]] != AACC2[[axis,1]])
               ||(AACC1[[axis,1]] != AACC3[[axis,1]])
       ,   mean = N[ ( AACC1[[axis,1]]
                          +AACC2[[axis,1]]
                          +AACC3[[axis,1]]) / 3, 30];  (* 30 dig. accuracy *)
               std = N[ Sqrt[ (( (AACC1[[axis,1]] - mean)^2
                              +(AACC2[[axis,1]] - mean)^2
                              +(AACC3[[axis,1]] - mean)^2
                             ) / 3)
                           ], 30];        (* 30 digits of accuracy *)
               If [debug==1,
                      Print["ax[[",axis,",1]]:: mean=",mean," & std=",std];
                      Print["ax[[",axis,",1]]=",aacc[[axis]] ];
                      Print["Perform std-compare..."]
                 ];
               If [ Abs[ mean - AACC0[[axis,1]] ] > (ASCALE std)
          ,  AACC0[[axis,1]] = N[mean,30];   (* eliminate outlier numbers *)
                 ASTATUS0[[axis]] = failed;
                 If [debug==1, Print["------axis[[",axis,",1]] = mean value"]]
          ]
       ]   (* close second If statement *)
     ]   (* close first If statement *)
] (* close for loop *)
If [debug==1, Print["After:"] ]
If [debug==1, Print[ASTATUS0] ]
If [debug==1, Print[ASTATUS1] ]
If [debug==1, Print[ASTATUS2] ]
If [debug==1, Print[ASTATUS3] ]
If [debug==1, Print[ASTATUS4] ]

If [debug==1,  Print["Finninhed ASP !!"] ]
```

GP

```
(**************************************************************************)
(* Filename : gp.m                                                        *)
(*                                                                        *)
(* Description:                                                           *)
(*                                                                        *)
(* This file contains the Mathematica code to calculate the expected values *)
(* for GP.                                                                *)
(* The following assumptions are made:                                    *)
(*     1) the data related to the 4 GCS data stores are pre-loaded        *)
(*     2) the specific data for a test case is also loaded                *)
(**************************************************************************)

(* rotate the variables *)


GPATT4 = GPATT3
GPATT3 = GPATT2
GPATT2 = GPATT1
GPATT1 = GPATT0

GPVEL4 = GPVEL3
GPVEL3 = GPVEL2
GPVEL2 = GPVEL1
GPVEL1 = GPVEL0

GPALT4 = GPALT3
GPALT3 = GPALT2
GPALT2 = GPALT1
GPALT1 = GPALT0

(* Runga-Kutta *)

h = 2.*DELTAT

(* first estimates *)

k1 = N[h (GPROT2.GPATT2), 30]

Array[GPATI3, {3,1}]

GPATI3 = N[{{GPATT2[[1,3]]}, {GPATT2[[2,3]]}, {GPATT2[[3,3]]}}, 30]

Array [gprv, {3,1}]
Array [tdlgpv, {3,1}]
Array [l1, {3,1}]
Array [GPROT2, {3,3}]
Array [GPVEL2, {3,1}]

gprv = N[GPROT2.GPVEL2, 30]
tdlgpv = N[TDLRVEL2 - GPVEL2, 30]
Ktdlgpv = N[KMATRIX2 . tdlgpv, 30]
```

```
Print [ALPHA]

l1 = N[h (gprv + GRAVITY GPATI3 + AACC2 + Ktdlgpv), 30]

Array [GPATI1, {1,3}]
GPATI1 = N[{GPATT2[[1,3]], GPATT2[[2,3]], GPATT2[[3,3]]}, 30]
m =N[h (-GPATI1.GPVEL2 + KALT[[3]]*(ARALT2-GPALT2)), 30]
m1 = N[m[[1]], 30]

(* second estimates *)

K12 = N[.5 k1, 30]
L12 = N[.5 l1, 30]
M12 = N[.5*m1, 30]

GPATI = N[GPATT2 + K12, 30]
k2 = N[h (GPROT1.(GPATI)), 30]

Array [l2, {3,1}]

gprv = N[GPROT1.(GPVEL2+L12), 30]
tdlgpv = N[TDLRVEL1 - (GPVEL2+L12), 30]
Ktdlgpv = N[KMATRIX1 . tdlgpv, 30]

GPATI3 = N[{{GPATI[[1,3]]}, {GPATI[[2,3]]}, {GPATI[[3,3]]}}, 30]
l2 = N[h (gprv + GRAVITY GPATI3 + AACC1 + Ktdlgpv), 30]

GPATI1 = N[{GPATI[[1,3]], GPATI[[2,3]], GPATI[[3,3]]}, 30]
m =N[h (-GPATI1.(GPVEL2+L12) + KALT[[2]]*(ARALT1 - (GPALT2+M12))), 30]
m2 = N[m[[1]], 30]

(* third estimates *)

K22 = N[.5 k2, 30]
L22 = N[.5 l2, 30]
M22 = N[.5*m2, 30]

GPATI = N[GPATT2 + K22, 30]
k3 = N[h (GPROT1.(GPATI)), 30]

Array [l3, {3,1}]

gprv = N[GPROT1.(GPVEL2+L22), 30]
tdlgpv = N[TDLRVEL1 - (GPVEL2+L22), 30]
Ktdlgpv = N[KMATRIX1 . tdlgpv, 30]

GPATI3 = N[{{GPATI[[1,3]]}, {GPATI[[2,3]]}, {GPATI[[3,3]]}}, 30]
l3 = N[h (gprv + GRAVITY GPATI3 + AACC1 + Ktdlgpv), 30]

GPATI1 = N[{GPATI[[1,3]], GPATI[[2,3]], GPATI[[3,3]]}, 30]
m =N[h (-GPATI1.(GPVEL2+L22) + KALT[[2]]*(ARALT1 - (GPALT2+M22))), 30]
m3 = N[m[[1]], 30]

(* forth estimates *)

GPATI = N[GPATT2 + k3, 30]
```

```
k4 = N[h (GPROT0.(GPATI)), 30]

Array [l4, {3,1}]

gprv = N[GPROT0.(GPVEL2+l3), 30]
tdlgpv = N[TDLRVEL0 - (GPVEL2+l3), 30]
Ktdlgpv = N[KMATRIX0 . tdlgpv, 30]

GPATI3 = N[{{GPATI[[1,3]]}, {GPATI[[2,3]]}, {GPATI[[3,3]]}}, 30]
l4 = N[h (gprv + GRAVITY GPATI3 + AACC0 + Ktdlgpv), 30]

GPATI1 = N[{GPATI[[1,3]], GPATI[[2,3]], GPATI[[3,3]]}, 30]
m =N[h (-GPATI1.(GPVEL2+l3) + KALT[[1]]*(ARALT0 - (GPALT2+m3))), 30]
m4 = N[m[[1]], 30]

(* calculate new values of GP_ATTITUDE, GP_VELOCITY and GP_ALTITUDE *)

GPATT0 = N[GPATT2 + 1./6. (k1 + 2. (k2 + k3) + k4), 30]
GPVEL0 = N[GPVEL2 + 1./6. (l1 + 2. (l2 + l3) + l4), 30]
GPALT0 = N[GPALT2 + 1./6.*(m1 + 2.*(m2 + m3) + m4), 30]

(****************************)
(*                          *)
(* equation from table 5.9  *)
(*                          *)
(** Old code before V1 Spec. Mod. 2.3-7 requires this change **********)
(*  X = N[2.*GRAVITY*GPALT0, 30]                         *)
(*  DUM = N[Sqrt[X] + GPVEL0[[1,1]], 30]                      *)
(*  X1 = N[DUM, 30]                                                  *)
(*-----------------------------------------------------------------*)
(* New code for V1 - added Max function to avoid negative square root *)
  X = N[2. * GRAVITY * Max[GPALT0,0], 30]
  DUM = N[Sqrt[X] + GPVEL0[[1,1]], 30]
  X1 = N[DUM, 30]
(***************** end of mod for V1 ********************************)

(* This section implements table 5.9 *)
(* AE_SWITCH = 0, OFF           *)
(* AE_SWITCH = 1, ON            *)
(* TD_SENSED = 0, TD NOT SENSED *)
(* TD_SENSED = 1, TD SENSED     *)

(* tengon is a local variable, it is only used to indicate that the engines  *)
(* are off and will be turned on after GP is exited.  This is a problem with *)
(* AE_SWITCH being turned on for the first time ... but the engines are still*)
(* off as far as the GP_PHASE condition meter is concerned                   *)

tengon = 0

If[AESWITCH == 1 && GPALT0 <= DROPHEIGHT && X1 <= MAXNORMVEL &&
   TDSENSED ==0,
    AESWITCH = 0;
    RESWITCH = 0
]

If[AESWITCH == 1 && TDSENSED == 1,
```

```
    AESWITCH = 0;
    RESWITCH = 0
]

If[AESWITCH == 0 && GPALT0 <= ENGONALT && TDSENSED ==0 && RESWITCH == 1,
    FRMENGIGN = FRAMECOUNTER;
    AESWITCH = 1;
    tengon = 1
]

(* DETERMINE OPT_VEL : Find the present altitude in CONTALT and locate the *)
(* corresponding velocity in CONTVEL. Interpolate if necessary             *)

(* first put CONTALT and CONTVEL into the correct units *)

KCONTALT = N[1000. CONTALT]
KCONTVEL = N[1000. CONTVEL]

(* NOTE : fix this in case there are more than 18 values *)

ALTMIN = KCONTALT[[1]]
ALTMAX = KCONTALT[[2]]
VELMIN = KCONTVEL[[1]]
VELMAX = KCONTVEL[[2]]

Do[
   If[GPALT0 > KCONTALT[[i]],
      ALTMIN = KCONTALT[[i]];
      VELMIN = KCONTVEL[[i]];
      ALTMAX = KCONTALT[[i+1]];
      VELMAX = KCONTVEL[[i+1]];
   ], {i, 17}]

(* compute the optimal_velocity *)

OPTVEL = GPVEL0[[1,1]]

If[ALTMIN != ALTMAX,
  SLOPE = N[(VELMAX - VELMIN)/(ALTMAX - ALTMIN), 30];
  OPTVEL = N[SLOPE*(GPALT0 - ALTMIN) + VELMIN, 30]
]

(* Print [StringForm["OPTVEL = ``", OPTVEL]] *)

(* compute VELOCITY_ ERROR *)

DUM = N[GPVEL0[[1,1]] - OPTVEL, 30]
VELERR = N[DUM, 30]

(* CONT_CROSSED = 0, Contour not crossed *)
(* CONT_CROSSED = 1, Contour crossed     *)

If[GPALT0 <= ENGONALT && CONTCROSSED == 0 && VELERR >= 0.,
   CONTCROSSED = 1]

(* Determine GP_PHASE                    *)
```

```
(* AE_SWITCH = 0, Engins OFF             *)
(* AE_SWITCH = 1, Engins ON              *)
(* TD_SENSED = 0, TD NOT SENSED          *)
(* TD_SENSED = 1, TD SENSED              *)
(* CONT_CROSSED = 0, Contour not crossed *)
(* CONT_CROSSED = 1, Contour crossed     *)
(* CHUTE_REL = 0, Chute attached         *)
(* CHUTE_REL = 1, Chute released         *)

(* PHASE 2 *)

If[GPPHASE == 1 && GPALT0 <= ENGONALT
  ,GPPHASE = 2]

(*If[GPPHASE == 1, idum = 1]                           *)
(*If[GPALT0 <= ENGONALT && idum == 1, idum = 2]             *)
(*Print [StringForm["idum = ``",idum]]                     *)
(*If[idum == 2, GPPHASE = 2]                           *)

(* PHASE 3 *)

If[GPPHASE == 2 && CHUTEREL ==1
                  && AETEMP == 2
  ,GPPHASE = 3]

If[GPPHASE == 2 && TDSENSED == 1
  ,GPPHASE = 5]

(* PHASE 4 *)

If[GPPHASE == 3 && GPALT0 <= DROPHEIGHT
                  && TDSENSED == 0
                  && TDSSTATUS == 0
                  && X1 <= MAXNORMVEL
  ,GPPHASE = 4]

If[GPPHASE == 3 && GPALT0 <= DROPHEIGHT
                  && TDSSTATUS == 1
  ,GPPHASE = 5]

If[GPPHASE == 3 && TDSENSED == 1
  ,GPPHASE = 5]

If[GPPHASE == 4 && TDSENSED == 1
  ,GPPHASE = 5]

If[GPPHASE == 4 && TDSSTATUS == 1
  ,GPPHASE = 5]

(* Determine the value of CL *)

(* CL = 1: First  *)
(* CL = 2: Second *)

(* The difference has to be used in for comparisons in Mathematica model. *)
```

```
(* because the model uses approximations upto 30 digits. *)
diff = .00000001

od = N[OPTVEL - DROPSPEED, 30]

Print [StringForm["od = ``", od]]
Print [StringForm["GPVEL0[[1,1]] = ``", GPVEL0[[1,1]]]]
Print [StringForm["OPTVEL = ``", OPTVEL]]
Print [StringForm["TEI = ``",TEI]]
Print [StringForm["DROPSPEED = ``", DROPSPEED]]

dum1 = DROPSPEED
dum1 = N[GPVEL0[[1,1]] - dum1, 30]
dum2 = 0.

Print [StringForm["dum1 = ``", dum1]]

If[CL == 1, Print [StringForm["test : cl = 1"]]]
If[(Abs[od] <= diff), Print [StringForm["test : Abs(od) <= diff"]]]
If[dum1 < dum2, Print [StringForm["(GPVEL0[[1,1]] < DROPSPEED)"]]]


If[CL == 1 && Abs[od] <= diff && dum1 < dum2,
   CL = 2;
   TEI = 0.0
]

Print [StringForm["od = ``", od]]
Print [StringForm["GPVEL0[[1,1]] = ``", GPVEL0[[1,1]]]]
Print [StringForm["OPTVEL = ``", OPTVEL]]
Print [StringForm["CL = ``", CL]]
Print [StringForm["TEI = ``",TEI]]
```

GSP

```
(****************************************************************************)
Filename :      gsp.m
Create Date :  6-30-94
Description:
This file contains the Mathematica code to calculate expected values
for GSP functional unit.  The following assumptions are made:
        data related to the 4 GCS data stores are pre-loaded.
        the specific data for a test case is also loaded
(****************************************************************************)

(* Local variables added for readability *)
healthy = 0       (* used for G_STATUS *)
failed  = 1       (* used for G_STATUS *)

(**** Rotate history variables ****)
(* G_ROTATION *)
GROT4 = GROT3
GROT3 = GROT2
GROT2 = GROT1
GROT1 = GROT0

(**** Adjust A_GAIN() for temperature ****)
ggain = { GGAIN0[[1]] + G3 ATMTEMP + G4 ATMTEMP^2     (* x axis *)
,GGAIN0[[2]] + G3 ATMTEMP + G4 ATMTEMP^2     (* y axis *)
,GGAIN0[[3]] + G3 ATMTEMP + G4 ATMTEMP^2     (* z axis *)
}

(**** Convert G_COUNTER to G_ROTATION ****)
For[ i=1, i<=3, i++,
If [ (GCOUNTER[[i]] > 0)                    (* Get G_COUNTER sign *)
, sign = 1
, sign = -1
];
counter = sign Mod[GCOUNTER[[i]], 2^14];      (* Get lower 14 bits *)
GROT0[[i]] = GOFFSET[[i]]                    (* Calculate G_ROTATION *)
+ ggain[[i]] counter
]

(**** Set Gyroscope status to healthy ****)
GSTATUS = healthy
```

RECLP

```
(***************************************************************************)
(* Filename : reclp.tc.code                                                *)
(*                                                                         *)
(* Description:                                                            *)
(*                                                                         *)
(* This file contains the Mathematica code to calculate the expected values *)
(* for RECLP.                                                              *)
(* The following assumptions are made:                                     *)
(*     1) the data related to the 4 GCS data stores are pre-loaded         *)
(*     2) the specific data for a test case is also loaded                 *)
(***************************************************************************)


Print [ALPHA]
GROT0[[1]] = GROT

GPALT0 = GPALT[[1]]
GPALT1 = GPALT[[2]]
GPALT2 = GPALT[[3]]
GPALT3 = GPALT[[4]]
GPALT4 = GPALT[[5]]

(* compute  the new value of THETA *)

DG =N[DELT*GROT]
THETA = N[THETA + DG]

(* check for all areas *)

If[Abs[THETA] <= THETA1, ITH = 1, ITH = 0]
If[Abs[THETA] <= THETA2 && Abs[THETA] > THETA1, ITH = 2]
If[Abs[GROT] <= P1, IP = 1, IP = 0]
If[Abs[GROT] <= P2 && Abs[GROT] > P1, IP = 2]
If[Abs[GROT] <= P3 && Abs[GROT] > P2, IP = 3]
If[Abs[GROT] <= P4 && Abs[GROT] > P3, IP = 4]

If[GROT > 0. && IP == 1 && THETA > 0. && ITH == 1, IROLL = 1, IROLL = 0]
If[GROT > 0. && IP == 1 && THETA > 0. && ITH == 2, IROLL = 3]
If[GROT > 0. && IP == 1 && THETA > 0. && ITH == 0, IROLL = 7]
If[GROT > 0. && IP <= 3 && THETA < 0. && ITH == 0, IROLL = 6]
If[GROT > 0. && IP <= 3 && THETA < 0. && ITH != 0, IROLL = 1]

If[GROT > 0. && IP == 2 && THETA > 0. && ITH != 0, IROLL = 5]
If[GROT > 0. && IP == 2 && THETA > 0. && ITH == 0, IROLL = 7]

If[GROT > 0. && IP >= 3 && THETA > 0., IROLL = 7]

If[GROT > 0. && IP == 4 && THETA <= 0., IROLL = 1]

If[GROT > 0. && IP == 0, IROLL = 7]

If[GROT < 0. && IP <= 3 && THETA > 0. && ITH != 0, IROLL = 1]
```

```
If[GROT < 0. && IP <= 3 && THETA > 0. && ITH == 0, IROLL = 7]
If[GROT < 0. && IP == 1 && THETA < 0. && ITH == 1, IROLL = 1]
If[GROT < 0. && IP == 1 && THETA < 0. && ITH == 2, IROLL = 2]
If[GROT < 0. && IP == 1 && THETA < 0. && ITH == 0, IROLL = 6]

If[GROT < 0. && IP == 2 && THETA < 0. && ITH <= 2, IROLL = 4]
If[GROT < 0. && IP == 2 && THETA < 0. && ITH == 0, IROLL = 6]

If[GROT < 0. && IP >= 3 && THETA < 0., IROLL = 6]

If[GROT < 0. && IP == 4 && THETA > 0., IROLL = 1]

If[GROT < 0. && IP == 0, IROLL = 6]

If[THETA == 0. && IP != 0, IROLL = 1]
If[THETA == 0. && IP == 0 && GROT > 0., IROLL = 7]
If[THETA == 0. && IP == 0 && GROT < 0., IROLL = 6]

If[GROT == 0. && Abs[THETA] <= THETA2, IROLL = 1]
If[GROT == 0. && THETA > THETA2, IROLL = 7]
If[GROT == 0. && THETA < -THETA2, IROLL = 6]
```

TDLRSP

```
(***************************************************************************
 Filename :     tdlrsp.m
 Create Date :  6-30-94
 Description:
    This file contains the Mathematica code to calculate expected values
    for TDLRSP functional unit.  The following assumptions are made:
        1) data related to the 4 GCS data stores are pre-loaded.
        2) the specific data for a test case is also loaded
 History:
    6-30-94  V0  created
    3-30-95  V1  Removed the use of KonAxisOK varaible
***************************************************************************)
debug = 1    (* debug prints 1=on 0=0ff *)

(* Local variables added for readability *)
   healthy = 0       (* used for TDLR_STATUS *)
   failed  = 1       (* used for TDLR_STATUS *)
   unlocked = 0      (* used for TDLR_STATE *)
   locked   = 1      (* used for TDLR_STATE *)
   good     = 1      (* used for deciding whether KonAxis is OK *)
   bad      = 0      (* used for deciding whether KonAxis is OK *)

(**** Rotate history variables ****)
   (* TDLR_VELOCITY *)
   TDLRVEL4 = TDLRVEL3
   TDLRVEL3 = TDLRVEL2
   TDLRVEL2 = TDLRVEL1
   TDLRVEL1 = TDLRVEL0

   (* K_MATRIX *)
   KMATRIX4 = KMATRIX3
   KMATRIX3 = KMATRIX2
   KMATRIX2 = KMATRIX1
   KMATRIX1 = KMATRIX0

If [ debug==1, Print["starting k_matrix = ",MatrixForm[KMATRIX0] ] ]

(**** Determine radar beam status ****)
If [debug==1, Print["--- evaluate beam ---"] ]
For [ i=1, i<=4, i++,
      If [debug==1, Print["TDLRSTATE[",i,"] = ",TDLRSTATE[[i]] ] ];
      If [debug==1, Print["TDLRCOUNT[",i,"] = ",TDLRCOUNT[[i]] ] ];
      If [debug==1, Print["FRBUNLOCK[",i,"] = ",FRBUNLOCK[[i]] ] ];
      If [debug==1, Print["FRAMECOUNTER = ", FRAMECOUNTER  ] ];
      Which[
            (* Row 1 of table 5.11 *)
                 (TDLRSTATE[[i]] == locked)
               && (TDLRCOUNT[[i]] == 0)
               , TDLRSTATE[[i]] = unlocked;
                 FRBUNLOCK[[i]] = FRAMECOUNTER;
                 If [debug==1, Print["Table 5.11 Row 1"] ]
            (* Row 2 of table 5.11 *)
```

```
            ,  (TDLRSTATE[[i]] == unlocked)
            && (TDLRCOUNT[[i]] != 0)
            && ((DELTAT (FRAMECOUNTER - FRBUNLOCK[[i]])) >= TDLRLT)
            , TDLRSTATE[[i]] = locked;
              If [debug==1, Print["Table 5.11 Row 2"] ]
          (* Row 3 of table 5.11 *)
            ,  (TDLRSTATE[[i]] == unlocked)
            && (TDLRCOUNT[[i]] == 0)
            && ((DELTAT (FRAMECOUNTER - FRBUNLOCK[[i]])) >= TDLRLT)
          , FRBUNLOCK[[i]] = FRAMECOUNTER;
            If [debug==1, Print["Table 5.11 Row 3"] ]
      ];
     If [ debug==1, Print["Frame_beam_unlocked[",i,"] = ",FRBUNLOCK[[i]] ]]
   ]

If [ debug==1, Print["at 2 k_matrix = ",MatrixForm[KMATRIX0] ] ]
(**** Determine beam velocity ****)
B = { N[(TDLROFF + TDLRGAIN TDLRCOUNT[[1]]),30]
    ,N[(TDLROFF + TDLRGAIN TDLRCOUNT[[2]]),30]
    ,N[(TDLROFF + TDLRGAIN TDLRCOUNT[[3]]),30]
    ,N[(TDLROFF + TDLRGAIN TDLRCOUNT[[4]]),30]
    }
If [ debug==1, Print["B = ",B] ]

If [ debug==1, Print["at 3 k_matrix = ",MatrixForm[KMATRIX0] ] ]
(**** Process the beam velocities ****)
  (* NOTE: In Mathematica, the WHICH statement works like a Pascal CASE *)
BonAxis = { 0, 0, 0 }  (* case where none or only 1 beam is locked  *)
KonAxis = { 0, 0, 0 }
Which[ TDLRSTATE[[1]] == TDLRSTATE[[2]] == TDLRSTATE[[3]]
          == TDLRSTATE[[4]] == locked
            , N[ BonAxis[[1]] = ( B[[1]] + B[[2]] + B[[3]] + B[[4]] )/4, 30];
              N[ BonAxis[[2]] = ( B[[1]] - B[[2]] - B[[3]] + B[[4]] )/4, 30];
              N[ BonAxis[[3]] = ( B[[1]] + B[[2]] - B[[3]] - B[[4]] )/4, 30];
              KonAxis = { 1, 1, 1 };
(* V1         KonAxisOK = good;  *)
              If [ debug==1, Print["Row 16"] ]
     ,TDLRSTATE[[2]] == TDLRSTATE[[3]] == TDLRSTATE[[4]] == locked
            , N[ BonAxis[[1]] = ( B[[2]] + B[[4]] )/2, 30];
              N[ BonAxis[[2]] = ( B[[4]] - B[[3]] )/2, 30];
              N[ BonAxis[[3]] = ( B[[2]] - B[[3]] )/2, 30];
              KonAxis = { 1, 1, 1 };
(* V1         KonAxisOK = good; *)
              If [ debug==1, Print["Row 15"] ]
     ,TDLRSTATE[[1]] == TDLRSTATE[[3]] == TDLRSTATE[[4]] == locked
            , N[ BonAxis[[1]] = ( B[[1]] + B[[3]] )/2, 30];
              N[ BonAxis[[2]] = ( B[[4]] - B[[3]] )/2, 30];
              N[ BonAxis[[3]] = ( B[[1]] - B[[4]] )/2, 30];
              KonAxis = { 1, 1, 1 };
(* V1         KonAxisOK = good; *)
              If [ debug==1, Print["Row 14"] ]
     ,TDLRSTATE[[1]] == TDLRSTATE[[2]] == TDLRSTATE[[4]] == locked
            , N[ BonAxis[[1]] = ( B[[2]] + B[[4]] )/2, 30];
              N[ BonAxis[[2]] = ( B[[1]] - B[[2]] )/2, 30];
              N[ BonAxis[[3]] = ( B[[1]] - B[[4]] )/2, 30];
              KonAxis = { 1, 1, 1 };
```

```
(* V1          KonAxisOK = good; *)
               If [ debug==1, Print["Row 13"] ]
     ,TDLRSTATE[[1]] == TDLRSTATE[[2]] == TDLRSTATE[[3]] == locked
              , N[ BonAxis[[1]] = ( B[[1]] + B[[3]] )/2, 30];
               N[ BonAxis[[2]] = ( B[[1]] - B[[2]] )/2, 30];
               N[ BonAxis[[3]] = ( B[[2]] - B[[3]] )/2, 30];
               KonAxis = { 1, 1, 1 };
(* V1          KonAxisOK = good; *)
               If [ debug==1, Print["Row 12"] ]
     ,TDLRSTATE[[3]] == TDLRSTATE[[4]] == locked
              , N[ BonAxis[[2]] = ( B[[4]] - B[[3]] )/2, 30];
               KonAxis = { 0, 1, 0 };
(* V1          KonAxisOK = good; *)
               If [ debug==1, Print["Row 11"] ]
     ,TDLRSTATE[[2]] == TDLRSTATE[[4]] == locked
              , N[ BonAxis[[1]] = ( B[[2]] + B[[4]] )/2, 30];
               KonAxis = { 1, 0, 0 };
(* V1          KonAxisOK = good; *)
               If [ debug==1, Print["Row 10"] ]
     ,TDLRSTATE[[2]] == TDLRSTATE[[3]] == locked
              , N[ BonAxis[[3]] = ( B[[2]] - B[[3]] )/2, 30];
               KonAxis = { 0, 0, 1 };
(* V1          KonAxisOK = good; *)
               If [ debug==1, Print["Row 9"] ]
     ,TDLRSTATE[[1]] == TDLRSTATE[[4]] == locked
              , N[ BonAxis[[3]] = ( B[[1]] - B[[4]] )/2, 30];
               KonAxis = { 0, 0, 1 };
(* V1          KonAxisOK = good; *)
               If [ debug==1, Print["Row 8"] ]
     ,TDLRSTATE[[1]] == TDLRSTATE[[3]] == locked
              , N[ BonAxis[[1]] = ( B[[1]] + B[[3]] )/2, 30];
               KonAxis = { 1, 0, 0 };
(* V1          KonAxisOK = good; *)
               If [ debug==1, Print["Row 7"] ]
     ,TDLRSTATE[[1]] == TDLRSTATE[[2]] == locked
              , N[ BonAxis[[2]] = ( B[[1]] - B[[2]] )/2, 30];
               KonAxis = { 0, 1, 0 };
(* V1          KonAxisOK = good; *)
               If [ debug==1, Print["Row 6"] ]
     ]

If [ debug==1, Print["at 4 k_matrix = ",MatrixForm[KMATRIX0] ] ]
(**** Convert to body velocities ****)
For[ i=1, i<=3, i++,                    (* RAD angles *)
    TDLRVEL0[[i,1]] = N[(BonAxis[[i]] 1/N[ Cos[TDLRANG[[i]]],30]),30]
   ]
If [ debug==1, Print["tdlr_velocity = ",TDLRVEL0] ]

(**** Set values in K_MATRIX ****)
(*****************  old code from V0  **************)
(*If [KonAxisOK                              *)
(*     , KMATRIX0 = { {0,0,0}, {0,0,0}, {0,0,0} }; *)(* initialize K_MATRIX *)
(*       KMATRIX0[[1,1]] = KonAxis[[1]];       *)
(*          KMATRIX0[[2,2]] = KonAxis[[2]];                          *)
(*          KMATRIX0[[3,3]] = KonAxis[[3]]                  *)
(*   ]                                                      *)
```

```
(*---------------- new code for V1  ----------------*)
If [ debug==1, Print["KonAxis = ",KonAxis] ]
KMATRIX0 = { {0,0,0}, {0,0,0}, {0,0,0} }  (* initialize K_MATRIX *)
KMATRIX0[[1,1]] = KonAxis[[1]]
KMATRIX0[[2,2]] = KonAxis[[2]]
KMATRIX0[[3,3]] = KonAxis[[3]]

(***********************************************)

If [ debug==1, Print["k_matrix = ",MatrixForm[KMATRIX0] ] ]

(**** Set TDLR_STATUS to healthy ****)
For[ i=1, i<=4, i++,
    TDLRSTATUS[[i]] = healthy
  ]
```

TDSP

```
(***************************************************************************
Filename :     tdspsp.m
Create Date :  7-5-94
Description:
This file contains the Mathematica code to calculate expected values
for TDSP functional unit.  The following assumptions are made:
        data related to the 4 GCS data stores are pre-loaded.
        the specific data for a test case is also loaded
***************************************************************************)

(* Local variables added for readability *)
healthy = 0       (* used for TDS_STATUS *)
failed  = 1       (* used for TDS_STATUS *)
sensed     = 1    (* used for TD_SENSED *)
notsensed  = 0    (* used for TD_SENSED *)
allzeros = 0      (* used for TD_COUTNER *)
allones  = 65536  (* used for TD_COUTNER *)

(**** Determine status of touch down sensor & whether it has been sensed ****)
If[ (TDSSTATUS == healthy)
, Switch [ TDCOUTNER
,allzeros, TDSENSED = notsensed
,allones,  TDSENSED = sensed
,_, TDSENSED = notsensed;
TDSSTATUS = failed
]
, (* according to the Spec:
if TDS_STATUS failes, GP determins when touch down occures *)
]
```

TSP

```
(**************************************************************************
Filename :     tsp.m
Create Date :  7-5-94
Description:
This file contains the Mathematica code to calculate expected values
for TSP functional unit.  The following assumptions are made:
        data related to the 4 GCS data stores are pre-loaded.
        the specific data for a test case is also loaded
**************************************************************************)

(* Local variables added for readability *)
healthy = 0       (* used for TS_STATUS *)
failed  = 1       (* used for TS_STATUS *)

(**** Calculate the solid state temperature ****)
SSslope = (T2 - T1) / (M2 - M1)
SSyint = T1 - (SSslope M1)
sst = (SSslope SSTEMP) + SSyint


(**** Determine upper and lower range of thermocouple temperature ****)
LowerLimit = M3 - ( 0.15 (M4 - M3) )  (* lower bound for valid THERMO_TEMP *)
UpperLimit = M4 + ( 0.15 (M4 - M3) )  (* upper bound for valid THERMO_TEMP *)
THslope = (T4 - T3) / (M4 - M3)       (* THERMO_TEMP  linear range slope *)
THyint = T3 - (THslope M3)
hL = M3 + (THslope/2)
kL = T3 + (THslope/2)^2
LowerParaTemp = - ( LowerLimit - hL )^2 + kL
hU = M4 - (THslope/2)
kU = T4 - (THslope/2)^2
UpperParaTemp =   ( UpperLimit - hU )^2 + kU


(**** Determine which sensor to use, & calculate thermo-temp if necessary ****)
If[ (sst < LowerParaTemp) || (sst > UpperParaTemp)
    , ATMTEMP = sst
    , Which[(THERMOTEMP >= M3) && (THERMOTEMP <= M4)
        ,ATMTEMP = (THslope THERMOTEMP) + THyint
        ,THERMOTEMP < M3
        ,ATMTEMP = - (THERMOTEMP - hL)^2 + kL
        ,THERMOTEMP > M4
        ,ATMTEMP = (THERMOTEMP - hU )^2 + kU
    ]
]


(**** Set both elements of TS_STATUS to healthy ****)
For[ i=1, i<=2, i++,
TSSTATUS[[i]] = healthy
]

(*  debug use only *)
```

```
(*
Print ["sstemp = ",sst]
Print ["UpperParaTemp = ",UpperParaTemp]
Print ["LowerParaTemp = ",LowerParaTemp]
Print ["Atm_Temp = ",ATMTEMP]
*)
```

A.14 Sample Test Case

This section contains an example of a test case input file and an expected values file. Both are generated by *Mathematica* based on the inputs that the Verification Analyst selects for the particular test case. Each of these files are simply a series of FORTRAN namelists that the Test Case Driver will use as input. The full test case consists of a Test case file and an expected-results file with the following naming convention:

Test case input file: *<functional unit name>_<NR or RO>_<a unique number>.TC*

Expected-results file: *<functional unit name>_<NR or RO>_<a unique number>.EX*

Both files are needed to run the test case. The *NR* designation indicates a “normal range” test of all valid values, both input and output. The *RO* designation indicates a “robustness” test case. These include those instances where the input is valid, but an invalid output occurs, as well as invalid input cases. Each “robustness” test case tests only one invalid input, but a single invalid input may produce several invalid outputs.

Note that this is a functional unit test case example only. The test case input files and expected results files for CP are generated on the VAX and not by *Mathematica*. Additionally, the subframe and frame test cases differ in that the expected values of the data element "PACKET" is not generated until the test case is actually executed. The example follows:

Sample Test Case Input

```
*****************************************************
*   File: gp_nr_001.tcNull
*   Date of Mathematica Model Run: 9-7-1994
*   Time of Mathematica Model Run: 8:13:5
*   Description:
*   Tester:  Debbie Taylor (CSC CORP)
*   DATE: July 15, 1994
*   Unit Test for Functional Unit GP
*
*   Test case 1
*   Initial GP Frame
*   All valid inputs
*
*  Tests Equivalence Classes: A_ACCELERATION.1
*                             GP_ALTITUDE.1
*                             GP_ATTITUDE.1
*                             GP_VELOCITY.1
*                             G_ROTATION.1
*                             TDLR_VELOCITY.1
*
*****************************************************
```

```
$RUN_PARAMETERS_NML
A_BIAS = -20., -20., -20.,
A_GAIN_0 = 0.012, 0.012, 0.012,
A_SCALE = 1,
ALPHA_MATRIX =
     0, 0,
     1, 0,
     0, 1,
AR_FREQUENCY = 2.45e9,
COMM_SYNC_PATTERN = -9806,
CONTOUR_ALTITUDE = -0.01,
     0.003048, 0.018288, 0.019, 0.0196, 0.0225,
     0.02617, 0.03648, 0.0506, 0.06855, 0.0903,
     0.14542, 0.21583, 0.30145, 2., 0.,
CONTOUR_VELOCITY = 0.002,
     0.002, 0.002, 0.0031, 0.0035, 0.0046,
     0.00538, 0.01222, 0.0162, 0.0203, 0.0245,
     0.0333, 0.0427, 0.0528, 0.1225, 0.,
DELTA_T = 0.02,
DROP_HEIGHT = 1.,
DROP_SPEED = 2.,
ENGINES_ON_ALTITUDE = 1500.,
FULL_UP_TIME = 5.,
G1 = 6.67e-7,
G2 = 4.e-9,
G3 = 3.e-9,
G4 = 2.22e-11,
G_GAIN_0 = 0.0003, 0.0003, 0.0003,
G_OFFSET = 0., 0., 0.,
GA = 0.01,
GAX = 3.,
GP1 = 0.852,
GP2 = -0.426,
GPY = 0.892,
GQ = 3., 7.,
GR = 3.,7.,
GRAVITY = 3.75,
GV = 5.,7.,
GVE = 200.,
GVEI = 40.,20.,
GW = 5.,7.,
GWI = 0.5,1.,
M1 = 10000.,
M2 = 10040.,
M3 = 1000.,
M4 = 1010.,
MAX_NORMAL_VELOCITY = 3.35,
OMEGA = 1.,
P1 = 0.00354,
P2 = 0.00827,
P3 = 0.01,
P4 = 0.015708,
PE_MAX = 0.524,0.062,
PE_MIN = -0.524,-0.062,
T1 = -200.,
T2 = 200.,
```

```
T3 = -38.46,
T4 = 38.46,
TDLR_ANGLES = 0.361367,1.31812, 1.31812,
TDLR_GAIN = 0.015625,
TDLR_LOCK_TIME = 0,
TDLR_OFFSET = -100.,
TE_DROP = 0.1,
TE_INIT = 0.1,
TE_MAX = 0.9,0.498,
TE_MIN = 0.1,0.1,
THETA1 = 0.004363,
THETA2 = 0.006109,
YE_MAX = 0.524,0.042,
YE_MIN = -0.524,-0.042,
$end

$EXTERNAL_NML
A_COUNTER = 1665, 1524, 1524,
AE_CMD = 0, 0, 0,
AR_COUNTER = 24464,
FRAME_COUNTER = 1,
G_COUNTER = 292, 161, 7,
PACKET =
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
RE_CMD = 1,
SS_TEMP = 0.,
SUBFRAME_COUNTER = 1,
TD_COUNTER = 0,
TDLR_COUNTER = 9920, 9770, 9852, 10002,
THERMO_TEMP = 992,
$end

$SENSOR_OUTPUT_NML
A_ACCELERATION =
1.63739825110955, -0.202263936687537, 1.99439462855677,
1.63739825110955, -0.202263936687537, 1.99439462855677,
1.63739825110955, -0.202263936687537, 1.99439462855677,
1.63739825110955, -0.202263936687537, 1.99439462855677,
1.63739825110955, -0.202263936687537, 1.99439462855677,
AR_ALTITUDE = 1497.79591836735, 1497.81166815946, 1497.81166815946, 1497.81166815946,
1497.81166815946,
ATMOSPHERIC_TEMP = -147.586,
G_ROTATION =
0.087614454, 0.0483079695, 0.0021003465,
```

0.0876579642295837, 0.0485167264938354, 0.00239650011062622,
0.0876579642295837, 0.0485167264938354, 0.00239650011062622,
0.0876579642295837, 0.0485167264938354, 0.00239650011062622,
0.0876579642295837, 0.0485167264938354, 0.00239650011062622,
TD_SENSED = 0,
TDLR_VELOCITY =
58.2295430434468, 4.68750002153857, -2.56250001177442,
58.2371395855674, 58.2371395855674, 58.2371395855674,
58.2371395855674, 58.2371395855674, 58.2371395855674,
58.2371395855674, 58.2371395855674, 58.2371395855674,
58.2371395855674, 58.2371395855674, 58.2371395855674,
$end

$GUIDANCE_STATE_NML
A_STATUS =
    0, 0,
    0, 0,
    0, 0,
    0, 0,
AE_STATUS = 0,
AE_SWITCH = 0,
AE_TEMP = 0,
AR_STATUS = 0, 0, 0, 0, 0,
C_STATUS = 0,
CHUTE_RELEASED = 0,
CL = 1,
CONTOUR_CROSSED = 0,
FRAME_BEAM_UNLOCKED =
    0, 0, 0,
FRAME_ENGINES_IGNITED = 1,
G_STATUS = 0,
GP_ALTITUDE = 1497.81166815946, 1497.81166815946, 1497.81166815946, 1497.81166815946,
1497.81166815946,
GP_ATTITUDE =
-0.0404392391645691, 0.958516512228094, -0.282153794448846,
-0.0747709982983742, 0.278690021002445, 0.957466015066395,
0.99638043223924, 0.0598161180598313, 0.0603992240926737,
-0.0404392391645691, 0.958516512228094, -0.282153794448846,
-0.0747709982983742, 0.278690021002445, 0.957466015066395,
0.99638043223924, 0.0598161180598313, 0.0603992240926737,
-0.0404392391645691, 0.958516512228094, -0.282153794448846,
-0.0747709982983742, 0.278690021002445, 0.957466015066395,
0.99638043223924, 0.0598161180598313, 0.0603992240926737,
-0.0404392391645691, 0.958516512228094, -0.282153794448846,
-0.0747709982983742, 0.278690021002445, 0.957466015066395,
0.99638043223924, 0.0598161180598313, 0.0603992240926737,
-0.0404392391645691, 0.958516512228094, -0.282153794448846,
-0.0747709982983742, 0.278690021002445, 0.957466015066395,
0.99638043223924, 0.0598161180598313, 0.0603992240926737,
GP_PHASE = 1,
GP_ROTATION =
    , -0.00210035, 0.0483079695,
0.0021003465, 0., -0.0876145,
-0.048308, 0.087614454, 0.,
GP_VELOCITY =
58.2371395855674, 4.67148327843904, -2.55368425934921,

58.2371395855674, 4.67148327843904, -2.5536842534921,
58.2371395855674, 4.67148327843904, -2.55368425934921,
58.2371395855674, 4.67148327843904, -2.55368425934921,
58.2371395855674, 4.67148327843904, -2.55368425934921,
INTERNAL_CMD = 0, 0, 0,
K_ALT = 1, 1, 1, 1, 1,
K_MATRIX =
    , 0., 0.,
    , 1., 0.,
    , 0., 1.,
    , 0., 0.,
    , 1., 0.,
    , 0., 1.,
    , 0., 0.,
    , 1., 0.,
    , 0., 1.,
    , 0., 0.,
    , 1., 0.,
    , 0., 1.,
    , 0., 0.,
    , 1., 0.,
    , 0., 1.,
PE_INTEGRAL = 0.,
RE_STATUS = 0,
RE_SWITCH = 1,
TDLR_STATE = 1, 1, 1, 1,
TDLR_STATUS = 0, 0, 0, 0,
TDS_STATUS = 0,
TE_INTEGRAL = 0.,
TE_LIMIT = 0.,
THETA = 0.00257,
TS_STATUS = 0, 0,
VELOCITY_ERROR = -43.348030923942914,
YE_INTEGRAL = 0.,
$END

Sample Expected Results

```
******************************************************
*   File: gp_nr_001.exNull
*   Date of Mathematica Model Run: 9-7-1994
*   Time of Mathematica Model Run: 8:13:10
*   Description:
*   Tester:  Debbie Taylor (CSC CORP)
*   DATE: July 15, 1994
*   Unit Test for Functional Unit GP
*
*   Test case 1
*   Initial GP Frame
*   All valid inputs
*
*  Tests Equivalence Classes: A_ACCELERATION.1
*                          GP_ALTITUDE.1
*                          GP_ATTITUDE.1
*                          GP_VELOCITY.1
*                          G_ROTATION.1
*                          TDLR_VELOCITY.1
*
******************************************************


$EX_RUN_PARAMETERS_NML
EX_A_BIAS = -20., -20., -20.,
EX_A_GAIN_0 = 0.012, 0.012, 0.012,
EX_A_SCALE = 1,
EX_ALPHA_MATRIX =
     0, 0,
     1, 0,
     0, 1,
EX_AR_FREQUENCY = 2.45e9,
EX_COMM_SYNC_PATTERN = -9806,
EX_CONTOUR_ALTITUDE = -10.,
     3.048, 18.288, 19., 19.6, 22.5,
     26.17, 36.48, 50.6, 68.55, 90.3,
     , 145.42, 215.83, 301.45, 2000., 0.,
EX_CONTOUR_VELOCITY = 2.,
     , 2., 2., 3.1, 3.5, 4.6,
     5.38, 12.22, 16.2, 20.3, 24.5,
     33.3, 42.7, 52.8, 122.5, 0.,
EX_DELTA_T = 0.02,
EX_DROP_HEIGHT = 1.,
EX_DROP_SPEED = 2.,
EX_ENGINES_ON_ALTITUDE = 1500.,
EX_FULL_UP_TIME = 5.,
EX_G1 = 6.67e-7,
EX_G2 = 4.e-9,
EX_G3 = 3.e-9,
EX_G4 = 2.22e-11,
EX_G_GAIN_0 = 0.0003, 0.0003, 0.0003,
EX_G_OFFSET = 0., 0., 0.,
```

```
EX_GA = 0.01,
EX_GAX = 3.,
EX_GP1 = 0.852,
EX_GP2 = -0.426,
EX_GPY = 0.892,
EX_GQ = 3., 7.,
EX_GR = 3.,7.,
EX_GRAVITY = 3.75,
EX_GV = 5.,7.,
EX_GVE = 200.,
EX_GVEI = 40.,20.,
EX_GW = 5.,7.,
EX_GWI = 0.5,1.,
EX_M1 = 10000.,
EX_M2 = 10040.,
EX_M3 = 1000.,
EX_M4 = 1010.,
EX_MAX_NORMAL_VELOCITY = 3.35,
EX_OMEGA = 1.,
EX_P1 = 0.00354,
EX_P2 = 0.00827,
EX_P3 = 0.01,
EX_P4 = 0.015708,
EX_PE_MAX = 0.524,0.062,
EX_PE_MIN = -0.524,-0.062,
EX_T1 = -200.,
EX_T2 = 200.,
EX_T3 = -38.46,
EX_T4 = 38.46,
EX_TDLR_ANGLES = 0.361367,1.31812, 1.31812,
EX_TDLR_GAIN = 0.015625,
EX_TDLR_LOCK_TIME = 0,
EX_TDLR_OFFSET = -100.,
EX_TE_DROP = 0.1,
EX_TE_INIT = 0.1,
EX_TE_MAX = 0.9,0.498,
EX_TE_MIN = 0.1,0.1,
EX_THETA1 = 0.004363,
EX_THETA2 = 0.006109,
EX_YE_MAX = 0.524,0.042,
EX_YE_MIN = -0.524,-0.042,
$end

$EX_EXTERNAL_NML
EX_A_COUNTER = 1665, 1524, 1524,
EX_AE_CMD = 0, 0, 0,
EX_AR_COUNTER = 24464,
EX_FRAME_COUNTER = 1,
EX_G_COUNTER = 292, 161, 7,
EX_PACKET =
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
EX_RE_CMD = 1,
EX_SS_TEMP = 0.,
EX_SUBFRAME_COUNTER = 1,
EX_TD_COUNTER = 0,
EX_TDLR_COUNTER = 9920, 9770, 9852, 10002,
EX_THERMO_TEMP = 992,
$end

$EX_SENSOR_OUTPUT_NML
EX_A_ACCELERATION =
1.63739825110955, -0.202263936687537, 1.99439462855677,
1.63739825110955, -0.202263936687537, 1.99439462855677,
1.63739825110955, -0.202263936687537, 1.99439462855677,
1.63739825110955, -0.202263936687537, 1.99439462855677,
1.63739825110955, -0.202263936687537, 1.99439462855677,
EX_AR_ALTITUDE = 1497.79591836735, 1497.81166815946, 1497.81166815946, 1497.81166815946,
1497.81166815946,
EX_ATMOSPHERIC_TEMP = -147.586,
EX_G_ROTATION =
0.087614454, 0.0483079695, 0.0021003465,
0.0876579642295837, 0.0485167264938354, 0.00239650011062622,
0.0876579642295837, 0.0485167264938354, 0.00239650011062622,
0.0876579642295837, 0.0485167264938354, 0.00239650011062622,
0.0876579642295837, 0.0485167264938354, 0.00239650011062622,
EX_TD_SENSED = 0,
EX_TDLR_VELOCITY =
58.2295430434468, 4.68750002153857, -2.56250001177442,
58.2371395855674, 58.2371395855674, 58.2371395855674,
58.2371395855674, 58.2371395855674, 58.2371395855674,
58.2371395855674, 58.2371395855674, 58.2371395855674,
58.2371395855674, 58.2371395855674, 58.2371395855674,
$end

$EX_GUIDANCE_STATE_NML
EX_A_STATUS =
    0, 0,
    0, 0,
    0, 0,
    0, 0,
EX_AE_STATUS = 0,
EX_AE_SWITCH = 1,
EX_AE_TEMP = 0,
EX_AR_STATUS = 0, 0, 0, 0, 0,
EX_C_STATUS = 0,
EX_CHUTE_RELEASED = 0,
EX_CL = 1,
EX_CONTOUR_CROSSED = 0,
EX_FRAME_BEAM_UNLOCKED =
    0, 0, 0,
```

EX_FRAME_ENGINES_IGNITED = 1,
EX_G_STATUS = 0,
EX_GP_ALTITUDE = 1495.521749022006, 1497.81166815946, 1497.81166815946, 1497.81166815946, 1497.81166815946,
EX_GP_ATTITUDE =
-0.03979878822126675, 0.957525014194865, -0.2855904474112915,
-0.07660037852440182, 0.282052052010574, 0.956336249426609,
0.99626725253411, 0.05993736023322743, 0.06212144864759948,
-0.0404392391645691, 0.958516512228094, -0.282153794448846,
-0.0747709982983742, 0.278690021002445, 0.957466015066395,
0.99638043223924, 0.059816118059831 3, 0.0603992240926737,
-0.0404392391645691, 0.958516512228094, -0.282153794448846,
-0.0747709982983742, 0.278690021002445, 0.957466015066395,
0.99638043223924, 0.0598161180598313, 0.0603992240926737,
-0.0404392391645691, 0.958516512228094, -0.282153794448846,
-0.0747709982983742, 0.278690021002445, 0.957466015066395,
0.99638043223924, 0.0598161180598313, 0.0603992240926737,
-0.0404392391645691, 0.958516512228094, -0.282153794448846,
-0.0747709982983742, 0.278690021002445, 0.957466015066395,
0.99638043223924, 0.0598161180598313, 0.0603992240926737,
EX_GP_PHASE = 2,
EX_GP_ROTATION =
, -0.00210035, 0.0483079695,
0.0021003465, 0., -0.0876145,
-0.048308, 0.087614454, 0.,
EX_GP_VELOCITY =
58.45070383441093, 6.40601755948299, -0.3969432260435215,
58.2371395855674, 4.67148327843904, -2.55368425934921,
58.2371395855674, 4.67148327843904, -2.5536842534921,
58.2371395855674, 4.67148327843904, -2.55368425934921,
58.2371395855674, 4.67148327843904, -2.55368425934921,
EX_INTERNAL_CMD = 0, 0, 0,
EX_K_ALT = 1, 1, 1, 1, 1,
EX_K_MATRIX =
, 0., 0.,
, 1., 0.,
, 0., 1.,
, 0., 0.,
, 1., 0.,
, 0., 1.,
, 0., 0.,
, 1., 0.,
, 0., 1.,
, 0., 0.,
, 1., 0.,
, 0., 1.,
, 0., 0.,
, 1., 0.,
, 0., 1.,
EX_PE_INTEGRAL = 0.,
EX_RE_STATUS = 0,
EX_RE_SWITCH = 1,
EX_TDLR_STATE = 1, 1, 1, 1,
EX_TDLR_STATUS = 0, 0, 0, 0,
EX_TDS_STATUS = 0,
EX_TE_INTEGRAL = 0.,

```
EX_TE_LIMIT = 0.,
EX_THETA = 0.00257,
EX_TS_STATUS = 0, 0,
EX_VELOCITY_ERROR = -43.34803091395316,
EX_YE_INTEGRAL = 0.,
$END
```

A.15 Sample Test Stub

The Test stubs are simply FORTRAN shells that will call the source code for each functional unit. These shells are compiled and linked with the source code provided by the programmer. The resulting executable code is then run at least once for each test case. The drivers compare the data in the expected-results files to the actual data computed by the source code and prints out a file that prints the discrepancies.

```
C*************************************************************************
C
C  NAME:  test_gp.for
C
C  DATE:  12/29/94
C
C  PURPOSE:  Generic test driver for GCS Guidance Processing
C            module. Reads in a test case data file, *.TC, executes
C            the module to be tested, and compares the actual computed data to
C        the expected data in file, *.EX
C
C*************************************************************************
        program test_gp

        include 'struct.for_inc'
        include 'commons.for_inc/nolist'
C
C      List of module inputs

      namelist /EXTERNAL_NML/
     +  A_COUNTER, AE_CMD, AR_COUNTER, FRAME_COUNTER,
     +  G_COUNTER,PACKET, RE_CMD,SS_TEMP,SUBFRAME_COUNTER,
     +  TD_COUNTER, TDLR_COUNTER, THERMO_TEMP
C
      namelist /SENSOR_OUTPUT_NML/
     +  A_ACCELERATION, AR_ALTITUDE, ATMOSPHERIC_TEMP, G_ROTATION,
     +  TD_SENSED, TDLR_VELOCITY
C
      namelist /GUIDANCE_STATE_NML/
     +  A_STATUS, AE_STATUS, AE_SWITCH, AE_TEMP, AR_STATUS,
     +  C_STATUS, CHUTE_RELEASED, CL, CONTOUR_CROSSED,
     +  FRAME_BEAM_UNLOCKED, FRAME_ENGINES_IGNITED,
     +  G_STATUS, GP_ALTITUDE, GP_ATTITUDE, GP_PHASE,
     +  GP_ROTATION, GP_VELOCITY, INTERNAL_CMD, K_ALT,
     +  K_MATRIX, PE_INTEGRAL, RE_STATUS, RE_SWITCH, TDLR_STATE,
```

```
     + TDLR_STATUS, TDS_STATUS, TE_INTEGRAL, TE_LIMIT, THETA,
     + TS_STATUS, VELOCITY_ERROR, YE_INTEGRAL
c
      namelist /RUN_PARAMETERS_NML/
     + A_BIAS, A_GAIN_0, A_SCALE, ALPHA_MATRIX, AR_FREQUENCY,
     + COMM_SYNC_PATTERN, CONTOUR_ALTITUDE, CONTOUR_VELOCITY,
     + DELTA_T, DROP_HEIGHT, DROP_SPEED, ENGINES_ON_ALTITUDE,
     + FULL_UP_TIME, G1, G2, G3, G4, G_GAIN_0, G_OFFSET, GA,
     + GAX, GP1, GP2, GPY, GQ, GR, GRAVITY, GV, GVE, GVEI, GVI,
     + GW, GWI, M1, M2, M3, M4, MAX_NORMAL_VELOCITY, OMEGA, P1,
     + P2, P3, P4, PE_MAX, PE_MIN, T1, T2, T3, T4, TDLR_ANGLES,
     + TDLR_GAIN, TDLR_LOCK_TIME, TDLR_OFFSET, TE_DROP, TE_INIT,
     + TE_MAX, TE_MIN, THETA1, THETA2, YE_MAX, YE_MIN

      namelist /EX_EXTERNAL_NML/
     + EX_A_COUNTER, EX_AE_CMD, EX_AR_COUNTER, EX_FRAME_COUNTER,
     + EX_G_COUNTER, EX_PACKET, EX_RE_CMD, EX_SS_TEMP,
     + EX_SUBFRAME_COUNTER,
     + EX_TD_COUNTER, EX_TDLR_COUNTER, EX_THERMO_TEMP
C
      namelist /EX_SENSOR_OUTPUT_NML/
     + EX_ A_ACCELERATION, EX_AR_ALTITUDE, EX_ATMOSPHERIC_TEMP,
     + EX_G_ROTATION,
     + EX_TD_SENSED, EX_TDLR_VELOCITY
C
      namelist /EX_GUIDANCE_STATE_NML/
     + EX_A_STATUS, EX_AE_STATUS, EX_AE_SWITCH, EX_AE_TEMP,
     + EX_AR_STATUS,
     + EX_C_STATUS, EX_CHUTE_RELEASED, EX_CL, EX_CONTOUR_CROSSED,
     + EX_FRAME_BEAM_UNLOCKED, EX_FRAME_ENGINES_IGNITED,
     + EX_G_STATUS, EX_GP_ALTITUDE, EX_GP_ATTITUDE, EX_GP_PHASE,
     + EX_GP_ROTATION, EX_GP_VELOCITY, EX_INTERNAL_CMD, EX_K_ALT,
     + EX_K_MATRIX, EX_PE_INTEGRAL, EX_RE_STATUS, EX_RE_SWITCH,
     + EX_TDLR_STATE,
     + EX_TDLR_STATUS, EX_TDS_STATUS, EX_TE_INTEGRAL, EX_TE_LIMIT,
     + EX_THETA,
     + EX_TS_STATUS, EX_VELOCITY_ERROR, EX_YE_INTEGRAL
c

C****   Begin execution

C  Read in test case data
        call read_tc

C  Execute gp
        type *, 'executing gp...'
        call gp

C  Read in the expected results from the appropriate .EX file
        call read_ex

C  Compare the expected results with the actual results
        type *, 'compare_guid...'
        call compare_guidance
        type *, 'compare_sensor...'
      call compare_sensor
```

```
      type *, 'compare_runparam...'
   call compare_runpram
      type *, 'compare_extern...'
   call compare_external

C****   end execution
        end
```

A.16 Test Case Results Log

Test Case Results Log

TEST CASE NAME	EXECUTION DATE	CODE VERSION #	TEST CASE VERSION #	RESULTS (was .ANA file generated Y or N?)	PR #

This log will trace the results of each implementation's test runs. It serves as a history of test cases executions for each implementation. Due to the large number of test cases, grouping them logically is highly recommended. For example the Test Case Results Logs will be broken up into 15 different logs; one for each functional unit test suite, one for each subframe test suite and one for the frame test suite. The title of the log will be modified to indicate which test suite and which implementation is being logged. For example the Test Case Log for Mercury for AECLP would be titled : *MERCURY TEST CASE RESULTS LOG FOR AECLP*.

Each of the fields in the log are described below:

TEST CASE NAME:	The name of the test case being logged
DATE:	The date the test case was run. This is used to distinguish between multiple runs of the same test case.
CODE UNIT VERSION #:	The version of the code being tested. This is be used to distinguish between multiple runs of the same test case.
TEST CASE VERSION #:	The version of the test case being tested. This is be used to distinguish between multiple runs of the same test case.
RESULTS:	Was a .ANA file generated? If yes, a PR must be issued.
PR #:	The PR number generated as a result of a test failure.

A.17 References

A.1 Finelli, George B.: Results of Software Error-Data Experiments. In *AIAA/AHS/ASEE Aircraft Design, Systems and Operations Conference,* Atlanta, GA, September 1988.

A.2 "Software Considerations in Airborne Systems and Equipment Certification", Document No. RTCA/DO-178B, Dec. 1992.

A.3 "Technical Assessment Procedure for Design Review and Assessment", SEES document volume III.

A.4 Fagan, Michael E., "Design and Code Inspections to Reduce Errors in Program Development", IBM Systems Journal, Volume 15, No. 3, 1976.

A.5 Withers, B. Edward, *GCS_SIM User's Guide Guidance Control Software Release 1*, Research Triangle Institute.

A.6 Holmberg, Neil A. et al, *Viking '75 Spacecraft Design and Test Summary, Vol. I - Lander Design*, NASA Reference Publication 1027.

A.7 Wolfram, Stephen,. *Mathematica, A System for Doing Mathematics by Computer, Second Edition*. Addison-Wesley Publishing Company, Inc., 1991

A.8 Myers, Glenford J., *The Art of Software Testing,* ,Wiley-Interscience Pub. N.Y., N.Y., 1979.

Appendix B: Software Verification Results for the PLUTO Implementation of the Guidance and Control Software

Authors: Cuong C. Quach, NASA Langley Research Center

This document was produced as part of Guidance and Control Software (GCS) Project conducted at NASA Langley Research Center. Although some of the requirements for the Guidance and Control Software application were derived from the NASA Viking Mission to Mars, this document does not contain data from an actual NASA mission.

B. Contents

B. List of Tables

B.1 Introduction

The purpose of this document, as described in Section 11.14 of DO-178B, is to provide details about the results of software verification activities conducted for the PLUTO implementation of the Guidance and Control Software (GCS). As stated in other documents, the GCS project adheres to the DO-178B guidelines for Level A software. Accordingly, specific verification activities have been described in the *Software Verification Plan*, and *Software Verification Cases and Procedures* documents. This document gives the results of each of those activities as carried out on the Pluto implementation.

As stated in the *Software Verification Plan*, verification activities conducted for Pluto encompass two groups:

- Review and analysis of artifacts from the Design and Coding processes
- Development and execution of test cases

The review and analysis of the Pluto design and source code are performed following the procedure established in the *Software Verification Cases and Procedures* document. Test case development as well as test case execution are also performed in accordance with procedures described in that document. The three sections below are the main thrust of this document and describe the design review, code review, and test case execution results.

B.2. Review and Analysis Results

B.2.1 Design Review

Two reviews were held for the Pluto design. The first occurred between September 16, 1993 and October 15, 1993. Problem Reports (PR) 1 through 13 were issued based on deficiencies found during this review. Before the second review, a modification to the specification (Spec. Mod. 2.3-2) necessitated issuance of PR 14. On July 1, 1994 an overview meeting was held for the Pluto design. The second design review was held twelve days later on July 13, 1994. This review culminated with the issuance of PRs 15 through 19. During this review, the design portion of the Traceability Matrix for Pluto given in section B.5 was completed. Shortly there after, PR 20 was issued due to another change to the GCS Specification. There after, the Pluto design was considered reviewed.

B.2.2 Code Review

Only one review was held for the Pluto code. An overview meeting occurred on October 26, 1994. The actual code review occurred November 16, 1995. Based on the code inspection, PR 21 through 23 were issued to correct deficiencies found. During the code review, modules of the code were identified with their requirements in the Pluto Traceability Matrix, see section B.5. The code was deemed ready for testing there after.

B.3. Pluto Test Results

DO-178B requires that test cases provide the coverage as stated in Section 6.4.2 and Table B.5-7. As described in the Verification Cases document, test cases were developed to fulfill those requirements. Testing Pluto with the those test cases will ensure that the coverage has been satisfied for the implementation. Pluto testing proceeded in the order as specified in *Software Verification Cases and Procedures*:

Requirements-based functional unit testing

- Requirements-based Subframe testing
- Requirements-based Frame testing
- Requirements-based Trajectory testing
- Structural analysis and testing of functional units

The output from each test phase was a series of test logs indicating when the test cases were executed, and whether the test cases revealed any deficiencies. A condensed version of the test logs are included in the following sections. Each section starts with a list of code components tested and the test log for that functional unit. The test logs have been abbreviated here so that only the naming pattern is entered in each entry of the log. Only those test cases that failed are listed specifically. The full test log for each Pluto functional unit are stored and can be fetched from the CMS library for this project. The same naming conventions are used in the logs as are used in the Verification Cases document.

The Pluto code consists of 21 files, each termed a code component. A description of each component is given in Table B.1.

Table B.1: Description of Pluto Code Components.

Pluto Code Component	**Functional Description**
AECLP.FOR	Implements the AECLP functional unit
ARSP.FOR	Implements the ARSP functional unit
ASP.FOR	Implements the ASP functional unit
CLPSF.FOR	This implements the control law processing subframe
CP.FOR	This implements the CP functional unit
CRCP.FOR	This implements the CRCP functional unit.
EXTERNAL.FOR	This is the include file for the External data store
GP.FOR	Implements the GP functional unit
GPSF.FOR	Implement the guidance processing subframe.
GSP.FOR	Implements the GSP functional unit
GUIDANCE_STATE.FOR	This component is an include file for the Guidance_State data store.
PLUTO.FOR	Implements the high level interface into the Pluto code.
RECLP.FOR	Implements the RECLP functional unit
RUN_PARAMETERS.FOR	Include file for Run_Parameters data store
SENSOR_OUTPUT.FOR	Include file for Sensor_Output data store
SPSF.FOR	This routine implements the sensor processing subframe
TDLRSP.FOR	Implements the TDLRSP functional unit
TDSP.FOR	Implements the TDSP functional unit
TSP.FOR	Implements the TSP functional unit
UTILITY.FOR	This file contains routines that perform range checking, checking for zero, and negative values. The routines are used in all functional units.

B.3.1 Requirements Based Functional Unit Testing

The following sections gives the results of the requirements-based test cases for the Pluto implementation starting with the functional-unit level testing. A list of functional unit is given below followed by the results of each functional unit.

Axial Engine Control Law Processing	AECLP
Altimeter Radar Sensor Processing	ARSP
Accelerometer Sensor Processing	ASP
Communications Processing	CP
Chute Release Control Processing	CRCP
Guidance Processing	GP
Gyroscope Sensor Processing	GSP
Roll Engine Control Law Processing	RECLP
Touch Down Landing Radar Sensor Processing	TDLRSP
Touch Down Sensor Processing	TDSP
Temperature Sensor Processing	TSP

B.3.1.1 ARSP Functional Unit

Code components tested in this test suite are given in Table B.2. The test log for ARSP requirements-based testing is summarized in Table B.3. The "xxx" notation used in Table B.3 as well as other test log summaries in this document represent the test case number. Only test cases that revealed anomalies in the code are specifically listed.

Table B.2: ARSP code components.

EXTERNAL.FOR	ARSP.FOR
RUN_PARAMETERS.FOR	UTILITY.FOR
GUIDANCE_STATE.FOR	CONSTANTS.FOR
SENSOR_OUTPUT.FOR	

Total number of normal range (NR) test cases: 9

Total number of robustness (RO) test cases: 14

Table B.3: Summary of Requirements-based Testing on the ARSP Functional Unit.

TEST CASE NAME	EXECUTION DATE	RESULTS .ANA file/PR #	Reason for Test Run
ARSP_RO_xxx	1/5/95	N	Initial testing.
ARSP_NR_xxx		N	
ARSP_NR_017		Y/24	
ARSP_NR_022		Y/24	
ARSP_NR_023		Y/24	
ARSP_RO_xxx	1/13/95	N	Retesting because PR 24 changed
ARSP_NR_xxx		N	CONSTANT.FOR.
ARSP_RO_xxx	4/7/95	N	Retest after Cases & Procedures
ARSP_NR_xxx		N	finalized.

Note: an analysis file (.ANA file) is only generated when the results of the test case does not match the expected results. In the RESULTS column in Table B.3, a "Y" indicates that the test cases miscompared generating an ANA file. "N" indicates cases that did not have any miscompares.

B.3.1.2 ASP Functional Unit

Code components tested for this functional unit are given in Table B.4.

Table B.4: ASP code components.

EXTERNAL.FOR	ASP.FOR
RUN_PARAMETERS.FOR	UTILITY.FOR
GUIDANCE_STATE.FOR	CONSTANTS.FOR
SENSOR_OUTPUT.FOR	

Total number of normal range (NR) test cases: 8

Total number of robustness (RO) test cases: 36

Table B.5: Summary of Requirements-based Testing on the ASP Functional Unit.

TEST CASE NAME	EXECUTION DATE	RESULTS .ANA file/PR #	Reason for Test Run
ASP_NR_xxx	1/5/95	N	Initial testing
ASP_RO_xxx		N	
ASP_NR_xxx	1/17/95	N	Retesting because PR 24 changed CONSTANT.FOR.
ASP_RO_xxx		N	
ASP_NR_xxx	4/7/95	N	Retest after Cases & Procedures finalized.
ASP_RO_xxx		N	

B.3.1.3 GSP Functional Unit

Code components tested in this test suite are given in Table B.6.

Table B.6: GSP code components.

EXTERNAL.FOR	GSP.FOR
RUN_PARAMETERS.FOR	UTILITY.FOR
GUIDANCE_STATE.FOR	CONSTANTS.FOR
SENSOR_OUTPUT.FOR	

Total number of normal range (NR) test cases: 8

Total number of robustness (RO) test cases: 36

Table B.7: Summary of Requirements-based Testing on the GSP Functional Unit.

TEST CASE NAME	EXECUTION DATE	RESULTS .ANA file/PR #	Reason for Test Run
GSP_NR_xxx	1/5/95	N	Initial testing
GSP_RO_xxx		N	
GSP_NR_xxx	1/17/95	N	Retesting because PR 24 changed CONSTANT.FOR.
GSP_RO_xxx		N	
GSP_NR_xxx	4/7/95	N	Retest after Cases & Procedures finalized.
GSP_RO_xxx		N	

B.3.1.4 TSP Functional Unit

Code components tested in this suite are given in Table B.8.

Table B.8: TSP code components.

EXTERNAL.FOR	TSP.FOR
RUN_PARAMETERS.FOR	UTILITY.FOR
GUIDANCE_STATE.FOR	CONSTANTS.FOR
SENSOR_OUTPUT.FOR	

Total number of normal range (NR) test cases: 5
Total number of robustness (RO) test cases: 6

The first iteration of testing revealed some deficiencies in TSP. These were addressed in Problem Report 24. The second iteration of testing shows that all deficiencies were corrected except for TSP_RO_011 which still did not compare exactly for ATMOSPHERIC_TEMP. The ANA file shows that Pluto computed ATMOSPHERIC_TEMP to be $-0.1140537605916 \times 10^{10}$ while the expected value is $-0.1140537605916 \times 10^{10}$. Recall from the pass/fail criteria discussion in *Software Verification Cases and Procedures* that relative error is used as an accuracy check when ATMOSPHERIC_TEMP exceeds 1. Accordingly, the absolute error is deduced to be .001 (since the number is not printed in the ANA file to the full precision); the relative error is calculated to be (.001/114053760) 8.77×10^{-12}. This is less than the d for ATMOSPHERIC_TEMP given in Table 22 of *Software Verification Cases and Procedures*. Hence this test case is considered passed. Note additionally that the value given for ATMOSPHERIC_TEMP is also out of bounds. This is also acceptable because its a robustness test case.

Table B.9: Summary of Requirements-based Testing on the TSP Functional Unit.

TEST CASE NAME	**EXECUTION DATE**	**RESULTS .ANA file/PR #**	**Reason for Test Run**
TSP_NR_xxx	1/4/95	N	Initial testing
TSP_RO_xxx		N	
TSP_NR_006.TC		Y/24	
TSP_NR_007.TC		Y/24	
TSP_RO_008.TC		Y/24	
TSP_RO_009.TC		Y/24	
TSP_RO_010.TC		Y/24	
TSP_RO_011.TC		Y/24	
TSP_NR_xxx	1/13/95	N	Retesting due to PR 24 corrections.
TSP_RO_xxx		N	
TSP_RO_011.TC		Y*	
TSP_NR_xxx	4/7/95	N	Retest after Cases & Procedures finalized
TSP_RO_xxx		N	
TSP_RO_011.TC		Y*	

B.3.1.5 TDSP Functional Unit

Code components tested for TDSP are given in Table B.10.

Table B.10: TDSP code components.

EXTERNAL.FOR	TDSP.FOR
RUN_PARAMETERS.FOR	UTILITY.FOR
GUIDANCE_STATE.FOR	CONSTANTS.FOR
SENSOR_OUTPUT.FOR	

Total number of normal range (NR) test cases: 3

Total number of robustness (RO) test cases: 4

Table B.11: Summary of Requirements-based Testing on the TDSP Functional Unit.

TEST CASE NAME	**EXECUTION DATE**	**RESULTS .ANA file/PR #**	**Reason for Test Run**
TDSP_NR_xxx	1/4/95	N	Initial testing
TDSP_RO_xxx		N	
TDSP_NR_xxx	1/17/95	N	Retesting because PR 24 changed
TDSP_RO_xxx		N	CONSTANT.FOR.
TDSP_NR_xxx	4/7/95	N	Retest after Cases & Procedures
TDSP_RO_xxx		N	finalized.

B.3.1.6 TDLRSP Functional Unit

Code components tested for TDLRSP are given in Table B.12.

Table B.12: TDLRSP code components.

EXTERNAL.FOR	TDLRSP.FOR
RUN_PARAMETERS.FOR	UTILITY.FOR
GUIDANCE_STATE.FOR	CONSTANTS.FOR
SENSOR_OUTPUT.FOR	

Total number of normal range (NR) test cases: 18

Total number of robustness (RO) test cases: 10

The ANA file generated for TDLRSP_RO_026 involves a condition that is not specified in the SPEC. Although the results of this test run does not agree with the expected values, the results are just as valid because this robustness test case exercises a condition that is not defined in the Specification. More specifically, a value of "2" is assigned to the variable TDLR_STATE. Although a "2" is not defined as a legal value for this variable in the GCS Spec, it is a possible value since the variable is ultimately implemented as an integer. For robustness test cases, DO-178B requires only that the software not cause any detrimental effects to the system. For this specific test case, the PLUTO code leaves the values of K_MATRIX unchanged. This will not have a severe impact on the implementation's ability to deliver the required function for TDLRSP.

Table B.13: Summary of Requirements-based Testing on the TDLRSP Functional Unit.

TEST CASE NAME	EXECUTION DATE	RESULTS .ANA file/PR #	Reason for Test Run
TDLRSP_NR_xxx	1/4/95	N	Initial testing
TDLRSP_RO_xxx		N	
TDLRSP_RO_026		Y/24	
TDLRSP_NR_xx	1/13/95	N	Retesting due to PR 24.
TDLRSP_RO_xxx		N	
TDLRSP_RO_026		Y	
TDLRSP_NR_xx	4/7/95	N	Retest after Cases & Procedures finalized.
TDLRSP_RO_xxx		N	
TDLRSP_RO_026		Y	

B.3.1.7 GP Functional Unit

Code components tested for GP are given in Table B.14.

Table B.14: GP code components.

EXTERNAL.FOR	GP.FOR
RUN_PARAMETERS.FOR	UTILITY.FOR
GUIDANCE_STATE.FOR	CONSTANTS.FOR
SENSOR_OUTPUT.FOR	

Total number of normal range (NR) test cases: 14

Total number of robustness (RO) test cases: 103

In the initial run of all the GP test cases, there were some errors in the algorithm for calculating GP_ATTITUDE, GP_ALTITUDE, and GP_VELOCITY. This caused a mismatch with the expected results for all the test cases. Problem Report 24 addressed this deficiency. As indicated in the second iteration of tests, this deficiency has been eliminated. The third run of GP test cases test a change to CONSTANT.FOR.

Table B.15: Summary of Requirements-based Testing on the GP Functional Unit.

TEST CASE NAME	EXECUTION DATE	RESULTS .ANA file/PR #	Reason for Test Run
GP_NR_xxx	1/4/95	Y/24	Initial testing
GP_RO_xxx	1/4/95	Y/24	
GP_NR_xxx	1/13/95	N	Retesting after PR 24 changes
GP_RO_xxx	1/13/95	N	
GP_NR_xxx	3/1/95	N	Retesting due to SDCR 15
GP_RO_xxx	3/1/95	N	
GP_NR_xxx	4/7/95	N	Retest after Cases & Procedures finalized.
GP_RO_xxx	4/7/95	N	

B.3.1.8 AECLP Functional Unit

Code components tested for AECLP are given in Table B.16.

Table B.16: AECLP code components.

EXTERNAL.FOR	AECLP.FOR
RUN_PARAMETERS.FOR	UTILITY.FOR
GUIDANCE_STATE.FOR	CONSTANTS.FOR
SENSOR_OUTPUT.FOR	

Total number of normal range (NR) test cases: 14

Total number of robustness (RO) test cases: 43

There were three iterations of testing for this functional unit as can be seen from the test log. Although all test cases passed in the first iteration, the second iteration was necessitated by a change in the CONSTANTS.FOR documented in Problem Report #24.

Table B.17: Summary of Requirements-based Testing on the AECLP Functional Unit.

TEST CASE NAME	EXECUTION DATE	RESULTS .ANA file/PR #	Reason for Test Run
AECLP_NR_xxx	1/5/95	N	Initial testing
AECLP_RO_xxx		N	
AECLP_NR_xxx	1/18/95	N	Retesting because PR 24 changed
AECLP_RO_xxx		N	CONSTANT.FOR.
AECLP_NR_xxx	4/7/95	N	Retest after Cases & Procedures
AECLP_RO_xxx		N	finalized.

B.3.1.9 RECLP Functional Unit

Code components tested for RECLP are given in Table B.18.

Table B.18: RECLP code components

EXTERNAL.FOR	RECLP.FOR
RUN_PARAMETERS.FOR	UTILITY.FOR
GUIDANCE_STATE.FOR	CONSTANTS.FOR
SENSOR_OUTPUT.FOR	

Total number of normal range (NR) test cases: 64

Total number of robustness (RO) test cases: 4

For the first round of testing, even though an analysis file (.ANA) was not generated for these test cases, the limits checking prints messages to the screen for values of THETA that are in bounds. Further observations revealed that the upper and lower bounds constants were reversed in CONSTANTS.FOR. This has been addressed in Problem Report 24. Test cases were re-executed after this was corrected. Note that neither the RECLP code or the test cases had to be refetched. However, the CONSTANTS.FOR file was refetched and the code was recompiled to generate a new executable incorporating new changes from CONSTANTS.FOR.

Table B.19: Summary of Requirements-based Testing on the RECLP Functional Unit.

TEST CASE NAME	**EXECUTION DATE**	**RESULTS .ANA file/PR #**	**Reason for Test Run**
RECLP_NR_xxx	1/5/95	N/24	Initial testing
RECLP_RO_xxx		N/24	
RECLP_NR_xxx	1/13/95	N	Retesting because PR 24 changed
RECLP_RO_xxx		N	CONSTANT.FOR.
RECLP_NR_xxx	4/7/95	N	Retest after Cases & Procedures
RECLP_RO_xxx		N	finalized.

B.3.1.10 CRCP Functional Unit

Code components tested for CRCP are given in Table B.20.

Table B.20: CRCP code components.

CRCP.FOR	EXTERNAL.FOR
UTILITY.FOR	RUN_PARAMETERS.FOR
CONSTANTS.FOR	GUIDANCE_STATE.FOR
	SENSOR_OUTPUT.FOR

Total number of normal range (NR) test cases: 6

Total number of robustness (RO) test cases: 4

Table B.21: Summary of Requirements-based Testing on the CRCP Functional Unit.

TEST CASE NAME	**EXECUTION DATE**	**RESULTS .ANA file/PR #**	**Reason for Test Run**
CRCP_NR_xxx	1/5/95	N	Initial testing
CRCP_RO_xxx		N	
CRCP_NR_xxx	1/17/95	N	Retesting because PR 24 changed
CRCP_RO_xxx		N	CONSTANT.FOR.
CRCP_NR_xxx	4/7/95	N	Retest after Cases & Procedure finalized.
CRCP_RO_xxx		N	

B.3.1.11 CP Functional Unit

Code components tested for CP are given in Table B.22.

Table B.22: CP code components:

CP.FOR	EXTERNAL.FOR
UTILITY.FOR	RUN_PARAMETERS.FOR
CONSTANTS.FOR	GUIDANCE_STATE.FOR
	SENSOR_OUTPUT.FOR

Total number of normal range (NR) test cases: 5

Total number of robustness (RO) test cases: 0

Table B.23: Summary of Requirements-based Testing on the CP Functional Unit.

TEST CASE NAME	EXECUTION DATE	RESULTS .ANA file/PR #	Reason for Test Run
CP_NR_xxx	1/12/95	Y/25	Initial testing
CP_NR_xxx	1/19/95	N	Retesting after PR 25 modifications
CP_NR_xxx	4/7/95	N	Retest after Cases & Procedures finalized

B.3.2 Subframe Testing

While preparing the code for subframe and frame testing, errors were found that necessitated issuance of PR 26.

B.3.2.1 SP Subframe

Code components tested for SP subframe are given in Table B.24.

Table B.24: SP code components.

TSP.FOR	EXTERNAL.FOR
ARSP.FOR	RUN_PARAMETERS.FOR
ASP.FOR	GUIDANCE_STATE.FOR
GSP.FOR	SENSOR_OUTPUT.FOR
TDLRSP.FOR	CONSTANTS.FOR
TDSP.FOR	UTILITY.FOR
CP.FOR	

Total number of test cases: 1

Table B.25: Summary of Requirements-based Testing on the SP subframe.

TEST CASE NAME	EXECUTION DATE	RESULTS .ANA file/PR #	Reason for Test Run
SP_001	3/6/95	N	Initial testing
SP_001	4/7/95	N	Retest after Cases & Procedures finalized

B.3.2.2 GP Subframe

Code components tested for GP subframe are given in Table B.26.

Table B.26: GP subframe code components.

GP.FOR	EXTERNAL.FOR
CP.FOR	RUN_PARAMETERS.FOR
UTILITY.FOR	GUIDANCE_STATE.FOR
CONSTANTS.FOR	SENSOR_OUTPUT.FOR

Total number of test cases: 8

Table B.27: Summary of Requirements-based Testing on the GPSF subframe.

TEST CASE NAME	**EXECUTION DATE**	**RESULTS .ANA file/PR #**	**Reason for Test Run**
GPSF_xxx	3/6/95	N	Initial testing
GPSF_xxx	4/7/95	N	Retest after Cases & Procedures finalized

B.3.2.3 CLP Subframe

Code components tested for CLP subframe are given in Table B.28.

Table B.28: CLP subframe code components.

AECLP.FOR	EXTERNAL.FOR
RECLP.FOR	RUN_PARAMETERS.FOR
CRCP.FOR	GUIDANCE_STATE.FOR
CP.FOR	SENSOR_OUTPUT.FOR
UTILITY.FOR	CONSTANTS.FOR

Total number of test cases: 14

Table B.29: Summary of Requirements-based Testing on the CLP subframe.

TEST CASE NAME	**EXECUTION DATE**	**RESULTS .ANA file/PR #**	**Reason for Test Run**
CLP_xxx	3/6/95	N	Initial testing
CLP_xxx	4/7/95	N	Retest after Cases & Procedures finalized

B.3.3 Frame Testing

Code components tested during Frame testing are given in Table B.28.

Table B.30: Frame code components.

TSP.FOR	CRCP.FOR
ARSP.FOR	CP.FOR
ASP.FOR	UTILITY.FOR
GSP.FOR	EXTERNAL.FOR
TDLRSP.FOR	RUN_PARAMETERS.FOR
TDSP.FOR	GUIDANCE_STATE.FOR
GP.FOR	SENSOR_OUTPUT.FOR
AECLP.FOR	CONSTANTS.FOR
RECLP.FOR	

Total number of test cases: 9

Table B.31: Summary of Requirements-based Testing on for Frame.

TEST CASE NAME	EXECUTION DATE	RESULTS .ANA file/PR #	Reason for Test Run
FRAME_xxx	3/6/95	N	Initial testing
FRAME_xxx	4/7/95	N	Retest after Cases & Procedures finalized

B.3.4 Trajectory Testing

Code components tested during trajectory testing are in Table B.32.

Table B.32: Trajectory test code components.

PLUTO.FOR	AECLP.FOR
SPSF.FOR	RECLP.FOR
GPSF.FOR	CRCP.FOR
CLPSF.FOR	CP.FOR
TSP.FOR	UTILITY.FOR
ARSP.FOR	EXTERNAL.FOR
ASP.FOR	RUN_PARAMETERS.FOR
GSP.FOR	GUIDANCE_STATE.FOR
TDLRSP.FOR	SENSOR_OUTPUT.FOR
TDSP.FOR	CONSTANTS.FOR
GP.FOR	

Total number of test cases: 34

Table B.33: Summary of Requirements-based Trajectory Testing

TEST CASE NAME	EXECUTION DATE	FAILED FRAME NUMBER MATCHES	FAILED GP_PHASE MATCHES	Reason for Test Run
TRAJ_ATM_UD/IC_xxx	3/6/95	N	N	Initial testing
TRAJ_TD_UD/IC_xxx		N	N	
TRAJ_TD_UD/IC_019		Y/27	N	
TRAJ_TD_UD/IC_021		N	Y/27	
TRAJ_ATM_UD/IC_xxx	4/7/95	N	N	Retesting after PR 27 modifications.
TRAJ_TD_UD/IC_xxx		N	N	

B.3.5 Structural Analysis and Testing

Structural analysis of Pluto source code was performed with the aid of the ACT software. ACT was used to derive a decision tree for each functional unit code. These trees are included with their respective decision tables. Decision tables were then created to match test cases to the specific decisions in the code. Each decision entry in a table has a true and false test case to test the respective outcome for that decision. To assist in building the decision tables, ACT is also used to generate annotated listings that indicate the FORTRAN decisions associated with the node numbers in the trees and listed in the tables.

The objective of structural analysis is to ensure that DO-178B's required Modified Condition/Decision Coverage (MC/DC) has been met for the Pluto code. As stated in *Software Verification Cases and Procedures*, four conditions must be satisfied to provide coverage. This structural analysis has satisfied those four conditions in the following ways:

1) "Each decision takes on every possible outcome at least once."

   This is satisfied by the primary decision tables for each functional unit and subroutine. The primary table contains a TRUE and FALSE column for each decision -- a test case is given for each. Test cases followed by an "*" indicate that there are multiple requirements-based test cases that satisfy the specific decision. Subroutines that do not contain any decisions will not have a primary decision table, because any test case that enters the routine will exercise all the statements in the routine. Those test cases are just listed in the Entry/Exit tables to avoid duplication.

2) "Each condition in each decision takes on every possible outcome at least once."

   This is demonstrated in the pairs table given for each decision that has multiple conditions. Each pairs table has extra columns to the right of the test case column showing cases where the condition is tested at each possible out come value.

3) "Each entry and exit point is invoked at least once."

   This is demonstrated in the Entry/Exit tables for subroutines in each functional unit.

4) "Each condition is shown to independently effect the decision outcome."

   This is also demonstrated in the pairs table for each decision with multiple conditions. The independent impact of each condition on the final decision outcome is shown in the independence columns (e.g. "Ind. of con 1") to the right of the test case column. The "*" in the column give test cases in which the value of the condition drives the outcome of the decision.

Much of the Pluto code structure was already tested by the requirements coverage test cases. Structural test cases are created for only those conditions not covered by the requirements based test cases. Since complete path coverage is not an objective in MC/DC requirement, the decisions involving a loop counter that is not manipulated or calculated are not tested since any test case reaching that point will exercise the loop entirely. These decisions are appropriately denoted in the decision tables.

In the following structural analysis of the Pluto implementation, a section is dedicated for each functional unit with the last section for the utility subroutines that are used by all functional unit. For each functional unit, a decision tree is first given. The decision tree is generated using the ACT software as prescribed in the Verification Cases and Procedures Document. The decision tree shows all the branching that occurs in the functional unit and assigns a number for each branch. These numbers are used in the decision table to identify the decision being made. The decision tree is followed by one or more tables listing the decision made at the node and the test cases that exercise the decision.

The first table in each section is the primary decision table that lists all decisions occurring in the code for the functional unit. Decisions with multiple conditions have a separate pairs table for each. Where applicable, Entry/Exit tables are given for subroutines used in a functional unit. Decision tables for utility routines specific to each functional unit are placed in the same sections as the corresponding functional units.

B.3.5.1 ARSP Structural Analysis

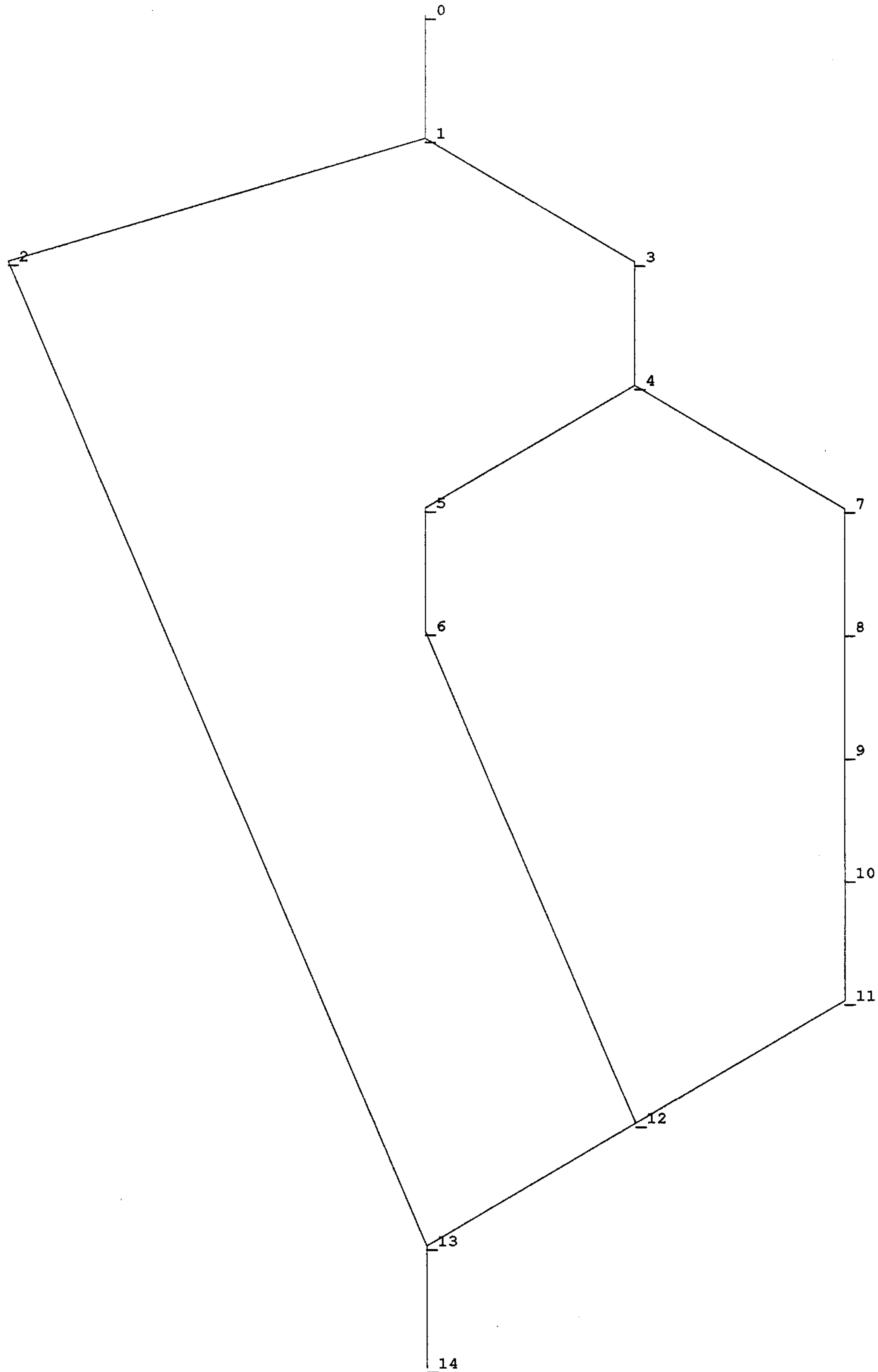


Figure B.1: ARSP Decision Tree:

Table B.34: ARSP Decision Table - see Figure B.1 for correspondence.

Graph Node Number	**ARSP Decisions**	**TRUE output test cases**	**FALSE output test case**
1	(AR_COUNTER .NE. -1)	ARSP_NR_017	ARSP_NR_012*
4	((AR_STATUS(1) .EQ. K$FAILED) .OR. (AR_STATUS(2) .EQ. K$FAILED) .OR. (AR_STATUS(3) .EQ. K$FAILED) .OR. (AR_STATUS(4) .EQ. K$FAILED))	See ARSP MC/DC table for decision node 4	

Table B.35: MC/DC Pairs table for decision node 4 of ARSP:

AR_STATUS(1) .EQ. K$FAILED **(Con 1)**	AR_STATUS(2) .EQ. K$FAILED **(Con 2)**	AR_STATUS(3) .EQ. K$FAILED **(Con 3)**	AR_STATUS(4) .EQ. K$FAILED **(Con 4)**	Final Decision	Test Case	Ind. of **Con 1**	Ind. of **Con 2**	Ind. of **Con 3**	Ind. of **Con 4**
0	0	0	0	0	ARSP_NR_011	*	*	*	*
0	0	0	1	1	ARSP_NR_015				*
0	0	1	0	1	ARSP_NR_014			*	
0	1	0	0	1	ARSP_NR_013		*		
1	0	0	0	1	ARSP_NR_012	*			

0 = FALSE value for the condition

1 = TRUE value for the condition

No structural test cases were developed for ARSP functional unit. The requirements based cases adequately tested the code structure.

B.3.5.2 ASP Structural Analysis

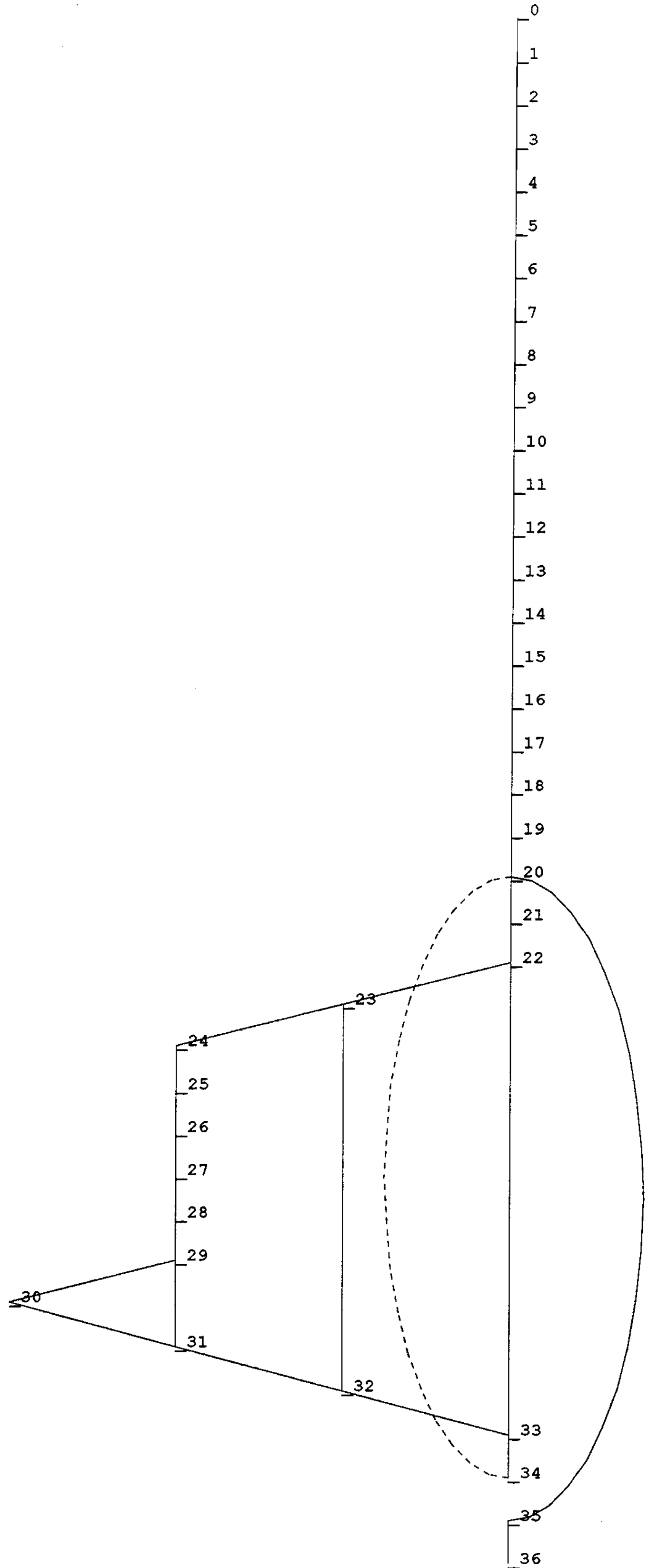


Figure B.2: ASP Decision Tree:

Table B.36: ASP Decisions Table - see Figure B.2 for correspondence

Graph Node Number	ASP Decisions	TRUE output test cases	FALSE output test case
20	I in range (loop based on I)	Not a calculated loop counter; Testing not required	
22	((A_STATUS(I,1) .EQ. K$HEALTHY) .AND. (A_STATUS(I,2) .EQ. K$HEALTHY) .AND. (A_STATUS(I,3) .EQ. K$HEALTHY))	See ASP MC/DC pairs table for Node 22	
23	(A_ACCELERATION(I,1) .NE. A_ACCELERATION(I,2)) .AND. (A_ACCELERATION(I,1) .NE. A_ACCELERATION(I,3))	See ASP MC/DC pairs table for Node 23	
29	temp .GT. A_SCALE * SD	ASP_NR_002	ASP_PST_002

Table B.37: MC/DC Pairs table for decision node 22 of ASP:

A_STATUS(I,1) .EQ. K$HEALTHY **(Con. 1)**	A_STATUS(I,2) .EQ. K$HEALTHY **(Con. 2)**	A_STATUS(I,3) .EQ. K$HEALTHY **(Con. 3)**	Final Decision	Test Case	Ind. of **Con 1**	Ind. of **Con 2**	Ind. of **Con 3**
1	1	0	0	ASP_NR_005			*
1	0	1	0	ASP_NR_004		*	
0	1	1	0	ASP_NR_003	*		
1	1	1	1	ASP_NR_001	*	*	*

0 = FALSE value for the condition
1 = TRUE value for the condition

Table B.38: MC/DC Pairs table for decision node 23 of ASP:

(A_ACCELERATION(I,1) .NE. A_ACCELERATION(I,2)) **(Con. 1)**	(A_ACCELERATION(I,1) .NE. A_ACCELERATION(I,3)) **(Con. 2)**	Final Decision	Test Case	Ind. of **Con 1**	Ind. of **Con 2**
0	0	0	ASP_PST_001	*	*
0	1	1	ASP_PST_003		*
1	0	1	ASP_PST_004	*	

0 = FALSE value for the condition
1 = TRUE value for the condition

B.3.5.3 ASP Structural Testing

Code components tested in ASP structural testing are in Table B.4. Recall from the Verification Cases & Procedures document that structural-based test are setup and executed in the same manner as requirements-based functional unit tests. Hence the code components tested in structural-based testing are also identical. Table B.39 gives the summary log of ASP structural testing. There are 4 structural test cases for ASP.

Table B.39: Summary of Structural Testing for ASP Functional Unit

TEST CASE NAME	**EXECUTION DATE**	**RESULTS .ANA file/PR #**	**Reason for Test Run**
ASP_PST_xxx	4/11/95	N	Initial testing

B.3.5.4 GSP Structural Analysis

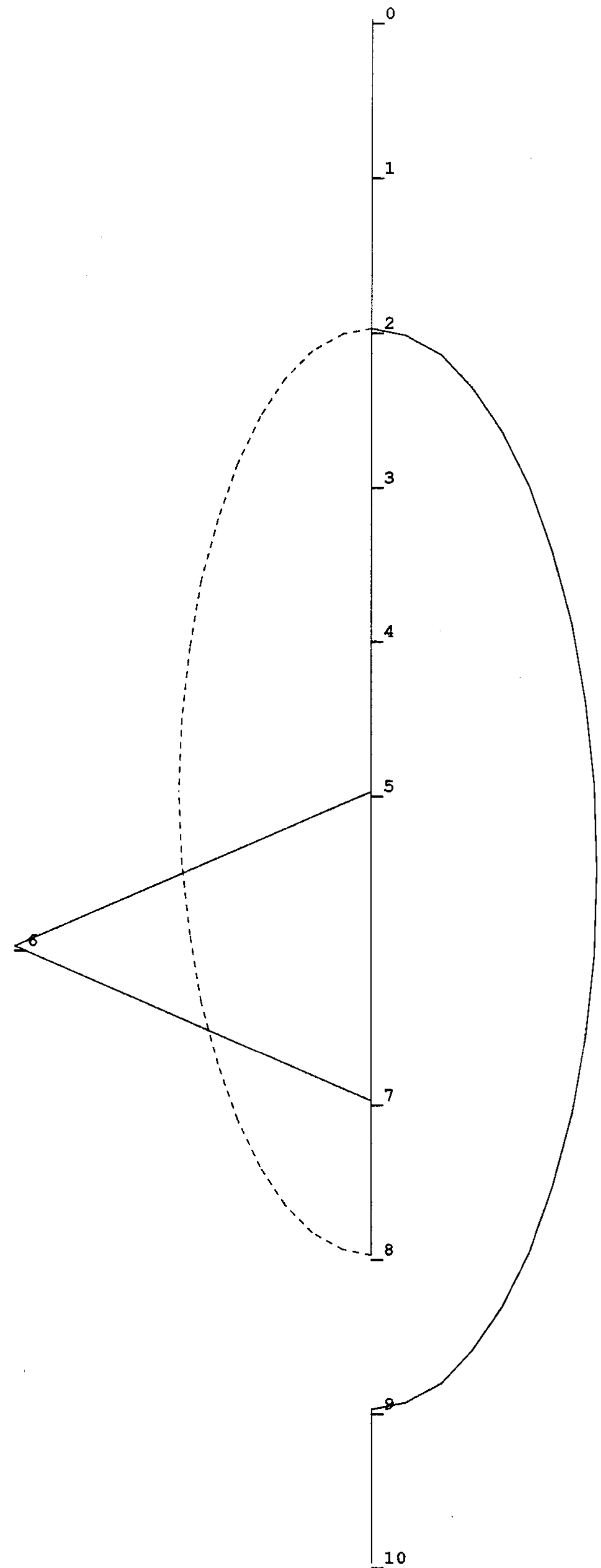


Figure B.3: GSP Decision Tree:

Table B.40: GSP Decision Table -- see Figure B.3 for correspondence.

Graph Node Number	**GSP Decisions**	**TRUE output test cases**	**FALSE output test case**
2	I in range (loop based on I)	Not a calculated loop counter; Testing not required	
5	BTEST(G_COUNTER(I), 15) .EQ. .TRUE.	GSP_NR_001	GSP_NR_004*

No structural test cases were developed for GSP functional unit. The requirements based cases adequately tested the code structure.

B.3.5.5 TSP Structural Analysis

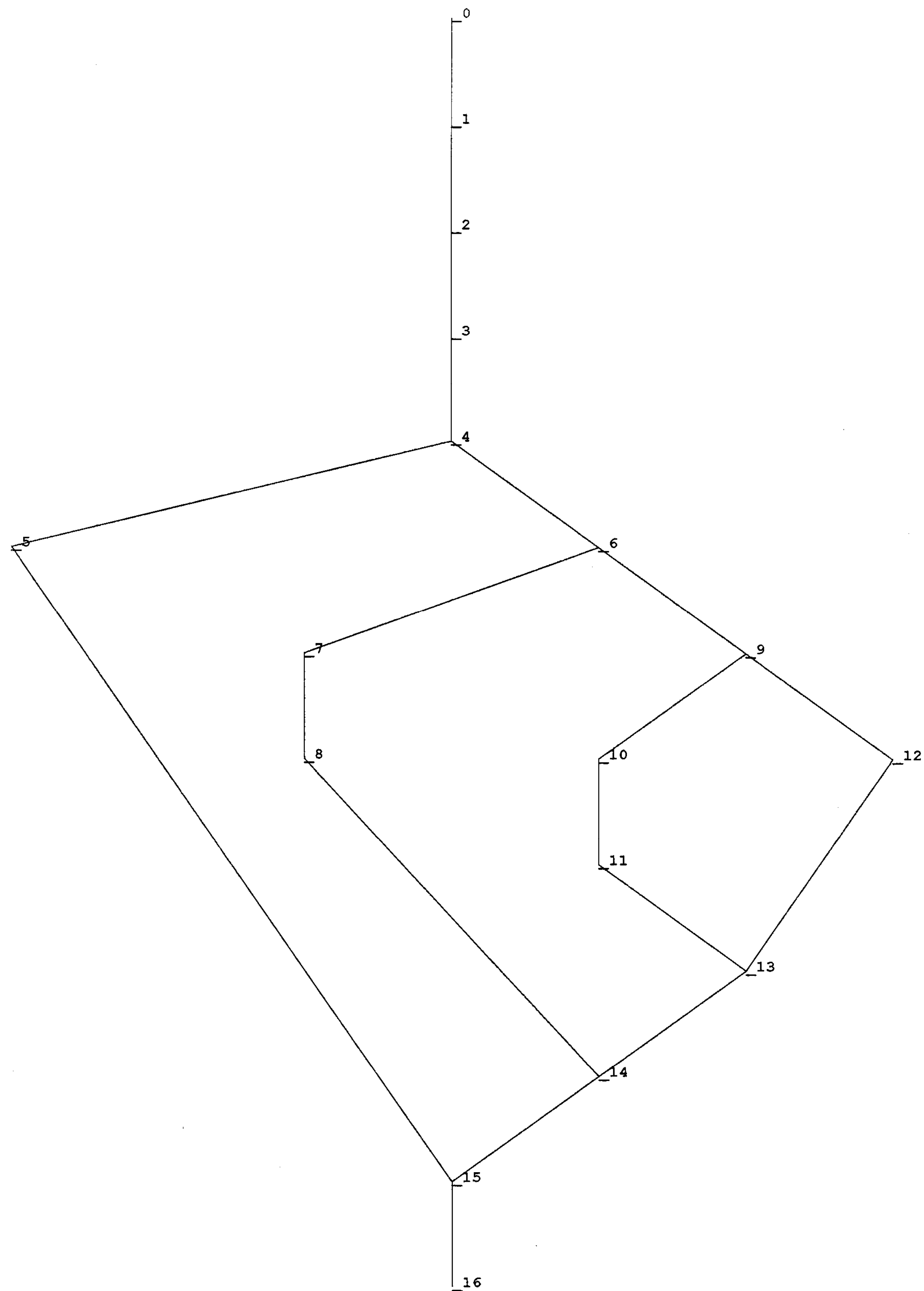


Figure B.4: TSP Decision Tree:

Table B.41: TSP Decision Table -- see TSP graph for correspondence.

Graph Node Number	TSP Decisions	TRUE output test cases	FALSE output test case
4	(SOLID_STATE_TEMP .LT. LOWER_PARABOLIC_TEMP_LIMIT) .OR. (SOLID_STATE_TEMP .GT. UPPER_PARABOLIC_TEMP_LIMIT)	TSP_NR_002*	TSP_NR_001*
6	THERMO_TEMP .LT. M3	TSP_NR_006	TSP_NR_001
9	THERMO_TEMP .GT. M4	TSP_NR_007	TSP_NR_001

Table B.42: MC/DC Pairs table for decision node 4 of TSP:

SOLID_STATE_TEMP .LT. LOWER_PARABOLIC_TEMP_LIMIT **(Con 1)**	SOLID_STATE_TEMP .GT. UPPER_PARABOLIC_TEMP_LIMIT **(Con 2)**	Final Decision	Test Case	Ind. of **Con 1**	Ind. of **Con 2**
0	0	0	TSP_NR_001*	*	*
0	1	1	TSP_NR_003		*
1	0	1	TSP_NR_002	*	

0 = FALSE value for the condition

1 = TRUE value for the condition

Table B.43: MC/DC Entry/Exit requirements -- for Modules inside TSP.FOR:

Module	Test Case
LOWER_PARABOLIC_FUNCTION	TSP_NR_001*
UPPER_PARABOLIC_FUNCTION	TSP_NR_001*

No structural test cases were developed for TSP functional unit. The requirements based cases adequately tested the code structure.

B.3.5.6 TDSP Structural Analysis

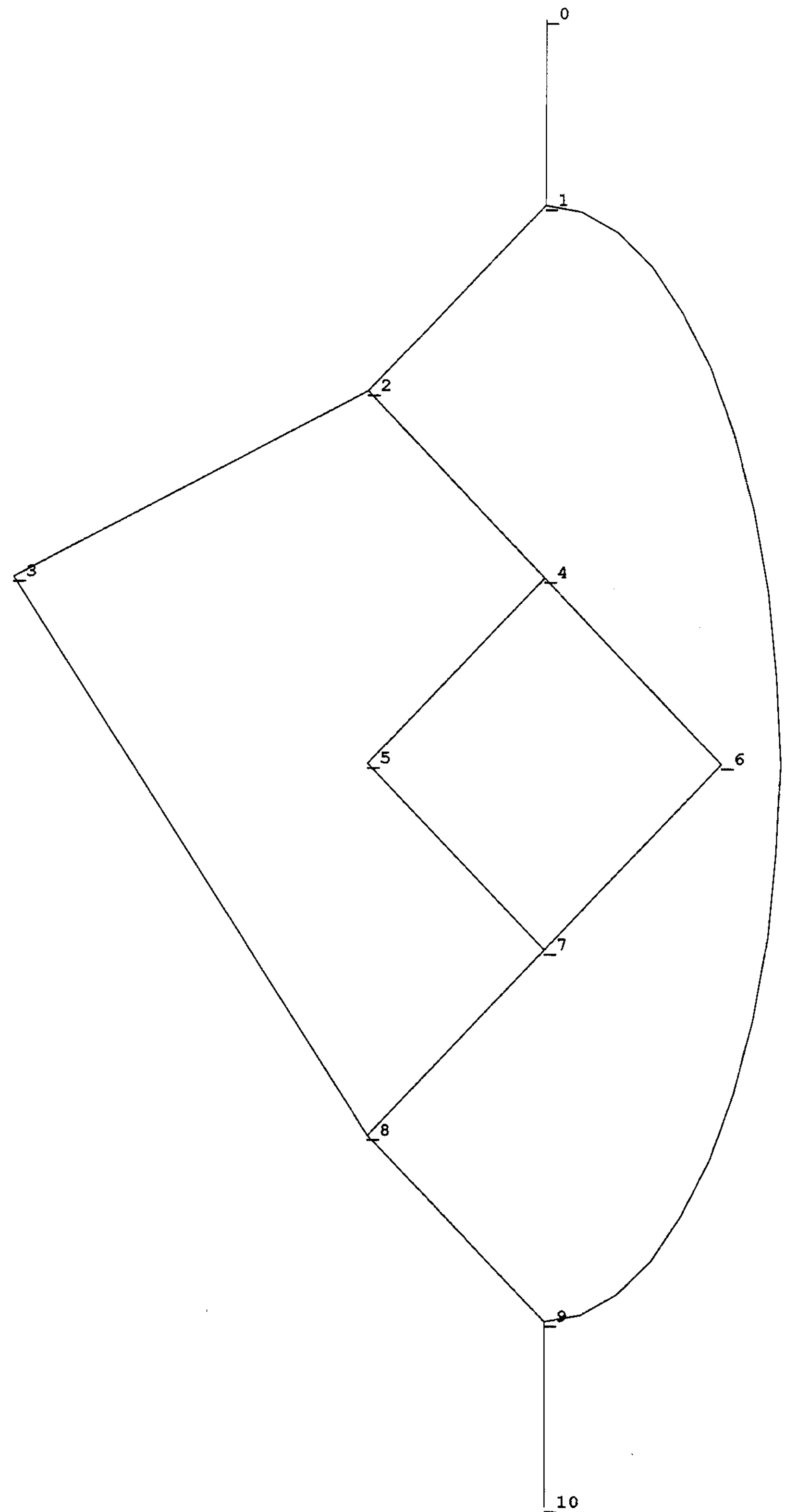


Figure B.5: TDSP Decision Tree:

Table B.44: TDSP Decisions -- see Figure B.5 for correspondence.

Graph Node Number	**TDSP Decisions**	**TRUE output test cases**	**FALSE output test case**
1	TDS_STATUS .EQ. K$HEALTHY	TDSP_NR_001*	TDSP_NR_004
2	TD_COUNTER .EQ. 0	TDSP_NR_001	TDSP_NR_002
4	TD_COUNTER .EQ. -1	TDSP_NR_002	TDSP_NR_003

No structural test cases were developed for TDSP functional unit. The requirements based cases adequately tested the code structure.

B.3.5.7 TDLRSP Structural Analysis

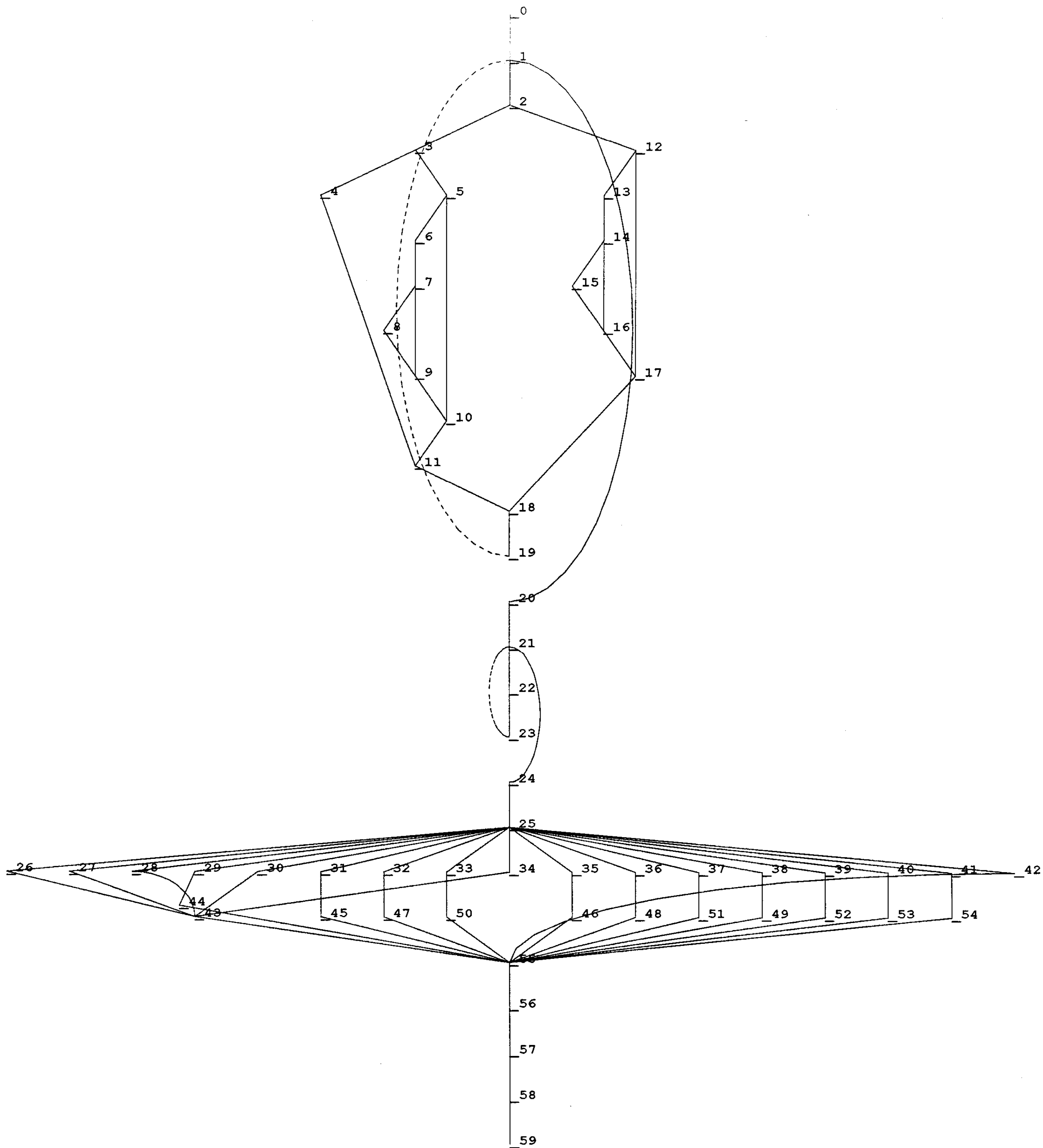


Figure B.6: TDLRSP Decision Tree:

Table B.45: TDLRSP Decisions -- see TDLRSP graph for correspondence.

Graph Node Number	TDLRSP Decisions	TRUE output test cases	FALSE output test case
1	I in range (loop based on I)	Not a calculated loop counter; Testing not required	
2	TDLR_COUNTER(I) .EQ. 0	TDLRSP_NR_003*	TDLRSP_NR_001*
3	TDLR_STATE(I) .EQ. K$BEAM_LOCKED	TDLRSP_NR_005	TDLRSP_NR_003
5	TDLR_STATE(I) .EQ. K$BEAM_UNLOCKED	TDLRSP_NR_003	TDLRSP_RO_026
7	ELAPSED_TIME .GE. TDLR_LOCK_TIME	TDLRSP_NR_003	TDLRSP_RO_004
12	TDLR_STATE(I) .EQ. K$BEAM_UNLOCKED	TDLRSP_NR_001	TDLRSP_RO_006
14	ELAPSED_TIME .GE. TDLR_LOCK_TIME	TDLRSP_NR_021	TDLRSP_RO_002
21	I in range (loop based on I)	Not a calculated loop counter; Testing not required	
25	This is a CASE statement implemented in VMS FORTRAN as a computed GOTO	See Table on Decision 25	

Table B.46: Expanded table for Decision 25.

TDLR_STATE(1) + 2*TDLR_STATE(2) + 4*TDLR_STATE(3) + 8*TDLR_STATE(4) + 1	Test Case
1	TDLRSP_NR_005
2	TDLRSP_NR_007
3	TDLRSP_NR_008
4	TDLRSP_NR_011
5	TDLRSP_NR_009
6	TDLRSP_NR_012
7	TDLRSP_NR_014
8	TDLRSP_NR_017
9	TDLRSP_NR_010
10	TDLRSP_NR_013
11	TDLRSP_NR_015
12	TDLRSP_NR_018
13	TDLRSP_NR_016
14	TDLRSP_NR_019
15	TDLRSP_NR_020
16	TDLRSP_NR_021
Out of range	TDLRSP_RO_026

No structural test cases were developed for TDLRSP functional unit. The requirements based cases adequately tested the code structure.

B.3.5.8 CP Structural Analysis

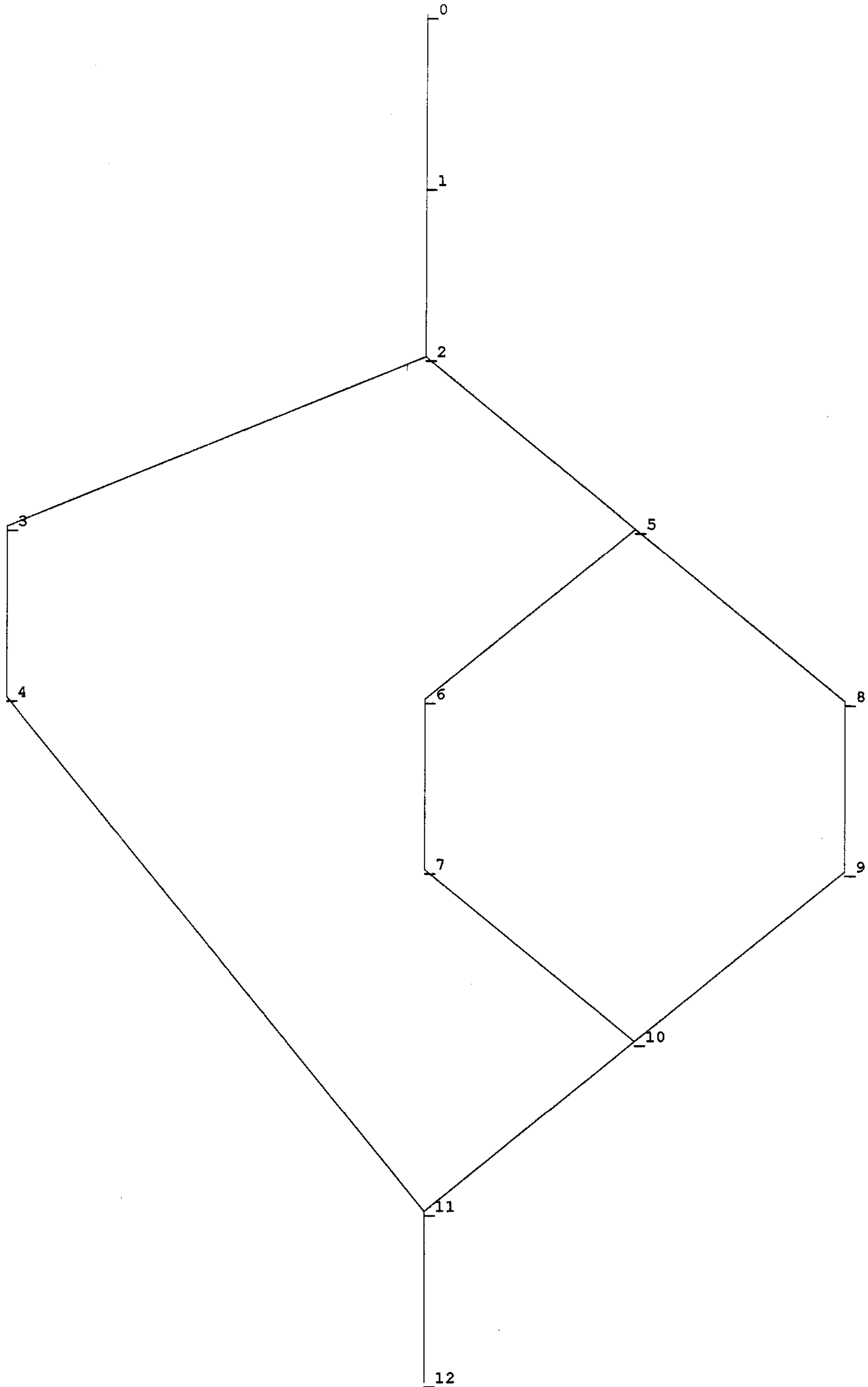


Figure B.7: CP Decision Tree.

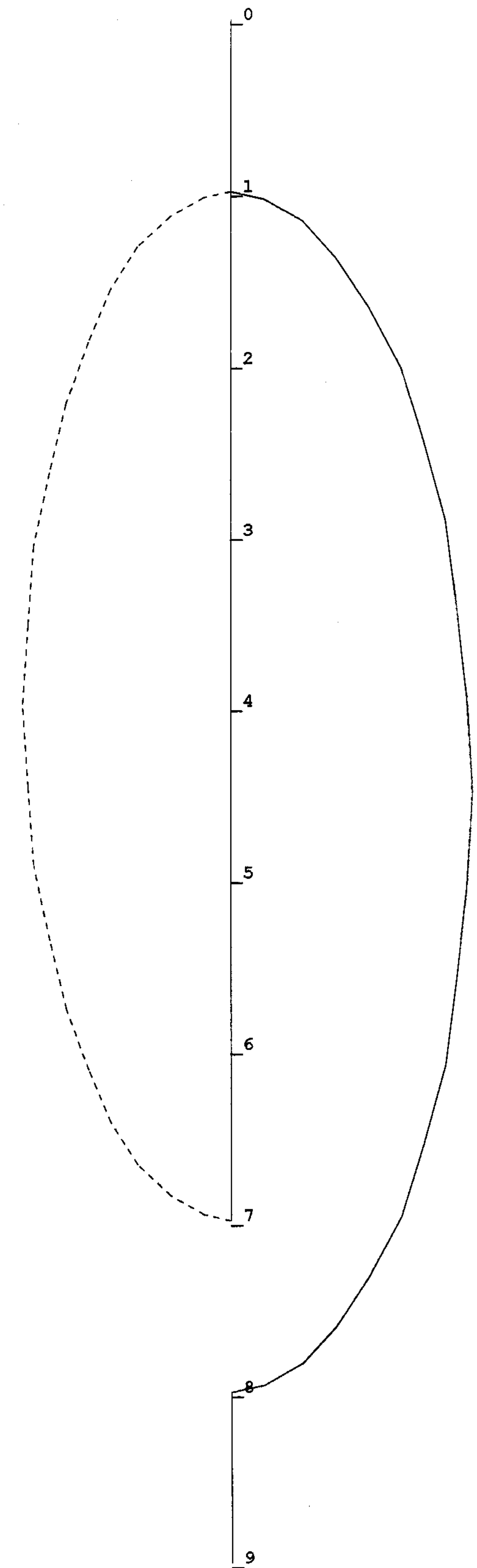


Figure B.8: CRC16 Decision Tree:

Table B.47: CP Decisions -- see CP graph for correspondence.

Graph Node Number	CP Decisions	TRUE output test cases	FALSE output test case
2	SUBFRAME_COUNTER .EQ. 1	CP_NR_001	CP_NR_002*
5	SUBFRAME_COUNTER .EQ. 2	CP_NR_002	CP_NR_003

Table B.48: CRC16 Decision.

Module	Test Case
I in range (loop based on I)	Not a calculated loop counter; Testing not required.

Table B.49: MC/DC Entry/Exit requirements for Module inside CP.FOR:

Module	Test Case
CRC16	CP_NR_001*

No structural test cases were developed for CP functional unit. The requirements based cases adequately tested the code structure.

B.3.5.9 GP Structural Analysis

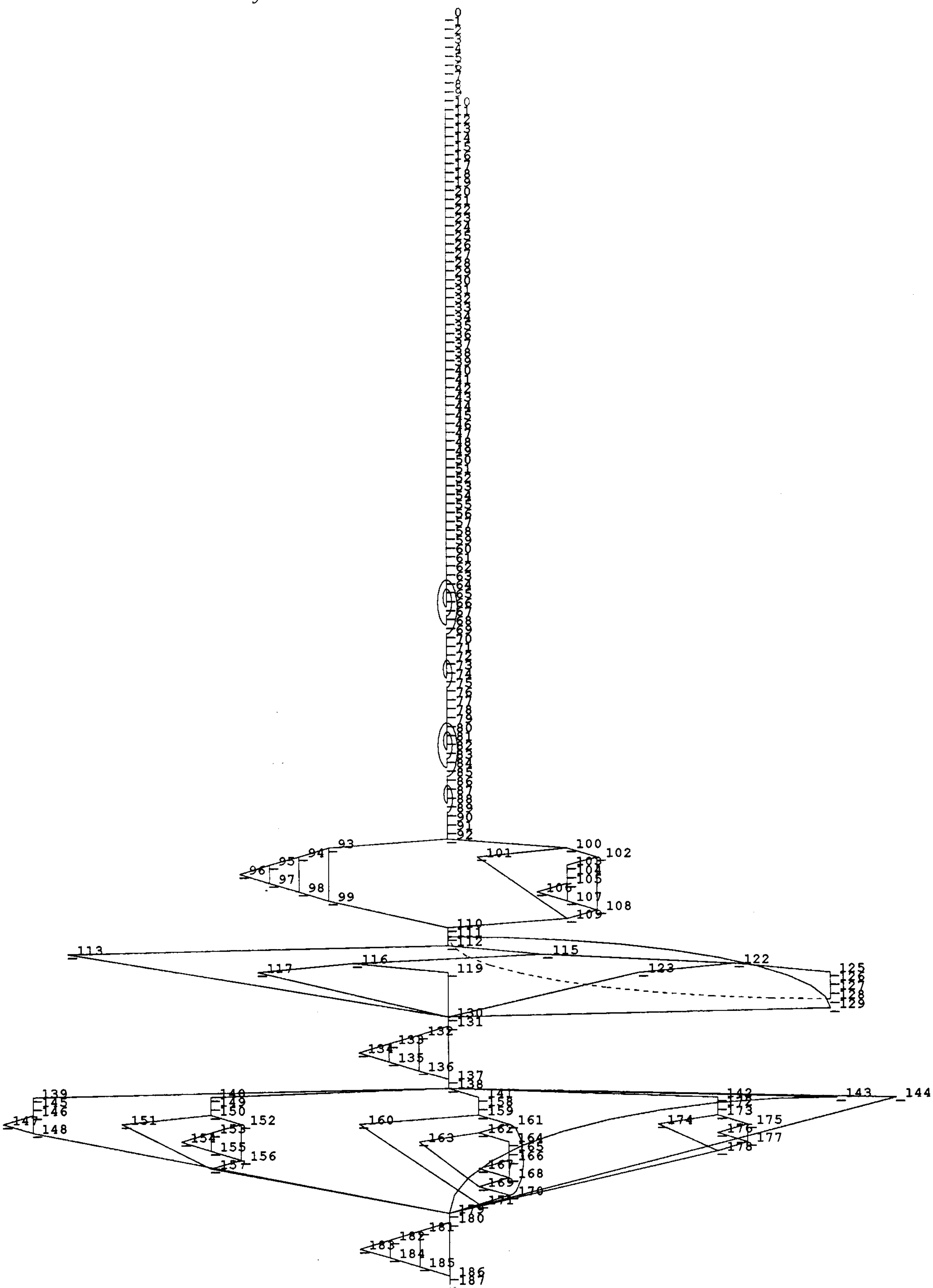


Figure B.9: GP Decision Tree:

Table B.50: GP Decisions -- see Figure B.9 for correspondence.

Graph Node Number	GP Decisions	TRUE output test cases	FALSE output test case
63	I in range (loop based on I)	Not a calculated loop counter; Testing not required	
64	J in range (loop based on I)	Not a calculated loop counter; Testing not required	
72	I in range (loop based on I)	Not a calculated loop counter; Testing not required	
79	I in range (loop based on I)	Not a calculated loop counter; Testing not required	
80	J in range (loop based on I)	Not a calculated loop counter; Testing not required	
86	I in range (loop based on I)	Not a calculated loop counter; Testing not required	
92	AE_SWITCH .EQ. K$AXIAL_ENGINES_ARE_OFF	GP_NR_001*	GP_NR_003*
93	RE_SWITCH .EQ. K$ROLL_ENGINES_ARE_ON	GP_NR_001*	GP_NR_105*
94	TD_SENSED .EQ. K$TOUCH_DOWN_NOT_SENSED	GP_NR_001*	GP_PST_001
95	GP_ALTITUDE(0) .LE. ENGINES_ON_ALTITUDE	GP_PST_003	GP_PST_002
100	TD_SENSED .EQ. K$TOUCH_DOWN_SENSED	GP_NR_003*	GP_NR_102*
102	GP_ALTITUDE(0) .LE. DROP_HEIGHT	GP_NR_007*	GP_NR_003*
106	SQRT(TEMP)+GP_VELOCITY(1,0) .LE. MAX_NORMAL_VELOCITY	GP_PST_004	GP_NR_007
112	I in range (loop based on I)	Not a calculated loop counter; Testing not required	
113	CONTOUR_ALTITUDE(I) .EQ. CUR_ALTITUDE	GP_PST_006	GP_PST_005
116	CONTOUR_ALTITUDE(I) .GT. CUR_ALTITUDE	GP_PST_005	GP_PST_007
117	I .GT. 1	GP_PST_005	GP_PST_008
123	(CONTOUR_ALTITUDE(I) .EQ. 0) .OR. (I .EQ. 100)	See MC/DC table for decision 123	
132	GP_ALTITUDE(0) .LE. ENGINES_ON_ALTITUDE	GP_PST_009	GP_PST_010
133	CONTOUR_CROSSED .EQ. K$CONTOUR_NOT_CROSSED	GP_PST_012	GP_PST_009
134	VELOCITY_ERROR .GE. 0	GP_PST_012	GP_PST_011
139-145	GOTO statement based on GP_PHASE	See Table based on GP_PHASE Decision	
147	GP_ALTITUDE(0) .LE. ENGINES_ON_ALTITUDE	GP_NR_001*	GP_RO_107
151	TD_SENSED .EQ. K$TOUCH_DOWN_SENSED	GP_NR_102	GP_NR_002*
153	AE_TEMP .EQ. K$HOT	GP_NR_004	GP_NR_003
154	CHUTE_RELEASED .EQ. K$CHUTE_RELEASED	GP_NR_004	GP_RO_110
160	TD_SENSED .EQ. K$TOUCH_DOWN_SENSED	GP_NR_104	GP_NR_005*
162	GP_ALTITUDE(0) .LE. DROP_HEIGHT	GP_NR_007*	GP_NR_008
163	TDS_STATUS .EQ. K$FAILED	GP_NR_006	GP_NR_008
168	SQRT(TEMP)+GP_VELOCITY(1,0) .LE. MAX_NORMAL_VELOCITY	GP_NR_008	GP_NR_007
175	TD_SENSED .EQ. K$TOUCH_DOWN_SENSED	GP_NR_105	GP_PST_013
177	TDS_STATUS .EQ. K$FAILED	GP_PST_014	GP_PST_013
182	CL .EQ. K$FIRST	GP_NR_001*	GP_NR_007*
183	OPTIMAL_VELOCITY .EQ. DROP_SPEED	GP_PST_015	GP_PST_019
184	GP_VELOCITY(1, 0) .LT. DROP_SPEED	GP_PST_016	GP_PST_015

"*" is used in the above table to indicate that there are more test cases that satisfy this decision branch but only one is listed for brevity.

Table B.51: MC/DC table for Decision 123.

CONTOUR_ALTITUDE(I) .EQ. 0 **(Con. 1)**	(I .EQ. 100) **(Con. 2)**	Final Decision	Test Case	Ind. of **Con 1**	Ind. of **Con 2**
0	0	0	GP_PST_007[A]	*	*
0	1	1	GP_PST_007[A]		*
1	0	1	GP_PST_017	*	

0 = FALSE value for the condition
1 = TRUE value for the condition
A: Test case GP_PST_007 iterates through decision-123 100 times. The first 99 iterations will exercise the 0,0 combination while the 100th iteration will exercise the 0,1 combination of the decision.

Table B.52: Expanded table for GP_PHASE Decision.

GP_PHASE	Test Case
1	GP_NR_001*
2	GP_NR_002*
3	GP_NR_005*
4	GP_NR_105
5	GP_PST_010
Out of range	GP_PST_020

Table B.53: MC/DC Entry/Exit requirements for Module inside GP.FOR:

Module	Test Case
DERIV_ATT	GP_NR_001*
DERIV_VEL	GP_NR_001*
DERIV_ALT	GP_NR_001*
MULT_ATT	GP_NR_001*
MULT_VEL	GP_NR_001*
AVG_ATT	GP_NR_001*
AVG_VEL	GP_NR_001*

3.5.10 GP Structural Testing

Code components tested for GP structural -based testing are identified in Table B.14. There are 21 test structure-based test cases. Only 20 are used to test GP code stricture. GP_PST_018 is not used for GP structural analysis because it test the same condition as GP_PST_007. It should also be noted that GP_PST_021 is used to test the ZERO_CHECK routine in the UTILITY.FOR file. This test case forces a negative-square-root to occur in the GP functional unit and is expected to cause a core dump. Hence even though an expected-values file is provided, it is not needed.

Table B.54: Summary of Structural Testing for GP Functional Unit.

TEST CASE NAME	EXECUTION DATE	RESULTS .ANA file/PR #	Reason for Test Run
GP_PST_xxx	4/11/95	N	Initial testing.

B.3.5.11 AECLP Structural Analysis

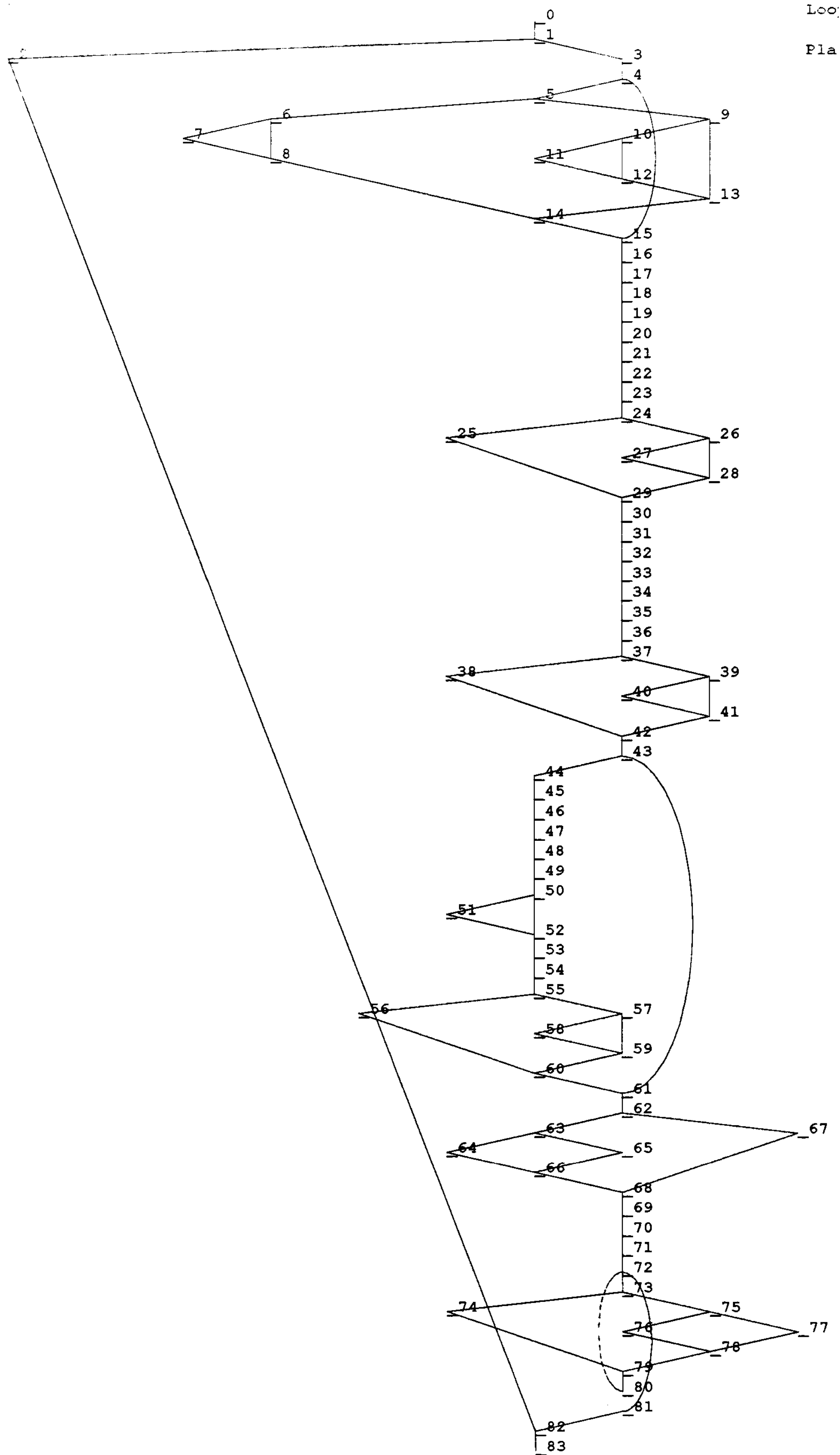


Figure B.10: AECLP Decision Tree

Table B.55: AECLP Decisions -- see Figure B.10 for correspondence

Graph Node Number	AECLP Decisions	TRUE output test cases	FALSE output test case
1	AE_SWITCH .EQ. K$AXIAL_ENGINES_ARE_OFF	AECLP_NR_008*	AECLP_NR_001*
4	GP_ALTITUDE(0) .LE. ENGINES_ON_ALTITUDE	AECLP_NR_001*	AECLP_RO_044*
5	AE_TEMP .EQ. K$COLD	AECLP_NR_001*	AECLP_NR_003*
6	(FRAME_COUNTER - FRAME_ENGINES_IGNITED) * DELTA_T .LT. FULL_UP_TIME	AECLP_NR_001*	AECLP_RO_41*
9	AE_TEMP .EQ. K$WARMING_UP	AECLP_NR_003*	AECLP_NR_005*
10	(FRAME_COUNTER - FRAME_ENGINES_IGNITED) * DELTA_T .GE. FULL_UP_TIME	AECLP_NR_003*	AECLP_RO_043*
24	PITCH_ERROR_LIMIT .LT. PE_MIN(CL)	AECLP_RO_027	AECLP_NR_001*
26	PITCH_ERROR_LIMIT .GT. PE_MAX(CL)	AECLP_RO_028	AECLP_NR_001*
37	YAW_ERROR_LIMIT .LT. YE_MIN(CL)	AECLP_RO_035	AECLP_NR_001*
39	YAW_ERROR_LIMIT .GT. YE_MAX(CL)	AECLP_RO_036	AECLP_NR_001*
43	CONTOUR_CROSSED .EQ. K$CONTOUR_CROSSED	AECLP_RO_005*	AECLP_NR_001*
50	OMEGA .NE. 0	AECLP_RO_005*	AECLP_PST_001
55	TE_LIMIT .LT. TE_MIN(CL)	AECLP_RO_029	AECLP_NR_001*
57	TE_LIMIT .GT. TE_MAX(CL)	AECLP_RO_030	AECLP_NR_001*
62	CHUTE_RELEASED .EQ. K$CHUTE_RELEASED	AECLP_NR_004*	AECLP_NR_001*
63	CONTOUR_CROSSED .EQ. K$CONTOUR_NOT_CROSSED	AECLP_NR_004	AECLP_NR_005
72	I IN RANGE	Not a calculated loop counter; Testing not required	
73	INTERNAL_CMD(I) .LT. 0	AECLP_NR_054	AECLP_NR_001*
75	INTERNAL_CMD(I) .LE. 1	AECLP_NR_001*	AECLP_NR_055

B.3.5.12 AECLP Structural Testing

Code components tested in AECLP structural testing are given in Table B.14. The results of structural testing is given in Table B.56. There are only two test cases in this suite. AECLP_PST_001 tests a decision in the AECLP functional unit. AECLP_PST_002 is designed to test the ZERO_CHECK subroutine in the UTILITY.FOR file. It forces a divide-by-zero to occur and is expected to cause a core dump. An expected values file is provided for this test case but is unnecessary. The objective of the test is to ensure that the exception message is displayed or printed. Hence this test case is not expected to run to completion.

Table B.56: Summary of Structural Testing for AECLP Functional Unit.

TEST CASE NAME	EXECUTION DATE	RESULTS .ANA file/PR #	Reason for Test Run
AECLP_PST_001	4/11/95	N	Initial testing
AECLP_PST_002		N	Initial testing

B.3.5.13 RECLP Structural Analysis

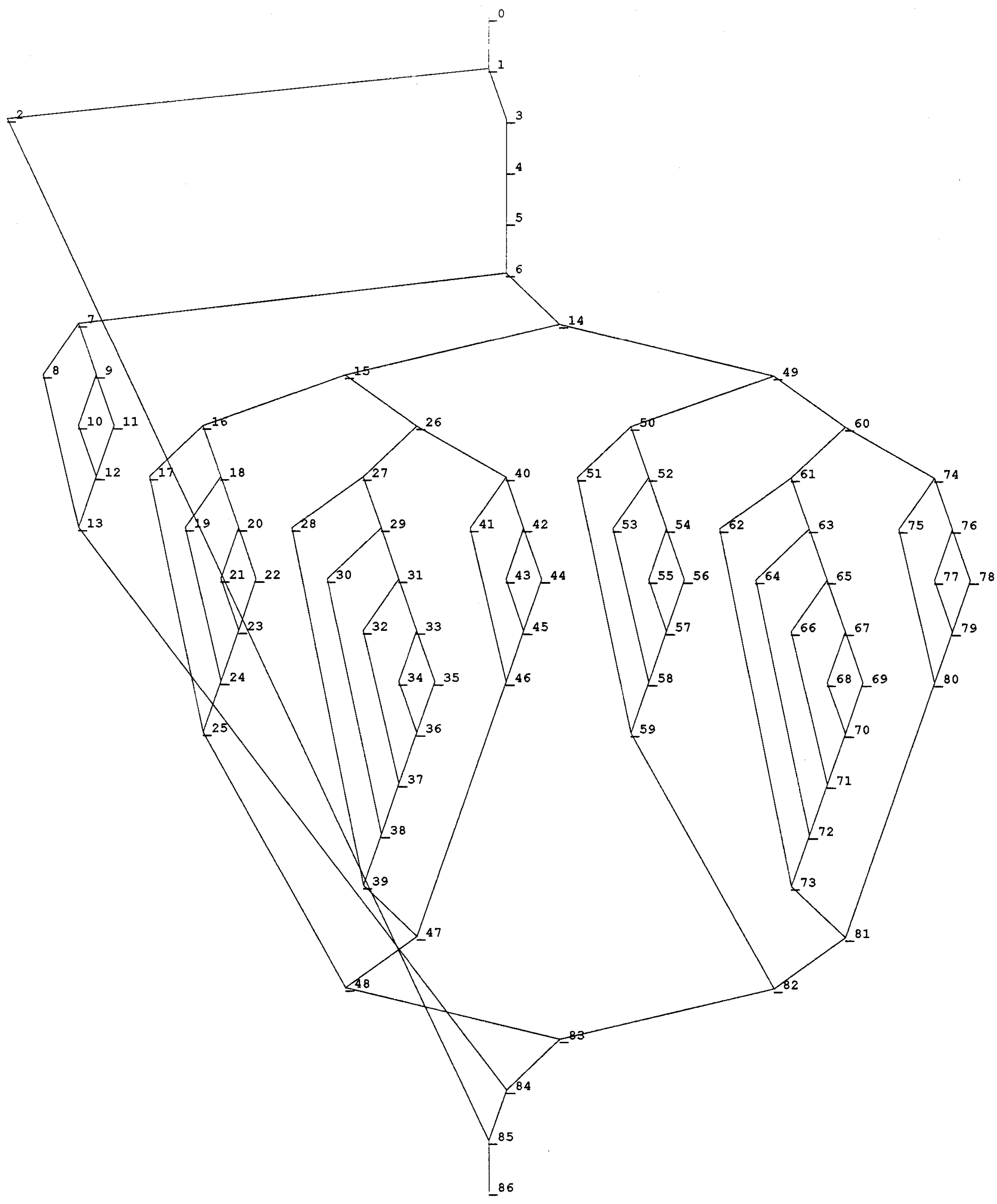


Figure B.11: RECLP Decision Tree:

Table B.57: RECLP Decisions -- see Figure B.11 for correspondence

Graph Node Number	RECLP Decisions	TRUE output test cases	FALSE output test case
1	RE_SWITCH .EQ. K$ROLL_ENGINES_ARE_OFF	RECLP_PST_003	RECLP_NR_001*
6	THETA .EQ. 0	RECLP_NR_059	RECLP_NR_001*
7	G_ROTATION(1, 0) .GT. P4	RECLP_NR_064	RECLP_NR_065
9	G_ROTATION(1, 0) .LT. -P4	RECLP_PST_011	RECLP_NR_065
14	THETA .GT. 0	RECLP_NR_001*	RECLP_NR_066*
15	THETA .LE. THETA1	RECLP_NR_001*	RECLP_NR_005
16	G_ROTATION(1, 0) .GT. P2	RECLP_NR_013*	RECLP_NR_001*
18	G_ROTATION(1, 0) .GT. P1	RECLP_PST_001	RECLP_NR_001*
20	G_ROTATION(1, 0) .GE. -P4	RECLP_NR_001	RECLP_PST_002
26	THETA .LE. THETA2	RECLP_NR_005*	RECLP_NR_009
27	G_ROTATION(1, 0) .GT. P2	RECLP_PST_004	RECLP_NR_005*
29	G_ROTATION(1, 0) .GT. P1	RECLP_NR_021	RECLP_NR_005*
31	G_ROTATION(1, 0) .GT. 0.0	RECLP_NR_008	RECLP_NR_005*
33	G_ROTATION(1, 0) .GE. -P4	RECLP_NR_005	RECLP_NR_043
40	G_ROTATION(1, 0) .GT. -P3	RECLP_NR_012*	RECLP_NR_039
42	G_ROTATION(1, 0) .GE. -P4	RECLP_NR_039	RECLP_PST_005
49	THETA .GE. -THETA1	RECLP_NR_002	RECLP_NR_063
50	G_ROTATION(1, 0) .GT. P4	RECLP_PST_006	RECLP_NR_002*
52	G_ROTATION(1, 0) .GE. -P1	RECLP_NR_002	RECLP_PST_007
54	G_ROTATION(1, 0) .GE. -P2	RECLP_PST_007	RECLP_PST_008
60	THETA .GE. -THETA2	RECLP_NR_006	RECLP_NR_010
61	G_ROTATION(1, 0) .GT. P4	RECLP_PST_009	RECLP_NR_006
63	G_ROTATION(1, 0) .GE. 0.0	RECLP_NR_007	RECLP_NR_006
65	G_ROTATION(1, 0) .GE. -P1	RECLP_NR_006	RECLP_NR_023
67	G_ROTATION(1, 0) .GE. -P2	RECLP_NR_023	RECLP_PST_010
74	G_ROTATION(1, 0) .GT. P4	RECLP_RO_063	RECLP_NR_010
76	G_ROTATION(1, 0) .GE. P3	RECLP_NR_010	RECLP_NR_011

B.3.5.14 RECLP Structural Testing

Table B.18 gives the code components tested by RECLP structural testing. The results are summarized in Table B.58 below. There are 11 test structure-based test cases in this suite.

Table B.58: Summary of Structural Testing for RECLP Functional Unit.

TEST CASE NAME	EXECUTION DATE	DATE CODE FETCHED	DATE TEST CASE FETCHED	RESULTS (was .ANA file generated Y or N?)	PR #
RECLP_PST_xxx	4/11/95	4/6/95	4/10/95	N	

B.3.5.15 CRCP Structural Analysis

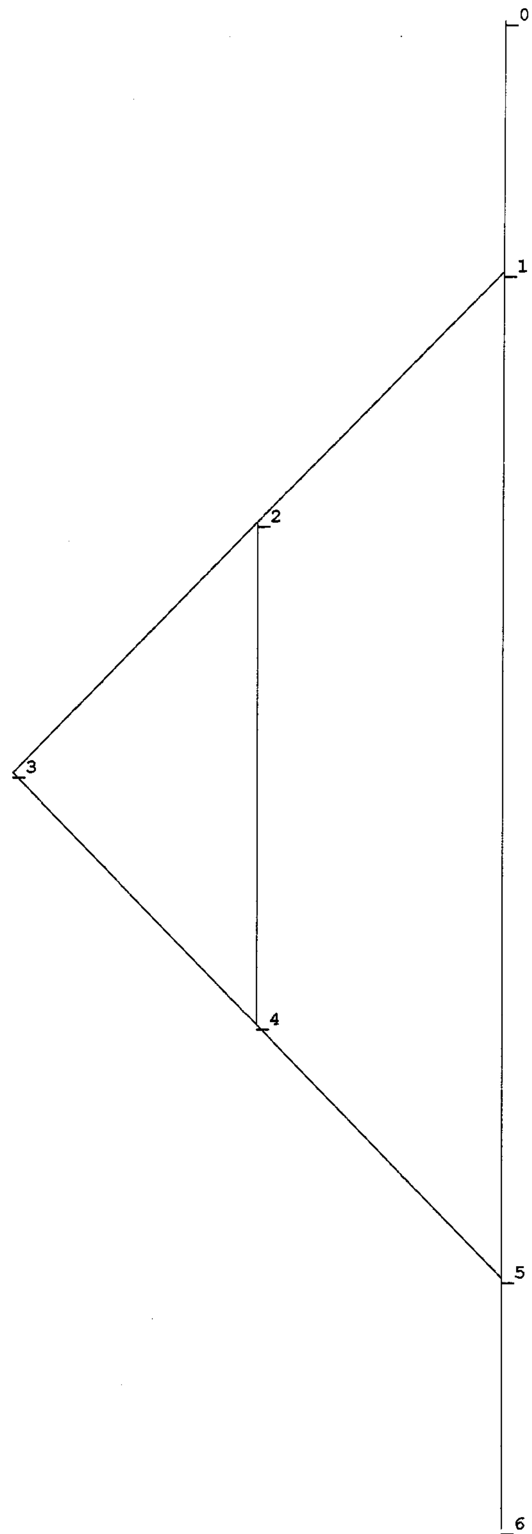


Figure B.12: CRCP Decision Tree

Table B.59: CRCP Decisions -- see Figure B.12 for correspondence

Graph Node Number	CRCP Decisions	TRUE output test cases	FALSE output test case
1	CHUTE_RELEASED .EQ. K$CHUTE_ATTACHED	CRCP_NR_001*	CRCP_NR_002*
2	AE_TEMP .EQ. K$HOT	CRCP_NR_005	CRCP_NR_001*

No structure-based test cases are needed for CRCP.

B.3.5.16 Utility Subroutines Structural Analysis

Utility routines are used throughout the various functional units for range checking, as well as checking for negative and zero numbers. These test cases are executed along with the functional units.

Table B.60: RANGE_CHECK Subroutine Decisions:

Graph Node Number	RANGE_CHECK Decisions	TRUE output test cases	FALSE output test case
	source .LT. lower_bound	GSP_RO_002*	ASP_NR_001*
	source .GT. upper_bound	GP_RO_003	ASP_NR_001*

Table B.61: NEG_VALUE_CHECK Subroutine Decisions:

Graph Node Number	NEG_VALUE_CHECK Decisions	TRUE output test cases	FALSE output test case
	source .LT. 0	GP_PST_021	GP_NR_007

Table B.62: ZERO_CHECK Subroutine Decisions:

Graph Node Number	ZERO_CHECK Decisions	TRUE output test cases	FALSE output test case
	source .EQ. 0	AECLP_PST_002	AECLP_NR_001

B.4 Traceability Matrix for Pluto Design and Code

This section gives the traceability matrix to match Pluto design and code elements to the GCS requirements.

Table B.4-1: Pluto Traceability Matrix

Functional Requirements	DESIGN	CODE
0-1 Specify four separate, globally accessible data stores: EXTERNAL, GUIDANCE_STATE, RUN_PARAMETERS, and SENSOR_OUTPUT.	DFD 1 DFD 2 DFD 3	Guidance_state.for Run_Parameters.for Sensor_output.for External.for
2-1 Control flow of the frame processing.		
2-1.1 The appropriate control flow for a frame is: call to GCS_SIM_RENDEZVOUS. Satisfy the Sensor Processing subframe requirements (2-2). call to GCS_SIM_RENDEZVOUS. Satisfy Guidance Processing subframe requirements (2-3). call to GCS_SIM_RENDEZVOUS fulfill Control Law Processing subframe requirements (2-4) or terminate (2-1.2).	PAT 0-s1 PAT 1-s1 PAT 2-s1 PAT 3-s1	Program Pluto Subroutine SPSF Subroutine GPSF Subroutine CLPSF
2-1.2 The implementation is to terminate immediately upon completion of the Control Law Processing subframe requirements during the frame in which GP_PHASE is set to 5.	DFD 2 P_Spec. 2.2 PAT 0-s1	Program Pluto
2-2 Sensor Processing subframe requirements.		
2-2.1 Satisfy the TSP requirements (2.1.5) prior to fulfilling any of the other requirements in (2.1.1 and 2.1.4).	PAT 1-s1	Subroutine SPSF
2-2.2 Satisfy all requirements in the sensor processing requirements hierarchy (2.1).	PAT 1-s1	Subroutine SPSF
2-2.3 Satisfy all requirements in the communications processing requirements (2.4) upon satisfying 2-2.1.	PAT 1-s1 P_Spec. 1.8	Subroutine SPSF Subroutine CP
2-2.4 Adhere to the functional unit scheduling in Table 4.3 of the GCS specification.	PAT 0-s1	Subroutine SPSF
2-3 The Guidance Processing subframe requirements.		
2-3.1 Satisfy all requirements in the guidance processing requirements (2.2).	PAT 2-s1	Subroutine GPSF
2-3.2 Satisfy all requirements in the communications processing requirements (2.4) upon satisfying 2-3.1.	PAT 2-s1 P_Spec. 2.3	Subroutine GPSF Subroutine CP
2-4 The Control Law Processing subframe requirements.		
2-4.1 Satisfy the AECLP requirements (2.3.1) prior to fulfilling any of the CRCP requirements (2.3.3).	PAT 3-s1	Subroutine CLPSF Subroutine AECLP
2-4.2 Satisfy all requirements in the control law processing requirements hierarchy (2.3).	PAT 3-s1	Subroutine CLPSF
2-4.3 Satisfy all requirements in the communications processing requirements (2.4) upon satisfying 2-4.1.	PAT 3-s1 P_Spec. 3.5	Subroutine CLPSF Subroutine CP
2-4.4 Adhere to the functional unit scheduling in Table 4.3 of the GCS specification.	PAT 3-s1	Subroutine CLPSF
2.1 SP -- Sensor Processing		
2.1.1 ASP -- Accelerometer Sensor Processing		
2.1.1-1 Rotate variables.	P_Spec 1.3 (step 1)	Subroutine ASP
2.1.1-2 Adjust gain for temperature.	P_Spec 1.3 (step 3)	Subroutine ASP
2.1.1-3 Remove characteristic bias.	P_Spec 1.3 (step 3)	Subroutine ASP
2.1.1-4 Correct for misalignment.	P_Spec 1.3 (step 3)	Subroutine ASP
2.1.1-5 Determine Accelerations.		
2.1.1-5.1 Acceleration based on current A_COUNTER.	P_Spec 1.3 (step 3)	Subroutine ASP
2.1.1-5.2 Acceleration based on mean of previous accelerations.	P_Spec 1.3 (step 3)	Subroutine ASP
2.1.1-6 Determine Accelerometer Status		
2.1.1-6.1 A_STATUS = healthy	P_Spec 1.3 (step 2)	Subroutine ASP
2.1.1-6.2 A_STATUS = unhealthy	P_Spec 1.3 (step 2)	Subroutine ASP
2.1.2 ARSP -- Altimeter Radar Sensor Processing		
2.1.2-1 Rotate variables.	P_Spec 1.2 (step 1)	Subroutine ARSP
2.1.2-2 Determine altitude when echo is received. (based on AR_COUNTER)	P_Spec 1.2 (step 3A)	Subroutine ARSP
2.1.2-3 Determine altitude when echo is not received		
2.1.2-3.1 Determine altitude based on third-order polynomial.	P_Spec 1.2 (step 2B)	Subroutine ARSP
2.1.2-3.2 Determine altitude based on previous calculation.	P_Spec 1.2 (step 2C)	Subroutine ARSP

2.1.2-4	Set altimeter radar status.		
2.1.2-4.1	AR_STATUS = healthy	P_Spec 1.2 (step 2)	Subroutine ARSP
2.1.2-4.2	AR_STATUS = failed	P_Spec 1.2 (step 2)	Subroutine ARSP
2.1.2-5	Set values of K_ALT.		
2.1.2-5.1	K_ALT = 1	P_Spec 1.2 (step 2)	Subroutine ARSP
2.1.2-5.2	K_ALT = 0	P_Spec 1.2 (step 2)	Subroutine ARSP
2.1.3	**TDLRSP -- Touch Down Landing Radar Sensor Processing**		
2.1.3-1	Rotate variables	P_Spec 1.5 (step 1)	Subroutine TDLRSP
2.1.3-2	Determine state for each radar beam.		
2.1.3-2.1	TDLR_STATE = unlocked.	P_Spec 1.5 (step 3A)	Subroutine TDLRSP
2.1.3-2.2	TDLR_STATE = locked.	P_Spec 1.5 (step 3A)	Subroutine TDLRSP
2.1.3-3	Determine Whether to set FRAME_BEAM_UNLOCKED		
2.1.3-3.1	Set FRAME_BEAM_UNLOCKED to FRAME_COUNTER	P_Spec 1.5 (step 3A)	Subroutine TDLRSP
2.1.3-3.2	Leave FRAME_BEAM_UNLOCKED unchanged	P_Spec 1.5 (step 3A)	Subroutine TDLRSP
2.1.3-4	Calculate the beam velocities	P_Spec 1.5 (step 3B)	Subroutine TDLRSP
2.1.3-5	Process beam velocities based on which beam(s) locked.		
2.1.3-5.1	no beams locked	P_Spec 1.5 (step 3C)	Subroutine TDLRSP
2.1.3-5.2	Beam1 locked	P_Spec 1.5 (step 3C)	Subroutine TDLRSP
2.1.3-5.3	Beam2 locked	P_Spec 1.5 (step 3C)	Subroutine TDLRSP
2.1.3-5.4	Beam3 locked	P_Spec 1.5 (step 3C)	Subroutine TDLRSP
2.1.3-5.5	Beam4 locked	P_Spec 1.5 (step 3C)	Subroutine TDLRSP
2.1.3-5.6	Beam1 & Beam2 locked	P_Spec 1.5 (step 3C)	Subroutine TDLRSP
2.1.3-5.7	Beam1 & Beam3 locked	P_Spec 1.5 (step 3C)	Subroutine TDLRSP
2.1.3-5.8	Beam1 & Beam4 locked	P_Spec 1.5 (step 3C)	Subroutine TDLRSP
2.1.3-5.9	Beam2 & Beam3 locked	P_Spec 1.5 (step 3C)	Subroutine TDLRSP
2.1.3-5.10	Beam2 & Beam4 locked	P_Spec 1.5 (step 3C)	Subroutine TDLRSP
2.1.3-5.11	Beam3 & Beam4 locked	P_Spec 1.5 (step 3C)	Subroutine TDLRSP
2.1.3-5.12	Beam1, Beam2, & Beam3 locked	P_Spec 1.5 (step 3C)	Subroutine TDLRSP
2.1.3-5.13	Beam1, Beam2, & Beam4 locked	P_Spec 1.5 (step 3C)	Subroutine TDLRSP
2.1.3-5.14	Beam1, Beam3, & Beam4 locked	P_Spec 1.5 (step 3C)	Subroutine TDLRSP
2.1.3-5.15	Beam2, Beam3, & Beam4 locked	P_Spec 1.5 (step 3C)	Subroutine TDLRSP
2.1.3-5.16	Beam1, Beam2, Beam3, & Beam4 locked	P_Spec 1.5 (step 3C)	Subroutine TDLRSP
2.1.3-6	Convert to body velocities.	P_Spec 1.5 (step 3D)	Subroutine TDLRSP
2.1.3-7	Set values in K_MATRIX.		
2.1.3-7.1	Kx = 0	P_Spec 1.5 (step 4)	Subroutine TDLRSP
2.1.3-7.2	Kx = 1	P_Spec 1.5 (step 4)	Subroutine TDLRSP
2.1.3-7.3	Ky = 0	P_Spec 1.5 (step 4)	Subroutine TDLRSP
2.1.3-7.4	Ky = 1	P_Spec 1.5 (step 4)	Subroutine TDLRSP
2.1.3-7.5	Kz = 0	P_Spec 1.5 (step 4)	Subroutine TDLRSP
2.1.3-7.6	Kz = 1	P_Spec 1.5 (step 4)	Subroutine TDLRSP
2.1.3-8	Set TDLR_STATUS.	P_Spec 1.5 (step 2)	Subroutine TDLRSP
2.1.4	**GSP -- Gyroscope Sensor Processing**		
2.1.4-1	Rotate variables.	P_Spec 1.4 (step 1)	Subroutine GSP
2.1.4-2	Determine the vehicle rotation rates along each of the vehicle's three axes.		
2.1.4-2.1	Adjust gain.	P_Spec 1.4 (step 3)	Subroutine TDLRSP
2.1.4-2.2	Convert G_COUNTER.	P_Spec 1.4 (step 3)	Subroutine TDLRSP
2.1.4-3	Set gyroscope status to healthy.	P_Spec 1.4 (step 2)	Subroutine TDLRSP
2.1.5	**TSP -- Temperature Sensor Processing**		
2.1.5-1	Calculate solid state temperature	P_Spec 1.7 (step 2A)	Subroutine TSP
2.1.5-2	Calculate Thermal Temperature	P_Spec 1.7 (step 2C)	Subroutine TSP
2.1.5-3	Determine which Temperature to use (SS or Thermocouple)		
2.1.5-3.1	Calculate the Thermo sensor upper limit	P_Spec 1.7 (step 2B)	Function UPPER_PARABOLIC _FUNCTION
2.1.5-3.2	Calculate the Thermo sensor lower limit	P_Spec 1.7 (step 2B)	Function LOWER_PARABOLIC _FUNCTION
2.1.5-4	Determine Atmospheric Temperature	P_Spec 1.7 (step 2B & 2C)	Subroutine TSP
2.1.5-5	Set status to healthy.	P_Spec 1.7 (step 1)	Subroutine TSP
2.1.6	**TDSP -- Touch Down Sensor Processing**		
2.1.6-1	Determine status of touch down sensor.	P_Spec 1.6 (step 1)	Subroutine TDSP
2.1.6-2	Determine whether touch down has been sensed.	P_Spec 1.6 (step 2)	Subroutine TDSP
2.2	**GP -- Guidance Processing**		
2.2-1	Rotate variables.	P_Spec 2.2 (step 1)	Subroutine GP
2.2-2	Determine the attitude, velocities, and altitude.		
2.2-2.1	Set up the GP_ROTATION matrix.	P_Spec 2.2 (step 2)	Subroutine DERIV_ATT

2.2-2.2	Calculate new values of attitude, velocity, and altitude.	P_Spec 2.2 (step 2)	Subroutine: GP DERIV_ATT DERIV_VEL DERIV_ATT MULT_ATT MULT_VEL MULT_ATT
2.2-3	Determine if the engines should be on or off.		
2.2-3.1	Engines on	P_Spec 2.2 (step 3)	Subroutine GP
2.2-3.2	Engines off	P_Spec 2.2 (step 3)	Subroutine GP
2.2-4	Set FRAME_ENGINES_IGNITED	P_Spec 2.2 (step 3)	Subroutine GP
2.2-5	Determine velocity error.	P_Spec 2.2 (step 4)	Subroutine GP
2.2-6	Determine optimal velocity	P_Spec 2.2 (step 4)	Subroutine GP
2.2-7	Determine if contour has been crossed.	P_Spec 2.2 (step 5)	Subroutine GP
2.2-8	Determine guidance phase.		
2.2-8.1	GP_PHASE = 1	P_Spec 2.2 (step 6)	Subroutine GP
2.2-8.2	GP_PHASE = 2	P_Spec 2.2 (step 6)	Subroutine GP
2.2-8.3	GP_PHASE = 3	P_Spec 2.2 (step 6)	Subroutine GP
2.2-8.4	GP_PHASE = 4	P_Spec 2.2 (step 6)	Subroutine GP
2.2-8.5	GP_PHASE = 5	P_Spec 2.2 (step 6)	Subroutine GP
2.2-9	Determine which set of control law parameters to use.		
2.2-9.1	CL = 1	P_Spec 2.2 (step 7)	Subroutine GP
2.2-9.2	CL = 2	P_Spec 2.2 (step 7)	Subroutine GP
2.3	**CLP -- Control Law Processing**		
2.3.1	**AECLP -- Axial Engine Control Law Processing**		
2.3.1-1	Generate the appropriate axial engine commands when AE_CMD=ON.		
2.3.1-1.1	Determine engine temperature		
2.3.1-1.1.1	AE_TEMP = COLD	P_Spec 3.2 (step 2A)	Subroutine AECLP
2.3.1-1.1.2	AE_TEMP = WARM	P_Spec 3.2 (step 2A)	Subroutine AECLP
2.3.1-1.1.3	AE_TEMP = HOT	P_Spec 3.2 (step 2A)	Subroutine AECLP
2.3.1-1.2	Compute limiting errors for pitch	P_Spec 3.2 (step 2B)	Subroutine AECLP
2.3.1-1.3	Compute limiting error for yaw	P_Spec 3.2 (step 2C)	Subroutine AECLP
2.3.1-1.4	Compute limiting error for thrust	P_Spec 3.2 (step 2D)	Subroutine AECLP
2.3.1-1.5	Compute pitch, yaw, and thrust errors.		
2.3.1-1.5.1	CHUTE_RELEASED = 1	P_Spec 3.2 (step 2E)	Subroutine AECLP
2.3.1-1.5.2	CHUTE_RELEASED = 0	P_Spec 3.2 (step 2E)	Subroutine AECLP
2.3.1-1.5.3	CONTOUR_CROSSED = 1	P_Spec 3.2 (step 2E)	Subroutine AECLP
2.3.1-1.5.4	CONTOUR_CROSSED = 0	P_Spec 3.2 (step 2E)	Subroutine AECLP
2.3.1-1.6	Compute INTERNAL_CMD	P_Spec 3.2 (step 2F)	Subroutine AECLP
2.3.1-1.7	Compute axial engine valve settings (AE_CMD).		
2.3.1-1.7.1	when INTERNAL_CMD < 0.0	P_Spec 3.2 (step 2G)	Subroutine AECLP
2.3.1-1.7.2	when 0.0 £ INTERNAL_CMD ≥ 1.0	P_Spec 3.2 (step 2G)	Subroutine AECLP
2.3.1-1.7.3	when 1.0 < INTERNAL_CMD	P_Spec 3.2 (step 2G)	Subroutine AECLP
2.3.1-2	Generate the appropriate axial engine commands when AE_CMD=OFF.		
2.3.1-2.1	Set AE_CMD = 0	P_Spec 3.2 (step 2)	Subroutine AECLP
2.3.1-3	Set axial engine status to healthy.	P_Spec 3.2 (step 1)	Subroutine AECLP
2.3.2	**RECLP -- Roll Engine Control Law Processing**		
2.3.2-1	Generate the appropriate roll engine command.	P_Spec 3.4 (step 2)	Subroutine RECLP
2.3.2-2	Set roll engine status to healthy.	P_Spec 3.4 (step 1)	Subroutine RECLP
2.3.3	**CRCP -- Chute Release Control Processing**		
2.3.3-1	Determine appropriate parachute release command.		
2.3.3-1.1	AE_TEMP = COLD	P_Spec 3.3	Subroutine CRCP
2.3.3-1.2	AE_TEMP = WARM	P_Spec 3.3	Subroutine CRCP
2.3.3-1.3	AE_TEMP = HOT	P_Spec 3.3	Subroutine CRCP
2.3.3-1.4	CHUTE_RELEASED = 0	P_Spec 3.3	Subroutine CRCP
2.3.3-1.5	CHUTE_RELEASED = 1	P_Spec 3.3	Subroutine CRCP
2.4	CP -- Communications Processing		
2.4-1	Set communicator status to healthy.	P_Spec 1.8 (step 1)	Subroutine CP
2.4-2	Get synchronization pattern.	P_Spec 1.8 (step 2A)	Subroutine CP
2.4-3	Determine sequence number.	P_Spec 1.8 (step 2B)	Subroutine CP
2.4-4	Prepare sample mask.		
2.4-4.1	Subframe 1 mask	P_Spec 1.8 (step 2C)	Subroutine CP
2.4-4.2	Subframe 2 mask	P_Spec 1.8 (step 2C)	Subroutine CP
2.4-4.3	Subframe 3 mask	P_Spec 1.8 (step 2C)	Subroutine CP
2.4-5	Prepare data section.		
2.4-5.1	Use subframe 1 data	P_Spec 1.8 (step 2D)	Subroutine CP
2.4-5.2	Use subframe 2 data	P_Spec 1.8 (step 2D)	Subroutine CP
2.4-5.3	Use subframe 3 data	P_Spec 1.8 (step 2D)	Subroutine CP
2.4-2.5	Calculate checksum.	P_Spec 1.8 (step 2E)	Function CRC16

Appendix C: Review Records for the Pluto Implementation of the Guidance and Control Software

Author: Kelly J. Hayhurst, NASA Langley Research Center

This document was produced as part of Guidance and Control Software (GCS) Project conducted at NASA Langley Research Center. Although some of the requirements for the Guidance and Control Software application were derived from the NASA Viking Mission to Mars, this document does not contain data from an actual NASA mission.

C. Contents

C.1 Pluto Preliminary Design Review

Attendees: Kelly Hayhurst (SQA representative/Moderator)
Rob Angellatta (Verification Analyst/Recorder, Inspector)
Paul Carter (Programmer/Reader, Inspector)
Bernice Becher (System Analyst/Inspector)

C.1.1 Review Notes from Preliminary Design Review

Session 1: 9/16/93 9:30 a.m. - 11:30 p.m.

Reviewed Design Review procedures and roles prior to starting review

High-Level Structured Analysis Diagrams

Context diagram

B - 1 -- Initialization Data not used exactly as in spec; also mix of control and data flow

INIT_RUN_GCS does not show AE_SWITCH and RE_SWITCH as outputs (should)

B - 50 -- FRAME_COUNTER and SUBFRAME_COUNTER not shown as input from GCS_SIM

INIT_RUN_GCS

B-78, R-4 -- INIT_GCS -- no need for design to redo initialization

B-16 -- problems with data stores RUN_PARAMETERS, GUIDANCE_STATE, and SENSOR_OUTPUT

B-52 -- complete set of flows into and out of EXTERNAL are missing

problem with the external data flow -- not consistent with spec

B-81 -- problem with raw sensor data

GENERATE_SEQUENCE_PARAMS -- need algorithmic solution -- more detail

B-109, R-6 -- problem with order of activation

------------------------------- END OF SESSION 1 ------------------------------------

Session 2: 9/17/93 9:30 a.m. - 11:30 a.m.

High Level Diagrams

B-119, R-7 -- unclear how RENDEZVOUS is invoked

B-73 -- unclear function of RENDEZVOUS_CNTL and RENDEZVOUS_CNTL_STORE

B-98 -- unclear need of P-Spec COPY_CONTROL_DATA -- need to clarify and justify

B-100, B-125 -- copying and use of SUBFRAME_COUNTER is unclear

B-81 -- unclear function/need for STORE_RAW_SENSOR_DATA

B-84 -- unclear function/need for INIT_RUN_PARAM_STORE

B-86 -- unclear function/need of INIT_GUIDANCE_STATE_STORE

B-92, B-6, R10 -- TS_STATUS is not an input to TSP

B-96, B-108, B-21 -- inconsistent use of labels for bubbles for the functional units

PAT for RUN_GCS

B-70 -- PAT seems to be changing SUBFRAME_COUNTER -- but should not

B-71 -- some processes that should be activated are not activated when ITH_FRAME_2 and ITH_FRAME_5

B-72 -- some processes which should be activated are not from line 3 of PAT

P_Spec INIT_GCS

B-74 -- used same label for stores and processes

B-75 -- ambiguous notation

B-76, B-79 -- problem with copying group flow names

------------------------------- END OF SESSION 2 -------------------------------------

Session 3: 9/20/93 1:00 p.m. - 3:00 p.m.

__TSP__

B-7, B-19 -- what is need/function of Data Expand and Data Compress? -- appears that process is trying to manipulate names (same is true for ARSP, B-19; TDSP, B-140; TDLRSP, B-29; RECLP, B-226; GP, B-170; GSP, B-140; CRCP, B-193; CP, B-230; ASP, B-148; AECLP, B-198)

B-11-12 -- need more explanation of approach to determine solid state temperature

implicit assumption in the spec that M4>M3 and T4>T3 --> MAY WANT TO MOD SPEC

B-135 -- TS_STATUS has not been checked for limits violations --> MAY NEED TO REWORD SPEC ON EXCEPTION HANDLING

B-8 -- problem with conditions involving THERMO_TEMP for setting ATMOSPHERIC_TEMP -- may have introduced a condition that is not necessary

all locals are real*4 as opposed to real*8 -- where all reals in spec are real*8 --> MAY WANT TO MOD SPEC WITH REGARD TO PRECISION

__ARSP__

B-20, R-13 -- need to provide more data for Shift Data and clarify
need to consistently notate comments

B-22 -- problem with rotating/shifting data at right time -- need to correct

B-23, R-14 -- notation ".*" is confusing/inconsistent

B-121, B-15 -- notation ".[previous value]" also confusing

B-134 -- Is it necessary to check all history variables; not clear which variables are being checked --> MAY NEED TO MOD SPEC

B24 -- AR_FREQUENCY does not have 0 in the valid range -- no need to check this variable since it is a RUN_PARAMETER

B25 -- need to make sure AR_FREQUENCY*2 is in denominator -- with given notation, it is not clear

B-15, R-18 -- problem with first step in Newton Divided Differences -- need to specify order

B-105 -- need to specify order of subtraction

R-19 -- need to clarify references to indexes so that they are consistent -- consistency between using "last" and "most recent"

R20 --is AR_ALTITUDE checked for limit exceeded where it needs to be?

B-82 -- check on consistency of applying limit checks

------------------------------- END OF SESSION 3 ------------------------------------

Session 4: 9/23/93 9:30 a.m. - 11:30 a.m.

TDLRSP

R-25 -- reference to TDLR_VELOCITYV seems to be a typo

B-32 -- seems that things are being rotated twice as often as necessary

B-130-131 -- not clear which time history is being checked for TDLR_STATE, FRAME_BEAM_UNLOCKED

B-33, R-26 -- FRAME_BEAM_UNLOCKED is not supposed to be changed (see line 2 Table 5.11)

B-34 -- FRAME_BEAM_UNLOCKED needs to be set as per line 3 Table 5.11 -- but it is not

B-136 -- some confusion about processing a table --> MAY WANT TO MOD SPEC

B-32, R-27 -- insufficient detail provided for calculating average beam velocities -- need to give equations (also reference to table should be corrected)

B-28 -- TDLR_STATUS is shown as input to TDLRSP - but it should not be

B-126 -- not clear where control loops must be

TDSP

B-141 -- extra functionality present by having statement “TDS_STATUS has bad value ...”

------------------------------- END OF SESSION 4 ------------------------------------

Session 5: 9/30/93 9:30 a.m. - 11:00 a.m.

ASP

B-156, R30 -- locals are declared just as real -- when some are real*8

B-158 -- no apparent reason to make assignment of ATMOSPHERIC_TEMP

B-159, R32 -- when calculating abbreviations accel.* -- it is not clear there is a matrix multiplication

B-161 -- need to check limits for A_ACCELERATION; also problem when all accelerations are equal when you go to calculate standard deviation

GSP

R-34 -- G_STATUS is not an input

B-162 -- there is no limit check for G_STATUS

------------------------------- END OF SESSION 5 ------------------------------------

Session 6: 10/6/93 9:30 a.m. - 11:30 a.m.

GP

B-180 -- variable END_GCS is missing from the output section

some control signals are being used in high-level diagrams -- but they have not been seen at the lower-level p-specs. Why are some set and others not?

not clear what is a comment and what is pseudo-code -- the design should have a convention for comments and pseudo-code

B-175 -- need to use the simultaneous Runge Kutta method (Current design uses a sequential approach)

B-176 -- GP_ROTATION matrix is not handled properly.

B-179, B59 -- combined tables 5.9 and 5.10 into 1 algorithm -- but it is not done correctly.

B-184, R-50 -- CONTOUR_VELOCITY array -- this is not the right array to be searched. Also, numerous ambiguities in the description of the search

B-186 -- the computation of VELOCITY_ERROR is done conditionally in the design -- but should be done unconditionally

B-188, R-52 -- velocity error is not being calculated correctly

B-189 -- in determining contour crossed -- in the conditional expression GP_ALTITUDE <= ENGINES_ON_ALTITUDE and VELOCITY_ERROR . 0 -- this is not correct

B-192 -- the term optimal velocity is not explained

B-190 -- references to GP at 2.7 should be 2.6

B-178 -- the term "tnow" has not been defined

B-181, R45-46, R-48 -- problem with limit checks

B-191 -- in the description of doing Runge Kutta -- the equations for the derivatives do not provide sufficient detail to be translated into code

CRCP

general comments -- Expand/Compress functions not needed and title consistency

-------------------------------- END OF SESSION 6 -------------------------------------

Session 7: 10/12/93 9:30 a.m -- 11:30 a.m.

AECLP

B-197, R-53 -- not clear how to determine axial engine temperature

B-215, R-60 -- conditional (page 6) dealing with AE_TEMP appears to be added functionality

B-200, R-54 -- problem with dimensions of arrays

B-209, R-57 -- check for upper bound of CONTOUR_CROSSED is not correct; also general problems with limit checks

B-201, R-55-56 -- need absolute values in calculating THETA

B-212 -- THETA is declared as a local variable -- but THETA is in a global data store

B-202 -- calculation of limiting pitch error is unnecessarily broken down into 2 steps

B-204 -- equation for Q_TEMP is incorrect

B-205, R-58 -- error in calculating TE_LIMIT -- does not properly reflect bounding process

B-214 -- the nested Ifs may not be verifiable/modifiable

B-206 -- problem with correctly giving error messages (unnecessary duplication of error messages)

B-216 -- 3 variable, PITCH_ERROR, YAW_ERROR, and THRUST_ERROR are needlessly set there -- but not used

B-210 -- introduces INT_CMD -- not necessary

B-211 -- need to show derivation of TE_LIMIT

B-207 -- does not show rounding of AE_CMD

RECLP

R-61 -- RE_STATUS is shown as an input - but should not be

B-220 -- problem with notation of G_ROTATION

B-222, R-62 -- need additional detail in determining roll engine command

B-224 -- PI is not defined

B-225 -- problems with limit checking regarding THETA and missing for RE_CMD, RE_STATUS

B-221 -- reference to Fig 5.1 pg 60 is not correct

B-223 00 need to define term “lowest bit” (need more precise description)

B-219 -- duplication of giving error message

------------------------------- END OF SESSION 7 ------------------------------------

Session 8: 10/14/93 1:30 p.m. -- 3:30 p.m.

CP

B-254 -- remove stuff (like end of CP P-spec) that is not necessary

B-232 -- ITH_FRAME_2 and ITH_FRAME_5, and NBYTES and BYTE_PACKET are missing from input/output section

B-249 -- BYTE_PACKET is not accurately defined in data dictionary

B-259 -- need to define notation “B” used in defining INIT_SAMPLE_MASK

B-240 -- GP_ROTATION and K_MATRIX are missing from the packet variables table

B-243 -- K_ALT and K_MATRIX are missing from the list of variables for sample mask when ITH_FRAME_2 is true and ITH_FRAME_5 is false

B-244, B-252-253 -- the variables to be loaded are ambiguously described

B-251 -- bits for K_ALT and K_MATRIX are missing

B-255 -- uses SUB_FRAME_COUNTER -- which is not defined -- should be SUBFRAME_COUNTER

B-245 -- insufficient detail in determining the total number of bytes

B-242, B-257 -- comment refers to "lower" 16 bits of CHECKSUM -- but CHECKSUM has only 16 bits -- comment needs to be more precise

B-258 -- the action to set C_STATUS to healthy is not done to calculating CHECKSUM and loading BYTE_PACKET

B-246 -- when K_MATRIX and/or GP_ROTATION are loaded -- these are supposed to be stated in a special way and the design does not address this -- but need to

B-247 -- the design is not specific about which history variables are being loaded

B-248 -- need a better explanation of getting masks and packets

R-63 -- several variables are shown as input on the CFD/DFD but are not shown in the spec as input

should not have 2 different P-Specs for Expand (it appears that one is never called)

B-238 -- CFD/DFD does not show packet going into GUIDANCE_STATE

<u>CRC</u>

B-264, R-73 -- need to reference or derive the CRC-16 algorithm

the statement that says that CRC-16 must be calculated at each call is false -- it should be removed

B-262 -- need to define "logical shift"

B-263-264 -- need to make description of forming CRC more precise

B-261 -- need more precision in the description of forming the table -- spell out all steps

-------------------------------- END OF SESSION 8 -------------------------------------

Session 9: 10/15/93 8:30 a.m. - 10:30 a.m.

IMPORTANT NOTE: DESIGN DOES NOT BALANCE.
According to the Software Development Standards for the Design Process, the Design should have been balanced prior to bringing it to Design Review. This, of course, explains the many many problems we have found.

<u>Data Dictionary</u>

B-12 -- EXTERNAL data store not consistent with spec

B-11 -- GUIDANCE_STATE is missing from the data dictionary

B-14 -- RUN_PARAMETERS and SENSOR_OUTPUT are missing from the data dictionary

Lots of miscellaneous stuff:
B-102, 268, 166, 228-229, 164-165, 238 -- See inspection logs for individual entries with problems

Introduction

B-39 -- in the top level description, the term “four phases” is not accurate

B-41 -- need to improve clarity of Module Descriptions

B-43 -- need to state which version of the spec this design complies with

B-103 -- clarify statement “code of the design”

B-104 -- need to refer to spec and mods appropriately

B-45 -- do not need a status section

B-47, B49, B122 -- need to clean up notation

B-111 -- need to include a description of the call structure

B-112 -- need to include an overview of scheduling procedures

B-116 -- need a section describing the syntax for the pseudocode

C.1.2 Review Logs from Preliminary Design Review

Review Log from System Analyst

Individual Inspection Preparation Log #1 (page 1)

Name: Bernice Becher | Date Log Submitted: October 15, 1993
Implementation: Pluto | Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

INDEX

Introduction

Structured Analysis Diagrams

Data Dictionary

INIT_GCS, P-Spec 1

AECLP, P-Spec 2.1

ARSP, P-Spec 2.2

ASP, P-Spec 2.3

CP, P-Spec 2.4

CRCP, P-Spec 2.5

GSP, P-Spec 2.6

GP, P-Spec 2.7

RECLP, P-Spec 2.8

TDLRSP, P-Spec 2.9

TDSP, P-Spec 2.10

TSP, P-Spec 2.11

Miscellaneous P-Specs (not the eleven functional units)

Miscellaneous

Typographic Errors, Style, Grammar

Suggestions for the Future

Individual Inspection Preparation Log #1 (Page 2)

Name: Bernice Becher   Date Log Submitted: October 15, 1993

Implementation: Pluto   Date of Inspection October 15, 1993

Role: Inspector

Defects/Clarity Problems/Concerns

INTRODUCTION

Introduction, page 1 40

1.1 Top-Level Description, all items with "*)"

Question: What does "*)" mean?

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

Introduction, page 1 39

1.1 Top-Level Description, first "*)" item "four phases" is not accurate.

*Requirement: Accuracy (Reference: DO-178B 6.3.2b).

(see Software Requirements Figure 1.2 and Table 5.10)

Introduction, page 2 41

1.3 Module Descriptions, third paragraph. Question: What does "empirical notations" mean?

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

Introduction, page 3 43

2.4 Transition History, second statement.

This statement does not include the fact that the code of the design should conform to the GCS Software Requirements document 2.2 and all existing formal modifications (1-26) to the Software Requirements document.

*Requirement: Reference: Software Develpment Standards, "Software Design Standards", "Design Documentation", "III Transition History", "If changes, additions, or deletions are made in response to a formal modification, the formal modification number should be referenced." *Requirement: Completeness (Reference: DO-178B 11.0b)

Introduction, page 3 103

2.4 Transition History, second statement. Question: What does "code of the design" mean?

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

Introduction, page 3 117

2.4 Transition History

No mention was made of changes to the design to comply with the Software Development Standards. Were there any changes to the design made in response to this requirement? If so, they should be mentioned in the transition history, as per this requirement.

*Requirement: Software Development Standards, "Instructions to Programmers Regarding the Transitional Design Phase", #1, "Modifying the orignial design...so that the new detailed design meets...the standards set forth in this document in the chapter "Software Design Standards"".

Introduction, page 4 104

2.6 References

The reference "GCS Development specifications" is not the correct name for the specification document and does not include the version number of the document. In addition, this statement does not include the numbers of all existing formal modifications (1-26) to the Software Requirements document.

*Requirement: Completeness (Reference: DO-178B 11.0b)

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

Individual Inspection Preparation Log #1 (Page 3)

Name: Bernice Becher | Date Log Submitted: October 15, 1993
Implementation: Pluto | Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

Introduction, page 5 45

3. Status of Pluto GCS Design, first paragraph

"The pluto version ... has the GCS modification number 1 to the 2.1 Release...incorporated into it."
The meaning of this statement is not clear.
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

Introduction, page 5 46

3. Status of Pluto GCS Design, second paragraph

The reference "version 2.2 of the GCS Development specifications" is not the correct name for the specification document. In addition, this statement does not include the fact that the design should have been modified to also incorporate all existing formal modifications (1-26) to the Software Requirements document.
*Requirement: Completeness (Reference: DO-178B 11.0b)
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

Introduction, Pages 6-7: 47

4. Notation in Pluto Version of GCS Design

It is not clear what the following mean:
"the * oldest",
"the * FIFO",
"does * not"
"noun * indicates"
"body * axis"
"performed * three"
"an array * with"
"performed * four times"
"array, * independently"
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

Introduction, pages 6-7 49

4. Notation in Pluto Version of GCS Design

Page 6, last comment box, and page 7, first and last comment box It seems the design is using one notation, namely ".*" to mean two different things (3 lander body axes as well as three independent calculations). Is this what was intended? Also, is "three times on each of the three elements in the vector..." intended to mean nine times? Is "four times on each of the four elements in the array" intended to mean sixteen times? That's probably not the intention, but is the meaning.
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

Introduction, page 6 122

4. Notation in Pluto Version of GCS Design

Page 6, last comment box "Also, anIndividual element of a vector can be referenced using the following notation: GP_VELOCITY.x " Problem: It is not clear what this notation means. Does ".x" mean ".x" or ".y" or ".z", and if so, exactly what does each represent?
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

Individual Inspection Preparation Log #1 (Page 4)

Name: Bernice Becher Date Log Submitted: October 15, 1993
Implementation: Pluto Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

Introduction 111

Description of Call Structure

The software structure needed to implement the requirements is missing from the design document. Even though some of the software structure is implicit in the DFDs and the PATs, a description as described in the reference below is missing.
*Requirement: Reference: Software Development Standards, "Software Design Standards", "II. Design Structure", "a)Description of Call Structure".
*Requirement: Reference:DO-178B, 11.10b

Introduction 112

Scheduling

An overview of the scheduling procedures is not contained in the design document. Even though the scheduling is implicit in the DFDs and PATs, the overview is missing. *Requirement: Reference:Software Development Standards, "Software Design Standards", "II. Design Structure", "d) Scheduling".
*Requirement: Reference:DO-178B, 11.10f

Introduction 123

An overview of the flow of control for any given frame is not contained in the design document. Even though this flow of control is implicit in the DFDs and PATs, the overview is missing. What is especially needed is a discussion of the invocation of the Individual subframes, the invocation of rendezvous, and the ending of GCS. *Requirement: Reference:DO-178B, 11.10f

Introduction 115

Comments on Method

Discussion is missing concerning which structured analysis/design method was used. *Requirement: Follow a particular design method (References: Software Development Standards,"Software Design Standards", "Design Methods, Rules, and Tools", "...using the structured analysis ...by Hatley and Pirbhai or...", and "Design Documentation", "...document should follow...GCS specification or the Hatley book...")
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

Introduction 116

Syntax for Pseudocode

An explanation of the syntax used in the pseudocode in the P-Specs is not present in the design. It is not important what type of pseudocode or structured English is used, but it is very important that the pseudocode be completely unambiguous. The inspection of the design is hampered unless the syntax is unambiguous.

In order to insure that the pseudocode is unambiguous, the design should supply either a reference to a source which describes the syntax in detail, or should itself supply a detailed description of the syntax. This design has not done so. The designer, during the overview meeting, stated that the pseudocode followed Fortran 77, and in some cases it does, but unfortunately there are exceptions:

Individual Inspection Preparation Log #1 (Page 5)

Name: Bernice Becher | Date Log Submitted: October 15, 1993
Implementation: Pluto | Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

1. The design frequently uses the construct which is not strictly FORTRAN 77:

```
if (expression
)   statement
  statement
  statement
endif
```

2. The design uses "==" which is not FORTRAN syntax.

3. In some cases, as for example in P-Spec 2.2.3, page 3, the design uses plain Engish text in the middle of a nested if.

4. The design stretches nested ifs over several pages, which is difficult to follow.

5. It is not always clear what is a comment and what is actually part of the design. Sometimes, the comments are boxed in with *, but sometimes they are not. An example of this is TDLRSP, P-Spec 2.9.2, pages 3 and 4. The design seems to use single asterisks for comments sometimes but never explicitly states the syntax for a comment. Sometimes a comment is the only entry inside an else or else if clause, and it is not completely clear if this means the clause is null.
   *Requirement: Nonambiguity (Reference: DO-178B 11.0a).

Individual Inspection Preparation Log #1 (Page 6)

Name:_________Bernice Becher____ Date Log Submitted:______October 15, 1993

Implementation:___Pluto ___________ Date of Inspection________October 15, 1993

Role:___________ Inspector_________

Defects/Clarity Problems/Concerns

STRUCTURED ANALYSIS DIAGRAMS

Context Diagram GCS and Data Dictionary entries for INITIALIZATION_DATA and for INIT_END_GCS 1

In the context diagram there is a solid arc labeled INITIALIZATION_DATA. This element has the same name as a data flow name in the Software Requirements document. There is some confusion because the data flow name in the design includes "INIT_END_GCS" which is not in the Software Requirements document. This in itself may not be a requirement violation, but it is confusing. There is, however, another problem. INIT_END_GCS is listed in the dictionary as a control flow. It is included in the group flow name INITIALIZATION_DATA which is a data flow name and appears on the GCS Context Diagram as a data flow. The control flow INIT_END_GCS should not be included on a solid data flow line.

*Requirement: Consistency (DO-178B 5.2.2a, 6.3.2b, and 11.0d)

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

Context Diagram GCS 2

The bubble INIT_RUN_GCS does not show as input the variables FRAME_COUNTER and SUBFRAME_COUNTER coming from GCS_SIM. These need to be shown as input for every frame and subframe after the initialization frame and subframe because the simulator updates them after each frame/subframe respectively. (see Software Requirements document, Figure 2.2)

*Requirement: Accuracy (Reference: DO-178B 6.3.2b).

*Requirement: Completeness (Reference: DO-178B 11.0b)

Context Diagram GCS 3

The bubble INIT_RUN_GCS does not show the variables AE_SWITCH and RE_SWITCH as output to the engines. These variables control the turning on/off of the axial engines and the turning off of the roll engines. (see Software Requirements document, Figure 2.3)

*Requirement: Completeness (Reference: DO-178B 11.0b)

*Addition 09/22/93 (revised 9/27/93)

Context Diagram GCS and DFD/CFD INIT_RUN_GCS 167

(see #3 and #51)

It turns out that AE_SWITCH and RE_SWITCH do not need to appear on the context diagram at all because they are in not in the EXTERNAL data store. They are merely used internally in the GCS software; therefore #3 and #51 can be canceled.

DFD/CFD INIT_RUN_GCS 50

The bubble RUN_GCS does not show the variables FRAME_COUNTER and SUBFRAME_COUNTER as input coming from GCS_SIM. These need to be shown as input for every frame and subframe after the initialization frame and subframe because the simulator updates them after each frame/subframe respectively.

*Requirement: Completeness (Reference: DO-178B 11.0b)

DFD/CFD INIT_RUN_GCS 51

The bubble RUN_GCS does not show the variables AE_SWITCH and RE_SWITCH as output to the engines. These variables control the turning on/off of the axial engines and the turning off of the roll engines.

*Requirement: Completeness (Reference: DO-178B 11.0b)

Individual Inspection Preparation Log #1 (Page 7)

Name: Bernice Becher Date Log Submitted: October 15, 1993
Implementation: Pluto Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

DFD/CFD INIT_RUN_GCS 16
The data stores RUN_PARAMETERS, GUIDANCE_STATE and SENSOR_OUTPUT and the flows into and out of them are missing from this diagram. These stores should appear because the data moves from INIT_GCS to RUN_GCS through these data stores. (see Requirements document, Figures 2.4 and 2.5)
*Requirement: Accuracy (Reference: DO-178B 6.3.2b).
*Requirement: Completeness (Reference: DO-178B 11.0b)

DFD/CFD INIT_RUN_GCS 52
Data flows into and out of the EXTERNAL store are missing. (see Software Requirements document, Figures 2.4 and 2.5)
*Requirement: Accuracy (Reference: DO-178B 6.3.2b).
*Requirement: Completeness (Reference: DO-178B 11.0b)

DFD/CFD INIT_RUN_GCS 124
The solid arc labeled RAW_SENSOR_DATA should not flow directly from outside to RUN_GCS. (see Requirements document, Figures 2.4 and 2.5)
*Requirement: Accuracy (Reference: DO-178B 6.3.2b).
*Requirement: Completeness (Reference: DO-178B 11.0b)

PAT INIT_RUN_GCS 17
Second line of table: "START_GCS INIT_DONE RUN_DONE 1 3 2"
Question: Since this is merely the line that lists the input names, what is the meaning of the right side of the second line with " 1 3 2"?
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

PAT INIT_RUN_GCS 109
Fifth line of table: ""TRUE" "TRUE" "FALSE" 0 1 1"
Problem: This line shows that the order of activation of GENERATE_SEQUENCE_PARMS and RUN_GCS doesn't matter; however, the INIT_END_GCS DFD shows that for a given frame, GENERATE_SEQUENCE_PARMS must be executed before RUN_GCS because the variables ITH_FRAME_2 and ITH_FRAME_5 flow from GENERATE_SEQUENCE_PARMS to RUN_GCS.
*Requirement: Consistency (DO-178B 5.2.2a, 6.3.2b, and 11.0d)

DFD/CFD RUN_GCS 67
On this diagram, SUBFRAME_COUNTER appears as a control flow as input and output to the cspec and as input to SUBFRAME_COUNTER_STORE. Problem: In the data dictionary, SUBFRAME_COUNTER is a data flow.
*Requirement: Consistency (DO-178B 5.2.2a, 6.3.2b, and 11.0d)

DFD/CFD RUN_GCS 98
It is not clear what is the purpose of the process "COPY CONTROL DATA" or whether it is actually needed at all. If it is the case that it has a function, it is not clear whether that function is traceable to the Software Requirements document.
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

Individual Inspection Preparation Log #1 (Page 8)

Name: Bernice Becher Date Log Submitted: October 15, 1993

Implementation: Pluto Date of Inspection October 15, 1993

Role: Inspector

Defects/Clarity Problems/Concerns

DFD/CFD RUN_GCS 81

The top left-hand bubble labeled "STORE RAW SENSOR DATA" seems to perform no useful function. The raw sensor data is included in the group flow INITIALIZATION_DATA which means it is initialized by the simulator. In addition, in the Software Requirements document, Figure 2.2, it is shown that the raw sensor data comes into GCS from the external sensors. There is therefore no requirement for GCS to put the raw sensor values into any data store, as they are already in the global data store EXTERNAL at the beginning of each frame.

*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

DFD/CFD RUN_GCS 84

The top middle bubble labeled "INIT RUN PARAM STORE"seems to perform no useful function. The run parameter data is included in the group flow INITIALIZATION_DATA which means it is initialized by the simulator. In addition, in the Software Requirements document, Figure 2.4, it is shown that the run parameter data is put into the store RUN_PARAMETERS by INIT_GCS which (according to the LEVEL 2 SPECIFICATION in the Software Requirements document) is "actually a part of GCS_SIM_RENDEZVOUS"). Figure 2.4 also shows that RUN_GCS does not store into the data store RUN_PARAMETERS. There is therefore no requirement for GCS to put the run parameter data into any data store, as they are already in the global data store RUN_PARAMETERS at the beginning of each frame.

*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

DFD/CFD RUN_GCS 86

The top right-hand bubble labeled "INIT GUIDANCE STATE STORE" seems to perform no useful function. The guidance state data (with the exception of INTERNAL_CMD which is not used as an input) is included in the group flow INITIALIZATION_DATA which means it is initialized by the simulator. In each frame, all of the data in the GUIDANCE_STATE store will be output by GCS. There is therefore no requirement for GCS to put the guidance state data into any data store as they are already in the global data store GUIDANCE_STATE at the beginning of each frame.

*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

DFD/CFD RUN_GCS 92

Input to TSP from GUIDANCE_STATE store with group flow name TEMP_GS_IN (which is element TS_STATUS) is incorrect. TS_STATUS is not an input to TSP.

*Requirement: Accuracy (Reference: DO-178B 6.3.2b).

DFD/CFD RUN_GCS 96

The labels on the eleven bubbles which represent the eleven functional units in the Software Requirements document are not exactly the same as the labels on the DFD/CFDs one level down, and this causes confusion. For example, the bubble for P-Spec 2.2 is "ARSP ALTIMETER RADAR", while the name one level down is "ARSP - Altimeter Radar Data Expand and Compress". The names for TDLRSP and TSP also do not match the names ones level down.

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

*Requirement: Consistency (DO-178B 5.2.2a, 6.3.2b, and 11.0d)

* DFD/CFD RUN_GCS 166

The two bubbles at the bottom of the page, namely SEND CHUTE RELEASE COMMAND and SEND ENGINE DATA do not seem to perform any function.

*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

Individual Inspection Preparation Log #1 (Page 9)

Name: Bernice Becher   Date Log Submitted: October 15, 1993

Implementation: Pluto   Date of Inspection October 15, 1993

Role: Inspector

Defects/Clarity Problems/Concerns

*Addition 09/22/93 (DELETED 9/27/93 - NEED REVISION TO SPEC - PUT CHUTE_RELEASED INTO EXTERNAL DATA STORE)

DFD/CFD RUN_GCS 168

It may be that CHUTE_RELEASE in the left-hand bottom corner should not appear on the diagram at all because it is in the GUIDANCE_STATE store rather than in the EXTERNAL store.
*Requirement: Accuracy (Reference: DO-178B 6.3.2b).

* DFD/CFD RUN_GCS 169

Question: In the bottom left-hand corner, the input to SEND ENGINE DATA is AE_RE_CMDS, while the output is ENGINE_DATA. Both flows contain the same data. Why are two different names used?
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

* DFD/CFD RUN_GCS 144

Input to GSP from GUIDANCE_STATE store with group flow name GYRO_GS_IN (which is element GS_STATUS) is incorrect because GS_STATUS is not an input to GSP.
*Requirement: Accuracy (Reference: DO-178B 6.3.2b).

* PAT RUN_GCS 229

There doesn't seem to be any mechanism for the eleven functional units to signal when they have finished executing activating continuously once they have been activated once.
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

PAT RUN_GCS 97

The process "COPY CONTROL DATA" which appears on the DFD/CFD for RUN_GCS is missing from this PAT, and therefore its order of activation is unknown.
*Requirement: Completeness (Reference: DO-178B 11.0b)

PAT RUN_GCS 68

Problem: In the first subframe (the first four lines of the table), this PAT has imposed an order of activation on the processes that is not stated in the Software Requirements document. The Software Requirements document does not state any specific order of activation for ARSP, TDLRSP, TDSP, ASP, or GSP with respect to each other; however, the PAT imposes an arbitrary order of activation.
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)
**Software Requirements 2.2 with Mods 1-26 Reference: Chapter 4, Level 3 Flow Diagrams and C-Specs, Scheduling

PAT RUN_GCS 69

Problem: In the third subframe (the last two lines of the table), this PAT has imposed an order of activation on the processes that is not stated in the Software Requirements document. The Software Requirements document does not state any specific order of activation for AECLP and RECLP with respect to each other; however, the PAT imposes an arbitrary order of activation.
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)
**Software Requirements 2.2 with Mods 1-26 Reference: Chapter 4, Level 3 Flow Diagrams and C-Specs, Scheduling

Individual Inspection Preparation Log #1 (Page 10)

Name: Bernice Becher Date Log Submitted: October 15, 1993
Implementation: Pluto Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

PAT RUN_GCS 70

Problem: The variable SUBFRAME_COUNTER appears in the output section of this PAT, and its value is therefore changed by the c-spec. This is not permitted. The Software Requirements document states in the section labeled LEVEL 2 SPECIFICATION that SUBFRAME_COUNTER will be initialized by INIT_GCS which is actually a part of GCS_SIM_RENDEZVOUS. The Software Requirements document also states in Chapter 4. LEVEL 3 FLOW DIAGRAMS AND C-SPECS, under SCHEDULING, that "On the first, and subsequent, calls to GCS_SIM_RENDEZVOUS, FRAME_COUNTER and SUBFRAME_COUNTER will be returned to the implementation containing the correct values for operation. There is no requirement anywhere in the Software Requirements document that the GCS software should change the value of SUBFRAME_COUNTER.

*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

**Software Requirements 2.2 with Mods 1-26 Reference: Chapter 4, Level 3 Flow Diagrams and C-Specs,

PAT RUN_GCS 125

Problem: The use of the name "SUBFRAME_COUNTER" is ambiguous because it appears in the stores EXTERNAL_OLD, SUBFRAME_COUNTER_STORE, and in the global store EXTERNAL (defined in the Software Requirements document but not in the store EXTERNAL in this design document). One can look at the RUN_GCS DFD and deduce that the intention is to store into SUBFRAME_COUNTER_STORE, but the P-Spec itself should be self-contained.

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

**Software Requirements 2.2 with Mods 1-26 Reference: Chapter 4, Level 3 Flow Diagrams and C-Specs,

PAT RUN_GCS 71

The first line of the table, where ITH_FRAME_2 IS F and ITH_FRAME_5 is F:

Problem: Some processes which should be activated are not activated.

*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)

**Software Requirements 2.2 with Mods 1-26 Reference: Chapter 4, Level 3 Flow Diagrams and C-Specs,

PAT RUN_GCS 72

The third line of the table, where ITH_FRAME_2 IS F and ITH_FRAME_5 is "TRUE":

Problem: Some processes which should be activated are not activated.

*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)

**Software Requirements 2.2 with Mods 1-26 Reference: Chapter 4, Level 3 Flow Diagrams and C-Specs,

PAT RUN_GCS 73

It is unclear, looking at the input and output columns for RENDEZVOUS_CNTL, how it functions. Question: How does the functioning of RENDEZVOUS_CNTL work? if WAITING= rendezvous was called, then what does RUNNING mean?

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

Individual Inspection Preparation Log #1 (Page 11)

Name:_________Bernice Becher____ Date Log Submitted:______October 15, 1993
Implementation:___Pluto ____________ Date of Inspection________October 15, 1993
Role:___________ Inspector__________

Defects/Clarity Problems/Concerns

*PAT RUN_GCS 228
Under output column, "GP_HAS_RUN", the value "DONT CARE" appears. This is not feasible. Normally "DONT CARE" represents any value for an input (it is not an actual value to be set on output). Note that the data dictionary only shows two values, namely "TRUE", and "FALSE" for GP_HAS_RUN.
*Requirement: Consistency (DO-178B 5.2.2a, 6.3.2b, and 11.0d)

*PAT RUN_GCS 164
There doesn't seem to be any mechanism for keeping the eleven functional unit processes from activating continuously once they have been activated once. There does not seem to be a way that they signal when their execution has completed so that they can be deactivated.
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

* PAT RUN_GCS 165
The processes "SEND CHUTE RELEASE COMMAND" and "SEND ENGINE DATA" do not seem to be traceable to the specification. (see #138) (see specification Figure 2.4 which shows data going off page)
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

* DFD/CFD GSP 143
Input to TDSP with group flow name GYRO_GS_IN (which is element GS_STATUS) is incorrect because G_STATUS is not an input to GSP.
*Requirement: Accuracy (Reference: DO-178B 6.3.2b).

* DFD/CFD CP 234
The following show as inputs to CP P-Spec 2.4.2, but are not actually inputs:
AE_SWITCH
C_STATUS
RE_SWITCH
TDLRSP_SWITCH
TDSP_SWITCH
TE_LIMIT THETA
FRAME_BEAM_UNLOCKED
FRAME_ENGINES_IGNITED
INTERNAL_CMD CL
*Requirement: Accuracy (Reference: DO-178B 6.3.2b).

* DFD/CFD CP, and DFD/CFD RUN_GCS 238
The variable PACKET is shown correctly as an output from CP, but the fact that it is also an output that goes into the GUIDANCE_STATE data store has not been shown on the diagrams.
*Requirement: Completeness (Reference: DO-178B 11.0b)

* DFD/CFD CP, and DFD/CFD RUN_GCS 239
The variable SUBFRAME_COUNTER is shown correctly as an input to CP, but the fact that it is also an input that comes from the EXTERNAL data store has not been shown on the diagrams.
*Requirement: Completeness (Reference: DO-178B 11.0b)

Individual Inspection Preparation Log #1 (Page 12)

Name: Bernice Becher Date Log Submitted: October 15, 1993
Implementation: Pluto Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

DATA DICTIONARY

DATA DICTIONARY 12
EXTERNAL
The actual data elements and order of the data elements in the store EXTERNAL do not agree with those in the Software Requirements document. Question: Why are the elements repeated several times? *Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)
**Software Requirements 2.2 with Mods 1-26 Reference: Introduction, under Requirements, under Global Data Store Organization.
*Requirement: Consistency (DO-178B 5.2.2a, 6.3.2b, and 11.0d)

DATA DICTIONARY 11
GUIDANCE_STATE
The store named GUIDANCE_STATE is missing from the data dictionary, even though it does appear on the RUN_GCS DFD/CFD, and descriptions of some elements in the design data dictionary state that they are in this store. It is therefore not possible for the inspector to check whether the data in this store is of the right data type and dimension, or whether the elements occur in the proper order.
*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)
**Software Requirements 2.2 with Mods 1-26 Reference: Introduction, under Requirements, under Global Data Store Organization.
*Requirement: Consistency (DO-178B 5.2.2a, 6.3.2b, and 11.0d)

DATA DICTIONARY 14
RUN_PARAMETERS
The store named RUN_PARAMETERS is missing from the data dictionary, even though it does appear on the RUN_GCS DFD/CFD, and descriptions of some elements in the design data dictionary state that they are in this store. It is therefore not possible for the inspector to check whether the data in this store is of the right data type and dimension, or whether the elements occur in the proper order.
*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)
**Software Requirements 2.2 with Mods 1-26 Reference: Introduction, under Requirements, under Global Data Store Organization.
*Requirement: Consistency (DO-178B 5.2.2a, 6.3.2b, and 11.0d)

DATA DICTIONARY
SENSOR_OUTPUT
The store named SENSOR_OUTPUT is missing from the data dictionary, even though it does appear on the RUN_GCS DFD/CFD, and descriptions of some elements in the design data dictionary state that they are in this store. It is therefore not possible for the inspector to check whether the data in this store is of the right data type and dimension, or whether the elements occur in the proper order.
*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)
**Software Requirements 2.2 with Mods 1-26 Reference: Introduction, under Requirements, under Global Data Store Organization.
*Requirement: Consistency (DO-178B 5.2.2a, 6.3.2b, and 11.0d)

Individual Inspection Preparation Log #1 (Page 13)

Name: Bernice Becher Date Log Submitted: October 15, 1993
Implementation: Pluto Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

DATA DICTIONARY

Use of Design-defined Control Stores 102

The following control stores (which are not defined in the Software Requirements document data dictionary) appear in the design data dictionary:

*END_GCS_STORE
EXTERNAL_OLD
GENERATE_SEQUENCE_PARMS
*GP_HAS_RUN_STORE
GUIDANCE_STATE_OLD
INIT_GCS
*RENDEZVOUS_CNTL_STORE
RUN_GCS
RUN_PARAMETERS_OLD
SENSOR_OUTPUT_OLD
*SUBFRAME_COUNTER_STORE
* = used in design

Problem 1: Only four of the above (END_GCS_STORE, GP_HAS_RUN_STORE, RENDEZVOUS_CNTL_STORE, and SUBFRAME_COUNTER_STORE) appear on the structured analysis diagrams in the design. It is not clear why these four stores are needed and exactly how they are used. *Requirement: Nonambiguity (Reference: DO-178B 11.0a).

Problem 2: It seems that the seven stores listed above which are not used at all should not be in the data dictionary. *Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

DATA DICTIONARY 268

The following entries are not used:

AECLP_DONE
ARSP_DONE
ASP_DONE
CLP_DONE
CP_DONE
CRCP_DONE
GP_DONE
GSP_DONE
RECLP_DONE
RENDEZVOUS
SP_DONE
TDLRSP_DONE
TDSP_DONE
TSP_DONE

DATA DICTIONARY 269

TDLR_ANGLES and THETA

In RANGE, "PI" is not defined.

DATA DICTIONARY

AR_FREQUENCY 267

RANGE upper value "2.45**9" is incorrect.

Individual Inspection Preparation Log #1 (Page 14)

Name: Bernice Becher　　Date Log Submitted: October 15, 1993
Implementation: Pluto　　Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

DATA DICTIONARY 195
AX_ENG_GS_IN
GP_ATTITUDE is an input to AECLP but is missing from AX_ENG_GS_IN. AE_STATUS is not an input to AECLP and therefore should not be included in AX_ENG_GS_IN.
*Requirement: Accuracy (Reference: DO-178B 6.3.2b).

DATA DICTIONARY 196
AX_ENG_RP_IN
DELTA_T is an input to AECLP, but is missing from AX_ENG_RP_IN. GRAVITY is an input to AECLP, but is missing from AX_ENG_RP_IN.
*Requirement: Accuracy (Reference: DO-178B 6.3.2b).

DATA DICTIONARY
AX_ENG_RP_IN 55
Doesn't state whether control or data flow
*Requirement: Completeness (Reference: DO-178B 11.0b)

DATA DICTIONARY
BYTE_PACKET 56
BYTE_PACKET is defined to be 188 of integer*1 which does not match the global data store variable PACKET which is 256 of integer*2. Also there is no type integer*1 in Fortran.
*Requirement: Accuracy (Reference: DO-178B 6.3.2b).
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

DATA DICTIONARY
CHECKSUM 57
The only attribute is the data type. More information is needed for understanding. *Requirement: Completeness (Reference: DO-178B 11.0b)

* DATA DICTIONARY (and Context Diagram GCS, DFD/CFD INIT_RUN_GCS, DFD/CFD RUN_GCS, DFD/CFD CRCP) 197
CHUTE_RELEASE
The name CHUTE_RELEASE is the same as CHUTE_RELEASED (except the first is a control flow and the second is a data flow).
Question: Why are they both required?

* DATA DICTIONARY 235
COMM_EXT_IN SUBFRAME_COUNTER is missing

* DATA DICTIONARY 236
COMM_EXT_OUT
This entry seems to be unnecessary as it does not appear to be used anywhere.
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

* DATA DICTIONARY 237
COMM_GS_IN
The variables C_STATUS and CL should not be included.
*Requirement: Accuracy (Reference: DO-178B 6.3.2b).

Individual Inspection Preparation Log #1 (Page 15)

Name: Bernice Becher Date Log Submitted: October 15, 1993
Implementation: Pluto Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

* DATA DICTIONARY 233
ROL_ENG_GS_IN
The variable RE_STATUS is not an input to RECLP, but it has been included in ROL_ENG_GS_IN.
*Requirement: Accuracy (Reference: DO-178B 6.3.2b).

DATA DICTIONARY
EXTERNAL_DATA 58
This is equivalent to FRAME_COUNTER only. It seems misleading to name it EXTERNAL_DATA and equate it to FRAME_COUNTER. What is its purpose?
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).
*Requirement: Consistency (DO-178B 5.2.2a, 6.3.2b, and 11.0d)
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

DATA DICTIONARY
FRAME_COUNTER 54
The attribute is data but it is shown as "data/control flow". Why is this? It does not appear on any diagrams as control.
*Requirement: Accuracy (Reference: DO-178B 6.3.2b).

DATA DICTIONARY
GENERATE_SEQUENCE_PARMS (store) 60
"*not-defined*"
Question: What does "not-defined" mean, and why is this element in here?
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

DATA DICTIONARY 211
GUIDE_SO_IN
ATMOSPHERIC_TEMP is not and input to GP, and therefore should not be included in GUIDE_SO_IN

DATA DICTIONARY 212
GUIDE_GS_OUT
TE_INTEGRAL and CL are outputs from GP, but are missing from GUIDE_GS_OUT.

DATA DICTIONARY 90
GUIDANCE_STATE_DATA and INIT_GS_OUT
Question: What is the reason for having two group flow names which contain exactly the same elements? It seems to overly complicate the design and make it more difficult to understand.
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

DATA DICTIONARY
INIT_END_GCS 4
This element has only one entry in the dictionary, namely "FALSE". The dictionary does not state what this entry is or what it means. It does not have brackets around it. Is it the range, or the initial value, or a constant? Can it assume only one value?
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

Individual Inspection Preparation Log #1 (Page 16)

Name: Bernice Becher Date Log Submitted: October 15, 1993
Implementation: Pluto Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

DATA DICTIONARY
INIT_EXT_OUT 61
"*not-defined*"
Question: What does "not-defined" mean, and why is this element in here?
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

DATA DICTIONARY
INIT_GCS 62
"*not-defined*"
Question: What does "not-defined" mean, and why is this element in here?
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

DATA DICTIONARY 91
INIT_GS_OUT
This group flow name contains two element names which are not in the data dictionary, namely TDLRSP_SWITCH and TDSP_SWITCH. It also is missing one data flow name, namely CL.
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)
*Requirement: Consistency (DO-178B 5.2.2a, 6.3.2b, and 11.0d)

DATA DICTIONARY
ITH_FRAME_2 and ITH_FRAME_5
Question: What is the meaning for "TRUE/FALSE"?
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

DATA DICTIONARY
NBYTES 65
"*integer*"
No information is given other than "*integer*". What is this element for?
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

DATA DICTIONARY 83
RAW_SENSOR_DATA and RAW_SENSOR_EXT_OUT
Question: What is the reason for having two group flow names which contain exactly the same elements? It seems to overly complicate the design and make it more difficult to understand.
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

DATA DICTIONARY
RENDEZVOUS_CNTL 120
Some description of the meaning of this variable is needed.
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

* DATA DICTIONARY 218
ROL_ENG_GS_OUT
The variable RE_SWITCH is not an output from RECLP, but it has been included in ROL_ENG_GS_OUT

Individual Inspection Preparation Log #1 (Page 17)

Name: Bernice Becher Date Log Submitted: October 15, 1993
Implementation: Pluto Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

DATA DICTIONARY
RUN_GCS 66
"*not-defined*"
What does "not-defined" mean, and why is this element in here?
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

DATA DICTIONARY 89
RUN_PARAMETER_DATA and INIT_RP_OUT
Question: What is the reason for having two group flow names which contain exactly the same elements? It seems to overly complicate the design and make it more difficult to understand.
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

DATA DICTIONARY
START_GCS 5
The dictionary gives the values FALSE, TRUE, and DONT_CARE but does not give the meanings for these. This causes confusion in understanding INIT_RUN_GCS PAT. It is not clear until one studies INIT_RUN_GCS PAT whether TRUE means that GCS is to be activated or whether it means that it has already been activated. By deduction, it appears to mean that it should be activated.
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

DATA DICTIONARY
SUBFRAME_COUNTER_STORE 101
This is a data store which contains the element named SUBFRAME_COUNTER. The global data store defined in the Software Requirements document as EXTERNAL contains a data element named SUBFRAME_COUNTER. This duplication of names causes ambiguity. (see P-Spec 2.18)
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

DATA DICTIONARY
TD_LND_RAD_RP_IN 95
This group flow name appears on the RUN_GCS DFD/CFD as input to TDLRSP. The group name incorrectly includes K_MATRIX and TDLR_STATE which are inputs to TDLRSP, but are not in the RUN_PARAMETERS store but rather in the GUIDANCE_STATE store. (K_MATRIX has already been correctly included in the group flow name TD_LND_RAD_GS_IN)
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

DATA DICTIONARY
TD_LND_RAD_GS_IN 94
This group flow name appears on the RUN_GCS DFD/CFD as input to TDLRSP. The group name incorrectly includes TDLR_STATUS which is not an input to TDLRSP.
*Requirement: Accuracy (Reference: DO-178B 6.3.2b).
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

DATA DICTIONARY
TD_LND_RAD_GS_IN 127
TDLR_STATE is missing from this list of inputs from GUIDANCE_STATE to TDLRSP.

Individual Inspection Preparation Log #1 (Page 18)

Name: Bernice Becher   Date Log Submitted: October 15, 1993

Implementation: Pluto   Date of Inspection October 15, 1993

Role: Inspector

Defects/Clarity Problems/Concerns

DATA DICTIONARY, pages 31 and 32 9

TDLRSP_SWITCH

TDLRSP_SWITCH is not a GCS common store variable and is not in the Data Dictionary but is listed under GUIDANCE_STATE_DATA and under GUIDANCE_STATE_OLD.(SEE FORMAL MODIFICATION 2.2-24.5)

*Requirement: Follow a particular design method (References: Software Development Standards,"Software Design Standards", "Design Methods, Rules, and Tools", "...using the structured analysis ...by Hatley and Pirbhai or...", and "Design Documentation", "...document should follow...GCS specification or the Hatley book...")

*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

DATA DICTIONARY, pages 31 and 32 10

TDSP_SWITCH

TDSP_SWITCH is not a GCS common store variable and is not in the Data Dictionary but is listed under GUIDANCE_STATE_DATA and under GUIDANCE_STATE_OLD.(SEE FORMAL MODIFICATION 2.2-24.6)

*Requirement: Follow a particular design method (References: Software Development Standards,"Software Design Standards", "Design Methods, Rules, and Tools", "...using the structured analysis ...by Hatley and Pirbhai or...", and "Design Documentation", "...document should follow...GCS specification or the Hatley book...")

*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

DATA DICTIONARY

TEMP_GS_IN 93

This control flow name is not required. TS_STATUS is not an input to TSP (as is incorrectly shown on the RUN_GCS DFD (see # 92)

*Requirement: Accuracy (Reference: DO-178B 6.3.2b).

*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a,

DATA DICTIONARY 139

The primitive elements which are in the design data dictionary but are not in the Software Requirements data dictionary in general do not contain enough information for their meaning and use to be unambiguous. Each should contain as a minimum a general description, the units, the name of the control store (if any) in which it appears, and if logical or boolean the meaning of the values. It would also be helpful to have the "USED IN" item, and the data type.

Name: Bernice Becher — Date Log Submitted: October 15, 1993
Implementation: Pluto — Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

INIT_GCS P-Spec 1

INIT_GCS, P-Spec 1, and Data Store INIT_GCS 74

There is confusion regarding the fact that there is a process named INIT_GCS and a data store named INIT_GCS (which is not used anywhere). *Requirement: Nonambiguity (Reference: DO-178B 11.0a).

INIT_GCS, P_Spec 1, page 1, middle of page 75

"copy INITIALIZATION_DATA.RUN_PARAMETER_DATA.* to RUN_PARAMETER_DATA.*"
"copy INITIALIZATION_DATA.GUIDANCE_STATE_DATA.* to GUIDANCE_STATE_DATA.*"
Problem:The notation INITIALIZATION_DATA.RUN_PARAMETER_DATA, INITIALIZATION_DATA.GUIDANCE_STATE_DATA, is not defined or explained anywhere.
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

INIT_GCS, P_Spec 1, page 1, middle of page 76

"copy INITIALIZATION_DATA.RUN_PARAMETER_DATA.* to RUN_PARAMETER_DATA.*"
"copy INITIALIZATION_DATA.GUIDANCE_STATE_DATA.* to GUIDANCE_STATE_DATA.*"

Problem: Neither one of these statements is feasible.
INITIALIZATION_DATA, RUN_PARAMETER_DATA, and GUIDANCE_STATE_DATA are merely group flow names. None of them is a data store, so how can any data be copied?
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).
*Requirement: Consistency (DO-178B 5.2.2a, 6.3.2b, and 11.0d)

INIT_GCS, P_Spec 1, page 1, middle of page 77

"copy INITIALIZATION_DATA.RUN_PARAMETER_DATA.* to RUN_PARAMETER_DATA.*"
"copy INITIALIZATION_DATA.GUIDANCE_STATE_DATA.* to GUIDANCE_STATE_DATA.*"
Problem: The copying of INITIALIZATION_DATA (the group flow name from the Software Requirements document) is not a function which can be traced to the Software Requirements document.
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

INIT_GCS, P_Spec 1, page 1, middle of page 78

"* Turn on Roll Engine *" through and including
"EXTERNAL_DATA.FRAME_COUNTER = INITIALIZATION_DATA.FRAME_COUNTER":
and
"*Initialize SUBFRAME_COUNTER *
INIT_SUBFRAME_COUNTER = 1"

Problem: All of the initialization of the global control store variables is to be performed by GCS_SIM, as stated in the Software Requirements document in Chaper 3. LEVEL 2 SPECIFIICATION.
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

INIT_GCS, P_Spec 1, page 1, middle of page 79

"* Turn on Roll Engine *" through and including
"EXTERNAL_DATA.FRAME_COUNTER = INITIALIZATION_DATA.FRAME_COUNTER":
Problem: These statements are not feasible because GUIDANCE_STATE_DATA and EXTERNAL_DATA are group data flow names rather than control store names, so how can any data be copied?
*Requirement: Consistency (DO-178B 5.2.2a, 6.3.2b, and 11.0d)

Individual Inspection Preparation Log #1 (Page 20)

Name: Bernice Becher Date Log Submitted: October 15, 1993

Implementation: Pluto Date of Inspection October 15, 1993

Role: Inspector

Defects/Clarity Problems/Concerns

AECLP, P-Spec 2.1

AECLP, P-Specs 2.1.1 and 2.1.3 (AECLP_EXPAND and AECLP_COMPRESS) 198

These P-Specs do not appear to have any useful function. A P-Spec should perform some function that converts its input elements to its outputs. These P-Specs seem to convert from control flow group names to element names and back, which is not an actual function. Why would there be a P- Spec with no body?

*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

AECLP P-Spec 2.1.2, page 1, TITLE 199

"AECLP - Axial Engine Contrl Law Processing (P-Spec 2.3.1)"

Problem: There is some confusion about the P-Spec number. At the top of the page is 2.1.2 which is the correct P-Spec number for this design. The title, on the other hand, contains P-Spec number 2.3.1 which is probably from the Software Requirements document. There needs to be some clarification here.

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

AECLP P-Spec 2.1.2 212

Top of Page 3:

"real*8 theta"

Middle and bottom of Page 4:

"theta = ..." "PE_INTEGRAL = PE_INTEGRAL + theta * DELTA_T"

Top of Page 5:

"theta = ..." "YE_INTEGRAL = YE_INTEGRAL + theta * DELTA_T"

Problem: According to the specification, THETA is a global data store variable in the GUIDANCE_STATE data store. In FORTRAN, the same name cannot be used as a local variable. This is an implementation detail, but since it has entered the design, it is a problem.

AECLP P-Spec 2.1.2, bottom of page 3 and top of page 4: 197

"First: determine the axial engines' temperature (AE_TEMP), as follows:.."

Problem: It is not made clear which (if any) column in the table at the top of page 4 actually represents the new value for AE_TEMP.

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

AECLP P-Spec 2.1.2, pages 4-5

Problem: There are instances where a global variable which has a time history subscript appears without any index for the time history. This leads to ambiguity. These instances are:

1. Top of page 4, within table:

"GP_ALTITUDE <= ..." (occurs in two places) 200

2. Middle of page 4:

"if (GP_VELOCITY(1) == 0)..."

"..= GP_VELOCITY(3) / GP_VELOCITY(1)"

3. Page 5:

"...= GP_VELOCITY(2) / GP_VELOCITY(1)"

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

Individual Inspection Preparation Log #1 (Page 21)

Name:___________Bernice Becher____ Date Log Submitted:_____October 15, 1993

Implementation:___Pluto ___________ Date of Inspection_______October 15, 1993

Role:___________ Inspector__________

Defects/Clarity Problems/Concerns

AECLP P-Spec 2.1.2, pages 4-5 201

Middle of page 4:
"theta = GP_VELOCITY(3) / GP_VELOCITY(1)"
Top of page 5:
"theta = GP_VELOCITY(2) / GP_VELOCITY(1)"

Problem: In each case for the denominator, the specification uses an absolute value, but the design doesn't.
*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)
"Software Requirements Reference: AECLP, COMPUTE LIMITING ERROR FOR PITCH and COMPUTE LIMITING ERROR FOR YAW

AECLP P-Spec 2.1.2, 202

Bottom of page 4 and middle of page 5:
Problem: The calculation for limiting_pitch_error and limiting_yaw_error seem to impose a two-step implementation for no other reason than to avoid a continuation line.
Question: Is there some other reason for this?
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

AECLP P-Spec 2.1.2, top of page 6: 203

Question: The possibility of a divide-by-zero can be avoided here if the design supplies a special solution for the differential equation for the case where OMEGA = 0. The specification does not, however, state this explicitly. Suggestion: perhaps modify the specification but do not cite a problem for the design now.

AECLP P-Spec 2.1.2, pages 6-7 215

Top of page 6:
"if (AE_TEMP == cold or AE_TEMP == WARMING_UP)
te_limit_temp = 0.
TE_LIMIT = 0.
else if (AE_TEMP == hot)"
Middle of page 7:
"end if * (AE_TEMP == ?) *"
Problem: This conditional dealing with AE_TEMP is added functionality because there is no such requirement in the specification.
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

AECLP P-Spec 2.1.2, top of page 7 204

"q_temp = -GAX(... * GP_ALTITUDE(1,3,0)...VELOCITY_ERROR
+ GVEI(CL * TE_INTEGRAL
q_over_omega = (GA * (q_temp + GVEI(CL) * TE_INTEGRAL)) / OMEGA"

Problem 1: In the equation for q_temp, the term "GP_ALTITUDE(1,3,0)" is incorrect.
Problem 2: In the equation for q_temp, the parentheses are unbalanced, thereby making the equation ambiguous.

Individual Inspection Preparation Log #1 (Page 22)

Name: Bernice Becher Date Log Submitted: October 15, 1993
Implementation: Pluto Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

Problem 3: After both of these equations are executed, the term q_over_omega will be incorrect because the term " GVEI(CL) * TE_INTEGRAL" will have been added in twice.
*Requirement: Accuracy (Reference: DO-178B 6.3.2b).
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

AECLP P-Spec 2.1.2, top of page 7 211
The design has not shown the derivation of the equation used to solve the differential equation for TE_LIMIT.

AECLP P-Spec 2.1.2, bottom of page 7 216
"pitch_error = 0.
yaw_error = 0.
thrust_error = 0."
Problem: These statements represent added functionality. In the case where AE_SWITCH is off, there is no requirement to set anything except AE_CMD and AE_STATUS, which is being done. In this case, pitch_error, yaw_error, and thrust_error will not be used.
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

AECLP, P-Spec 2.1.2, pages 3-7 214
In the pseudocode, nested if's spanning many pages makes the logic extremely difficult to follow and may lead to an error-prone inspection. In this P-Spec, one nested if begins on page 3, nests to a depth of four, and does not terminate until page 7. The problem is that it is very difficult to see the matching parts of each if block and therefore difficult to follow the logic.

AECLP P-Spec 2.1.2, pages 7-9 205
The variable TE_LIMIT may be in error (depending on the actual values) when this P-Spec is finished executing because it will not include the processing that took place during the bounding process using TE_MAX and TE_MIN.
*Requirement: Accuracy (Reference: DO-178B 6.3.2b).
*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)
**Software Requirements 2.2 with Mods 1-26 Reference: AECLP, COMPUTE LIMITING ERROR FOR THRUST

AECLP P-Spec 2.1.2, pages 7-8 206
Problem: In each of four different places, an error message is given if the variable is outside its acceptable range. In three of the cases, the exception condition has already been handled in a previous place, and therefore this is added functionality. The places where this occurs are:
Middle of page 7, "Give error message." (for AE_SWITCH)
Middle of page 8, "Give error message." (for CONTOUR_CROSSED)
Middle of page 8, "Give error message." (for CHUTE_RELEASED)
Top of page 9, "Give error message." (for AE_SWITCH)
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

Individual Inspection Preparation Log #1 (Page 23)

Name: Bernice Becher Date Log Submitted: October 15, 1993
Implementation: Pluto Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

AECLP P-Spec 2.1.2, top of page 9 207

"else if (int_cmd >= 0.0 & int_cmd <= 1.0) AE_CMD(i) = 127 * int_cmd"
Problem: The specification states that "Each value for AE_CMD is to be rounded to the nearest integer, where rounding is defined...". The design does not show that the value for AE_CMD is to be rounded.
*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)
**Software Requirements 2.2 with Mods 1-26 Reference: AECLP, COMPUTE AXIAL ENGINE VALVE SETTINGS.

AECLP P-Spec 2.1.2, pages 3-9 209

Limit Checking:

1. Bottom page 5:
   The upper limit check for CONTOUR_CROSSED is incorrect.

2. Bottom page 6:
   "if (GP_ALTITUDE(1,3,0) < ..."
   "else if (GP_ALTITUDE(1,3,0) > ..."
   Problem: GP_ALTITUDE only has one dimension, but the design uses three.

4. The following input variables to this P-Spec are not checked at all for limit violations:
   GP_ATTITUDE, CL, GP_VELOCITY

5. The following output variables to this P-Spec are not checked at all for limit violations:
   AE_CMD, AE_STATUS

AECLP P-Spec 2.1.2, pages 3-9 217

Problem: The fact that most of the limit checking is done inside nested if-then statements seriously obscures the flow of control of the P-Spec, and makes it difficult to check if limit checking has been done correctly.
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

AECLP P-Spec 2.1.2, bottom page 8, and top page 9 210

"int_cmd = INTERNAL_CMD(I)"
"if (int_cmd < ..."
"else if (int_cmd > 1.7..." etc. etc.
Problem: The use of the local variable int_cmd seems to serve no function and introduces added complexity.
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

Individual Inspection Preparation Log #1 (Page 24)

Name:_________Bernice Becher____ Date Log Submitted:______October 15, 1993

Implementation:___Pluto ___________ Date of Inspection________October 15, 1993

Role:___________ Inspector__________

Defects/Clarity Problems/Concerns

ARSP, P-Spec 2.2.2

ARSP P-Specs 2.2.1 and 2.2.3 (ARSP_EXPAND and ARSP_COMPRESS) 19

These P-Specs do not appear to have any useful function. A P-Spec should perform some function that converts its input elements to its outputs. These P-Specs seem to convert from control flow group names to element names and back, which is not an actual function. Why would there be a P- Spec with no body?

*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

ARSP P-Spec 2.2.2, page 1, TITLE 21

"ARSP - Altimeter Radar Sensor Processing (P-Spec 2.1.2)"

Problem: There is some confusion about the P-Spec number. At the top of the page is 2.2.2 which is the correct P-Spec number for this design. The title, on the other hand, contains P-Spec number 2.1.2 which is probably from the Software Requirements document. There needs to be some clarification here.

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

ARSP P-Spec 2.2.2, page 1, bottom of page: 20

"Shift Data in AR_ALTITUDE, AR_STATUS..."

Problem: More detail is needed.

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

*Requirement: Translatability to source code (Reference: Software Development Standards, Software Design Standards, "The low level requirements should be directly translatable into source code, with no further decomposition required.")

ARSP P-Spec 2.2.2, page 2, top of page: 22

"if (FRAME_COUNTER == even)
AR_ALTITUDE.* = AR_ALTITUDE.[previous value]
AR_STATUS.* = AR_STATUS.[previous value]
K_ALT.* = K_ALT.[previous value]"

Problem: The step before this in the design was to rotate these same variables unconditionally. These three assignments will cause a second rotation, which is incorrect.

*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)

**Software Requirements 2.2 with Mods 1-26 Reference: ARSP, Rotate Variables

ARSP P-Spec 2.2.2, page 2, top of page: 23

"AR_ALTITUDE.* = AR_ALTITUDE.[previous value]
AR_STATUS.* = AR_STATUS.[previous value]
K_ALT.* = K_ALT.[previous value]"

Problem: The notation ".*" is confusing here. Pages 6 and 7 of the design says this refers to independent iteration over 3 axes, or to three independent axes, but there are no axes involved here. It seems what is intended is rotation, but this rotation is ambiguous.

*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B
6.3.2a and 11.10a)

**Software Requirements 2.2 with Mods 1-26 Reference: ARSP, Rotate Variables

*Requirement: Translatability to source code (Reference: Software Development Standards, Software Design Standards, "The low level requirements should be directly translatable into source code, with no further decomposition required.")

Individual Inspection Preparation Log #1 (Page 25)

Name: Bernice Becher   Date Log Submitted: October 15, 1993
Implementation: Pluto   Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

ARSP P-Spec 2.2.2, page 2, top of page: 121

"AR_ALTITUDE.* = AR_ALTITUDE.[previous value]
AR_STATUS.* = AR_STATUS.[previous value]
K_ALT.* = K_ALT.[previous value]"

Problem: The notation ".[previous value]" has not been explained prior to its use.
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

ARSP P-Spec 2.2.2, page 2, middle of page: 134

Limit checks for AR_ALTITUDE, AR_STATUS, and K_ALT:

Problem 1: In each case, the notation ".*" is used; however in each case, the element is a scalar, except for the time history. Are all elements of the time history to be checked? (Note that on the page 3 limit check, only AR_ALTITUDE.[0] is checked (which is correct)).

Problem 2: The rotations have been done, but the new values have not been calculated, so which history value is being checked?

ARSP P-Spec 2.2.2, page 3, top of page: 24

"if (AR_FREQUENCY == 0)..."

Problem: AR_FREQUENCY does not contain zero in its valid range.
Problem: In the case where an echo is not received, AR_FREQUENCY is not used, but this check is made whether or not an echo is received.
*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)
**Software Requirements 2.2 with Mods 1-26 Reference: ARSP, Determine Altitude
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

ARSP P-Spec 2.2.2, page 3, top of page: 25

"AR_ALTITUDE.[0] = (AR_COUNTER * 3 * 10**8) / AR_FREQUENCY * 2 "

Problem: In the overview meeting, the designer stated that the syntax for the pseudocode was FORTRAN 77. If that is the case, then, due to the hierarchy of operations in FORTRAN, " / AR_FREQUENCY * 2 " means that AR_FREQUENCY is part of the numerator, which is incorrect.
*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)
**Software Requirements 2.2 with Mods 1-26 Reference: ARSP, Determine altitude if echo received

ARSP P-Spec 2.2.2, page 3, top of page: 26

Problem: A lower limit check is made for AR_ALTITUDE(0), but the upper limit check is not made.
*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)
**Software Requirements 2.2 with Mods 1-26 Reference: Introduction, Exception Handling, Upper Limit Exceeded

Individual Inspection Preparation Log #1 (Page 26)

Name: Bernice Becher — Date Log Submitted: October 15, 1993

Implementation: Pluto — Date of Inspection October 15, 1993

Role: Inspector

Defects/Clarity Problems/Concerns

ARSP P-Spec 2.2.2, page 3, middle of the page 15

Starts with "Construct a table of divided differences....":

"1. The first column of the table holds the four previous altitudes."

Problem: The design does not state the specific order for the four previous altitudes, that is, is the entry in the first row the most recent or the oldest altitude? This order must be stated because it will affect the result.

*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)

**Software Requirements 2.2 with Mods 1-26 Reference: ARSP, Determine Altitude if all previous values of AR_STATUS are healthy, by fitting a polynomial and then evaluating it.

ARSP P-Spec 2.2.2, page 3, middle of the page 105

Starts with "Construct a table of divided differences....":

"2. The 2nd column holds the differences between..."

Problem: The design does not state the order for the subtraction, i.e., is it:

diff = element in row i - element in row i+1, or

diff = element in row i+1 - element in row i?

The order must be stated because it will affect the result.

*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)

**Software Requirements 2.2 with Mods 1-26 Reference: ARSP, Determine Altitude if all previous values of AR_STATUS are healthy, by fitting a polynomial and then evaluating it.

ARSP P-Spec 2.2.2, page 3, middle of the page 106

Starts with "Construct a table of divided differences....":

Question:

Is it possible to give a reference for this method? I have found several texts which present this method, but only show the equation for the coefficients of the interpolating polynomial. The step which is missing is going from the polynomial to the direct evaluation at the current point by doing the summation. I have convinced myself that the resulting evaluation is exactly equivalent to the Lagrange method, and therefore am convinced this method is correct, but would like to see the reference text or the derivation, if possible.

*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)

**Software Requirements 2.2 with Mods 1-26 Reference: ARSP, Determine Altitude if all previous values of AR_STATUS are healthy, by fitting a polynomial and then evaluating it.

ARSP P-Spec 2.2.2, page 4, middle of page: 27

"AR_ALTITUDE.* = AR_ALTITUDE.[previous value]"

Problem: The "*" notation is used here but there are no axes involved.

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

ARSP P-SPEC 2.2.2 82

The variables AR_STATUS and K_ALT are checked for limits exceeded in the case where FRAME_COUNTER is even, but they are not checked for limits exceeded in the case where FRAME_COUNTER is odd.

Individual Inspection Preparation Log #1 (Page 27)

Name:_________Bernice Becher____ Date Log Submitted:_____October 15, 1993

Implementation:___Pluto ___________ Date of Inspection_______October 15, 1993

Role:___________ Inspector__________

Defects/Clarity Problems/Concerns

ASP, P-Spec 2.3.2

ASP, P-Specs 2.3.1 and 2.3.3 (ASP_EXPAND and ASP_COMPRESS) 148

These P-Specs do not appear to have any useful function. A P-Spec should perform some function that converts its input elements to its outputs. These P-Specs seem to convert from control flow group names to element names and back, which is not an actual function. Why would there be a P- Spec with no body?

*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

ASP P-Spec 2.3.2, page 1, TITLE 153

"ASP - Accelerometer Sensor Processing (P-Spec 2.1.1)"

Problem: There is some confusion about the P-Spec number. At the top of the page is 2.3.2 which is the correct P-Spec number for this design. The title, on the other hand, contains P-Spec number 2.1.1 which is probably from the Software Requirements document. There needs to be some clarification here.

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

ASP P-Spec 2.3.2, page 2, middle of page: 149

"Shift Data the A_ACCELERATION.* AND..."

Problem 1: More detail is needed.

Problem 2: ".*" notation

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

*Requirement: Translatability to source code (Reference: Software Development Standards, Software Design Standards, "The low level requirements should be directly translatable into source code, with no further decomposition required.")

ASP P-Spec 2.3.2, page 1, bottom of page: 156

"BEGIN LOCAL TYPE DEFS
real a_gain.*
.
.
real hold
END LOCAL TYPE DEFS"

Question: Is there any special significance here when "real" is used, while most other P-Specs use "real*8" or "real*4"?

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

ASP P-Spec 2.3.2, pages 1-3 157

Problem: The "*." notation used throughout the entire P-Spec is very confusing and ambiguous.

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

ASP P-Spec 2.3.2, pages 1 and 2 158

"real at"

"at = "ATMOSPHERIC_TEMP"

Question: What is the purpose for this step? It doesn't seem to accomplish anything.

*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

Individual Inspection Preparation Log #1 (Page 28)

Name:______________Bernice Becher____ Date Log Submitted:_____October 15, 1993

Implementation:___Pluto ____________ Date of Inspection_______October 15, 1993

Role:____________ Inspector__________

Defects/Clarity Problems/Concerns

ASP P-Spec 2.3.2, page 3, top of page 159

"accel.* = ALPHA_MATRIX.*.* * accel.*"

Problem: This is supposed to be a matrix multiplication, but as stated here it appears to be a scalar multiplication.

*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)

*Software Requirements Document, ASP, CORRECT FOR MISALIGNMENT

ASP P-Spec 2.3.2, page 3 160

"if [A_STATUS.*.[all 1..3]..."

Problem: The notation ".[all 1..4] is explained but not ".[all 1..3]"

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

ASP P-Spec 2.3.2 161

Question: This P-Spec does not provide special handling for the case where all three values of A_ACCELERATION are exactly equal, in order to avoid roundoff and a possible negative square root error later in the standard deviation. I don't really believe this is required, but it was brought up in a previous meeting of the GCS team.

ASP P-Spec 2.3.2 163

The variable A_ACCELERATION has not been checked for limits exceeded.

**Software Requirements 2.2 with Mods 1-26 Reference:

Name: Bernice Becher Date Log Submitted: October 15, 1993
Implementation: Pluto Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

CP, P-Spec 2.4

CP, P-Specs 2.4.1 and 2.4.3 (CP_EXPAND and CP_COMPRESS) 230

These P-Specs do not appear to have any useful function. A P-Spec should perform some function that converts its input elements to its outputs. These P-Specs seem to convert from control flow group names to element names and back, which is not an actual function. Why would there be a P- Spec with no body?

*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

CP P-Spec 2.4.2, page 1, TITLE 231

"CP - Communications Processing (P-Spec 2.4)"

Problem: There is some confusion about the P-Spec number. At the top of the page is 2.4.2 which is the correct P-Spec number for this design. The title, on the other hand, contains P-Spec number 2.4 which is probably from the Software Requirements document. There needs to be some clarification here.

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

CP P-Spec 2.4.2 232

INPUT/OUTPUT Section:

1. The control flows ITH_FRAME_2 and ITH_FRAME_5 which are shown on the CP DFD/CFD diagram and which are used as inputs to this P-Spec, are missing from the INPUT/OUTPUT section.
2. The data flows NBYTES and BYTE_PACKET both of which appear on the CP DFD/CFD as outputs from CP to CALCULATE CRC-16, are missing from the INPUT/OUTPUT section.
3. The data flow CHECKSUM which appears on the CP DFD/CFD as an input to CP from CALCULATE CRC-16 is missing from the INPUT/OUTPUT section. *Requirement:Completeness (Reference: DO-178B 11.0b)

CP P-Spec 2.4.2 249

See #56 in DATA DICTIONARY problems for BYTE_PACKET.

CP P-Spec 2.4.2, page 3 259

Definitions for init_sample_mask_sub_fr_1 and 2 and 3:

Problem: Since the notation "B'...' " is not FORTRAN notation, it is ambiguous.

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

CP P-Spec 2.4.2, bottom half of page 3 and top half of page 4 240

This table which unfortunately spans two pages and shows the variable names vertically is in such a format that it is virtually impossible to read and interpret, and some variables (eg. GP_ROTATION and K_MATRIX) are missing. This table is important and should be presented in an easily understandable format, eg, horizontally.

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

*Requirement: Accuracy (Reference: DO-178B 6.3.2b).

CP P-Spec 2.4.2, bottom page 4 243

"Set bits for AR_ALTITUDE...every other frame:"

1. The variables K_ALT and K_MATRIX are missing from this list.

Individual Inspection Preparation Log #1 (Page 30)

Name: Bernice Becher   Date Log Submitted: October 15, 1993
Implementation: Pluto   Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

CP P-Spec 2.4.2, bottom page 4

1. Bottom of page 4: 244
   "Starting with byte eight, load BYTE_PACKET alphabetically with (subframe 1 variables, AR_ALTITUDE,...TS_STATUS)."...TDLR_VELOCITY, and TS_STATUS)."
2. Top of page 5: "Starting with byte eight, load ...TDS_STATUS and TD_SENSED)."
3. Middle of page 5: "Starting with byte eight...subframe 1 variables, and...TS_STATUS)."
4. Bottom of page 5: "Starting...with subframe 1 data:...G_STATUS."

Problem: In each one of these cases for subframe 1, the design is attempting to state which variables must be loaded into the data section of the packet.
1. There is some ambiguity in the statement of exactly which variables are to be loaded. There is a comment above describing "Subframe one's variables", but this is merely a comment and not a strict definition (see #250) as part of the design. In addition the statement of which variables to load(except for #4 above) does not include the word "and" or any synonym for "and", and thus one could be misled into thinking that the variables listed in the load statement are the only variables to be loaded, which would be incorrect.
2. #4 above uses the term "subframe 1 data" which in this case actually means the variable names that follow, which lends to even more confusion.
3. It would be significantly more clear if for each case, the design would state alphabetically in one list all the variables to be loaded for that case.

CP P-Spec 2.4.2, bottom page 4

1. Top of page 6: 252
   "Starting with byte eight, load BYTE_PACKET alphabetically with subframe two's data."

   Problem: For subframe 2, the design is attempting to state which variables must be loaded into the data section of the packet. There is some ambiguity in the statement of exactly which variables are to be loaded. There is a comment above describing "Subframe two's data", but this is merely a comment and not a strict definition (see #250) as part of the design.

CP P-Spec 2.4.2, bottom page 4 253

1. Bottom of page 6:
   "Starting with byte eight, load BYTE_PACKET alphabetically with (subframe three's variables and CHUTE_RELEASED)." "Starting with byte eight, load BYTE_PACKET alphabetically with subframe three's variables."

   Problem: For subframe 3, the design is attempting to state which variables must be loaded into the data section of the packet. There is some ambiguity in the statement of exactly which variables are to be loaded. There is a comment above describing "Subframe three's variables ", but this is merely a comment and not a strict definition (see #250) as part of the design. It would be significantly more clear if for each case, the design would state alphabetically in one list all the variables to be loaded for that case.

Individual Inspection Preparation Log #1 (Page 31)

Name: Bernice Becher   Date Log Submitted: October 15, 1993
Implementation: Pluto   Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

CP P-Spec 2.4.2, pages 4 - 6: 242
The terms "first", "lowest", and "last" are used in many places, but are ambiguous. More specific wording is needed. Some examples of occurrences are:
Top of page 4: "Load the MSB of COMM_SYNC_PATTERN first."
Top of page 4: "Load the lowest 8 bits..."
Bottom of page 4: "Copy sample_mask's LSB first and its MSB last.
Bottom of page 4: "their MSB first and their LSB last."
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

CP P-Spec 2.4.2, pages 4 - 6 250
1. Middle of page 4: "* Subframe one's variables consist...values described *"
2. Bottom of page 5 to top of page 6: "Subframe two's data...VELOCITY_ERROR"
3. Middle of page 6: "Subframe three's variable's ...YE_INTEGRAL"
Problem: All of the above are formatted as comments but in reality apparently were actually intended to be part of the design. See Problem # 244.

CP P-Spec 2.4.2, middle of page 5: 251
"Set bits 3, 5, 6, 7...28, and 29...sample_mask."
Problem: The bits for K_ALT and K_MATRIX are missing.

CP P-Spec 2.4.2, pages 4 - 6 255
"if (sub_frame_counter == 1)"
"else if (sub_frame_counter == 2)"
"else if (sub_frame_counter == 3)"
Problem: The variable name "sub_frame_counter" is not correct.
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

CP P-Spec 2.4.2, bottom page 4 through bottom of page 6 245
"Set NBYTES with the total number of bytes stored in BYTE_PACKET."
Problem: The statement above occurs seven times. In each case, the design has not provided the actual number of bytes, nor has it provided an algorithm for obtaining this number.
*Requirement: Translatability to source code (Reference: Software Development Standards, Software Design Standards, "The low level requirements should be directly translatable into source code, with no further decomposition required.")

CP P-Spec 2.4.2, page 7, and DFD/CFD for CP. 256
"call CALCULATE CRC-16...CHECKSUM)"
Problem: The issue of one P-Spec calling another and how this is to be treated on the DFD/CFD at this point is unresolved.

CP P-Spec 2.4.2, page 7 257
"Store lower 16 bits of CHECKSUM in BYTE_PACKET in locations NBYTES+1 AND NBYTES+2"
Problem 1: Since CHECKSUM is only 16 bits, why the statement "lower 16 bits"?
Problem 2: In what order are the two bytes to be stored?

Individual Inspection Preparation Log #1 (Page 32)

Name:_________Bernice Becher____ Date Log Submitted:______October 15, 1993

Implementation:___Pluto ___________ Date of Inspection________October 15, 1993

Role:___________ Inspector__________

Defects/Clarity Problems/Concerns

CP P-Spec 2.4.2, page 7 258

"Set CSTATUS to healthy"

Problem: This step must be done at any point prior to calculating the CHECKSUM and prior to loading C_STATUS into BYTE_PACKET and PACKET.

*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)

**Software Requirements 2.2 with Mods 1-26 Reference: CP, SET COMMUNICATOR STATUS TO HEALTHY."

CP P-Spec 2.4.2, top of page 7 254

"Give error '(CP...value"

Problem: This represents added functionality. There is no requirement for checking the limits on SUBFRAME_COUNTER because it is in EXTERNAL data store.

CP P-Spec 2.4.2 246

General:

There are several places in this P-Spec where the variables K_MATRIX and/or GP_ROTATION are loaded into the packet, but nowhere is it stated that the constant terms (the off-diagonal elements of K_MATRIX and the diagonal elements of GP_ROTATION) should not be loaded.

*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)

**Software Requirements 2.2 with Mods 1-26 Reference: CP, PREPARE DATA SECTION.

CP P-Spec 2.4.2 247

General:

The design has not stated that only current history variables should be loaded into the packet.

*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)

**Software Requirements 2.2 with Mods 1-26 Reference: CP, PREPARE SAMPLE MASK

CP P-Spec 2.4.2 248

General:

For each particular unique mask and packet, the design needs to explain how it was derived, i.e., specifically which modules are executing for that case.

. CP, P-Spec 2.4.2 266

General:

No limit checking is done in this P-Spec, which actually is as it should be; however it might be a good idea to modify the specification to explicitly state this.

. CALCULATE CRC-16, P-Spec 2.4.5, bottom page 1 261

"For each of the 16 integer*4 entries in the table, store the zero-origin index (0 throught 15) into a temporary variable."

Problem: The intent here is that all the steps beginning with "store the zero-origin index..." through and including "When 1) through 3)...(table index)." be done for each of the 16 integer*4 entries in the table; however what has been stated is that only the very first step be done for all 16 entries.

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

Individual Inspection Preparation Log #1 (Page 33)

Name: Bernice Becher Date Log Submitted: October 15, 1993
Implementation: Pluto Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

CALCULATE CRC-16, P-Spec 2.4.5, bottom page 1 262

Bottom of page 1: "2. "Logically shift the temporary variable...right."
Middle of page 2: "2. Shift the crc right four bits, using a logical shift."

Problem: The terms "logically shift" and "shift...using a logical shift" should be precisely defined. Specifically, what rule is used to fill in the bits vacated on the left?
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

. CALCULATE CRC-16, P-Spec 2.4.5, middle page 2 263

"1. Exclusive-or the first byte in BYTE_PACKET into the lower eight bits of the crc."

Problem 1: "first" is not an accurate description of which byte is to be used on any except the first iteration.

Problem 2: This sentence does not state with what data the byte in BYTE_PACKET is to be exclusive-ored.

*Requirement: Completeness (Reference: DO-178B 11.0b)

. CALCULATE CRC-16, P-Spec 2.4.5, middle page 2 264

"3. Using the four bits...Exclusive-or the indexed table entry with the results of the shifted crc."

Problem: This sentence does not state where the new exclusive-ored value is to be placed.
*Requirement: Completeness (Reference: DO-178B 11.0b)

. CALCULATE CRC-16, P-Spec 2.4.5 265

The design should provide either a derivation or a reference to the derivation of the algorithms used to create the table and then to use the table to calculate the crc.

CRCP, P-Spec 2.5

CRCP, P-Specs 2.5.1 and 2.5.3 (CRCP_EXPAND and CRCP_COMPRESS) 193

These P-Specs do not appear to have any useful function. A P-Spec should perform some function that converts its input elements to its outputs. These P-Specs seem to convert from control flow group names to element names and back, which is not an actual function. Why would there be a P- Spec with no body?
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

CRCP P-Spec 2.5.2, page 1, TITLE 194

"CRCP - Chute Release Control Processing(P-Spec 2.3.3)"

Problem: There is some confusion about the P-Spec number. At the top of the page is 2.5.2 which is the correct P-Spec number for this design. The title, on the other hand, contains P-Spec number 2.3.3 which is probably from the Software Requirements document. There needs to be some clarification here.

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

Individual Inspection Preparation Log #1 (Page 34)

Name: Bernice Becher Date Log Submitted: October 15, 1993
Implementation: Pluto Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

GSP, P-Spec 2.6.2

GSP, P-Specs 2.6.1 and 2.6.3 (GSP_EXPAND and GSP_COMPRESS) 140

These P-Specs do not appear to have any useful function. A P-Spec should perform some function that converts its input elements to its outputs. These P-Specs seem to convert from control flow group names to element names and back, which is not an actual function. Why would there be a P- Spec with no body?

*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

GSP P-Spec 2.10.2, page 1, TITLE 151

"GSP - Gyroscope Sensor Processing (P-Spec 2.1.4)"

Problem: There is some confusion about the P-Spec number. At the top of the page is 2.6.2 which is the correct P-Spec number for this design. The title, on the other hand, contains P-Spec number 2.1.4 which is probably from the Software Requirements document. There needs to be some clarification here.

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

GSP P-Spec 2.10.2, page 1 155

"real*4 at

real*4 g_gain.*

Problem: Precision will be lost because of the data type "real*4". Requirement: ???

GSP P-Spec 2.6.2, page 2, top of page: 145

"Shift Data in G_ROTATION..."

Problem: More detail is needed.

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

*Requirement: Translatability to source code (Reference: Software Development Standards, Software Design Standards, "The low level requirements should be directly translatable into source code, with no further decomposition required.")

GSP, P-Spec 2.6.2 146

page 1:

"real*4 g_gain.*"

"real*4 count.*"

page 2:

"g_gain.* = ..." through "write (6,99) "G_ROTATION.*", G_ROTATION.*"

The ".*" notation is ambiguous.

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

GSP, P-Spec 2.6.2, page 1 and page 2, middle and end of page 147

"real*4 at"

"at = "ATMOSPHERIC_TEMP"

Question: What is the purpose for this step? It doesn't seem to accomplish anything.

GSP, P-Spec 2.6.2 162

The variable G_STATUS (in GUIDANCE_STATE) has not been checked for limits exceeded.

*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)

**Software Requirements 2.2 with Mods 1-26 Reference: Introduction, Exception Handling, Upper or Lower Limit Exceeded

Individual Inspection Preparation Log #1 (Page 35)

Name: Bernice Becher   Date Log Submitted: October 15, 1993

Implementation: Pluto   Date of Inspection October 15, 1993

Role: Inspector

Defects/Clarity Problems/Concerns

GP, P-Spec 2.7

GP, P-Specs 2.7.1 and 2.7.3 (GP_EXPAND and GP_COMPRESS) 170

These P-Specs do not appear to have any useful function. A P-Spec should perform some function that converts its input elements to its outputs. These P-Specs seem to convert from control flow group names to element names and back, which is not an actual function. Why would there be a P- Spec with no body?

*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

GP P-Spec 2.7.2, page 1, TITLE 171

"GP - Guidance Processing (P-Spec 2.2)"

Problem: There is some confusion about the P-Spec number. At the top of the page is 2.7.2 which is the correct P-Spec number for this design. The title, on the other hand, contains P-Spec number 2.2 which is probably from the Software Requirements document. There needs to be some clarification here.

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

GP, P-Spec 2.7.2, pages 1-2, INPUT/OUTPUT section. 180

The variable END_GCS which is an output has been omitted from this section.

*Requirement: Completeness (Reference: DO-178B 11.0b)

GP, P-Spec 2.7.2, page 2, bottom of page: 173

"BEGIN LOCAL TYPE DEFS
real interpolated_velocity
.
real new_gp_attitude.*.*
END LOCAL TYPE DEFS"

Problem: Precision may be lost if real (default real*4) is used.

GP, P-Spec 2.7.2, page 3, middle of page: 172

"Shift Data in the GP_VELOCITY...during this time step

Problem: More detail is needed.

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

*Requirement: Translatability to source code (Reference: Software Development Standards, Software Design Standards, "The low level requirements should be directly translatable into source code, with no further decomposition required.")

GP, P-Spec 2.7.2, Notation Problems: 174

Pages 2 and 3: The ".*" and ".*.*" notation throughout these pages is ambiguous.

Page 8, middle of page:

"GP_VELOCITY.[1]..."

Pages 9 - 11:

"alpha.[n], beta.[n], correction_term[n]", where n = 0 or 1 or 2, are ambiguous.

Pages 2-11:

It is not always clear what is a comment and what is pseudocode. Also there are random "*" in some of the comments.

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

Individual Inspection Preparation Log #1 (Page 36)

Name: Bernice Becher Date Log Submitted: October 15, 1993
Implementation: Pluto Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

GP, P-Spec 2.7.2, page 3: 175
"Calculate new attitude.*.*" through "Calculate new altitude:..."
This design calculates completely all the attitude values, then calculates completely all the velocity values, and then calculates completely all the altitude values. The specification says to calculate all three (attitude, velocity, and altitude) simultaneously.
*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)
*Software Requirements Document Reference: APPENDIX C, first paragraph, "If the Runge-Kutte method is used, it is required that the three equations be solved as a set of simultaneous equations.

GP, P-Spec 2.7.2, pages 3 and 9: 176
Problem: The set up of the GP_ROTATION MATRIX is not handled properly.
The only information given regarding the GP_ROTATION matrix is given at the top of page 9. This is neither a correct nor a sufficient explanation for setting up or for the use of the matrix. The specification states that one should "...use a temporary variable during calculation to hold the time histores of GP_ROTATION or to use elements directly from G_ROTATION; however, GP_ROTATION does describe...should contain the correct values for the present time step." All of this statement is being violated by this design. In addition, the correct setup must be done during or before the Runge-Kutte method is executed (on page 3)
*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)
*Software Requirements Document Reference: GP, SET UP THE GP_ROTATION MATRIX.

GP, P-Spec 2.7.2 181
Limit Checking, pages 4 - 8
1. The lower limit used for GP_ALTITUDE is incorrect.
2. The lower limit used for FRAME_ENGINES_IGNITED is incorrect.
3. There is no limit check for the upper bound on AE_TEMP.
4. The following input/output variables to this P-Spec are not checked at all for limit violations:
   GP_ATTITUDE, A_ACCELERATION, K_ALT, AE_SWITCH, AR_ALTITUDE, CONTOUR_CROSSESD, G_ROTATION, K_MATRIX, RE_SWITCH, VELOCITY_ERROR, TE_INTEGRAL, GP_ROTATION
5. The following variables are only checked for one case, namely the case where GP_PHASE = 1, and GP_ALTITUDE <= ENGINES_ON_ALTITUDE:
   FRAME_ENGINES_IGNITED, AE_TEMP, BHUTE_RELEASED, TDS_STATUS, GP_VELOCITY, TD_SENSED

*Requirement: Accuracy (Reference: DO-178B 6.3.2b).

GP, P-Spec 2.7.2, page 4, bottom 178
"if (GP_PHASE == one and GP_ALTITUDE[tnow] <= ENGINES_ON_ALTITUDE"
The term "tnow" has not been defined or explained.
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

Individual Inspection Preparation Log #1 (Page 37)

Name: Bernice Becher　　　Date Log Submitted: October 15, 1993

Implementation: Pluto　　　Date of Inspection October 15, 1993

Role: Inspector

Defects/Clarity Problems/Concerns

* GP, P-Spec 2.7.2, page 4, bottom through page 7,top 179

Algorithm for determining GP_PHASE, AE_SWITCH, RE_SWITCH, FRAME_ENGINES_IGNITED, and END_GCS:

Begins "if (GP_PHASE==one", ends with "end if"

Note on Notation: Let SQT represent sqrt (2 x GRAVITY X GP_ALTITUDE) + x component of GP_VELOCITY

Problems:

1. "hold = (2*GRAVITY*GP_ALTITUDE+GP_VELOCITY.[1])" "result = sqrt(2*GRAVITY*GP_ALTITUDE+GP_VELOCITY.[1])"
   Problem: Both of these statements incorrectly include the term "+GP_VELOCITY.[1]"
2. The case where GP_PHASE = 2 and AE_TEMP = hot and TDS_STATUS = healthy should not be setting GP_PHASE to 5. (line 3)
3. The case GP_PHASE = 3 and GP_ALTITUDE <= DROP_HEIGHT and TDS_STATUS is healthy should not be setting GP_PHASE to 4 or turning off engines without checking SQT and TD_SENSED. (line 4)
4. The two places under GP_PHASE == three that have the conditional: "else if (GP_ALTITUDE == DROP_HEIGHT and TDS_STATUS = failed"
   Problem 1: There is no reason to make of special case for GP_ALTITUDE exactly equal to DROP_HEIGHT since the specification doesn't make it a special case.
   Problem 2: Control will never reach the second conditional, and so it will never be executed. There is a contradiction in that these are the same conditionals yet call for two different types of processing to take place.
   Problem 3: It is not clear if control can ever reach the first conditional because it is actually a subset of the conditional that is executed before it, namely:
   "else if (GP_ALTITUDE <= DROP_HEIGHT and TDS_STATUS == failed"
   Problem 4: Under the second conditional is another contradiction. It contains the conditional:
   "if (result <= MAX_NORMAL_VELOCITY and TDS_STATUS == healthy"
   Control would never have reached here unless TDS_STATUS were failed, so this conditional can never be true.
5. The case where GP_PHASE = 3 and GP_ALTITUDE = DROP_HEIGHT, is not turning the engines off or setting END_GCS to TRUE. In fact, this case has already been treated correctly for GP_ALTITUDE <= DROP_HEIGHT and did not need to be handled again. (line 7)
6. The case where GP_PHASE=4 and TDS_STATUS = healthy and TD_SENSED is not sensed, should not be setting END_GCS to TRUE. (line 12)
7. At the top of page 7, " else (GP_PHASE == ?)" is ambiguous. (line 13)
8. Middle of page 6:
   "GP_STATUS = five"
   GP_STATUS is not a defined variable. (line 7)
9. The terms GP_ALTITUDE and GP_VELOCITY are used many time throughout these pages without any subscripts. This is ambiguous.
10. Middle of page 6:
    "result = ..GP_VELOCITY.[1]"
    This notation is not clear.,

11, "and TDS_STATUS = failed"

All other places use "==". Is this a typo, or does it have a different meaning.

Individual Inspection Preparation Log #1 (Page 38)

Name: Bernice Becher Date Log Submitted: October 15, 1993
Implementation: Pluto Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

Conclusions: This design has attempted to include the processing for Tables 5.9 and 5.10 into one large multi-page nested if-then-elseif statement. The merging of the two tables, and the many errors and inconsistencies make the design very confusing and very difficult to understand. The approach is so complicated, and there are enough errors in the control-handling logic, that it is impossible at this stage of the review to be certain that all the cases have been handled correctly. In fact, it seems that it would not be feasible to modify/maintain this particular design with an acceptable degree of accuracy.
*Requirement: Accuracy (Reference: DO-178B 6.3.2b).
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

GP, P-Spec 2.7.2, middle page 7 182
"This process would normally be done only once at initialization time;"
Question: what does this mean?
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

GP, P-Spec 2.7.2, middle page 7 184
"This process would normally be done...
Search the CONTOUR_VELOCITY array for a zero value... index value...while accessing the zero value."

Problem 1: "Search the CONTOUR_VELOCITY array for a zero value..."
Problem: The variable name here is incorrect.
*Requirement: Accuracy (Reference: DO-178B 6.3.2b).
Problem 2: The comments do not state explicitly of what we are attempting to find the size.
Problem 3: It is not impossible to determine the algorithm implied here, but the language used is imprecise and could lead to ambiguity. It is not an algorithmic solution.
Problem 4: "If off end of array, set size...if zero value is found, set size..."
Problem: There is no local variable named "size". In addition, there is no place in the pseudocode where "size" is explicitly used.
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

GP, P-Spec 2.7.2, middle page 7 186
"if (GP_ALTITUDE <= ENGINES_ON_ALTITUDE)"
Problem: According to the specification, the determination of the VELOCITY_ERROR is unconditional: therefore, this conditional is incorrect and introduces additional functionality.
*Requirement: Accuracy (Reference: DO-178B 6.3.2b).
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

GP, P-Spec 2.7.2, middle page 7 187
"Do a binary search in the...spurious results in his case."
Problem: The procedures for doing a binary search, interpolation and extrapolation are not explained in sufficient detail to represent an actual algorithmic solution.
*Requirement: Translatability to source code (Reference: Software Development Standards, Software Design Standards, "The low level requirements should be directly translatable into source code, with no further decomposition required.")

Individual Inspection Preparation Log #1 (Page 39)

Name: Bernice Becher   Date Log Submitted: October 15, 1993

Implementation: Pluto   Date of Inspection October 15, 1993

Role: Inspector

Defects/Clarity Problems/Concerns

GP, P-Spec 2.7.2, bottom page 7 188

"hold = ((GP_VELOCITY.x) ^ 2 + ... - interpolated velocity"

Problem 1: The specification states to use the x component of GP_VELOCITY. The design is using the magnitude of GP_VELOCITY.

Problem 2: The design is checking for an exceptional condition on a term which is not really the argument of the square root. (this problem may go away when problem 1 is fixed.)

*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)

*Software Requirements Reference: GP, DETERMINE VELOCITY ERROR

GP, P-Spec 2.7.2, top page 8 189

"if GP_ALTITUDE <= ENGINES_ON_ALTITUDE
and
VELOCITY_ERROR > 0 ..."

Problem: The relational operator ">" in the phrase VELOCITY_ERROR > 0" is incorrect.

*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)

*Software Requirements Reference: GP, DETERMINE IF CONTOUR HAS BEEN CROSSED

GP, P-Spec 2.7.2, middle page 8 192

"if (CL = first and optimal_velocity == ..."

Problem: The term "optimal_velocity" is not defined nor explained in this design and is therefore ambiguous.

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

GP, P-Spec 2.7.2, top page 9 190

"...the definitions of these terms given in section 2.7 GP..."

Problem: GP is no longer section 2.7 in the specification.

*Requirement: Accuracy (Reference: DO-178B 6.3.2b).

GP, P-Spec 2.7.2, bottom of page 8 through end of page 11 191

"NOTES:..."

Problem 2: The equations given for the derivatives do not provide sufficient detail to be translatable into code. TheIndividual matrix equation for the derivative of each of attitude, velocity, and altitude, should be explicity given. In each equation, it should be made completely clear which time history value or calculated value should be used for any of the three variables GP_ATTITUDE, GP_VELOCITY, and/or GP_ALTITUDE, as well as which time history values should be used for the sensor variables (G_ROTATION, A_ACCELERATION, K_MATRIX, TDLR_VELOCITY, K_ALT, and/or AR_ALTITUDE) which appear in that particular equation. The derivative equations being referenced here are:

d/dt(Vbl.[2])
d/dt(Vbl.[1]est_A)
d/dt(Vbl.[1]est_B)
d/dt(Vbl.[0]est_C)

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

Individual Inspection Preparation Log #1 (Page 40)

Name: Bernice Becher　　Date Log Submitted: October 15, 1993

Implementation: Pluto　　Date of Inspection October 15, 1993

Role: Inspector

Defects/Clarity Problems/Concerns

RECLP, P-Spec 2.8

RECLP, P-Specs 2.8.1 and 2.8.3 (RECLP_EXPAND and RECLP_COMPRESS) 226

These P-Specs do not appear to have any useful function. A P-Spec should perform some function that converts its input elements to its outputs. These P-Specs seem to convert from control flow group names to element names and back, which is not an actual function. Why would there be a P- Spec with no body?

*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

RECLP P-Spec 2.8.2, page 1, TITLE 227

"RECLP - Roll Engine Contrl Law Processing (P-Spec 2.3.2)"

Problem: There is some confusion about the P-Spec number. At the top of the page is 2.8.2 which is the correct P-Spec number for this design. The title, on the other hand, contains P-Spec number 2.3.2 which is probably from the Software Requirements document. There needs to be some clarification here.

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

RECLP P-Spec 2.8.2 220

Middle of page 2: "if (G_ROTATION.x < -1"

Bottom of page 2: "x_roll_rate = G_ROTATION.x"

Problem: The variable G_ROTATION is a history variable, but no history subscript is indicated here.

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

RECLP P-Spec 2.8.2, bottom of page 2 224

"if (THETA < PI)"

"else if (THETA > PI)"

Problem: No definition has been given for "PI".

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

RECLP P-Spec 2.8.2 225

Limit Checking

1. Bottom page 2:

The lower and upper limit checks for THETA taken together imply that THETA must be exactly equal to some number PI, which is not correct according to the Data Dictionary of the specification.

2. Limit checks are missing for the following output variables: RE_CMD, RE_STATUS, THETA

*Requirement: Accuracy (Reference: DO-178B 6.3.2b).

*Requirement: Completeness (Reference: DO-178B 11.0b)

RECLP P-Spec 2.8.2, bottom of page 2 221

"* Determine which region of the graph (Figure 5.10 pg 60 of spec..."

Problem: Neither the Figure Number nor the page number is correct.

*Requirement: Accuracy (Reference: DO-178B 6.3.2b).

Individual Inspection Preparation Log #1 (Page 41)

Name: Bernice Becher   Date Log Submitted: October 15, 1993
Implementation: Pluto   Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

RECLP P-Spec 2.8.2, bottom of page 2 and top of page 3 222
"Use "if" statments constructed using...command should be used."
Problem: This description of how to find the correct region is inadequate and does not provide enough detail to be an algorithmic solution which can be translated to code. There is nothing in this description to even indicate which variables (other than the RUN_PARAMETER variables) are involved in finding the region nor how they are to be used.
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).
*Requirement: Translatability to source code (Reference: Software Development Standards, Software Design Standards, "The low level requirements should be directly translatable into source code, with no further decomposition required.")

RECLP P-Spec 2.8.2, page 3 223
Problem: The term "lowest bit(s)" is used in three different places. It needs a precise definition.
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

RECLP P-Spec 2.8.2, middle of page 3 219
"Give error message."
The variable RE_SWITCH has already been checked and handled properly for values outside its acceptable range. This error message represents added functionality which cannot be traced to the specification.
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

Individual Inspection Preparation Log #1 (Page 42)

Name: Bernice Becher Date Log Submitted: October 15, 1993
Implementation: Pluto Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

TDLRSP, P-Spec 2.9.2

TDLRSP DFD 28
TDLR_STATUS appears as an input to P-Spec 2.9.2. It is not an input to TDLRSP.(also see #94) (SEE FORMAL MODIFICATION 2.2-16.2)
*Requirement: Accuracy (Reference: DO-178B 6.3.2b).

TDLRSP P-Specs 2.9.1 and 2.9.3(TDLRSP_EXPAND and TDLRSP_COMPRESS) 29
These P-Specs do not appear to have any useful function. A P-Spec should perform some function that converts its input elements to its outputs. These P-Specs seem to convert from control flow group names to element names and back, which is not an actual function. Why would there be a P- Spec with no body?
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

TDLRSP P-Spec 2.9.2, page 1, TITLE 107
"TDLRSP - Touch Down Lander Radar Sensor Processing (P-Spec 2.1.3)"
Problem: There is some confusion about the P-Spec number. At the top of the page is 2.9.2 which is the correct P-Spec number for this design. The title, on the other hand, contains P-Spec number 2.1.3 which is probably from the Software Requirements document. There needs to be some clarification here.
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

TDLRSP P-Spec 2.9.2, page 2, TOP of page: 30
"Shift the data in the FIFOS: TDLR_VELOCITY.#..."
Problem: More detail is needed.
*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)
**Software Requirements 2.2 with Mods 1-26 Reference: TDLRSP, Rotate Variables
*Requirement: Translatability to source code (Reference: Software Development Standards, Software Design Standards, "The low level requirements should be directly translatable into source code, with no further decomposition required.")

TDLRSP P-Spec 2.9.2, page 2 128
"if (TDLR_VELOCITY.x < -100)"
"else if (TDLR_VELOCITY.x > 100)"
Problem 1: It is not clear exactly what the notation ".x" means (see #122). If it refers to just the first element, why is an additional check being made for all elements at the bottom of page 4?
Problem 2: It is not clear which elements in the time history are being checked. If the most recent are implied, then there is a problem because the rotation has already taken place but the new element has not yet been calculated..
*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)
**Software Requirements 2.2 with Mods 1-26 Reference: Introduction, Exception Handling, Upper or Lower Limit Exceeded
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

Individual Inspection Preparation Log #1 (Page 43)

Name: Bernice Becher — Date Log Submitted: October 15, 1993
Implementation: Pluto — Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

TDLRSP P-Spec 2.9.2, page 2 129
"if (K_MATRIX.*<0)"
"else if (K_MATIX.* > 1)"
Problem 1: K_MATRIX has three dimensions. It's not clear here which dimensions are being checked.
Problem 2: It is not clear which elements in the time history are being checked. If the most recent are implied, then there is a problem because the rotation has already taken place but the new elements have not yet been calculated.
*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)
**Software Requirements 2.2 with Mods 1-26 Reference: Introduction, Exception Handling, Upper or Lower Limit Exceeded
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

TDLRSP P-Spec 2.9.2, page 2, MIDDLE of page: 31
"if (FRAME_COUNTER == even)
set TDLR_VELOCITY.* to previous value of TDLR_VELOCITY.*
set K_MATRIX.* to previous value of K_MATRIX.*
exit..."
Problem: The step before this in the design was to rotate these same variables unconditionally. These assignments will cause a second rotation, which is incorrect.
*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)
**Software Requirements 2.2 with Mods 1-26 Reference: TDLRSP, PERFORM ALTERNATE PROCESSING IF THIS IS AN EVEN-NUMBERED FRAME

TDLRSP P-Spec 2.9.2, page 2 130
"if (TDLR_STATE<0)"
"else if (TDLR_STATE > 1)"
Problem: It is not clear which element in the time history is being checked. If the most recent is implied, then there is a problem because the rotation has already taken place but the new elements have not yet been calculated.
*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)
**Software Requirements 2.2 with Mods 1-26 Reference: Introduction, Exception Handling, Upper or Lower Limit Exceeded
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

TDLRSP P-Spec 2.9.2, page 3 131
"if (FRAME_BEAM_UNLOCKED<0)"
"else if (TDLR_STATE > 1)"
Problem 1: It is not clear which element in the time history is being checked.
Problem 2: The variable FRAME_BEAM_UNLOCKED should also be set later on this page (see #34); therefore, this is either the incorrect place to check for limits or else they must also be checked elsewhere in addition.
*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)

Individual Inspection Preparation Log #1 (Page 44)

Name: Bernice Becher   Date Log Submitted: October 15, 1993
Implementation: Pluto   Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

**Software Requirements 2.2 with Mods 1-26 Reference: Introduction, Exception Handling, Upper or Lower Limit Exceeded
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

TDLRSP P-Spec 2.9.2, page 3, MIDDLE of page: 33
"Test to determine if a beam has locked again:"

"The beam can now be used *
TDLR_STATE.# = locked
FRAME_BEAM_UNLOCKED.# = 0"
Problem: In the Software Requirements document, Table 5.11, line 2, FRAME_BEAM_UNLOCKED is not supposed to be changed, but the design does change it in this case.
*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)
**Software Requirements 2.2 with Mods 1-26 Reference: TDLRSP, DETERMINE RADAR BEAM STATES

TDLRSP P-Spec 2.9.2, page 3, MIDDLE of page: 34
"Test to determine if a beam has locked again:"
"The beam can now be used *
...
else
*Beam is unlocked and remains unlocked *
endif
Problem: In the Software Requirements document, Table 5.11, line 3, FRAME_BEAM_UNLOCKED should be set to the value of FRAME_COUNTER, but in this case (between the "else" and the "endif") it is not being changed.
*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)
**Software Requirements 2.2 with Mods 1-26 Reference: TDLRSP, DETERMINE RADAR BEAM STATES

TDLRSP P-Spec 2.9.2, page 3, most of page: 136
The design checks for conditions in line 1 of Table 5.11 (in specification) and then even if it has found and processed the conditions in line 1, it goes on to check and process for conditions for lines 2 and 3. The intention in the specification was to only process one line of the table. It is possible that line 1 would be processed, setting TDLR_STATE to locked, and then line 3 could also be processed. This would not cause a problem in this case, but this was not the intent of the specification. Perhaps a modification to the specification is required.

TDLRSP P-Spec 2.9.2 32
Page 3, bottom of page:
"Average(resolve)...using the table named AVERAGING DOPPLER RADAR BEAMS IN LOCK..."
Page 4, middle of page, under CLASS 2, CLASS 3, and CLASS 4:
In each case it states to calculate average_velocity, but does not give the particular equations. One can deduce by going back to the comment quoted above from page 3, that one is to use the equations in the table, but the comment is not specific enough and has an incorrect table name and does not give the table number. In any case, neither the table reference nor the actual equation is given in the design body on page 4.

Individual Inspection Preparation Log #1 (Page 45)

Name: Bernice Becher   Date Log Submitted: October 15, 1993

Implementation: Pluto   Date of Inspection October 15, 1993

Role: Inspector

Defects/Clarity Problems/Concerns

*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)

**Software Requirements 2.2 with Mods 1-26 Reference: TDLRSP, PROCESS THE BEAM VELOCITIES

*Requirement: Translatability to source code (Reference: Software Development Standards, Software Design Standards, "The low level requirements should be directly translatable into source code, with no further decomposition required.")

TDLRSP P-Spec 2.9.2, page 4 132

"if (TDLR_VELOCITY.* < -100)"

"else if (TDLR_VELOCITY.* > 100)"

Problem 1: It is not clear which elements in the time history are being checked.

Problem 2: Why is one element being checked on page 2 and all elements being checked on page 4?

*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)

**Software Requirements 2.2 with Mods 1-26 Reference: Introduction, Exception Handling, Upper or Lower Limit Exceeded

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

TDLRSP P-Spec 2.9.2 133

TDLR_STATUS has not been checked for limits exceeded.

*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)

**Software Requirements 2.2 with Mods 1-26 Reference: Introduction, Exception Handling, Upper or Lower Limit Exceeded

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

TDLRSP P-SPEC 2.9.2, pages 3 and 4 126

The notation ".#"

Problem 1: Because of this notation, it is not clear where the control loops must be, or, if in fact it makes any difference where the loops are.

Problem 2: The Averaging of the beams beginning at the bottom of page 3 and continuing to page 4 cannot be done until all the steps through "Calculate all four RADAR beam velocities" has been completed (for all four beams). Because of the confusion due to ".#" over where the loops must be, the above fact stated in the previous sentence may not be explicitly clear.

Individual Inspection Preparation Log #1 (Page 46)

Name: Bernice Becher   Date Log Submitted: October 15, 1993

Implementation: Pluto   Date of Inspection October 15, 1993

Role: Inspector

Defects/Clarity Problems/Concerns

TDSP, P-Spec 2.10.2

TDSP, P-Specs 2.10.1 and 2.10.3 (TDSP_EXPAND and TDSP_COMPRESS) 140
These P-Specs do not appear to have any useful function. A P-Spec should perform some function that converts its input elements to its outputs. These P-Specs seem to convert from control flow group names to element names and back, which is not an actual function. Why would there be a P- Spec with no body?
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

TDSP P-Spec 2.10.2, page 1, TITLE 150
"TDSP - Touch Down Sensor Processing (P-Spec 2.1.6)"
Problem: There is some confusion about the P-Spec number. At the top of the page is 2.10.2 which is the correct P-Spec number for this design. The title, on the other hand, contains P-Spec number 2.1.6 which is probably from the Software Requirements document. There needs to be some clarification here.
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

TDSP, P-Spec 2.10.2, page 2, middle of page 141
"else
Give message "TDS_STATUS has bad value..."

Problem: The only way control could get here is for TDS_STATUS to have a value outside of its range. This problem would have already been handled by the exception handling on page 1, so this pseudocode represents additional functionality over what the specification has required.
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

TDSP, P-Spec 2.10.2, page 1 and page 2 142
Question: Why is the TDS_STATUS limit check made before the variable is set, while the TD_SENSED limit check is made after the variable is set.
*Requirement: Consistency (DO-178B 5.2.2a, 6.3.2b, and 11.0d)

Individual Inspection Preparation Log #1 (Page 47)

Name: Bernice Becher Date Log Submitted: October 15, 1993

Implementation: Pluto Date of Inspection October 15, 1993

Role: Inspector

Defects/Clarity Problems/Concerns

TSP, P-Spec 2.11.2

TSP 2.11 DFD 6

TS_STATUS shows as input to TSP, but it is not an input to TSP. It also incorrectly appears as "TEMP_GS_IN" (see #93). (See Formal Modification 2.2-17.2).

*Requirement: Accuracy (Reference: DO-178B 6.3.2b).

TSP P_Spec 2.11.1 and 2.11.3 (Data Expand and Data Compress) 7

These P-Specs do not appear to have any useful function. A P-Spec should perform some function that converts its input elements to its outputs. These P-Specs seem to convert from control flow group names to element names and back, which is not an actual function. Why would there be a P- Spec with no body?

*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

TSP P-Spec 2.11.2, page 1, TITLE 108

"TSP - Temperature Sensor Processing (P-Spec 2.1.5)"

Problem: There is some confusion about the P-Spec number. At the top of the page is 2.11.2 which is the correct P-Spec number for this design. The title, on the other hand, contains P-Spec number 2.1.5 which is probably from the Software Requirements document. There needs to be some clarification here.

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

TSP P-Spec 2.11.2, page 4 8

"Determine which expression to use to calculate THERMOCOUPLE temperature:"

```
"if (THERMO_TEMP >= lo_meas_limit_tc
          and
     THERMO_TEMP < M3..."    ...
            .
            .
"ELSE IF (THERMO_TEMP > m4
          AND
     THERMO_TEMP <= hi_meas_limit_tc)"
```

Problem: In the first conditional, the first relational expression is unnecessary, and in the second conditional the second relational expression is unnecessary. It is conceivable that these expressions could cause a problem. It has previously been determined that the thermocouple sensor should be used, and therefore we should not exit from this section without setting ATMOSPHERIC_TEMP to some value base on THERMO_TEMP. Since there may be a roundoff in the calculations of lo_meas_limit_tc and/or hi_meas_limit_tc, it is possible these unnecessary expressions might cause the "if" to yield "false" where it might otherwise yield "true", and the result would be an undefined value for ATMOSPHERIC_TEMP.

*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)

**Software Requirements 2.2 with Mods 1-26 Reference: TSP, Calculate the Thermocouple Temperature, "Use the value of THERMO_TEMP to determine whether the temperature lies in the thermocouple linear or the upper parabolic or the lower parabolic region."

*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

Individual Inspection Preparation Log #1 (Page 48)

Name: Bernice Becher Date Log Submitted: October 15, 1993

Implementation: Pluto Date of Inspection October 15, 1993

Role: Inspector

Defects/Clarity Problems/Concerns

TSP P-Spec 2.11.2 135

The variable TS_STATUS has not been checked for upper/lower limit exceeded.

*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)

**Software Requirements 2.2 with Mods 1-26 Reference: Introduction, Exception Handling, Upper or Lower Limit Exceeded

* TSP P-Spec 2.11.2, page 1 154

"BEGIN LOCAL TYPE DEFS" real*4 ell . . real*4 hold
END LOCAL TYPE DEFS"

MISCELLANEOUS P-Specs (not the eleven functional units)

GENERATE_SEQUENCE_PARMS (store) and GENERATE_SEQUENCE_PARMS P-Spec 2.18 118

The fact that GENERATE_SEQUENCE_PARMS is used as the name for a data store and as the name for a process is confusing. Since the store is "not- defined", and since it doesn't appear as a store on any of the DFD/CFD diagrams, it's not clear what is its function, if any. (see #60)

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

COPY CONTROL DATA, P-SPEC 2.18, page 1 (see #98 also) 100

"Copy INIT_SUBFRAME_COUNTER to SUBFRAME_COUNTER"
There are several problems with this statement.

Problem: It seems there is no reason to copy SUBFRAME_COUNTER to anywhere since it already exists in the global data store EXTERNAL. Why is this being done, and what is the purpose for the store "SUBFRAME_COUNTER_STORE

*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

Problem: INIT_SUBFRAME_COUNTER is a control flow, while SUBFRAME_COUNTER is a data flow.

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

Problem: The use of the name "SUBFRAME_COUNTER" is ambiguous because it appears in the stores EXTERNAL_OLD, SUBFRAME_COUNTER_STORE, and in the global store EXTERNAL (defined in the Software Requirements document but missing from the store EXTERNAL in this design document). One can look at the RUN_GCS DFD to see that the intention is to store into SUBFRAME_COUNTER_STORE, but the P-Spec itself should be self-contained.

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

Potential Problem: If it is intended that SUBFRAME_COUNTER in the store EXTERNAL is to be changed, then this would be a violation of the requirements because in the Software Requirements document, SUBFRAME_COUNTER is not an output for any functional unit.

*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a,

Individual Inspection Preparation Log #1 (Page 49)

Name: Bernice Becher Date Log Submitted: October 15, 1993
Implementation: Pluto Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

Miscellaneous P-Specs with no body 138
INIT EXTERNAL STORE (P-SPEC 2.12)
STORE RAW SENSOR DATA (P-SPEC 2.13)
INIT RUN PARM STORE (P-Spec 2.14)
INIT GUIDANCE STATE STORE (P-Spec 2.15)
SEND CHUTE RELEASE COMMAND (P-Spec 2.16)
SEND ENGINE DATA (P-Spec 2.17)

It is not immediately clear what is the function of these P-Specs.
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

* Miscellaneous P-Specs with no body 196
The design states that "This P-Spec exists because Teamwork cannot send data flows off page (so an intervening bubble is required)" ; however, this seems to be contradicted by Figures 2.4 and 2.5 in the Specification. Could this be looked into?
*Requirement: Completeness (Reference: DO-178B 11.0b)

Miscellaneous

Invocation of Rendezvous 119
There is nothing in the design which states exactly how and when to invoke the rendezvous routine.
*Requirement: Fullfillment of requirements in specification (References: DO-178B 6.3.2a and 11.10a; Software Requirements 2.2 with Mods 1-26:
*Requirement: Reference: Software Requirements 2.2 with Mods 1-26, Appendix B, "Process", "The calling convention for this GCS_SIM provided support utility is as follows: 9 GCS_SIM_RENDEZVOUS (requires no parameters) "

Teamwork Balancing 137
Question: Has the Teamwork balancing been done? Should this be included in the design?

General 213
There are many places in the design where the name of a variable which contains a time history is used with no history subscript. An example is GP, P-Spec 2.7.2 where GP_ALTITUDE and GP_VELOCITY are used with no history subscripts.The design should find some method for removing this type of ambiguity.
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

Individual Inspection Preparation Log #1 (Page 50)

Name: Bernice Becher Date Log Submitted: October 15, 1993
Implementation: Pluto Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

Typographic Errors, Style, and Grammar

Introduction
2.5 Revision History, second statement. Grammar: It's not considered a good practice to use the pronoun "I" in a technical document.

Introduction
2.4 Transition History, second statement. Clarity: This is not a correct grammatical sentence as it has no subject.

P-Spec 2.11.2
"SERSOR'S" should be "SENSOR'S"

P-Spec 2.2.2
"reciept" should be "receipt"

P-Spec 2.9.2
"TDLR_VELOCITYV" should be "TDLR_VELOCITY"

P-Spec 2.4.5, bottom of page 1
"accesed" should be "accessed"

Data Dictionary
A_COUNTER: "accelerating" should be "accelerations"

Data Dictionary
A_SCALE: "RUN_PAREMETERS" should be "RUN_PARAMETERS"

Data Dictionary
ALPHA_MATRIX: "rea*8" should be "real*8"

DATA DICTIONARY GUIDE_SO_IN "ATMOSPHEREIC_TEMP" should be "ATMOSPHERIC_TEMP"

DATA DICTIONARY TDLR_ANGLES "y;" should be "gamma;"

Individual Inspection Preparation Log #1 (Page 51)

Name: Bernice Becher Date Log Submitted: October 15, 1993
Implementation: Pluto Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

Suggestions for the Future

It would be helpful if the entire design document were numbered sequentially from beginning to end.
There doesn't seem to be an attempt on part of the designer to simplify equations. (see TSP equations for parabola. see eqn for note 1 m3 - m_lo = m3 - [m3...]). Perhaps we could request this.

Constants used for limit checking make modification difficult and error- prone.

In the pseudocode, nested if's spanning many pages makes the logic extremely difficult to follow and may lead to an error-prone inspection. As an example, see AECLP, P-Spec 2.1.2, where one nested if begins on page 3, nests to a depth of four, and does not terminate until page 7. The suggestion is that the standards require that any time an if statement spans more than one page, that the if, else, elseif, and endif (or whatever syntax is used) be meticulously labeled in all places so that the scope of each "if" is immediately obvious.

It would be very helpful if the designer, when using an algorithm that is not in the specification, gave either a text reference or the derivation of the algorithm and well as an explanation as to how it is being applied to GCS.

The SA charts and tables and entries in the Data Dictionary seem overly complicated and difficult to follow. Is there any way we can ask for simplicity of design? We might want to simplify the structured analysis diagrams in the specification (minimize the use of control flows).

It would be helpful if the titles for diagrams could say what that diagram is, eg, DFD/CFD, PAT etc

Is there some way we can add something to the standards to keep designers/coders from using code that is completely superfluous?

Can we add something to the standards to force the designer to be explicit about what is a comment and what is actual pseudocode/structured English?

Can we add something to the standards to force the designer to use very specific non-ambiguous language?

Require that a Teamwork Balance Report (with no errors) be included as part of the design.

Individual Inspection Preparation Log #1 (Page 52)

Name: Bernice Becher Date Log Submitted: October 15, 1993
Implementation: Pluto Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

QUESTIONABLE ITEMS

Introduction 48

4. Notation in Pluto Version of GCS Design Page 6, middle paragraph "Another syntax..." There is no statement about whether the array under discussion for the current subframe has been rotated or is about to be rotated
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

DATA DICTIONARY
INIT_RP_OUT 88
The intention here seems to be to duplicate the data flow names in RUN_PARAMETERS data store. If this is the case, then MAX_NORMAL_VELOCITY has been omitted.
*Requirement: Accuracy (Reference: DO-178B 6.3.2b).

DATA DICTIONARY
F 59
This has one value, "FALSE". It seems to be a misuse of the Data Dictionary to put a constant into it. It is supposed to be used for data flows, control flows and/or data conditions.
*Requirement: Follow a particular design method (References: Software Development Standards,"Software Design Standards", "Design Methods, Rules, and Tools", "...using the structured analysis ...by Hatley and Pirbhai or...", and "Design Documentation", "...document should follow...GCS specification document or the Hatley book...")

DATA DICTIONARY
INIT_RENDEZVOUS_CNTL 63
"RUNNING"
Question: This element is a constant. Why is it in the data dictionary?
*Requirement: Follow a particular design method (References: Software Development Standards,"Software Design Standards", "Design Methods, Rules, and Tools", "...using the structured analysis ...by Hatley and Pirbhai or...", and "Design Documentation", "...document should follow...GCS specification or the Hatley book...")

DATA DICTIONARY
INIT_SUBFRAME_COUNTER 64
"1"
Question: This element is a constant. Why is it in the data dictionary?
*Requirement: Follow a particular design method (References: Software Development Standards,"Software Design Standards", "Design Methods, Rules, and Tools", "...using the structured analysis ...by Hatley and Pirbhai or...", and "Design Documentation", "...document should follow...GCS specification or the Hatley book...")

Individual Inspection Preparation Log #1 (Page 53)

Name: Bernice Becher   Date Log Submitted: October 15, 1993
Implementation: Pluto   Date of Inspection October 15, 1993
Role: Inspector

Defects/Clarity Problems/Concerns

NOT USED

DATA DICTIONARY 126
TD_LND_RAD_GS_IN TDLR_STATUS is not an input from GUIDANCE_STATE store to TDLRSP.

STORE RAW SENSOR DATA, P-Spec 2.13 82
This P-Spec does not perform any function traceable to the Software Requirements document. (see #81)
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

INIT RUN PARM STORE, P-Spec 2.14 85
This P-Spec does not perform any function traceable to the Software Requirements document. (SEE #84)
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

INIT GUIDANCE STATE STORE, P-Spec 2.15 87
This P-Spec does not perform any function traceable to the Software Requirements document. (SEE #86)
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

DATA DICTIONARY, pages 18 and 19 113
The element EXTERNAL is defined as a store but yet is shown as a group flow name. This is not consistent.
*Requirement: Consistency (DO-178B 5.2.2a, 6.3.2b, and 11.0d)

DATA DICTIONARY, pages 18 and 19 114
The element EXTERNAL is shown as a group flow name; however, some of the primitive elements in the group (e.g. AE_CMD, FRAME_COUNTER) are repeated several times in the list).
*Requirement: Consistency (DO-178B 5.2.2a, 6.3.2b, and 11.0d)
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a,

DATA DICTIONARY,page 43 18

Introduction
CALLING STRUCTURE (reword???) 110
The calling structure, especially in terms of rendezvous, is not shown directly.

Introduction 38
1.1 Top-Level Description, first paragraph
The wording "touch-down switch" is not an accurate description.
* Requirement: Accuracy (DO-178B 6.3.2 b)

Individual Inspection Preparation Log #1 (Page 54)

Name: Bernice Becher Date Log Submitted: October 15, 1993

Implementation: Pluto Date of Inspection October 15, 1993

Role: Inspector

Defects/Clarity Problems/Concerns

1

*Requirement: Accuracy (Reference: DO-178B 6.3.2b).

2

*Requirement: Nonambiguity (Reference: DO-178B 11.0a).

3

*Requirement: Follow a particular design method (References: Software Development Standards,"Software Design Standards", "Design Methods, Rules, and Tools", "...using the structured analysis ...by Hatley and Pirbhai or...", and "Design Documentation", "...document should follow...GCS specification or the Hatley book...")

4

*Requirement: Consistency (DO-178B 5.2.2a, 6.3.2b, and 11.0d)

5

*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)

6

*Requirement: Translatability to source code (Reference: Software Development Standards, Software Design Standards, "The low level requirements should be directly translatable into source code, with no further decomposition required.")

8

*Requirement: Completeness (Reference: DO-178B 11.0b)

10

*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)

11

*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)

**Software Requirements 2.2 with Mods 1-26 Reference: Introduction, Exception Handling, Upper or Lower Limit Exceeded

==

Added since last design review of 10/15/93

Two typos were by mistake in the data dictionary. They were moved to the typos.

A suggestion was added to require that the design include a balancing report.

Review Log from Verification Analyst

Individual (Design) Inspection Log 10/15/93 (Final)
Rob Angellatta
Pluto

General Deficiencies

The overall quality of the Pluto design is disappointing. Listed below are several general comments supporting this opinion. Preparing a problem report for the listed items is probably unnecessary, however some feedback on the “sloppiness" of the design may prove beneficial.

- o The syntax for referencing array data as described on page 6 of the "Design Description Document - Pluto" is confusing and inconsistently followed. For example, as found on page 6, "The '.*' in equations following a variable name or comment indicates independent iteration over each of the 3 lander body axis directions: x, y, & z." P-Spec 2.2.2 ARSP contains the following operation: "AR_ALTITUDE.* = ..." The data element AR_ALTITUDE is an array and represents history data, not vehicle axial data. Thus, the reference is inconsistent with the defined usage. Additionally, also in P-Spec 2.2.2 ARSP, the current value of the vehicle altitude is referenced in one location as AR_ALTITUDE.[0], and in another location referenced as AR_ALTITUDE.*, providing another example of inconstant (and incorrect) use of the defined syntax.

- o There are several instances where the design should contain a brief description of the designer's intentions. For instance, in P-Spec 2.11.2 TSP several operations are presented for computing the temperature from the solid-state temperature sensor. A brief narration of the intent of the operations is in order.

- o And then, there are several instances where a description of a solution is provided, but no algorithm for implementing the solution is presented. For instance, in P-Spec 2.2.2 ARSP a thorough (although incorrect) description of the Newton Divided Difference Method for extrapolation is provided, but no algorithm for implementing the method is presented. This example is poignant because the description of the method is flawed. So the question arises, does the designer really understand the method (and its application) merely mistaken in the explanation or does the designer not really understand the method? An algorithm implementing the solution would certainly provide the necessary insight into the designers understanding of the problem and the proposed solution.

- o At the design level of abstraction a data element of type "logical" can assume one of two values, namely "TRUE" or "FALSE." The Pluto design contains many references to data element of type logical assigning values of "0", "1", "healthy", "failed", and so forth. Technically, it is incorrect to refer to a "logical" data type as any value other than "TRUE" or "FALSE." I do not attribute this deficiency to the Pluto design so much as to the GCS Programming Specification. The spec is full of such references and it is this type of mistake which significantly contributes to the "sloppiness" and "amateur" appearance of both the programming specification and the Pluto design.

- o A comparison of the Pluto data dictionary entries (DDE) with the DDE's of the programming specification uncovers defects for the following entries:

- o A_COUNTER -- A typo in the "description" field.
- o COMM_SYNC_PATTERN -- The value specified in the "range" field is ambiguous. The value is apparently a bit pattern, however the chosen syntax for expressing this value does not make this fact clear. This defect also appears in the programming specification.
- o GP_DONE -- Missing the field "data type."
- o K_MATRIX -- The value of the field "accuracy" is inconsistent with the programming spec.
- o THETA1 -- The field "data store location" does not exist.

Defects/Clarity Problems/Concerns

Location Description

1 EXTERNAL data store The Pluto design contains a data store labeled "EXTERNAL." However, Pluto's "EXTERNAL" data store is inconsistent with the programming specification (missing data elements "PACKET" and "SUBFRAME COUNTER") . (Note, Pluto's "EXTERNAL_OLD" data store is consist with the programming specifications "EXTERNAL" data store.) See the SPEC pgs. 13-14 for requirements and table 6.2 on page 98 for data store description.

2 GUIDANCE STATE data store The Pluto design contains a data store labeled "GUIDANCE STATE." However, Pluto's "GUIDANCE STATE" data store is inconsistent with the programming specification (contains additional data elements "TDLRSP_SWITCH" and "TDSP_SWITCH"). See the SPEC pgs. 13-14 for requirements and table 6.1 on page 97 for data store description.

3 SENSOR_OUTPUT data store The Pluto design does not contain the required data store labeled "SENSOR_OUTPUT." However, Pluto's "SENSOR_DATA" data store appears to be consistent with the programming specification for the data store "SENSOR_OUTPUT." See the SPEC pgs. 13-14 for requirements and table 6.3 on page 98 for data store description.

4 P-Spec 1, INIT_GCS The low-level specifications for this process should not be specified in the Pluto design. The programming spec, page 31 "LEVEL 2 SPECIFICATION" , clearly states, "INIT_GCS ... are not the responsibility of the programmer."

5 P-Spec 3, GENERATE_SEQUENCE_PARAMS Missing algorithms. Insufficient detail is specified as to how to determine the state of theredata elements "ITH_FRAME_2" and "ITH_FRAME_5." It is not clear weather or not the designer truly understands which frames are the "ith_frame_2s" and which frames are "ith_frame_5s." Assume that the Pluto design description was given to a programmer for code implementation. Would the programmer clearly understand which frames to designate "ith_frame_2" and which frames to designate "ith_frame_5" from this description?

6 P-Spec 0-s1, RUN_GCS PAT Inconsistent with P-Spec 2.s1. The PAT appear to specify that the order of process execution for the processes "RUN_GCS" and "GENERATE_SEQUENCE_PARMS" is insignificant. However, P-Spec 2.s1 which controls the processing within the process RUN_GCS clearly depends upon the value of the data elements "ITH_FRAME_2" and "ITH_FRAME_5" which are updated in the process "GENERATE_SEQUENCE_PARMS."

7 ?? GCS_SIM_RENDEZVOUS ?? There is an obvious absence of the process GCS_SIM_RENDEZVOUS. The programming spec page 31, LEVEL 2 SPECIFICATION clear states "There should be a call to GCS_SIM_RENDEZVOUS, prior to executing each subframe."

8 P-Spec 2, RUN_GCS There are a number of control signals defined, data elements like "AECLP_DONE", "ASP_DONE" and so forth. Where are these control signals set/reset? I can not find any evidence to suggest these signals are properly manipulated? It's frustrated -- we "know" how they are supposed to be manipulated, but how would a programmer "know" from just the design?

Defects/Clarity Problems/Concerns

	Location	Description
9	P-Spec 2.11.2;10 TSP	The design assumes that (M3,T3) < (M4,T4). This is a valid assumption only because figure 5.4 implies this is true. Other then figure 5.4, the spec is not clear on this point.
10	P-Spec 2.11.2;10 TSP	??? Inconsistency. The data element "TS_STATUS" is designated Input/output on the bubble diagram 2.11. The data element "TS_STATUS" is designated output in the P-Spec 2.11.2;10. The programming specification lists "TS_STATUS" as output.
11	P-Spec 2.11.2;10 TSP	I believe that the method for computing the temperature from the solid state temperature sensor requires an explanation, some narration.
12	P-Spec 2.11.2;10 TSP	The method for computing the upper and lower limits of the thermocouple temperature sensor range most definitely requires an explanation, some narration.
13	P-Spec 2.2.2;22	ARSP
14	P-Spec 2.2.2;22 ARSP	The syntax AR_ALTITUDE.*, AR_STATUS.*, and K_ALT.* is inconsistent with the definition of ".*" as specified in the "Design Description Document -- Pluto" page 7.
15	P-Spec 2.2.2;22 ARSP	There are several instances where a data element is assigned a previously computed value of a data element, denoted by the expression "[previous value." In these instances, four previously computed values are available for the assignment. The intent is to assign the most recently computed value, not just any previously computed value. Thus, in these instances the design is ambiguous as to which previously computed value is used for the assignment operation.
16	P-Spec 2.2.2;22 ARSP	When computing the altitude in the case where an echo is received, a check for the exception condition "upper limit exceeded" is absent.
17	P-Spec 2.2.2;22 ARSP	The description of the Newton Dividend Difference method for extrapolation -- I expect to see this description in the "Design Description Document." However, here in the design itself, I expect to see an algorithm implementing this method. Thus, I believe that the design provides insufficient detail.
18	P-Spec 2.2.2;22 ARSP	The description of the Newton Dividend Difference method for extrapolation -- The first step under "construct a table of divided differences" states "The first column of the table holds the four previous altitudes." The ordering of the four previous value is significant, however the ordering of the four previous values if unspecified. Thus, the statement is ambiguous.
19	P-Spec 2.2.2;22 ARSP	The second, third, and forth steps under "build a polynomial" state "... the first (most recent) index in column ... " These references are inconsistent with step one where the most recent value is located in the last element of the column.

Defects/Clarity Problems/Concerns

	Location	Description
20	P-Spec 2.2.2;22 ARSP	When computing the altitude in the case where a value must be extrapolated from the previous computations, there is an absence of checks for the exception conditions "lower limit exceeded" and "upper limit exceeded."
21	P-Spec 2.2.2;22 ARSP	When reporting the altitude as the most recent previously reported value, the statement "AR_ALTITUDE.* = AR_ALTITUDE.[previous value" is deficient. First, the syntax is "AR_ALTITUDE.* " is inconsistent with with the definition of ".*" as specified in the "Design Description Document -- Pluto" page 7. This is really sloppy as the appropriate syntax is used earlier in the P-Spec. Second, the statement "[previous value" is ambiguous.
22	P-Spec 2.9.2;17 TDLRSP	Rotate Variables -- Insufficient detail in description of the proposed method. The phase "shift the data" is ambiguous.
23	P-Spec 2.9.2;17 TDLRSP	The syntax "TDLR_VELOCITY.*" and "K_MATRIX.*" is inconsistent with the definition of ".*" as specified in the "Design Description Document -- Pluto" page 7.
24	P-Spec 2.9.2;17 TDLRSP	The statements "set ... to previous value of ..." are ambiguous.
25	P-Spec 2.9.2;17 TDLRSP	Typo. "TDLR_VELOCITYV.*"
26	P-Spec 2.9.2;17 TDLRSP	Questionable assignment: "FRAME_BEAM_UNLOCKED.# = 0." Technically, table 5.11 (case 2) on page 69 of the programming specification clearly indicates that this assignment should not be made. However, I don't really see a problem with this action
27	P-Spec 2.9.2;17 TDLRSP	Insufficient detail. There is a thorough description of processing the beam velocities. However, the description is merely a prose version of the programming specifications table 5.12. Reference is made to "calculating average velocities," but a description of how to calculate average velocities is noticeably absent. An algorithm implementing the solution is in order (or merely a reference to table 5.12 may suffice).
28	P-Spec 2.10.2;12 TDSP	COSMETIC. Valid values for the status of the touchdown are healthy (0) and failed (1). The P- Spec references the failed status as "unhealthy." This inconsistency with the programming specification is potentially confusing.
29	P-Spec 2.10.2;12 TDSP	SYNTAX. The local integer constant "all_ones" has a value of -1. An assumption is made that integers will be represented in two's complement -- thus in a 16-bit value of -1 all 16 bits are set (ie. '1'). I question the validity of this assumption. Note, P-Spec 2.2.2;22 ARSP declares a similiar constant using a preferred syntax.
30	P-Spec 2.3.2;21 ASP	QUESTION? When declaring the 'local type defs' should these variable have type real*8? what precision is required?
31	P-Spec 2.3.2;21 ASP	INSUFFICIENT DETAIL. The description for "rotating the variables" is ambiguous. The phase "shift data" is ambiguous.

Defects/Clarity Problems/Concerns

	Location	Description
32	P-Spec 2.3.2;21 ASP	INSUFFIECENT DETAIL. When "correcting for misalignment of the accels" it is not clear if a matrix multiplication is specified.
33	P-Spec 2.3.2;21 ASP	AMBIGUITY. When computing the standard deviation the syntax of the mathimatical operation is not clear.
34	P-Spec 2.6;1 GSP dfd	DEVIATION FROM SPEC. The data element G_STATUS apears as input to GSP.
35	P-Spec 2.6.2;9 GSP	INSUFFICIENT DETAIL. The description for "rotating the variables" is ambiguous. The phase "shift data" is ambiguous.
36	P-Spec 2.6.2;9 GSP	QUESTION. The local data element "at" is used to buffer the value found in the element "atmospheric_temp." Note, "at" is of type real*4 while "atmospheric_temp" is of type real*8. Is the lost of numeric precision acceptable? How about the precesion of the other local data elements?
37	P-Spec 2.6.2;9 GSP	QUESTION. The use of the operator "IAND." Is this acceptable and what is the operation? This is not provided in FORTRAN-88.
38	P-Spec 2.6.2;9 GSP	????? When converting to twos-comp, the then case -- The proposed solution is not a twos-comp function.
39	P-Spec 2.6.2;9 GSP	????? When converting to twos-comp, the "else" case is not necessary.
40	P-Spec 2.7.2;29 GP	INSUFFICIENT DETAIL. The description for "rotating the variables" is ambiguous. The phase "shift data" is ambiguous.
41	P-Spec 2.7.2;29 GP	AMBIGUITY. The ".*" syntax is not used as defined in the document description documentation.
42	P-Spec 2.7.2;29 GP	AMBIGUITY. There are several reference to a one diminsional data element GP_VELOCITY, GP_ALTITUDE, and GP_ATTITUDE.
43	P-Spec 2.7.2;29 GP	AMBIGUITY. When computing the current values of the vehicle altitude, velocity, and altitude, the assignment statements are inconsist with the assignment operator "=".
44	P-Spec 2.7.2;29 GP	AMBIGUITY. The data element "tnow" is used but not defined.
45	P-Spec 2.7.2;29 GP	DEVIATION FROM SPEC. The lower limit of GP_ALTITUDE is incorrectly evaluated with the value -1.
46	P-Spec 2.7.2;29 GP	DEVIATION FROM SPEC. The lower limit of FRAME_ENGINE_IGNITED is incorrectly evaluated with the value -1.
47	P-Spec 2.7.2;29 GP	??????? The statemant "else if (FRAME_ENGINES_IGNITED > 2**31-1)" is not valid (or necessary). The data element FRAME_ENGINES_IGNITED is specd as Integer*4. The maximum value for this data type is 2**(31-1).

Defects/Clarity Problems/Concerns

	Location	Description
48	P-Spec 2.7.2;29 GP	DEVIATION FROM SPEC. The data element AE_TEMP is not examined for exceeding the upper limit.
49	P-Spec 2.7.2;29 GP	I have some difficulty following the determination of the current phase. Some portions are clearly incorrect.
50	P-Spec 2.7.2;29 GP	INSUFFICIENT DETAIL. Need some algorithms for interpolation and extraplolation for computing the velocity error.
51	P-Spec 2.7.2;29 GP	AMBIGUITY. The data element "second" is referenced, but not defined.
52	P-Spec 2.7.2;29 GP	DEVIATION FROM SPEC. The appearant computation of the velocity error is incorrect.
53	P-Spec 2.1.2;31 AECLP	In the section "determining the axial engines' temperature -- is this the algorithm or a comment? I do not see the actual data assignment.
54	P-Spec 2.1.2;31 AECLP	AMBIGUITY -- There are references to a one diminsional array data element GP_VELOCITY. "the" GP_VELOCITY is a two diminsional array data element.
55	P-Spec 2.1.2;31 AECLP	DEVIATION FROM SPEC. When computing the PE_INTEGRAL, there is a noticable absence of the abs functions for the GP_VELOCITY(1) term when computing the local data element theta.
56	P-Spec 2.1.2;31 AECLP	DEVIATION FROM SPEC. When computing the YE_INTEGRAL, there is a noticable absence of the abs functions for the GP_VELOCITY(1) term when computing the local data element theta.
57	P-Spec 2.1.2;31 AECLP	DEVIATION FROM SPEC. The upper bounds check of the data element CONTOUR_CROSSED is flawed.
58	P-Spec 2.1.2;31 AECLP	DEVIATION FROM SPEC. The data element TE_LIMIT is not updated with the proper value.
59	P-Spec 2.1.2;31 AECLP	DEVIATION FROM SPEC. Page 7, "if (AE_SWITCH == off)" condition, the processing is not defined in the spec. Is it appropriate?
60	P-Spec 2.1.2;31 AECLP	
61	DFD 2.8;4 RECLP	The data element RE_STATUS is displayed as an input to the process RECLP.
62	P-Spec 2.8.2;13 RECLP	INSUFFICIENT DETAIL. When determining the roll engine command from the graph.

Defects/Clarity Problems/Concerns

	Location	Description
63	DFD 2.4;16 CP	DEVIATION FROM SPEC. There are a number of data elements displayed as input to CP which as not specified in the spec. AE_SWITCH, RE_SWITCH, TDLRSP_SWITCH, TDSP_SWITCH, TE_LIMIT, THETA, FRAME_BEAM_UNLOCKED, FRAME_ENGINES_IGNITED, INTERNAL_CMD, CL.
64	DFD 2.4;16 CP	DEVIATION FROM SPEC. The control signal SUBFRAME_COUNTER appears as input to the process CP.
65	P-Spec 2.4.2;25 CP	DEVIATION FROM SPEC. "subframe 1, ith_frame_2 and not ith_frame_5" (class f) processing -- the comments do not refer to the data elements K_ALT and K_MATRIX bits as set in the sample mask. Then, the K_ALT bit is set, however the K_MATRIX bit is not set.
66	P-Spec 2.4.2;25 CP	DEVIATION FROM SPEC. Class F processing -- the data element K_MATRIX is not loaded into the packet correctly.
67	P-Spec 2.4.2;25 CP	DEVIATION FROM SPEC. subframe 1, not ith_frame_2 and ith_frame_5 (class G) processing -- when accually loading the data elements into the data buffer, the following data elements are not loaded: A_ACCELERATION, A_STATUS, C_STATUS, G_ROTATION, and G_STATUS.
68	P-Spec 2.4.2;25 CP	DEVIATION FROM SPEC. "subframe 1, ith_frame_2, ith_frame_5" (class A) processing -- the data elements K_ALT and K_MATRIX bits are not set in the sample mask.
69	P-Spec 2.4.2;25 CP	DEVIATION FROM SPEC. Class A processing -- when actually loading the data elements into the data buffer, the following data elements are not loaded: A_ACCELERATION, A_STATUS, C_STATUS, G_ROTATION, G_STATUS, K_ALT, and K_MATRIX.
70	P-Spec 2.4.2;25 CP	DEVIATION FROM SPEC. subframe 2, (Class B) -- the data element GP_ROTATION is not loaded into the packet correctly.
71	P-Spec 2.4.2;25 CP	AMBIGUITY. The calling syntax and argument usage of the process CRC-16 is not clear
72	P-Spec 2.4.2;25 CP	The data elements BYTE_PACKET, NBYTES, and CHECKSUM are reference but never defined.
73	P-Spec 2.4.5;8 CRC-16	?????? It is not clear how the algorithm for computing the CRC operates. Some narration and/or reference is required.

C.2 Pluto Design Review

Attendees: Kelly Hayhurst (SQA representative/Moderator)
Patrick Quach (Verification Analyst/Recorder, Inspector)
Rob Angellatta (Programmer/Reader, Inspector)
Bernice Becher (System Analyst/Inspector)

C.2.1 Review Notes from Design Review

Pluto Design Review

July 13, 1994

Session 1: 9:30 a.m. - 11:30 p.m.

High-Level Structured Analysis Diagrams

<u>Context diagram:</u>

Telemetry packet flow not illustrated. Need modify to include

DFD GCS Level 0 specification
B - 358 -- Lower level diagrams should reflect changes for telemetry packet

DFD GCS Level 1 specification
B - 354 -- Unlabeled data flows to and from GCS_SIM_RENDEZVOUS - comment to be added in introduction

DFD GCS Level 2 specification
B - 355 -- Bubbles .1 & .3 should reference their counter part in DFD 1.

DFD GCS Level 3 specification
B - 356.3 - INTERNAL_CMD not shown as input into AECLP. Need to add to the Dataflow.

<u>GCS_SIM_RENDEZVOUS</u>

B - 342 - Extra unnecessary comment using personal pronoun. - To be deleted

<u>Altimeter Radar Sensor Processing (ARSP)</u>

B - 318, P-1 -- FRAME_COUNTER is not an input to ARSP -- should be removed

B - 319 -- Syntax problem -- The use of E in the constant for the transmission speed (if FORTRAN notation is going to be used -- should use D instead of E for accuracy)

B - 316.2, P-2 -- Problem with limit checks for AR_ALTITUDE - Limit checking missing for AR_ALTITUDE before using for extrapolation

Accelerometer Sensor Processing (ASP)

B - 307, B - 316, P - 3 -- Limits checking for A_ACCELERATION -- -- need to check for negative square root
Question: Does the range checking have to be performed on A_ACCELERATION before it is used to calculate the mean and standard deviation for each axis. It is a real*8 from SENSOR_OUTPUT data store.

Gyroscope Sensor Processing (GSP)

B - 308 -- problem with whether G_COUNTER(I) has a negative sign -- current syntax may not be appropriate.

Temperature Sensor Processing (TSP)

P-7 -- Lower parabolic function (pg. 3):
There appears to be a typo in the substitution of "h" into the parabolic equation. Either there is an extra set of parentheses or the sign after the M3 should be a "+"

B - 313 -- Incorrect term in the comments in upper parabolic function derivation. The first equation should be y = (1/4*p) * (x - h)^2 + k

Touch Down Landing Radar Processing (TDLRSP)

B - 320 -- The total number of radar beams is not explicitly expressed in the P-Spec. Only implicit in the table. The same indication should be used for maximum number of axis in other P-Spec.

B - 322, P-4 -- Concerning the set of IF statements for determining radar beam states (Table 5.11) The design meets all the requirements but has extra branches that are not specified in the Requirements.

B - 323 -- case 15 while computing “b” there is an incorrect operator; in equation for “pbvY”, there is an incorrect operator

B - 321 -- elapsed time calculation should not be within comments

P-6 -- problem with range for TDLR_ANGLES in Data Dictionary

B - 309, P-5 -- should off-diagonal elements of K_MATRIX be set?

Touch Down Sensor Processing (TDSP)

No Problems
-------------------------- END OF SESSION 1 --

Session 2: July 13, 1994 1:00 p.m. - 3:00 p.m.

Guidance Processing (GP)

B - 328 -- TE_INTEGRAL not an input for GP

B - 330 -- Comment and Pseudo-code not clearly delineated

B - 331 -- The algorithm does not specify which history variable to use when calculating the altitude (need more detail) -- current pseudocode not directly translatable to source

B - 303 -- The derivation for GP_VELOCITY uses GP_ROTATION, but no explanation is given on how its derived from G_ROTATION

B - 332, P-9 -- wrong history variable is used in setting up GP_ROTATION (pg. 5): Question: Should the most recent values for G_ROTATION be used to build GP_ROTATION?

B - 333 -- Negative square-root check not performed in the "if" statement on page 7

B - 335 -- Divide by zero check -- there is added information in the exception handling messages.

B - 336, P-9 -- The Else branch for "CONTOUR_ALTITUDE(i) < cur_altitude" (pg. 8): The index is missing from the first part of the IF condition. It should be "CONTOUR_ALTITUDE(i)".

B - 338 -- The END_GCS signal should not appear in the P-Spec if its not implemented. Use GP_PHASE instead.

B - 316.4 - missing range checking for variables used in the RK method.

Axial Engine Control Law Processing (AECLP)

B - 301 -- problem with order of execution of operators

P-16 -- problem with <=

B - 304, P-15 -- value of e is not correct

P-12 -- an extra check is made for divide-by-zero

B- - 302, P-13 -- problem with computation of yaw_error_limit -- it contains an incorrect term

P-14 -- problem with process step enumeration.

Roll Engine Control Law Processing (RECLP)

B - 311, P-11 -- there are 3 cases where RE_CMD is not set correctly

B - 312 -- in the "else" statement for deriving roll engine command, the sign of THETA2 is incorrect

Chute Release Control Processing (CRSP)

B - 339 -- problem with “released” (released not used in this process
B - 340, P - 17 -- problem with limit checks -- format statements not needed

Communications Processing (CP)

B - 400 -- presentation of crc table -- need more detail
B - 401 -- subscript incorrect in K_MATRIX
B - 402.1 -- syntax problems -- use of "^" for pointers
B - 402.2 -- need to note number of bits in CRC
B - 402 3 -- In the looping through bytes, the byte order is not specified.
B - 403 - The XOR operation is does not specify specifically that the lower byte of the CRC is to be used

Data Dictionary

B - 345 - Order within data stores needs to be explicitly stated.
B - 349-352, P-21-29 -- several elements have problems: AE_TEMP, CL, CONTOUR_CROSSED, DROP_HEIGHT, G1, G2, GVEI, K_MATRIX, TDLR_ANGLES, TE_DROP, GP_GS_IN

General

B - 325 - use of "RETURN" at the end of some P-Specs should be consistent (Is use of RETURN appropriate in a P-Spec?).

--------------------------- END OF SESSION 2 ---

C.2.2 Review Logs from Design Review

Review Log from System Analyst

Individual Inspection Preparation Log #1 (Page 1)

Name:___________Bernice Becher____ Date Log Submitted:___July 12, 1994

Implementation:___Pluto ___________ Date of Inspection:___July 7, 1994

Role:___________ Inspector_________

Defects/Clarity Problems/Concerns

INDEX

Individual Inspection Preparation Log #1 (Page 2)

Name:___________Bernice Becher____ Date Log Submitted:___July 12, 1994
Implementation:___Pluto ___________ Date of Inspection:___July 7, 1994
Role:___________ Inspector_________

Defects/Clarity Problems/Concerns

General Problems

NOTATION 306

"^" is used in ASP, page 3, first equation, and in other places without being explained.
"==" is used without being explained.
*Requirement: nonamb

PSEUDOCODE SYNTAX 359

The pseudocode syntax used in the P-Specs has not been described.

GLOBAL DATA STORES 317

The design does not give instructions for a FORTRAN program for declaring the four global data stores as labeled common blocks and how to name them.
Even though this is a coding detail, it is given in the specification.
*Requirement: comp, nonamb

GCS_SIM_RENDEZVOUS 327

The design does not state that in the code, this process must actually be called by the name GCS_SIM_RENDEZVOUS. Even though this is a coding detail, it is stated in the specification.
*Requirement: comp, nonamb

"return" in P-Specs 325

Several of the P-Specs contain a "return" before the "END P_SPEC". Since "return" is really a coding entity, it does not seem appropriate in a P-Spec. The processes which contain this are:
ASP
ARSP
GSP
RECLP
TDLRSP
TDSP
TSP
GP

*Requirement: trace

ARRAY NOTATION(nnp) 329

In most (or all) cases (see eg, GP, TDLRSP) where a rotation is to be done, the array notation does not use variable indices. This cannot be considered an error; however if this notation were to be carried over into the implementation into code, it would be quite error-prone, difficult to check for errors, difficult to maintain in the case of changes to the requirements, and involves many more lines of code than would otherwise be necessary. This notation is also used in other places besides rotation, as for example in AECLP where INTERNAL_CMD is converted to AE_CMD.
*Requirement: modif

Individual Inspection Preparation Log #1 (Page 3)

Name:___________Bernice Becher____ Date Log Submitted:___July 12, 1994
Implementation:___Pluto ___________ Date of Inspection:___July 7, 1994
Role:___________ Inspector_________

Defects/Clarity Problems/Concerns

Limit Checks

GENERAL LIMIT CHECKING: 315

In each case in which the design says to produce limit checking output, it uses the term "display-error" which does not seem to be defined anywhere. Specifically, the information missing in the design is the output logical unit number and the formats for FORTRAN code. Even though these are coding details, they are stated in the specification. It is not necessary to give this information for each incidence of a limit check, but it could be done at least once.

*Requirement: trans, comp
(Reference: Software Development Standards, Software Design Standards, "The low level requirements should be directly translatable into source code, with no further decomposition required.")

SPECIFIC LIMIT CHECKING PROBLEMS 316

2. Limit check for AR_ALTITUDE is not being done in the first subframe before it is used to fit a polynomial.

3. In CRCP, the variables CHUTE_RELEASED and AE_TEMP are being subjected to limit checks, but neither of these is of type real*8.

4. The following variables are not being checked for limits in the second subframe before being used in the Runge-Kutte calculations:

A_ACCELERATION
AR_ALTITUDE
GP_ALTITUDE
GP_ATTITUDE
GP_VELOCITY
G_ROTATION
TDLR_VELOCITY

*Requirement: spec

Individual Inspection Preparation Log #1 (Page 4)

Name:___________Bernice Becher____ Date Log Submitted:___July 12, 1994

Implementation:___Pluto ___________ Date of Inspection:___July 7, 1994

Role:___________ Inspector_________

Defects/Clarity Problems/Concerns

INTRODUCTION

Derivation Notes

Abbreviations 343

Any abbreviations used in any of the notes on derivations should be precisely defined. Some examples are GRAV, VEL_ERROR, ATT.

AECLP, Derivation of Solution for Differential Equation 305

Problem 1: On the second page, there is an equation which is not correct, namely TE_LIMIT = Q/OMEGA + C

Problem 2: On the second page, it is not clear at all how one goes from the equation

TE_LIMIT = Q/OMEGA + C x Q x e-Wt

to the final equation for TE_LIMIT. (specifically, how does one solve for C?)

*Requirement: acc, nonamb, comp

GP notes, Derivation for interpolation/extrapolation 344

1. The second paragraph which begins "Given the point..." has omitted some important defining information, namely:
   x0 < x < x1 ("which is less than the desired point" is not specific")
   x0 and x1 are contiguous points in the table
   xi represents altitude; f(xi) represents desired velocity at xi

2. (not an error) The discussion has not included the case where x = xi (even though it is handled in the p-spec).

3. (not an error) The text does not note that all three equations are exactly the same (which could make the GP p-spec simpler and more straightforward).
   *Requirement: nonamb, modif, comp

GP notes, Derivation for attitude, velocity, and altitude:

1. The equation for the derivative of GP_VELOCITY has the order of GP_VELOCITY and GP_ROTATION reversed in the matrix multiplication.
2. ACCEL is not a 1 x 3 matrix.
3. Coding syntax, such as the do loop on page 2, is not appropriate for a derivation.
4. The equations for the derivatives of GP_ATTITUDE, GP_VELOCITY, and GP_ALTITUDE, are given on page 2 and then are repeated on page 5-6 (with the GP_ATTITUDE equation incorrect on page 5).
   *Requirement: spec, acc, modif

Individual Inspection Preparation Log #1 (Page 5)

Name:___________Bernice Becher____ Date Log Submitted:___July 12, 1994
Implementation:___Pluto ___________ Date of Inspection:___July 7, 1994
Role:___________ Inspector_________

Defects/Clarity Problems/Concerns

STRUCTURED ANALYSIS DIAGRAMS

GCS Context Diagram 353

PACKET does not appear on any flow going out from the bubble GCS to the telemetry external sink.

GCS DFD/CFD 358

PACKET does not appear on flows coming from each of the three subframe bubbles and going off-page.

Sensor Processing Subframe DFD/CFD 354

1. It has not been made clear that the data flows going out from GCS_SIM_RENDEZVOUS to GUIDANCE_STATE, RUN_PARAMETERS, and SENSOR_OUTPUT are actually valid only for the first frame.

2. The data flow coming out of GCS_SIM_RENDEZVOUS and going to EXTERNAL indicates that all variables in that store are on the flow, but this is not correct.

3. PACKET does not appear on a flow out from GCS_SIM_RENDEZVOUS to off-page.

Guidance Processing Subframe DFD/CFD 355

1. PACKET does not appear on a flow out from GCS_SIM_RENDEZVOUS to off-page.

optional

2. The bubble 2.1 does not contain "(1.1)" inside, and the bubble 2.3 does not contain "(1.8)" inside. See Hatley, page 143 regarding notation for multiple use of one process.

3. The data flow coming out of GCS_SIM_RENDEZVOUS and going to EXTERNAL indicates that all variables in that store are on the flow, but this is not correct.

4. There should be no flows from GCS_SIM_RENDEZVOUS to GUIDANCE_STATE, SENSOR_OUTPUT, or RUN_PARAMETERS.

Control Law Processing Subframe DFD/CFD 356

1. PACKET does not appear on a flow out from GCS_SIM_RENDEZVOUS to off-page.

2. The bubble 3.1 does not contain "(1.1)" inside, and the bubble 3.5 does not contain "(1.8) inside". See Hatley, page 143 regarding notation for multiple use of one process.

3. (nnp)The data flow coming from GUIDANCE_STATE to AECLP does not include INTERNAL_CMD, but it is an input to AECLP. (Note: this is a result of Formal Modification 2.3-3.2).

4. The data flow coming out of GCS_SIM_RENDEZVOUS and going to EXTERNAL indicates that all variables in that store are on the flow, but this is not correct.

5. There should be no flows from GCS_SIM_RENDEZVOUS to GUIDANCE_STATE, SENSOR_OUTPUT, or RUN_PARAMETERS.

Individual Inspection Preparation Log #1 (Page 6)

Name:___________Bernice Becher____ Date Log Submitted:___July 12, 1994
Implementation:___Pluto ___________ Date of Inspection:___July 7, 1994
Role:___________ Inspector_________

Defects/Clarity Problems/Concerns

ARSP, P-Spec 1.2

ARSP, INPUT/OUTPUT Section 318
FRAME_COUNTER is not an input to this process. This is probably due to an error in the specification, which will be modified.
*Requirement: acc, trace

ARSP, first line of page 3: 319
It is not necessary to use FORTRAN floating point notation for a constant in the design, but if it is used, the "D" format rather than the "E" format should be used for accuracy.
*Requirement: acc

ASP, P-Spec 1.3

ASP P-Spec 2.3, page 4 307
Problem: Even though a check is being made for all accelerations being equal, it is still required that a check be made for a negative argument for the square root, as all equals may not be the only case with a roundoff problem.
*Requirement: spec, Reference: introduction, exception handling

GSP, P-Spec 1.4

GSP, P-Spec 1.4, top of page 3 308
"if ((G_COUNTER(I) & 0x8000 == 1)"
Problem: The intent here is to determine whether G_COUNTER(I) has a negative sign. The partial statement above will not work because if the sign is negative, the resulting "anded" value is not equal to 1.
*Requirement: spec, acc

Individual Inspection Preparation Log #1 (Page 7)

Name:___________Bernice Becher____ Date Log Submitted:___July 12, 1994
Implementation:___Pluto ___________ Date of Inspection:___July 7, 1994
Role:___________ Inspector_________

Defects/Clarity Problems/Concerns

TDLRSP, P-Spec 1.5

TDLRSP, P-Spec 1.5, page 4, top of page and bottom of page also: 320
"do (for each radar beam i)"
Problem: The design has not explicitly stated the number of radar beams.
*Requirement: comp, nonamb

TDLRSP, P-Spec 1.5, page 4, top of page: 321
Problem: The equation for elapsed_time is not given in the pseudocode itself, but only in a comment.
*Requirement: comp, nonamb

TDLRSP, P-Spec 1.5, page 4, middle of page: 322

LINE NUMBER

```
1  "if (elapsed_time >= TDLR_LOCK_TIME
2     tdlr_state(i)= 0  /*set unlocked */
3     FRAME_BEAM_UNLOCKED(i) = FRAME_COUNTER

4   else   /* the sensor has not recovered */
5     TDLR_STATE(i) = 0  /*set unlocked */
6     endif
7  endif
8  else   /* the sensor measurement != 0  */

9     if (TDLR_STATE(i) == 1  /* beam was locked */
10      TDLR_STATE(i) = 1    /* set locked */
11    else   /* beam was unlocked  */

12      if (elapsed_time >= TDLR_LOCK_TIME)
13        TDLR_STATE(i) = 1      /* set locked */
14      else  /* the sensor has not recovered  */
15        TDLR_STATE(i) = 0   /*  set unlocked */
```

Problem 1: Line #2 is not traceable to any requirement
Problem 2: Lines 4 and 5 are not traceable to any requirement
Problem 3: Line 10 is not traceable to any requirement
Problem 4: Lines 14 and 15 are not traceable to any requirement

*Requirement: spec

TDLRSP, P-Spec 1.5, top of page 5 309
The setting of the off-diagonal elements of K_MATRIX to zero is not a requirement in the specification. (the spec may require a formal mod to make this unambiguous).
*Requirement: trace

Individual Inspection Preparation Log #1 (Page 8)

Name:___________Bernice Becher____ Date Log Submitted:___July 12, 1994
Implementation:___Pluto ___________ Date of Inspection:___July 7, 1994
Role:___________ Inspector_________

Defects/Clarity Problems/Concerns

TDLRSP, P-Spec 1.5, Equation for pbvY: 323

Problem 1: Page 5, comments at bottom of page:
In the equation for "b", the operator in front of the term b(4) is incorrect.

Problem 2: Page 7, in case # 15:
In the equation for "pbvY", the operator in front of the term b(4) is incorrect.

*Requirement: acc, spec

TDLRSP, P-Spec 1.5 324
"where cos represents the cosine function"
Problem: This statement has not been marked as a comment.
*Requirement: nonamb

TDSP, P-Spec 1.6

TSP, P-Spec 1.7

TSP, P-Spec 1.7, middle of page 2 326
"Implementation note, if M1=M2 a divide by zero exception must be handled.

Problem 1: The divide-by-zero exception handling should happen prior to the divide.
*Requirement: spec

Problem 2: (nnp)While this cannot really be considered an error, the syntax and content for this exception is not presented in a consistent manner with the rest of the exception handling in this design. This looks like a comment but is actually pseudocode.
*Requirement: con

TSP P-Spec 2.11, middle of page 3 314
In the calculation for lower-parabolic-function, there is a division by (M4 - M3), but there is no provision for a check for divide-by zero in case M4 = M3.
*Requirement: spec, INTRODUCTION, Exception Handling

TSP P-Spec 2.11, 10th line from bottom of page 3 313
"y = 4*p..."
Problem: "4*p" is not correct
*Requirement: acc, nonamb

Individual Inspection Preparation Log #1 (Page 9)

Name:___________Bernice Becher____ Date Log Submitted:___July 12, 1994
Implementation:___Pluto ___________ Date of Inspection:___July 7, 1994
Role:___________ Inspector_________

Defects/Clarity Problems/Concerns

GP, P-Spec 2.2

GP, P-Spec 2.2, INPUT/OUTPUT section: 328
TE_INTEGRAL is not an input to this process.
*Requirement: acc, spec

GP, P-Spec 2.2, middle of page 4 through middle of page 5: 330
Application of Runge-Kutte:

In the text which describes the method for calculating attitude, velocity, and altitude, beginning with "A five step implentation of the RK method..." and ending with step 5, the comments are not clearly delineated from the pseudocode.
*Requirement: nonamb, comp, spec

GP, P-Spec 2.2, middle of page 4 through middle of page 5: 331
Application of Runge-Kutte:

Problem: In general, the pseudocode given is not directly translatable into souce code. More specifically: In each of the four parts labeled "A)":

Problem 1: It is not stated for any of the three derivatives which history values are to be used for the "sensor" variables.

Problem 2: In the case of the derivative of the velocity, it is not stated which values are to be used for the attitude.

Problem 3: In the case of the derivative of the altitude, it is not stated which values are to be used for the attitude or for the velocity.

Problem 4: The equations for the derivatives have not been included in the pseudocode.

GP, P-Spec 2.2 303
Problem: GP_ROTATION may not be used as an input to GP, yet it appears in the design's equations for the derivatives of GP_ATTITUDE and GP_VELOCITY. (see specification, page 130, under Notation)
*Requirement: req, comp

GP, P-Spec 2.2, bottom of page 5: 332
In the setting of the GP_ROTATION matrix, the wrong history subscript is being used for the G_ROTATION elements.
*Requirement: spec, acc

GP, P-Spec 2.2, second line of page 7, and fourth line of page 10:: 333
In each case there is no check for a negative argument before the square root is taken.
*Requirement: spec

Individual Inspection Preparation Log #1 (Page 10)

Name:___________Bernice Becher____ Date Log Submitted:___July 12, 1994
Implementation:___Pluto ___________ Date of Inspection:___July 7, 1994
Role:___________ Inspector_________

Defects/Clarity Problems/Concerns

GP, P-Spec 2.2, page 7 through 8: 334
(nnp)In several places " i = 101 " is used as a "coding" method for exiting the loop. While this is
not an error, it would be adequate (and preferable) in the design to state "exit the loop".

GP, P-Spec 2.2, page 7 and page 8: 335
In three separate places there is a check for divide-by-zero. In each case if there is an exception, the text "COMPUTATION OF OPTIMAL VELOCITY" is produced, which is not a requirement in the specification.
*Requirement: trace

GP, P-Spec 2.2, middle of page 8: 336
"if ((CONTOUR_ALTITUDE == 0) .or. (index == 100)) then

Problem 1: CONTOUR_ALTITUDE is a vector but has no subscript.
Problem 2: "index" is undefined.
*Requirement: nonamb, comp,acc, spec

GP, P-Spec 2.2, bottom of page 10: 338
(nnp)The designer stated at the overview that "END_GCS would not be implemented". If that is the case, it should not be set inside a process.
*Requirement: trans, trace

AECLP, P-Spec 2.1

AECLP P-Spec 2.1, top of page 4: 301
"if (FRAME_COUNTER - FRAME_ENGINES_IGNITED * DELTA_T <... {4}"
.
.
.
"if (FRAME_COUNTER - FRAME_ENGINES_IGNITED * DELTA_T >=... {4}"

Problem: In each of these statements, assuming the FORTRAN precedence rules, the order of execution of the operators is not correct.
*Requirement: acc, req

AECLP P-Spec 2.1, middle of page 7: 302
"yaw_error_limit = -GQ(CL) * GP_ROTATION(1,2) + ..."

Problem: This partial statement contains an incorrect term.
*Requirement: acc, req

AECLP P-Spec 2.1, top of page 9: 304
"let e = 2.718........"
Problem: This value given for e is not correct (it is not really necessary to define e).
*Requirement: acc

Individual Inspection Preparation Log #1 (Page 11)

Name:___________Bernice Becher____ Date Log Submitted:___July 12, 1994
Implementation:___Pluto ___________ Date of Inspection:___July 7, 1994
Role:___________ Inspector_________

Defects/Clarity Problems/Concerns

RECLP, P-Spec 3.4

RECLP P-Spec 2.8, pages 3&4, deriving roll engine command: 311
Problem: There are three specific cases for which the value set for RE_CMD is not correct. These cases are as follows: (LET P = G_ROTATION(1,0)
1. THETA = 0 and P > P2 and P <= P4
2. THETA = 0 and P <= P2 and P > P1
3. THETA < 0 and THETA >= -THETA1 and P < -P2
4. THETA < 0 and THETA >= -THETA1 and P = -P2
*Requirement: spec, acc

RECLP P-Spec 2.8, page 4, middle of page, deriving roll engine command: 312
"else if (THETA >= THETA2) then {3}"
Problem: The sign of THETA2 is not correct.
*Requirement: spec, acc

CRCP, P-Spec 3.3

CRCP, P-Spec 3.3, page 1, top of page 339
"log*1 released = 1"
Problem: "released" is not used in this process.
Problem: (nnp)While not an error, the declaration of local constants is not consistent with the syntax in the rest of the design.
*Requirement: trace, con

CRCP, P-Spec 3.3, page 1, middle of page 340
The three format statements are not needed (see limits section).
*Requirement: trace

CRCP, P-Spec 3.3, page 1, bottom of page 341
"IF (CHUTE_RELEASED == not_released)"
(nnp)Not an error, but why not used "chute attached" for consistency?
*Requirement: con

Individual Inspection Preparation Log #1 (Page 12)

Name:___________Bernice Becher____ Date Log Submitted:___July 12, 1994

Implementation:___Pluto ___________ Date of Inspection:___July 7, 1994

Role:___________ Inspector_________

Defects/Clarity Problems/Concerns

CP

CP, P-Spec 2.4 400

Problem with presentation of crc table (for purposes of verification by inspectors) (designer may include algorithm for table)

CP, P-Spec 2.4, PAGE 7 401

"DATA_PACKET.data.sp.k_matrix(3) =..."

Problem: first subscript for K_MATIRX is incorrect

CP, P-Spec 2.4 402

Ambiguitie

1. Page 5, "var data_packet:.....
   ......= PACKET"

2. Page 9:
   1. Is table to be read column first or row first?
   2. How many bits are in the CRC?
   3. In loops for bytes, start with first or last byte? ie, definition of next_byte?

CP, P-Spec 2.4 403

Page 9

"index = crc XOR next_byte"

Problem: Design has not stated that only the low-order byte of crc is to be used.

GCS_SIM_RENDEZVOUS, P-Specs 1.1, 2.1, and 3.1

GCS_SIM_RENDEZVOUS, P-Specs 1.1, 2.1, and 3.1 342

In the body of the P-Spec is the statement ", it is not our responsibility." This is not an appropriate statement to be inside a P-Spec, since the function of a P-Spec is merely to state the transformation from the inputs to the outputs.

*Requirement: trace

Individual Inspection Preparation Log #1 (Page 13)

Name:___________Bernice Becher____ Date Log Submitted:___July 12, 1994
Implementation:___Pluto ___________ Date of Inspection:___July 7, 1994
Role:___________ Inspector_________

Defects/Clarity Problems/Concerns

DATA DICTIONARY

Four Global Data Stores and Ordering with "+" Notation 345
Hatley (page 101) states that the "+" notation "does not imply order. If ordering is required, it is specified by a comment in the dictionary or PSPEC"; therefore, for the global data stores, there should be some such explicit statement.

Data Conditions (not an error) 346
All variables in the data dictionary that are listed with attribute of "data condition" could be changed to "data" (with the exception of GP_PHASE and CHUTE_RELEASED).

Notation 347
Hexadecimal notation is used in, for example, COMM_SYNC_PATTERN, but the syntax for the notation is given inside pspecs, for example, TDSP. The notation should be given in some central place, such as, for example, the data dictionary or the introduction.

END_GCS 348
This is a primitive data element but has no description at all.

G1 349
The units are not correct.

GP_GS_IN 350
This group flow includes TE_INTEGRAL which is not an input to GP.

K_MATRIX 351
The Accuracy is incorrect.

TDLR_ANGLES 352
In the range, the value PI/2 should not be included.

Individual Inspection Preparation Log #1 (Page 14)

Name: Bernice Becher   Date Log Submitted: July 12, 1994
Implementation: Pluto   Date of Inspection: July 7, 1994
Role: Inspector

Defects/Clarity Problems/Concerns

Typographic Errors

GSP, page 1, comments at bottom of page:
"diminsion" should be "dimension"

GSP, page 3, comment at top of page:
"hexidecimal" should be "hexadecimal"

ASP, page 3: "hexidecimal" should be "hexadecimal"

TDSP, page 1: In comment near bottom of page, "hexidecimal" should be "hexadecimal"

GP, bottom of page 8 to top of page 9
A line has been split in two.

GP, bottom of page 9
Should "=<" be "<=" ?

GP, page 8, in two different comments:
"Exapolation" and "Exapolate" should be "Extrapolation" and "Extrapolate"

ASP, page 4, comment at top of page:
"seperate" should be "separate"

Notes on high-level design, page 1:
Third paragraph: "have to input or output" should be "have no input or output"

Fourth paragraph, last sentence: "with of off_page" should be "with off-page"

Last paragraph: should "deficients" be "deficiencies"?

Data Dictionary

G2, in the units is "degree*"

TE_DROP, the last word of the description, namely "intersected" was cut off.

Individual Inspection Preparation Log #1 (Page 15)

Name:___________Bernice Becher____ Date Log Submitted:___July 12, 1994

Implementation:___Pluto ___________ Date of Inspection:___July 7, 1994

Role:___________ Inspector_________

Defects/Clarity Problems/Concerns

Suggestions for the Future

It would be helpful if the entire design document were numbered sequentially from beginning to end.

Constants used for limit checking make modification difficult and error-prone.

Can we add something to the standards to force the designer to be explicit about what is a comment and what is actual pseudocode/structured English?

Can we add something to the standards to force the designer to use very specific non-ambiguous pseudocode syntax?

Require that a Teamwork Balance Report (with no errors) be included as part of the design.

Individual Inspection Preparation Log #1 (Page 16)

Name:___________Bernice Becher____ Date Log Submitted:___July 12, 1994
Implementation:___Pluto ___________ Date of Inspection:___July 7, 1994
Role:___________ Inspector_________

Defects/Clarity Problems/Concerns

1
*Requirement: Accuracy (Reference: DO-178B 6.3.2b).
2
*Requirement: Nonambiguity (Reference: DO-178B 11.0a).
3
*Requirement: Follow a particular design method (References: Software Development Standards,"Software Design Standards", "Design Methods, Rules, and Tools", "...using the structured analysis ...by Hatley and Pirbhai or...", and "Design Documentation", "...document should follow...GCS specification or the Hatley book...")
4
*Requirement: Consistency (DO-178B 5.2.2a, 6.3.2b, and 11.0d)
5
*Requirement: Traceability (References: DO-178B 5.2.2a, 5.5b, 6.1b, 6.2a, 6.3.2a, and 11.0f)
6
*Requirement: Translatability to source code (Reference: Software Development Standards, Software Design Standards, "The low level requirements should be directly translatable into source code, with no further decomposition required.")
8
*Requirement: Completeness (Reference: DO-178B 11.0b)
10
*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)
11
*Requirement: Fullfillment of requirements in Software Requirements document (References: DO-178B 6.3.2a and 11.10a)
**Software Requirements 2.2 with Mods 1-26 Reference: Introduction, Exception Handling, Upper or Lower Limit Exceeded

*Requirement: acc
*Requirement: nonamb
*Requirement: des
*Requirement: con
*Requirement: trace
*Requirement: trans
*Requirement: comp
*Requirement: spec
*Requirement: modif

Review Log from Verification Analyst

Pluto Individual Inspection Log
Inspector: Patrick Quach
Date: July 11, 1994

The following is a list of deficiencies or possible deficiencies found in the Pluto design document. The comments are grouped under the heading of the P-Spec. or configuration item to which they pertain. No deficiencies were discovered in any DFD's or PAT's.

<u>ARSP (P-Spec 1.2)</u>

1. I/O Section
   FRAME_COUNTER is an unnecessary input to ARSP because it is not used in the P-Spec. It is, however, listed as an input in the Requirements Document. This may be a left over from the Spec. Mod. 2.3-3.3

2. Limits checking for AR_ALTITUDE
   Question: Does the range checking have to be performed on AR_ALTITUDE before using it in the Divided Difference Method. It is a real*8 from SENSOR_OUTPUT data store.
   CITATION: Spec. --- Exception Condition (pg. 16)

<u>ASP (P-Spec 1.3)</u>

3. Limits checking for A_ACCELERATION
   Question: Does the range checking have to be performed on A_ACCELERATION before it is used to calculate the mean and standard deviation for each axis. It is a real*8 from SENSOR_OUTPUT data store.
   CITATION: Spec. --- Exception Condition (pg. 16)

<u>TDLRSP (P-Spec 1.5)</u>

4. Concerning the set of IF statements for determining radar beam states (pg. 4)
   The design meets all the requirements but has extra branches that are not specified in the Requirements. However, these branches are innocuous and do not change any values. It may not be worth the risk to alter the design since it may introduce some logic errors.
   CITATION: Spec --- Use of Tables (pg. 15)

5. Concerning the table for setting K_MATRIX (pg. 5-7)
   The table uses the X, Y, Z indexes for the elements of K_MATRIX while the case statement uses the actual numerical indexes. It may be useful to clarify this in the text of the explanation.
   CITATION: 178B --- Non-Ambiguity (pg. 47 11.0A)

6. Divide by zero check(pg. 8)
   Question: In step 3D, should a divide by zero check (on the COS[TDLR_ANGLES]) be performed before TDLR_VELOCITY is computed.

<u>TSP (P-Spec 1.7)</u>

7. Lower parabolic function (pg. 3):
   There appears to be a typo in the substitution of "h" into the parabolic equation. Either there is an extra set of paren. or the sign after the M3 should be a "+"
   CITATION: 178B --- Non-Ambiguity (pg. 47 11.0A)

<u>GP (P-Spec 2.2)</u>

8. GP Algorithm notes (pg. 1)
   Typo on first word of second paragraph.

9. Setting up GP_ROTATION (pg. 5):
   Question: Should the most recent values for G_ROTATION be used to build GP_ROTATION.

10. The Else branch for "CONTOUR_ALTITUDE(i) < cur_altitude" (pg. 8):
    The index is missing from the first part of the IF condition. It should be "CONTOUR_ALTITUDE(i)".
    CITATION: 178B --- Non-Ambiguity (pg. 47 11.0A)

<u>RECLP (P-Spec 3.4)</u>

11. The If statement for determining roll engine intensity & direction (pg. 4)
    Typo in the case where:
    THETA >= -THETA & G_ROTATION < -P2,
    the value of RE_CMD should be:
    RE_CMD = 6 + 0.
    CITATION: 178B --- Non-Ambiguity (pg. 47 11.0A)

<u>AECLP (P-Spec 3.2)</u>

12. Divide-by-zero check (pg. 6)
    Question: Why is there an extra Divide-by-zero check in the yaw error limit calculation. The check was performed previously in the pitch error limit calculation?

13. Yaw_error_limit equation (pg. 7)
    Typo in the yaw_error_limit equation; the first gain should be "GR" instead of "GQ".
    CITATION: Compliance (pg. 27 6.3.2a)

14. Processing step enumeration (pg. 7-10)
    The enumeration of step "2C" on the middle of page 7 duplicates the previous numbering. This step should be "2D", Subsequent steps are also off by 1 letter.
    CITATION: 178B --- Non-Ambiguity (pg. 47 11.0A)

15. The value of "e" (pg. 9)
    Typo in the value of "e" e = 2.718281828459045235360.......
    CITATION: 178B --- Accuracy (pg. 27 6.3.2b)

16. Concerning setting of AE_CMD from INTERNAL_CMD (pg. 11)
    Typo in second branch of all 3 "If" statements; should read: "(INTERNAL_CMD(i) <= 1)"
    CITATION: 178B --- Non-Ambiguity (pg. 27 6.3.2b)

CRCP (P-Spec 3.3)

17. Limit checking (pg. 1)
    Limits checking is not necessary for CHUTE_RELEASED and AE_TEMP.
    CITATION: Spec. --- Exception Condition (pg. 16)

18. Local variable definition (pg. 1)
    Defining the local variable "hot" as an "int*2" is more accurate but does not agree with the Requirements.
    CITATION: 178B --- Non-Ambiguity (pg. 27 6.3.2b). See also Data Dictionary citation for AE_TEMP

19. Concerning the variable assignment (pg. 1)
    Typo in assignment for CHUTE_RELEASED.
    CITATION: 178B --- Non-Ambiguity (pg. 27 6.3.2b)

Data Dictionary

Open Issue: The Specification does not give the required accuracy for many data elements. Hence this field is also "TBD" in many instances in the Design Data Dictionary. Will this be determined before coding, before testing, or left to the programmer and tester's discretion?
CITATION: 178B --- Verifiability (pg. 27 6.3.1d)

AE_TEMP This element is specified as a "LOGICAL-1". In general, logical variables can have only 2 values, but this one has more. An enumerated type is more correct. The Requirements Data Dictionary also has this defined as a LOGICAL-1
CITATION: 178B --- Non-Ambiguity (pg. 27 6.3.1b & 6.3.2b)

CL Question: Should the range for this data element correspond with the TeamWork usage?
CITATION: 178B --- Non-Ambiguity (pg. 27 6.3.2b)

CONTOUR_CROSSED Typo in DESCRIPTION field: "velocity_altitude" should be "velocity-contour"
CITATION: 178B --- Non-Ambiguity (pg. 27 6.3.2b)

DROP_HEIGHT Typo in ACCURACY field: extra period

G1 Typo in UNITS field: should be "(meters/sec^2)/(degree_C)"
CITATION: Compliance (pg. 27 6.3.2a)

G2 Typo in UNITS field: should be "(meters/sec^2)/degree_C^2"
CITATION: Compliance (pg. 27 6.3.2a)

GVEI Typo in UNITS field: unit should be "/sec^2"
CITATION: Compliance (pg. 27 6.3.2a)

K_MATRIX Typo in ACCURACY field: does not agree with Specification. Spec. has "N/A".
CITATION: Compliance (pg. 27 6.3.2a)

TDLR_ANGLES Typo in DESCRIPTION: the "y" should be "gamma"
CITATION: 178B --- Non-Ambiguity (pg. 27 6.3.2b)
Typo in RANGE field: PI/2 should be excluded from the range according to the Spec.
CITATION: Compliance (pg. 27 6.3.2a)

TE_DROP Format error in DESCRIPTION field: missing last part of explanation

C.3 Pluto Code Review

Attendees: Kelly Hayhurst (SQA representative/Moderator)
Patrick Quach (Verification Analyst/Recorder, Inspector)
Philip Morris (Programmer/Reader, Inspector)
Bernice Becher (System Analyst/Inspector)

C.3.1 Review Notes from Code Review

Pluto Code Review

Session 1: 11/16/94 9:30 a.m. - 11:30 p.m.

Reviewed Comments on Design before examining Code

Design Issues

B-1 -- "RETURNS" should be removed from the design

B-18 -- need to correct statement about data and control flows

Need to include balance report as part of design documentation

B-2, B-3W -- P-Spec ASP: problem with computing standard deviation and comments about it
--> Related to Spec Mod 2.3-4.2

B-15 -- P-Spec CP: problem with type of SUBFRAME_T

B-18 -- P-Spec CP: problem with GP_ROTATION and GP_VELOCITY -- they need subscripts

P-Spec CP: typos, need] instead of) on page 7 of data packet stuff

B-17 -- P-Spec CP: need to define specify order for next_byte (not a code problem)

B-7 -- P-Spec GP: problem with equations of att_k2, vel_k2, alt_k2 and ... att_k3, vel_k3, alt_k3
--> Problem is also in Code -- see B-42 in code review log

B-8 -- P-Spec GP: extra range check after END_P_SPEC

B-11, B-4, P-12 -- P-Spec GP: need to comment a "where" statement

B-6 -- Data Dictionary: problem with attribute of "data condition" for several variables (not a code problem)

B-14 -- Data Dictionary: CHUTE_RELEASED -- should be in the EXTERNAL data store
--> Related to Spec Mod 2.3-4 **(should have been corrected in PR #20)**

-------------------------------- END OF SESSION 1 -------------------------------------

Session 2: 11/16/94 1:00 p.m. - 3:00 p.m.

Code Issues:

B-36 -- ADD TO DEVELOPMENT STANDARDS: require P-Spec numbers in part of module headers

B-35, B-36 -- floating point constants should all be double precision to avoid precision problems

B-56 -- DEVELOPMENT STANDARDS Bernice would like the list of arguments in the module header to include whether each argument is on input, output, or both

B-58 -- might want to consider deleting the requirement to note configuration date in the Development Standards

B-30 -- EXTERNAL.FOR: problem with clp_data_t -- data type is incorrect
--> SEE SPEC MOD 2.3-2 ?

NOTE: Want to require that inspection logs have all items uniquely ordered (to make notes easier to follow)

B-32 -- PLUTO.FOR -- check for termination should be done after subframe 3 -- not subframe 2
--> see Spec Mod 2.3-2.1

B-33 -- PLUTO.FOR: “go to 100” -- no unconditional gotos (this one is not justified)

B-34 -- AECLP.FOR -- need to check for divide-by-zero for OMEGA (both design and code need change)

NOTE: Require that listing file be turned in for review sessions

B-40 -- CRC16.FOR: generator polynomial is not correct -- chould also note that the bits are reversed

B-57 -- CRC16.FOR: in module header -- should say that it is returning checksum

B-41 -- GP: when calling mult_vel -- should be sending vel_k1 ... not att_k1

B-43 -- GP: the last argument for deriv_vel is incorrect

B-44 -- GP: unconditional go tos -- not justified

B-45, P-11 -- GP: problem with relational operator -- “.GE.” is not correct (design is correct)

Need range check for VELOCITY_ERROR in code (design is correct)

B-46, B-50 -- go to is okay -- but need safety net -- same as in TDLRSP

B-47 -- problem with EQUIVALENCE statement and the variables pv, qv, rv -- problem is in both DERIV_ATT and DERIV_VEL

P-8 -- MULT_ATT: problem in matrix multiplication

B-51 -- TDLRSP: missing go to statement

P-6 -- GSP: “counter” is mistyped -- it should be an integer

B-52 -- LOWER_PARABOLIC_FUNCTION.FOR -- problem in calculation of LOWER_PARABOLIC_FUNCTION -- term “M3 + half_slope” is incorrect (design is okay)

B-53 -- UPPER_PARABOLIC_FUNCTION: problem in calculation of upper parabolic function

B-55 -- UTILITY.FOR: problem with format statement 30

P-2 -- CONSTANTS.FOR -- AE_TEMP is mistyped --
--> See Spec Mod

C.3.2 Review Logs from Code Review

Review Log from System Analyst

Individual Inspection Preparation Log #1 (Page 1)

Name: Bernice Becher — Date Log Submitted: 11/15/94

Implementation: Pluto — Date of Inspection 11/16/94

Role: Inspector

Defects/Clarity Problems/Concerns

PLUTO DESIGN LOG III

GENERAL ISSUES

1 "Return" in P-Specs

Several of the P-Specs contain a "return" before the "END P_SPEC". Since "return" is purely a coding function, it is not appropriate in a P-Spec and in addition accomplishes no function with respect to how the inputs of the process are converted to its outputs. The processes which contain "return" are:

+CRCP (added since design review but not mentioned in action item 16)
+AECLP (added since design review but not mentioned in action item 16)
*ASP
*ARSP *TDLRSP
*GSP *TDSP
*RECLP *TSP
*GP (4 separate returns)

*=remains in from before design review
+=was added since design review
*Requirement: traceability

INTRODUCTION

13N It is difficult to refer to text because the pages of the introduction are not numbered.
*Requirement: modifiability

5W Section 1.3, Design Syntax Specifications, fourth paragraph.
It does not seem that the indirection symbol and Modula-2 record syntax were really needed, when the FORTRAN record structure syntax would have been adequate (and in fact was used in the code). It seems that the indirection added an unnecessary level of complication in the design, and was not used at all in the code.
*Requirement: traceability

18 Section 2.2, Data and Control Flows, fifth paragraph:
"...the consequence of this action will result in approximately 80 data flows requiring off-page connections..."

Individual Inspection Preparation Log #1 (Page 2)

Name:_________Bernice Becher____ Date Log Submitted:______11/15/94

Implementation:___Pluto ____________ Date of Inspection________11/16/94

Role:____________ Inspector__________

Defects/Clarity Problems/Concerns

It is not clear that this statement is entirely correct. It would seem that approximately 18 group-flows would be required, and they would not need to be off-page connectors. Perhaps a statement saying that the number of data flows would increase would be more accurate.
*Requirement: accuracy

STRUCTURED ANALYSIS CHARTS

9 **PAT 0**

- There is an empty input cell under the heading GP_PHASE. There should either be an entry in this cell or some explanation as to the meaning of an empty cell.
*Requirement: nonambiguity

- There is no explanation for what will happen in terms of activation when GP_PHASE has not yet been defined, which is the case before the first activation of GCS_SIM_RENDEZVOUS.
*Requirement: nonambiguity

- The statement " "GP_PHASE" is initialized to "1" during initialization" is not correct. It is possible that it might be initialized to any value from 1 to 5. The specification states that it will be initialized but does not state the initial value.
*Requirement: accuracy

19I Teamwork Balancing Report
Should a balancing report be included in the design document?
*Requirement: completeness

P-SPECS

ASP(1.3)

2 Page 4, comment at bottom of page:
"identical values" should be "identical or nearly identical values"
*Requirement: accuracy
*Requirement: completeness

3W Calculation of standard deviation (sd)
Design, Page 4, bottom, and page 5: :
The specification has been modified (mod 2.3-4.2) to give a formula for the standard deviation which cannot yield a negative square root. This design does not use the new formula and hence can produce a negative square root, for which a check is being made. The negative square root problem could be eliminated and there would be no need for a square root check if the formula in the Specification were used. Is the design OK as stands?
(affected code: lines 831 through 837
*Requirement: specification

Individual Inspection Preparation Log #1 (Page 3)

Name: Bernice Becher Date Log Submitted: 11/15/94
Implementation: Pluto Date of Inspection 11/16/94
Role: Inspector

Defects/Clarity Problems/Concerns

CP (1.8)

15 Page 5, bottom of page: (not a code problem)
"type subframe_t = (subframe_t, gp_data_t, clp_data_t)"
Problem: "subframe_t" ,on the right hand side of the assignment, is incorrect.
*Requirement: accuracy
*Requirement: specification

18 Page 8, top
In each of the assignment statements for GP_ROTATION and GP_VELOCITY, there is no subscript on the left hand side.
*Requirement: nonambiguity
*Requirement: specification

17 Page 9, bottom:
"do for each byte in the message next_byte"

"next_byte" is ambiguous in that it doesn't specify the order, i.e., first-to-last byte or vice versa.(not a code problem) (was #402)
*Requirement: nonambiguity

GP(2.2)

7 Page 6, top :
In each of the equations for att_k2, vel_k2, alt_k2, att_k3, vel_k3, and alt_k3, the right parenthesis preceding the term "/2" is not in the correct place., and thus the attitude, velocity, and altitude arguments for the derivative routines are not correct.
*Requirement: accuracy
*Requirement: specification

8 Page 14, middle
A range check for altitude follows "END P_SPEC" but has already been done where needed.
*Requirement: translatability

10I Page 8, top and page 11, middle:
Is check for negative square root really necessary, given that a valid GP_ALTITUDE(0) will be positive, and that GP_ALTITUDE(0) has been checked earlier and then not changed.

Individual Inspection Preparation Log #1 (Page 4)

Name: Bernice Becher   Date Log Submitted: 11/15/94
Implementation: Pluto   Date of Inspection 11/16/94
Role: Inspector

Defects/Clarity Problems/Concerns

11 Page 13, middle:

The following statements are comments and should be designated as such. In addition, there is some confusion because they appear in the middle of an equation.

"where
pv :=...

rv :=... "

*Requirement: nonambiguity

TDLRSP(1.5)

4 Page 7, bottom (previous #324):

"where cos represents the cosine function"
Problem: This statement has not been marked as a comment.
*Requirement: nonambiguity

DATA DICTIONARY

6 (previous number: 346) (not a code problem)

There are several elements in the data dictionary whose ATTRIBUTE is listed as "data condition". In fact, in the SA/SD charts, none of these is ever used as anything except a data flow. These elements are:
AE_SWITCH
AE_TEMP
CONTOUR_CROSSED
RE_SWITCH
TD_SENSED
TDLR_STATE
*Requirement: accuracy
*Requirement: consistency
*Requirement: nonambiguity

14 Element CHUTE_RELEASED (not a code problem)

The DATA STORE section says GUIDANCE_STATE, but according to Formal Modification 3.2.4-4, it should be EXTERNAL. Problem Report 20 states this has been changed, but it hasn't been.
This oversight causes one to wonder why there is not some type of error produced by Teamwork, because now all the DFDs show CHUTE_RELEASED coming from and going to EXTERNAL, while the data dictionary states that it is in GUIDANCE_STATE.
*Requirement: accuracy
*Requirement: consistency

Individual Inspection Preparation Log #1 (Page 5)

Name:___________Bernice Becher____ Date Log Submitted: 11/15/94
Implementation: Pluto Date of Inspection 11/16/94
Role: Inspector

Defects/Clarity Problems/Concerns

TYPOS

INTRODUCTION, Section 1.3, fourth paragraph:
"previous chosen to signify" should be "previous**ly** chosen to signify"
ARSP (1.2), page 2, bottom
"recieved" should be "received"

DATA DICTIONARY
COMM_SYNC_PATTERN
In the RANGE, "hexidecimal" should be "hexadecimal"

TDSP, page 1
"hexidecimal" should be "hexadecimal"

Introduction, Section 2.3, Module Description, discussion about GP_VELOCITY:
"oincide" should be "coincide"

Introduction, Section 2.3, Module Description, AECLP equations:
The entire discussion uses a Greek capital symbol for X double dot (acceleration), while the very last equation for Q reverts to the small x for acceleration.

ARSP, page 2, bottom:
"recieved" should be "received"

CP, page 2:
In comments, "consist of" should be "consists of"

CP, page 3:
Comment which gives total no of bytes for sensor processing, "127" should be "129"
CP, page 4:
In comments, "returns and integer" should be "returns an integer"

GP, page 9:
"Exapolation" should be ""Extrapolation" in two places
"Exapolate" should be "Extrapolate"

GP, page 10:
"range check the current altitude" should be "range check the VELOCITY_ERROR"

GP, page 11
"GP_ALTITUDE[0] =<" should be "GP_ALTITUDE[0] <=" for consistency

Individual Inspection Preparation Log #1 (Page 1)

Name: Bernice Becher   Date Log Submitted: 11/15/94

Implementation: Pluto   Date of Inspection 11/16/94

Role: Inspector

Defects/Clarity Problems/Concerns

PLUTO CODE LOG I

GENERAL ISSUES

36 All Modules

The correspondence between P-Specification number in the design and FORTRAN modules is not given

*Requirement: nonambiguity
*Requirement: traceability
*Requirement: completeness

37W Using enumeration of all combinations of subscripts vs. using DO loops:

In many cases where a rotation is to be done, or where range-checking is to be done for an array, loops with variable indices are not used, but rather a separate assignment statement is given for each element of the array. This cannot be considered an error; however, in the code it is quite error-prone, difficult to verify, difficult to maintain in the case of changes to the requirements, and involves many more lines of code than would otherwise be necessary. (see e.g., GP, lines 728-888; TDLRSP, lines 711-764)

*Requirement: verifiability
*Requirement: modifiability

35W Constants

Many of the floating point constants used in the code have not been written in a format which explicitly declares them as double precision constants. Whether this will cause a loss of precision seems to depend on other factors such as the use of parentheses and the order in which the operations are done. In some cases, it is clear from experience that there is definitely a problem, and in some cases, there is the potential for a problem. Also, many of the constants in floating point expressions are written without decimal points (which probably will not cause a problem).

Some particular cases noticed are:

AECLP.FOR:

None of the constants from lines 974 through 978 is explicitly declared as double precision. The one which is specifically in question in terms of possible precision problems is "0.5".

ASP.FOR

Lines 818 and 837, constant is 3.0

GP.FOR

Lines 995, 1007, and 1017, constant is 6.0
Lines 993, 1006, 1017, constant is 2.0
Line 1086, constant is 1000.0

Individual Inspection Preparation Log #1 (Page 2)

Name: Bernice Becher Date Log Submitted: 11/15/94

Implementation: Pluto Date of Inspection 11/16/94

Role: Inspector

Defects/Clarity Problems/Concerns

TDLRSP.FOR
Lines 925, 935, 948, 959, etc., constant is 2.0

TSP.FOR
Lines 738 AND 749, constant is 0.15

LOWER_PARABOLIC_FUNCTION.FOR
Line 178, constant is 2.0

UPPER_PARABOLIC_FUNCTION.FOR
Line 178, constant is 2.0

CONSTANTS.DAT
All of the upper and lower bounds.
K$THETA$UB and K$THETA$LB
*Requirement: accuracy
*Requirement: nonambiguity
*Requirement: specification

56H Modules with Arguments:

The Software Standards state that each module should list its arguments, and the Pluto modules do this. It would be very helpful if a comment would state for each argument whether it is an input, output or both. The modules in question are: CRC16, DERIV_ATT, DERIV_VEL, DERIV_ALT, MULT_ATT, MULT_VEL, (AVG_ATT), (AVG_VEL), RANGE_CHECK, NEG_VALUE_CHECK, and ZERO_CHECK.
*Requirement: nonambiguity
*Requirement: completeness

58 All Modules

The Software Standards state that each module header should include the "DATE FIRST SUBMITTED FOR CONFIGURATION MANAGEMENT". The Pluto modules do not appear to have this date, as 15-Sep-1994 is not the configuration management date, which apparently is 26-Sep-1994.
*Requirement: Software Standards

SPECIFIC PROBLEMS IN MODULES

30 EXTERNAL.FOR

Under "structure /clp_data_t/", the data type for ae_temp is incorrect.
*Requirement: accuracy
*Requirement: consistency

Individual Inspection Preparation Log #1 (Page 3)

Name: Bernice Becher   Date Log Submitted: 11/15/94
Implementation: Pluto   Date of Inspection 11/16/94
Role: Inspector

Defects/Clarity Problems/Concerns

32 PLUTO.FOR
The check for whether to terminate is being done at the end of the second subframe. Formal Modification 2.3-2.1 (Scheduling Section) states that this check should be done "immediately after executing the Control Law Processing subframe".
*Requirement: specification

33I PLUTO.FOR
The statement near the end of the loop, namely "go to 100" is an unconditional GO TO which is not permitted according to the Software Standards.
*Requirement: Software Standards

34 AECLP.FOR, between lines 895 and 897:
A divide-by-zero check is required for the variable OMEGA.
*Requirement: specification

36 ARSP.FOR:
Line 746: The constant "3E08" is not explicitly double precision and may cause a loss of precision.
*Requirement: specification

38W CRC.FOR, lines 41 through 136
Warning: The "data" statements used to initialize the array "table" are very tedious to check and the check may be prone to error.
*Requirement: verifiability

40 CRC.FOR
In line 37, the hexadecimal constant given for the CRC-16 generator polynomial is not correct.
*Requirement: accuracy

57 CRC16
In the section "Returns", it states that CRC16 is "the CRC-16" of the specified message." The "CRC-16 is a bit ambiguous, as it does not explicitly state it is the checksum or error code.
*Requirement: nonambiguity

41 GP.FOR
In line 909, the first argument , namely "att_k1", is incorrect.
*Requirement: accuracy

Individual Inspection Preparation Log #1 (Page 4)

Name: Bernice Becher Date Log Submitted: 11/15/94
Implementation: Pluto Date of Inspection 11/16/94
Role: Inspector

Defects/Clarity Problems/Concerns

42 GP.FOR

Lines 921 & 922, 925 & 926, 929, 939 & 940, 943 & 944, and 947 are not correct.

The subroutines avg_att and avg_vel are performing an incorrect function, and thus the second argument for each derivative call is incorrect. These problems directly relate to design problem #7

*Requirement: accuracy
*Requirement: traceability
*Requirement: specification

43 GP.FOR

In line 970, the last argument for deriv_vel, namely "1", is not correct.

*Requirement: accuracy
*Requirement: traceability
*Requirement: specification

44 GP.FOR

Lines 1095, 1114, 1132, 1156, 1203, 1224, and 1256 are unconditional "GO TOs, which are prohibited by the Software Standards, and which also differ from the design p-spec.

*Requirement:Software Standards

45 GP.FOR

In line 1178, the relational operator, namely ".GE.", is not correct.

*Requirement: specification
*Requirement: accuracy

46I GP.FOR

In line 1190, a computed GO TO (which is a variant of unconditional GO TO) is used. Is this permitted?
If it is permitted, then a fall-through statement may be needed in the case where GP_PHASE is not 1,2,3,4, or 5 (in which case no action should be taken as opposed to the action for GP_PHASE = 1).

*Requirement: accuracy
*Requirement: traceability
*Requirement: completeness
*Requirement: specification

47 DERIV_ATT.FOR

In lines 72-74, it was intended that the variables pv, qv, and rv will yield the appropriate values of G_ROTATION. The EQUIVALENCE statements do not accomplish what was intended, and therefore, lines 78 through 88 will yield incorrect results.

*Requirement: accuracy
*Requirement: specification

Individual Inspection Preparation Log #1 (Page 5)

Name: Bernice Becher   Date Log Submitted: 11/15/94

Implementation: Pluto   Date of Inspection 11/16/94

Role: Inspector

Defects/Clarity Problems/Concerns

47 DERIV_VEL.FOR

In lines 297-299, it was intended that the variables pv, qv, and rv will yield the appropriate values of G_ROTATION. The EQUIVALENCE statements do not accomplish what was intended, and therefore, lines 309, 316, AND 323 will yield incorrect results.

*Requirement: accuracy
*Requirement: specification

48 AVG_ATT.FOR

This subroutine is performing a function which is not required at all. This problem is related to design problem #7 and to code problem #42.

*Requirement: traceability

49 AVG_VEL.FOR

This subroutine is performing a function which is not required at all. This problem is related to design problem #7 and to code problem #42.

*Requirement: traceability

50 TDLRSP.FOR

In lines 906 through 909, a computed GO TO is used. Is this permitted?
If it is permitted, then a fall-through statement may needed in the case where the computed expression is less than 1 or greater than 15.

*Requirement: accuracy
*Requirement: traceability
*Requirement: completeness
*Requirement: specification

51 TDLRSP.FOR

Following line 963, there is no control statement, and so control will pass to line 967, which is not correct.

*Requirement: accuracy
*Requirement: traceability
*Requirement: specification

52 LOWER_PARABOLIC_FUNCTION.FOR

In line 181, the addition operator in the term "...M3 + half_slope..." is incorrect.

*Requirement: accuracy
*Requirement: specification

53 UPPER_PARABOLIC_FUNCTION.FOR

In line 181, both arithmetic operators immediately preceding "half_slope" (namely "-" and then "+") are incorrect.

*Requirement: accuracy
*Requirement: specification

Individual Inspection Preparation Log #1 (Page 6)

Name: Bernice Becher | Date Log Submitted: 11/15/94
Implementation: Pluto | Date of Inspection 11/16/94
Role: Inspector

Defects/Clarity Problems/Concerns

54I UTILITY.FOR (subroutine RANGE_CHECK)

The specification states to "...display the name of the data element in question"... In the case of an array, this implementation displays the name of the array, but not the subscript(s) of the element in question. Two issues arise: Should it be required that the subscripts be displayed? Should the specification be reworded?

*Requirement: completeness

55 UTILITY.FOR (subroutines RANGE_CHECK, NEG_VALUE_CHECK, and ZERO_CHECK)

In each of the three subroutines, FORMAT statement 30 is missing "x," immediately before the "I4".

*Requirement: accuracy

*Requirement: specification

TYPOS

EXTERNAL.FOR

Heading: "Originial" should be "Original"

ASP.FOR, page 6, comment on line 970:

"convertion" should be "conversion"

GP.FOR, page 9, lines 1125, 1141, and 1149:

"exapolat..." should be "extrapolat..."

Review Log from Verification Analyst

The following are deficiencies discovered in the Pluto code during the code review process. The list is organized by file name and in alphabetical order.

Reviewer: Cuong C. Quach

ARSP.FOR

1) Typo in the comment for step 3 C). ..
   "...mostly recently..."
   Citation: Typographical error.

CONSTANTS.FOR

1) AE_TEMP constants are of incorrect type. Should be Integer*2, not Logical*1
   Citation: Specification not followed.

CP.FOR

1) The variable name "PACKET.DATA_MASK" used to build the packet for subframe 1 is typographically different from the same variable used to build the packet for the other two subframes.
   Citation: Coding clarity is compromised.

2) The assignment of the sequence field directly from the MOD intrinsic function is erroneous. The MOD function returns a integer quantity but its assigned to a logical.
   Citation: Fortran syntax violated.

EXTERNAL.FOR

1) In the structure declaration for "clp_data_t", the element ae_temp is not declared correctly according to the Specification.
   Citation: Specification not followed.

GSP.FOR

1) The local variable "counter" is typed as a "real*8" when it should be an "integer*2"
   Citation: Specification not followed.

TDLRSP.FOR

1) In the table look-up scheme for obtaining beam velocities. The initial computation is offset by 1. This would cause selection of beam processing not to agree with the specification.
   Citation: Specification not followed.

GP.FOR

1) In the MULT_ATT subroutine, the second index, of the array element to be multiplied with the "factor", is incorrect for the following elements
   att(1,2)
   att(1,3)
   att(2,2)
   att(2,3)
   att(3,2)
   att(3,3)
   Citation: Specification not followed as a result of typographical error.

2) In the DERIV_VEL subroutine, the index for "temp(1)" is incorrect for the following statements:
   temp(1) = TDLR_VELOCITY(2,index) - vel(2)
   temp(1) = TDLR_VELOCITY(3,index) - vel(3)
   Citation: Specification not followed as a result of typographical error.

3) In GP, at step 4 of the RK-method where vel_k4 is calculated, the wrong history index is passed into the "deriv_vel" derivative routine.
   Citation: Specification not followed as a result of typographical error.

4) In step 5 - determining if contour-altitude has been crossed, the first if comparison should be ".LE."
   Citation: Specification not followed.

PLUTO.FOR

1) The third subframe is not executed when GP_PHASE =5. this is incorrect.
   Citation: Specification not followed.

Appendix D: Test Results Logs for the Pluto Implementation of the Guidance and Control Software

Author: Cuong C. Quach, NASA Langley Research Center

This document was produced as part of Guidance and Control Software (GCS) Project conducted at NASA Langley Research Center. Although some of the requirements for the Guidance and Control Software application were derived from the NASA Viking Mission to Mars, this document does not contain data from an actual NASA mission.

D. Contents

D.1 Pluto Test Case Results Log for AECLP

TEST CASE NAME	EXECUTION DATE	DATE CODE FETCHED	DATE TEST CASE FETCHED	RESULTS (was .ANA file generated Y or N?)	PR #
AECLP_NR_001	1/5/95	12/21/94	12/21/94	N	
AECLP_NR_002				N	
AECLP_NR_003				N	
AECLP_NR_004				N	
AECLP_NR_005				N	
AECLP_NR_006				N	
AECLP_NR_007				N	
AECLP_NR_008				N	
AECLP_NR_009				N	
AECLP_NR_010				N	
AECLP_NR_011				N	
AECLP_NR_012				N	
AECLP_RO_013				N	
AECLP_RO_014				N	
AECLP_RO_015				N	
AECLP_RO_016				N	
AECLP_RO_017				N	
AECLP_RO_018				N	
AECLP_RO_019				N	
AECLP_RO_020				N	
AECLP_RO_021				N	
AECLP_RO_022				N	
AECLP_RO_023				N	
AECLP_RO_024				N	
AECLP_RO_025				N	
AECLP_RO_026				N	
AECLP_RO_027				N	
AECLP_RO_028				N	
AECLP_RO_029				N	
AECLP_RO_030				N	
AECLP_RO_031				N	
AECLP_RO_032				N	
AECLP_RO_033				N	
AECLP_RO_034				N	
AECLP_RO_035				N	
AECLP_RO_036				N	

AECLP_RO_037				N	
AECLP_RO_038				N	
AECLP_RO_039				N	
AECLP_RO_040				N	
AECLP_RO_041				N	
AECLP_RO_042				N	
AECLP_RO_043				N	
AECLP_RO_044				N	
AECLP_RO_045				N	
AECLP_RO_046				N	
AECLP_RO_047				N	
AECLP_RO_048				N	
AECLP_RO_049				N	
AECLP_RO_050				N	
AECLP_RO_051				N	
AECLP_RO_052				N	
AECLP_RO_053				N	
AECLP_NR_054				N	
AECLP_NR_055				N	
AECLP_RO_056				N	
AECLP_RO_057				N	
AECLP_NR_001	1/18/95	12/21/94	12/21/94	N*	
AECLP_NR_002				N*	
AECLP_NR_003				N*	
AECLP_NR_004				N*	
AECLP_NR_005				N*	
AECLP_NR_006				N*	
AECLP_NR_007				N*	
AECLP_NR_008				N*	
AECLP_NR_009				N*	
AECLP_NR_010				N*	
AECLP_NR_011				N*	
AECLP_NR_012				N*	
AECLP_RO_013				N*	
AECLP_RO_014				N*	
AECLP_RO_015				N*	
AECLP_RO_016				N*	
AECLP_RO_017				N*	
AECLP_RO_018				N*	
AECLP_RO_019				N*	
AECLP_RO_020				N*	
AECLP_RO_021				N*	

AECLP_RO_022				N*	
AECLP_RO_023				N*	
AECLP_RO_024				N*	
AECLP_RO_025				N*	
AECLP_RO_026				N*	
AECLP_RO_027				N*	
AECLP_RO_028				N*	
AECLP_RO_029				N*	
AECLP_RO_030				N*	
AECLP_RO_031				N*	
AECLP_RO_032				N*	
AECLP_RO_033				N*	
AECLP_RO_034				N*	
AECLP_RO_035				N*	
AECLP_RO_036				N*	
AECLP_RO_037				N*	
AECLP_RO_038				N*	
AECLP_RO_039				N*	
AECLP_RO_040				N*	
AECLP_RO_041				N*	
AECLP_RO_042				N*	
AECLP_RO_043				N*	
AECLP_RO_044				N*	
AECLP_RO_045				N*	
AECLP_RO_046				N*	
AECLP_RO_047				N*	
AECLP_RO_048				N*	
AECLP_RO_049				N*	
AECLP_RO_050				N*	
AECLP_RO_051				N*	
AECLP_RO_052				N*	
AECLP_RO_053				N*	
AECLP_NR_054				N*	
AECLP_NR_055				N*	
AECLP_RO_056				N*	
AECLP_RO_057				N*	
AECLP_NR_001	4/7/95	4/6/95	4/7/95	N	
AECLP_NR_002				N	
AECLP_NR_003				N	
AECLP_NR_004				N	
AECLP_NR_005				N	

AECLP_NR_006				N	
AECLP_NR_007				N	
AECLP_NR_008				N	
AECLP_NR_009				N	
AECLP_NR_010				N	
AECLP_NR_011				N	
AECLP_NR_012				N	
AECLP_RO_013				N	
AECLP_RO_014				N	
AECLP_RO_015				N	
AECLP_RO_016				N	
AECLP_RO_017				N	
AECLP_RO_018				N	
AECLP_RO_019				N	
AECLP_RO_020				N	
AECLP_RO_021				N	
AECLP_RO_022				N	
AECLP_RO_023				N	
AECLP_RO_024				N	
AECLP_RO_025				N	
AECLP_RO_026				N	
AECLP_RO_027				N	
AECLP_RO_028				N	
AECLP_RO_029				N	
AECLP_RO_030				N	
AECLP_RO_031				N	
AECLP_RO_032				N	
AECLP_RO_033				N	
AECLP_RO_034				N	
AECLP_RO_035				N	
AECLP_RO_036				N	
AECLP_RO_037				N	
AECLP_RO_038				N	
AECLP_RO_039				N	
AECLP_RO_040				N	
AECLP_RO_041				N	
AECLP_RO_042				N	
AECLP_RO_043				N	
AECLP_RO_044				N	
AECLP_RO_045				N	
AECLP_RO_046				N	
AECLP_RO_047				N	

AECLP_RO_048				N	
AECLP_RO_049				N	
AECLP_RO_050				N	
AECLP_RO_051				N	
AECLP_RO_052				N	
AECLP_RO_053				N	
AECLP_NR_054				N	
AECLP_NR_055				N	
AECLP_RO_056				N	
AECLP_RO_057				N	

*: These test cases had to be re-executed because the include file CONSTANTS.FOR was changed in PR#24.

D.2 Pluto Test Case Results Log for ARSP

TEST CASE NAME	EXECUTION DATE	DATE CODE FETCHED	DATE TEST CASE FETCHED	RESULTS (was .ANA file generated Y or N?)	PR #
ARSP_RO_001.TC	1/5/95	12/21/94	12/21/94	N	
ARSP_RO_002.TC				N	
ARSP_RO_003.TC				N	
ARSP_RO_004.TC				N	
ARSP_RO_005.TC				N	
ARSP_RO_006.TC				N	
ARSP_RO_007.TC				N	
ARSP_RO_008.TC				N	
ARSP_RO_009.TC				N	
ARSP_RO_010.TC				N	
ARSP_NR_011.TC				N	
ARSP_NR_012.TC				N	
ARSP_NR_013.TC				N	
ARSP_NR_014.TC				N	
ARSP_NR_015.TC				N	
ARSP_NR_016.TC				N	
ARSP_NR_017.TC				Y	24
ARSP_RO_018.TC				N	
ARSP_RO_019.TC				N	
ARSP_RO_020.TC				N	
ARSP_RO_021.TC				N	
ARSP_NR_022.TC				Y	24
ARSP_NR_023.TC				Y	24
ARSP_RO_001.TC	1/13/95	1/13/95	12/21/94	N	
ARSP_RO_002.TC				N	
ARSP_RO_003.TC				N	
ARSP_RO_004.TC				N	
ARSP_RO_005.TC				N	
ARSP_RO_006.TC				N	
ARSP_RO_007.TC				N	
ARSP_RO_008.TC				N	
ARSP_RO_009.TC				N	
ARSP_RO_010.TC				N	
ARSP_NR_011.TC				N	
ARSP_NR_012.TC				N	
ARSP_NR_013.TC				N	
ARSP_NR_014.TC				N	
ARSP_NR_015.TC				N	

ARSP_NR_016.TC				N	
ARSP_NR_017.TC				N	
ARSP_RO_018.TC				N	
ARSP_RO_019.TC				N	
ARSP_RO_020.TC				N	
ARSP_RO_021.TC				N	
ARSP_NR_022.TC				N	
ARSP_NR_023.TC				N	
ARSP_RO_001.TC	4/7/95	4/6/95	4/6/95	N	
ARSP_RO_002.TC				N	
ARSP_RO_003.TC				N	
ARSP_RO_004.TC				N	
ARSP_RO_005.TC				N	
ARSP_RO_006.TC				N	
ARSP_RO_007.TC				N	
ARSP_RO_008.TC				N	
ARSP_RO_009.TC				N	
ARSP_RO_010.TC				N	
ARSP_NR_011.TC				N	
ARSP_NR_012.TC				N	
ARSP_NR_013.TC				N	
ARSP_NR_014.TC				N	
ARSP_NR_015.TC				N	
ARSP_NR_016.TC				N	
ARSP_NR_017.TC				N	
ARSP_RO_018.TC				N	
ARSP_RO_019.TC				N	
ARSP_RO_020.TC				N	
ARSP_RO_021.TC				N	
ARSP_NR_022.TC				N	
ARSP_NR_023.TC				N	

D.3 Pluto Test Case Results Log for ASP

TEST CASE NAME	EXECUTION DATE	DATE CODE FETCHED	DATE TEST CASE FETCHED	RESULTS (was .ANA file generated Y or N?)	PR #
ASP_NR_001.TC	1/5/95	12/21/94	12/21/94	N	
ASP_NR_002.TC				N	
ASP_NR_003.TC				N	
ASP_NR_004.TC				N	
ASP_NR_005.TC				N	
ASP_NR_006.TC				N	
ASP_NR_007.TC				N	
ASP_RO_008.TC				N	
ASP_RO_009.TC				N	
ASP_RO_010.TC				N	
ASP_RO_011.TC				N	
ASP_RO_012.TC				N	
ASP_RO_013.TC				N	
ASP_RO_014.TC				N	
ASP_RO_015.TC				N	
ASP_NR_016.TC				N	
ASP_RO_017.TC				N	
ASP_RO_018.TC				N	
ASP_RO_019.TC				N	
ASP_RO_020.TC				N	
ASP_RO_021.TC				N	
ASP_RO_022.TC				N	
ASP_RO_023.TC				N	
ASP_RO_024.TC				N	
ASP_RO_025.TC				N	
ASP_RO_026.TC				N	
ASP_RO_027.TC				N	
ASP_RO_028.TC				N	
ASP_RO_029.TC				N	
ASP_RO_030.TC				N	
ASP_RO_031.TC				N	
ASP_RO_032.TC				N	
ASP_RO_033.TC				N	
ASP_RO_034.TC				N	
ASP_RO_035.TC				N	
ASP_RO_036.TC				N	
ASP_RO_037.TC				N	
ASP_RO_038.TC				N	

ASP_RO_039.TC				N	
ASP_RO_040.TC				N	
ASP_RO_041.TC				N	
ASP_RO_042.TC				N	
ASP_RO_043.TC				N	
ASP_RO_044.TC				N	
ASP_NR_001.TC	1/17/95	12/21/94	12/21/94	N*	
ASP_NR_002.TC				N*	
ASP_NR_003.TC				N*	
ASP_NR_004.TC				N*	
ASP_NR_005.TC				N*	
ASP_NR_006.TC				N*	
ASP_NR_007.TC				N*	
ASP_RO_008.TC				N*	
ASP_RO_009.TC				N*	
ASP_RO_010.TC				N*	
ASP_RO_011.TC				N*	
ASP_RO_012.TC				N*	
ASP_RO_013.TC				N*	
ASP_RO_014.TC				N*	
ASP_RO_015.TC				N*	
ASP_NR_016.TC				N*	
ASP_RO_017.TC				N*	
ASP_RO_018.TC				N*	
ASP_RO_019.TC				N*	
ASP_RO_020.TC				N*	
ASP_RO_021.TC				N*	
ASP_RO_022.TC				N*	
ASP_RO_023.TC				N*	
ASP_RO_024.TC				N*	
ASP_RO_025.TC				N*	
ASP_RO_026.TC				N*	
ASP_RO_027.TC				N*	
ASP_RO_028.TC				N*	
ASP_RO_029.TC				N*	
ASP_RO_030.TC				N*	
ASP_RO_031.TC				N*	
ASP_RO_032.TC				N*	
ASP_RO_033.TC				N*	
ASP_RO_034.TC				N*	
ASP_RO_035.TC				N*	
ASP_RO_036.TC				N*	
ASP_RO_037.TC				N*	
ASP_RO_038.TC				N*	

ASP_RO_039.TC				N*	
ASP_RO_040.TC				N*	
ASP_RO_041.TC				N*	
ASP_RO_042.TC				N*	
ASP_RO_043.TC				N*	
ASP_RO_044.TC				N*	
ASP_NR_001.TC	4/7/95	4/6/95	4/6/95	N	
ASP_NR_002.TC				N	
ASP_NR_003.TC				N	
ASP_NR_004.TC				N	
ASP_NR_005.TC				N	
ASP_NR_006.TC				N	
ASP_NR_007.TC				N	
ASP_RO_008.TC				N	
ASP_RO_009.TC				N	
ASP_RO_010.TC				N	
ASP_RO_011.TC				N	
ASP_RO_012.TC				N	
ASP_RO_013.TC				N	
ASP_RO_014.TC				N	
ASP_RO_015.TC				N	
ASP_NR_016.TC				N	
ASP_RO_017.TC				N	
ASP_RO_018.TC				N	
ASP_RO_019.TC				N	
ASP_RO_020.TC				N	
ASP_RO_021.TC				N	
ASP_RO_022.TC				N	
ASP_RO_023.TC				N	
ASP_RO_024.TC				N	
ASP_RO_025.TC				N	
ASP_RO_026.TC				N	
ASP_RO_027.TC				N	
ASP_RO_028.TC				N	
ASP_RO_029.TC				N	
ASP_RO_030.TC				N	
ASP_RO_031.TC				N	
ASP_RO_032.TC				N	
ASP_RO_033.TC				N	
ASP_RO_034.TC				N	
ASP_RO_035.TC				N	
ASP_RO_036.TC				N	
ASP_RO_037.TC				N	
ASP_RO_038.TC				N	

ASP_RO_039.TC				N	
ASP_RO_040.TC				N	
ASP_RO_041.TC				N	
ASP_RO_042.TC				N	
ASP_RO_043.TC				N	
ASP_RO_044.TC				N	

*: These test cases had to be re-executed because the include file CONSTANTS.FOR was changed in PR#24.

D.4 Pluto Test Case Results Log for CP

TEST CASE NAME	EXECUTION DATE	DATE CODE FETCHED	DATE TEST CASE FETCHED	RESULTS (was .ANA file generated Y or N?)	PR #
CP_NR_001.TC	1/12/95	12/28/94	1/12/95	Y	25
CP_NR_002.TC				Y	25
CP_NR_003.TC				Y	25
CP_NR_004.TC				Y	25
CP_NR_005.TC				Y	25
CP_NR_001.TC	1/19/95	1/19/95	1/12/95	N	
CP_NR_002.TC				N	
CP_NR_003.TC				N	
CP_NR_004.TC				N	
CP_NR_005.TC				N	
CP_NR_001.TC	4/7/95	4/6/95	4/7/95	N	
CP_NR_002.TC				N	
CP_NR_003.TC				N	
CP_NR_004.TC				N	
CP_NR_005.TC				N	

D.5 Pluto Test Case Results Log for CRCP

TEST CASE NAME	EXECUTION DATE	DATE CODE FETCHED	DATE TEST CASE FETCHED	RESULTS (was .ANA file generated Y or N?)	PR #
CRCP_NR_001	1/5/95	12/21/94	12/21/94	N	
CRCP_NR_002				N	
CRCP_NR_003				N	
CRCP_NR_004				N	
CRCP_NR_005				N	
CRCP_NR_006				N	
CRCP_RO_007				N	
CRCP_RO_008				N	
CRCP_RO_009				N	
CRCP_RO_010				N	
CRCP_NR_001	1/17/95	12/21/94	12/21/94	N*	
CRCP_NR_002				N*	
CRCP_NR_003				N*	
CRCP_NR_004				N*	
CRCP_NR_005				N*	
CRCP_NR_006				N*	
CRCP_RO_007				N*	
CRCP_RO_008				N*	
CRCP_RO_009				N*	
CRCP_RO_010				N*	
CRCP_NR_001	4/7/95	4/6/94	4/7/94	N	
CRCP_NR_002				N	
CRCP_NR_003				N	
CRCP_NR_004				N	
CRCP_NR_005				N	
CRCP_NR_006				N	
CRCP_RO_007				N	
CRCP_RO_008				N	
CRCP_RO_009				N	
CRCP_RO_010				N	

*: These test cases had to be re-executed because the include file CONSTANTS.FOR was changed in PR#24.

D.6 Pluto Test Case Results Log for GP

TEST CASE NAME	EXECUTION DATE	DATE CODE FETCHED	DATE TEST CASE FETCHED	RESULTS (was .ANA file generated Y or N?)	PR #
GP_NR_001	1/4/95	12/21/94	1/4/95	Y	24
GP_NR_002				Y	24
GP_NR_003				Y	24
GP_NR_004				Y	24
GP_NR_005				Y	24
GP_NR_006				Y	24
GP_NR_007				Y	24
GP_NR_008				Y	24
GP_RO_009				Y	24
GP_RO_010				Y	24
GP_RO_011				Y	24
GP_RO_012				Y	24
GP_RO_013				Y	24
GP_RO_014				Y	24
GP_RO_015				Y	24
GP_RO_016				Y	24
GP_RO_017				Y	24
GP_RO_018				Y	24
GP_RO_019				Y	24
GP_RO_020				Y	24
GP_RO_021				Y	24
GP_RO_022				Y	24
GP_RO_023				Y	24
GP_RO_024				Y	24
GP_RO_025				Y	24
GP_RO_026				Y	24
GP_RO_027				Y	24
GP_RO_028				Y	24
GP_RO_029				Y	24
GP_RO_030				Y	24
GP_RO_031				Y	24
GP_RO_032				Y	24
GP_RO_033				Y	24
GP_RO_034				Y	24
GP_RO_035				Y	24
GP_RO_036				Y	24
GP_RO_037				Y	24
GP_RO_038				Y	24

GP_RO_039				Y	24
GP_RO_040				Y	24
GP_RO_041				Y	24
GP_RO_042				Y	24
GP_RO_043				Y	24
GP_RO_044				Y	24
GP_RO_045				Y	24
GP_RO_046				Y	24
GP_RO_047				Y	24
GP_RO_048				Y	24
GP_RO_049				Y	24
GP_RO_050				Y	24
GP_RO_051				Y	24
GP_RO_052				Y	24
GP_NR_053				Y	24
GP_RO_054				Y	24
GP_RO_055				Y	24
GP_RO_056				Y	24
GP_RO_057				Y	24
GP_RO_058				Y	24
GP_RO_059				Y	24
GP_RO_060				Y	24
GP_RO_061				Y	24
GP_RO_062				Y	24
GP_RO_063				Y	24
GP_RO_064				Y	24
GP_RO_065				Y	24
GP_RO_066				Y	24
GP_RO_067				Y	24
GP_RO_068				Y	24
GP_RO_069				Y	24
GP_RO_070				Y	24
GP_RO_071				Y	24
GP_RO_072				Y	24
GP_RO_073				Y	24
GP_RO_074				Y	24
GP_RO_075				Y	24
GP_RO_076				Y	24
GP_RO_077				Y	24
GP_RO_078				Y	24
GP_RO_079				Y	24
GP_RO_080				Y	24
GP_RO_081				Y	24
GP_RO_082				Y	24

GP_RO_083				Y	24
GP_RO_084				Y	24
GP_RO_085				Y	24
GP_RO_086				Y	24
GP_RO_087				Y	24
GP_RO_088				Y	24
GP_RO_089				Y	24
GP_RO_090				Y	24
GP_RO_091				Y	24
GP_RO_092				Y	24
GP_RO_093				Y	24
GP_RO_094				Y	24
GP_RO_095				Y	24
GP_RO_096				Y	24
GP_RO_097				Y	24
GP_RO_098				Y	24
GP_RO_099				Y	24
GP_RO_100				Y	24
GP_RO_101				Y	24
GP_NR_102				Y	24
GP_NR_103				Y	24
GP_NR_104				Y	24
GP_NR_105				Y	24
GP_NR_106				Y	24
GP_RO_107				Y	24
GP_RO_108				Y	24
GP_RO_109				Y	24
GP_RO_110				Y	24
GP_RO_111				Y	24
GP_RO_112				Y	24
GP_RO_113				Y	24
GP_RO_114				Y	24
GP_RO_115				Y	24
GP_RO_116				Y	24
GP_NR_001	1/13/95	1/13/95	1/4/95+	N	
GP_NR_002	1/13/95	1/13/95	12/21/94	N	
GP_NR_003				N	
GP_NR_004				N	
GP_NR_005				N	
GP_NR_006				N	
GP_NR_007				N	
GP_NR_008				N	
GP_RO_009				N	
GP_RO_010				N	

GP_RO_011				N	
GP_RO_012				N	
GP_RO_013				N	
GP_RO_014				N	
GP_RO_015				N	
GP_RO_016				N	
GP_RO_017				N	
GP_RO_018				N	
GP_RO_019				N	
GP_RO_020				N	
GP_RO_021				N	
GP_RO_022				N	
GP_RO_023				N	
GP_RO_024				N	
GP_RO_025				N	
GP_RO_026				N	
GP_RO_027				N	
GP_RO_028				N	
GP_RO_029				N	
GP_RO_030				N	
GP_RO_031				N	
GP_RO_032				N	
GP_RO_033				N	
GP_RO_034				N	
GP_RO_035				N	
GP_RO_036				N	
GP_RO_037				N	
GP_RO_038				N	
GP_RO_039				N	
GP_RO_040				N	
GP_RO_041				N	
GP_RO_042				N	
GP_RO_043				N	
GP_RO_044				N	
GP_RO_045				N	
GP_RO_046				N	
GP_RO_047				N	
GP_RO_048				N	
GP_RO_049				N	
GP_RO_050				N	
GP_RO_051				N	
GP_RO_052				N	
GP_NR_053				N	
GP_RO_054				N	

GP_RO_055				N	
GP_RO_056				N	
GP_RO_057				N	
GP_RO_058				N	
GP_RO_059				N	
GP_RO_060				N	
GP_RO_061				N	
GP_RO_062				N	
GP_RO_063				N	
GP_RO_064				N	
GP_RO_065				N	
GP_RO_066				N	
GP_RO_067				N	
GP_RO_068				N	
GP_RO_069				N	
GP_RO_070				N	
GP_RO_071				N	
GP_RO_072				N	
GP_RO_073				N	
GP_RO_074				N	
GP_RO_075				N	
GP_RO_076				N	
GP_RO_077				N	
GP_RO_078				N	
GP_RO_079				N	
GP_RO_080				N	
GP_RO_081				N	
GP_RO_082				N	
GP_RO_083				N	
GP_RO_084				N	
GP_RO_085				N	
GP_RO_086				N	
GP_RO_087				N	
GP_RO_088				N	
GP_RO_089				N	
GP_RO_090				N	
GP_RO_091				N	
GP_RO_092				N	
GP_RO_093				N	
GP_RO_094				N	
GP_RO_095				N	
GP_RO_096				N	
GP_RO_097				N	
GP_RO_098				N	

GP_RO_099				N	
GP_RO_100				N	
GP_RO_101				N	
GP_NR_102				N	
GP_NR_103				N	
GP_NR_104				N	
GP_NR_105				N	
GP_NR_106				N	
GP_RO_107				N	
GP_RO_108				N	
GP_RO_109				N	
GP_RO_110				N	
GP_RO_111				N	
GP_RO_112				N	
GP_RO_113				N	
GP_RO_114				N	
GP_RO_115				N	
GP_RO_116				N	
GP_NR_001	3/1/95	1/13/95	3/1/95	N	
GP_NR_002				N	
GP_NR_003				N	
GP_NR_004				N	
GP_NR_005				N	
GP_NR_006				N	
GP_NR_007				N	
GP_NR_008				N	
GP_RO_009				N	
GP_RO_010				N	
GP_RO_011				N	
GP_RO_012				N	
GP_RO_013				N	
GP_RO_014				N	
GP_RO_015				N	
GP_RO_016				N	
GP_RO_017				N	
GP_RO_018				N	
GP_RO_019				N	
GP_RO_020				N	
GP_RO_021				N	
GP_RO_022				N	
GP_RO_023				N	
GP_RO_024				N	
GP_RO_025				N	
GP_RO_026				N	

GP_RO_027				N	
GP_RO_028				N	
GP_RO_029				N	
GP_RO_030				N	
GP_RO_031				N	
GP_RO_032				N	
GP_RO_033				N	
GP_RO_034				N	
GP_RO_035				N	
GP_RO_036				N	
GP_RO_037				N	
GP_RO_038				N	
GP_RO_039				N	
GP_RO_040				N	
GP_RO_041				N	
GP_RO_042				N	
GP_RO_043				N	
GP_RO_044				N	
GP_RO_045				N	
GP_RO_046				N	
GP_RO_047				N	
GP_RO_048				N	
GP_RO_049				N	
GP_RO_050				N	
GP_RO_051				N	
GP_RO_052				N	
GP_NR_053				N	
GP_RO_054				N	
GP_RO_055				N	
GP_RO_056				N	
GP_RO_057				N	
GP_RO_058				N	
GP_RO_059				N	
GP_RO_060				N	
GP_RO_061				N	
GP_RO_062				N	
GP_RO_063				N	
GP_RO_064				N	
GP_RO_065				N	
GP_RO_066				N	
GP_RO_067				N	
GP_RO_068				N	
GP_RO_069				N	
GP_RO_070				N	

GP_RO_071				N	
GP_RO_072				N	
GP_RO_073				N	
GP_RO_074				N	
GP_RO_075				N	
GP_RO_076				N	
GP_RO_077				N	
GP_RO_078				N	
GP_RO_079				N	
GP_RO_080				N	
GP_RO_081				N	
GP_RO_082				N	
GP_RO_083				N	
GP_RO_084				N	
GP_RO_085				N	
GP_RO_086				N	
GP_RO_087				N	
GP_RO_088				N	
GP_RO_089				N	
GP_RO_090				N	
GP_RO_091				N	
GP_RO_092				N	
GP_RO_093				N	
GP_RO_094				N	
GP_RO_095				N	
GP_RO_096				N	
GP_RO_097				N	
GP_RO_098				N	
GP_RO_099				N	
GP_RO_100				N	
GP_RO_101				N	
GP_NR_102				N	
GP_NR_103				N	
GP_NR_104				N	
GP_NR_105				N	
GP_NR_106				N	
GP_RO_107				N	
GP_RO_108				N	
GP_RO_109				N	
GP_RO_110				N	
GP_RO_111				N	
GP_RO_112				N	
GP_RO_113				N	
GP_RO_114				N	

GP_RO_115				N	
GP_RO_116				N	
GP_NR_001	4/795	4/6/95	4/7/95	N	
GP_NR_002				N	
GP_NR_003				N	
GP_NR_004				N	
GP_NR_005				N	
GP_NR_006				N	
GP_NR_007				N	
GP_NR_008				N	
GP_RO_009				N	
GP_RO_010				N	
GP_RO_011				N	
GP_RO_012				N	
GP_RO_013				N	
GP_RO_014				N	
GP_RO_015				N	
GP_RO_016				N	
GP_RO_017				N	
GP_RO_018				N	
GP_RO_019				N	
GP_RO_020				N	
GP_RO_021				N	
GP_RO_022				N	
GP_RO_023				N	
GP_RO_024				N	
GP_RO_025				N	
GP_RO_026				N	
GP_RO_027				N	
GP_RO_028				N	
GP_RO_029				N	
GP_RO_030				N	
GP_RO_031				N	
GP_RO_032				N	
GP_RO_033				N	
GP_RO_034				N	
GP_RO_035				N	
GP_RO_036				N	
GP_RO_037				N	
GP_RO_038				N	
GP_RO_039				N	
GP_RO_040				N	
GP_RO_041				N	
GP_RO_042				N	

GP_RO_043				N	
GP_RO_044				N	
GP_RO_045				N	
GP_RO_046				N	
GP_RO_047				N	
GP_RO_048				N	
GP_RO_049				N	
GP_RO_050				N	
GP_RO_051				N	
GP_RO_052				N	
GP_NR_053				N	
GP_RO_054				N	
GP_RO_055				N	
GP_RO_056				N	
GP_RO_057				N	
GP_RO_058				N	
GP_RO_059				N	
GP_RO_060				N	
GP_RO_061				N	
GP_RO_062				N	
GP_RO_063				N	
GP_RO_064				N	
GP_RO_065				N	
GP_RO_066				N	
GP_RO_067				N	
GP_RO_068				N	
GP_RO_069				N	
GP_RO_070				N	
GP_RO_071				N	
GP_RO_072				N	
GP_RO_073				N	
GP_RO_074				N	
GP_RO_075				N	
GP_RO_076				N	
GP_RO_077				N	
GP_RO_078				N	
GP_RO_079				N	
GP_RO_080				N	
GP_RO_081				N	
GP_RO_082				N	
GP_RO_083				N	
GP_RO_084				N	
GP_RO_085				N	
GP_RO_086				N	

GP_RO_087				N	
GP_RO_088				N	
GP_RO_089				N	
GP_RO_090				N	
GP_RO_091				N	
GP_RO_092				N	
GP_RO_093				N	
GP_RO_094				N	
GP_RO_095				N	
GP_RO_096				N	
GP_RO_097				N	
GP_RO_098				N	
GP_RO_099				N	
GP_RO_100				N	
GP_RO_101				N	
GP_NR_102				N	
GP_NR_103				N	
GP_NR_104				N	
GP_NR_105				N	
GP_NR_106				N	
GP_RO_107				N	
GP_RO_108				N	
GP_RO_109				N	
GP_RO_110				N	
GP_RO_111				N	
GP_RO_112				N	
GP_RO_113				N	
GP_RO_114				N	
GP_RO_115				N	
GP_RO_116				N	

D.7 Pluto Test Case Results Log for GSP

TEST CASE NAME	EXECUTION DATE	DATE CODE FETCHED	DATE TEST CASE FETCHED	RESULTS (was .ANA file generated Y or N?)	PR #
GSP_NR_001.TC	1/5/95	12/21/94	12/21/94	N	
GSP_RO_002.TC				N	
GSP_RO_003.TC				N	
GSP_RO_004.TC				N	
GSP_RO_005.TC				N	
GSP_RO_006.TC				N	
GSP_RO_007.TC				N	
GSP_RO_008.TC				N	
GSP_RO_009.TC				N	
GSP_NR_001.TC	1/17/95	12/21/94	12/21/94	N*	
GSP_RO_002.TC				N*	
GSP_RO_003.TC				N*	
GSP_RO_004.TC				N*	
GSP_RO_005.TC				N*	
GSP_RO_006.TC				N*	
GSP_RO_007.TC				N*	
GSP_RO_008.TC				N*	
GSP_RO_009.TC				N*	
GSP_NR_001.TC	4/7/95	4/6/95	4/6/95	N	
GSP_RO_002.TC				N	
GSP_RO_003.TC				N	
GSP_RO_004.TC				N	
GSP_RO_005.TC				N	
GSP_RO_006.TC				N	
GSP_RO_007.TC				N	
GSP_RO_008.TC				N	
GSP_RO_009.TC				N	

*: These test cases had to be re-executed because the include file CONSTANTS.FOR was changed in PR#24.

D.8 Pluto Test Case Results Log for RECLP

TEST CASE NAME	EXECUTION DATE	DATE CODE FETCHED	DATE TEST CASE FETCHED	RESULTS (was .ANA file generated Y or N?)	PR #
RECLP_NR_001	1/5/95	12/21/94	12/21/94	N*	24
RECLP_NR_002				N*	24
RECLP_NR_003				N*	24
RECLP_NR_004				N*	24
RECLP_NR_005				N*	24
RECLP_NR_006				N*	24
RECLP_NR_007				N*	24
RECLP_NR_008				N*	24
RECLP_NR_009				N*	24
RECLP_NR_010				N*	24
RECLP_NR_011				N*	24
RECLP_NR_012				N*	24
RECLP_NR_013				N*	24
RECLP_NR_014				N*	24
RECLP_NR_015				N*	24
RECLP_NR_016				N*	24
RECLP_NR_017				N*	24
RECLP_NR_018				N*	24
RECLP_NR_019				N*	24
RECLP_NR_020				N*	24
RECLP_NR_021				N*	24
RECLP_NR_022				N*	24
RECLP_NR_023				N*	24
RECLP_NR_024				N*	24
RECLP_NR_025				N*	24
RECLP_NR_026				N*	24
RECLP_NR_027				N*	24
RECLP_NR_028				N*	24
RECLP_NR_029				N*	24
RECLP_NR_030				N*	24
RECLP_NR_031				N*	24
RECLP_NR_032				N*	24
RECLP_NR_033				N*	24
RECLP_NR_034				N*	24
RECLP_NR_035				N*	24
RECLP_NR_036				N*	24
RECLP_NR_037				N*	24
RECLP_NR_038				N*	24
RECLP_NR_039				N*	24
RECLP_NR_040				N*	24
RECLP_NR_041				N*	24

RECLP_NR_042				N*	24
RECLP_NR_043				N*	24
RECLP_NR_044				N*	24
RECLP_NR_045				N*	24
RECLP_NR_046				N*	24
RECLP_NR_047				N*	24
RECLP_NR_048				N*	24
RECLP_NR_049				N*	24
RECLP_NR_050				N*	24
RECLP_NR_051				N*	24
RECLP_NR_052				N*	24
RECLP_NR_053				N*	24
RECLP_NR_054				N*	24
RECLP_NR_055				N*	24
RECLP_NR_056				N*	24
RECLP_NR_057				N*	24
RECLP_NR_058				N*	24
RECLP_NR_059				N*	24
RECLP_RO_060				N*	24
RECLP_RO_061				N*	24
RECLP_RO_062				N*	24
RECLP_RO_063				N*	24
RECLP_NR_064				N*	24
RECLP_NR_065				N*	24
RECLP_NR_066				N*	24
RECLP_NR_067				N*	24
RECLP_NR_068				N*	24
RECLP_NR_001	1/13/95	12/21/94	12/21/94	N	
RECLP_NR_002				N	
RECLP_NR_003				N	
RECLP_NR_004				N	
RECLP_NR_005				N	
RECLP_NR_006				N	
RECLP_NR_007				N	
RECLP_NR_008				N	
RECLP_NR_009				N	
RECLP_NR_010				N	
RECLP_NR_011				N	
RECLP_NR_012				N	
RECLP_NR_013				N	
RECLP_NR_014				N	
RECLP_NR_015				N	
RECLP_NR_016				N	
RECLP_NR_017				N	
RECLP_NR_018				N	
RECLP_NR_019				N	
RECLP_NR_020				N	
RECLP_NR_021				N	

RECLP_NR_022				N	
RECLP_NR_023				N	
RECLP_NR_024				N	
RECLP_NR_025				N	
RECLP_NR_026				N	
RECLP_NR_027				N	
RECLP_NR_028				N	
RECLP_NR_029				N	
RECLP_NR_030				N	
RECLP_NR_031				N	
RECLP_NR_032				N	
RECLP_NR_033				N	
RECLP_NR_034				N	
RECLP_NR_035				N	
RECLP_NR_036				N	
RECLP_NR_037				N	
RECLP_NR_038				N	
RECLP_NR_039				N	
RECLP_NR_040				N	
RECLP_NR_041				N	
RECLP_NR_042				N	
RECLP_NR_043				N	
RECLP_NR_044				N	
RECLP_NR_045				N	
RECLP_NR_046				N	
RECLP_NR_047				N	
RECLP_NR_048				N	
RECLP_NR_049				N	
RECLP_NR_050				N	
RECLP_NR_051				N	
RECLP_NR_052				N	
RECLP_NR_053				N	
RECLP_NR_054				N	
RECLP_NR_055				N	
RECLP_NR_056				N	
RECLP_NR_057				N	
RECLP_NR_058				N	
RECLP_NR_059				N	
RECLP_RO_060				N	
RECLP_RO_061				N	
RECLP_RO_062				N	
RECLP_RO_063				N	
RECLP_NR_064				N	
RECLP_NR_065				N	
RECLP_NR_066				N	
RECLP_NR_067				N	
RECLP_NR_068				N	
RECLP_NR_001	4/7/95	4/6/95	4/7/95	N	

RECLP_NR_002				N	
RECLP_NR_003				N	
RECLP_NR_004				N	
RECLP_NR_005				N	
RECLP_NR_006				N	
RECLP_NR_007				N	
RECLP_NR_008				N	
RECLP_NR_009				N	
RECLP_NR_010				N	
RECLP_NR_011				N	
RECLP_NR_012				N	
RECLP_NR_013				N	
RECLP_NR_014				N	
RECLP_NR_015				N	
RECLP_NR_016				N	
RECLP_NR_017				N	
RECLP_NR_018				N	
RECLP_NR_019				N	
RECLP_NR_020				N	
RECLP_NR_021				N	
RECLP_NR_022				N	
RECLP_NR_023				N	
RECLP_NR_024				N	
RECLP_NR_025				N	
RECLP_NR_026				N	
RECLP_NR_027				N	
RECLP_NR_028				N	
RECLP_NR_029				N	
RECLP_NR_030				N	
RECLP_NR_031				N	
RECLP_NR_032				N	
RECLP_NR_033				N	
RECLP_NR_034				N	
RECLP_NR_035				N	
RECLP_NR_036				N	
RECLP_NR_037				N	
RECLP_NR_038				N	
RECLP_NR_039				N	
RECLP_NR_040				N	
RECLP_NR_041				N	
RECLP_NR_042				N	
RECLP_NR_043				N	
RECLP_NR_044				N	
RECLP_NR_045				N	
RECLP_NR_046				N	
RECLP_NR_047				N	
RECLP_NR_048				N	
RECLP_NR_049				N	

RECLP_NR_050				N	
RECLP_NR_051				N	
RECLP_NR_052				N	
RECLP_NR_053				N	
RECLP_NR_054				N	
RECLP_NR_055				N	
RECLP_NR_056				N	
RECLP_NR_057				N	
RECLP_NR_058				N	
RECLP_NR_059				N	
RECLP_RO_060				N	
RECLP_RO_061				N	
RECLP_RO_062				N	
RECLP_RO_063				N	
RECLP_NR_064				N	
RECLP_NR_065				N	
RECLP_NR_066				N	
RECLP_NR_067				N	
RECLP_NR_068				N	

* Even though an analysis file (.ANA) was not generated for these test cases, the limits checking prints messages to the screen for values of THETA that are in bounds. This indicates an error in the bounds checking code. Further observations revealed that the upper and lower bounds constants were reversed in the CONSTANTS.FOR file. The test cases were re-executed after this is corrected. Note that neither the RECLP code or the test cases had to be refetched. However, the CONSTANTS.FOR file was refetched and the code was recompiled.

D.9 Pluto Test Case Results Log for TDLRSP

TEST CASE NAME	EXECUTION DATE	DATE CODE FETCHED	DATE TEST CASE FETCHED	RESULTS (was .ANA file generated Y or N?)	PR #
TDLRSP_NR_001.TC	1/4/95	12/21/94	12/21/94	N	
TDLRSP_RO_002.TC				N	
TDLRSP_NR_003.TC				N	
TDLRSP_RO_004.TC				N	
TDLRSP_NR_005.TC				N	
TDLRSP_RO_006.TC				N	
TDLRSP_NR_007.TC				N	
TDLRSP_NR_008.TC				N	
TDLRSP_NR_009.TC				N	
TDLRSP_NR_010.TC				N	
TDLRSP_NR_011.TC				N	
TDLRSP_NR_012.TC				N	
TDLRSP_NR_013.TC				N	
TDLRSP_NR_014.TC				N	
TDLRSP_NR_015.TC				N	
TDLRSP_NR_016.TC				N	
TDLRSP_NR_017.TC				N	
TDLRSP_NR_018.TC				N	
TDLRSP_NR_019.TC				N	
TDLRSP_NR_020.TC				N	
TDLRSP_NR_021.TC				N	
TDLRSP_RO_022.TC				N	
TDLRSP_RO_023.TC				N	
TDLRSP_RO_024.TC				N	
TDLRSP_RO_025.TC				N	
TDLRSP_RO_026.TC				Y	24
TDLRSP_RO_027.TC				N	
TDLRSP_RO_028.TC				N	
TDLRSP_NR_001.TC	1/13/95	1/13/95	12/21/94	N	
TDLRSP_RO_002.TC				N	
TDLRSP_NR_003.TC				N	
TDLRSP_RO_004.TC				N	
TDLRSP_NR_005.TC				N	
TDLRSP_RO_006.TC				N	
TDLRSP_NR_007.TC				N	
TDLRSP_NR_008.TC				N	
TDLRSP_NR_009.TC				N	
TDLRSP_NR_010.TC				N	

TDLRSP_NR_011.TC				N	
TDLRSP_NR_012.TC				N	
TDLRSP_NR_013.TC				N	
TDLRSP_NR_014.TC				N	
TDLRSP_NR_015.TC				N	
TDLRSP_NR_016.TC				N	
TDLRSP_NR_017.TC				N	
TDLRSP_NR_018.TC				N	
TDLRSP_NR_019.TC				N	
TDLRSP_NR_020.TC				N	
TDLRSP_NR_021.TC				N	
TDLRSP_RO_022.TC				N	
TDLRSP_RO_023.TC				N	
TDLRSP_RO_024.TC				N	
TDLRSP_RO_025.TC				N	
TDLRSP_RO_026.TC				Y*	
TDLRSP_RO_027.TC				N	
TDLRSP_RO_028.TC				N	
TDLRSP_NR_001.TC	4/7/95	4/6/95	4/6/95	N	
TDLRSP_RO_002.TC				N	
TDLRSP_NR_003.TC				N	
TDLRSP_RO_004.TC				N	
TDLRSP_NR_005.TC				N	
TDLRSP_RO_006.TC				N	
TDLRSP_NR_007.TC				N	
TDLRSP_NR_008.TC				N	
TDLRSP_NR_009.TC				N	
TDLRSP_NR_010.TC				N	
TDLRSP_NR_011.TC				N	
TDLRSP_NR_012.TC				N	
TDLRSP_NR_013.TC				N	
TDLRSP_NR_014.TC				N	
TDLRSP_NR_015.TC				N	
TDLRSP_NR_016.TC				N	
TDLRSP_NR_017.TC				N	
TDLRSP_NR_018.TC				N	
TDLRSP_NR_019.TC				N	
TDLRSP_NR_020.TC				N	
TDLRSP_NR_021.TC				N	
TDLRSP_RO_022.TC				N	
TDLRSP_RO_023.TC				N	
TDLRSP_RO_024.TC				N	
TDLRSP_RO_025.TC				N	
TDLRSP_RO_026.TC				Y*	

TDLRSP_RO_027.TC				N	
TDLRSP_RO_028.TC				N	

*: The ANA file generated in this iteration of testing involves a condition that is not specified in the SPEC. Although the results of this test run does not agree with the expected values, the results are just as valid because this robustness test case exercises a condition that is not defined in the Specification. More specifically, a value of "2" is assigned to the variable TDLR_STATE. Although a "2" is not defined as a legal value for this variable in the GCS Spec, it is a possible value since the variable is ultimately implemented as an integer. For robustness test cases, DO-178B requires only that the software not cause any detrimental effects to the system. For this specific test case, the PLUTO code leaves the values of K_MATRIX unchanged. This will not have a severe impact on the implementation's ability to deliver the required function for TDLRSP.

D.10 Pluto Test Case Results Log for TDSP

TEST CASE NAME	EXECUTION DATE	DATE CODE FETCHED	DATE TEST CASE FETCHED	RESULTS (was .ANA file generated Y or N?)	PR #
TDSP_NR_001.TC	1/4/95	12/21/94	12/21/94	N	
TDSP_NR_002.TC				N	
TDSP_NR_003.TC				N	
TDSP_RO_004.TC				N	
TDSP_RO_005.TC				N	
TDSP_RO_006.TC				N	
TDSP_RO_007.TC				N	
TDSP_NR_001.TC	1/17/95	12/21/94	12/21/94	N*	
TDSP_NR_002.TC				N*	
TDSP_NR_003.TC				N*	
TDSP_RO_004.TC				N*	
TDSP_RO_005.TC				N*	
TDSP_RO_006.TC				N*	
TDSP_RO_007.TC				N*	
TDSP_NR_001.TC	4/7/95	4/6/95	4/6/95	N	
TDSP_NR_002.TC				N	
TDSP_NR_003.TC				N	
TDSP_RO_004.TC				N	
TDSP_RO_005.TC				N	
TDSP_RO_006.TC				N	
TDSP_RO_007.TC				N	

*: These test cases had to be re-executed because the include file CONSTANTS.FOR was changed in PR#24.

D.11 Pluto Test Case Results Log for TSP

TEST CASE NAME	EXECUTION DATE	DATE CODE FETCHED	DATE TEST CASE FETCHED	RESULTS (was .ANA file generated Y or N?)	PR #
TSP_NR_001.TC	1/4/95	12/21/94	12/21/94	N	
TSP_NR_002.TC				N	
TSP_NR_003.TC				N	
TSP_RO_004.TC				N	
TSP_RO_005.TC				N	
TSP_NR_006.TC				Y	24
TSP_NR_007.TC				Y	24
TSP_RO_008.TC				Y	24
TSP_RO_009.TC				Y	24
TSP_RO_010.TC				Y	24
TSP_RO_011.TC				Y	24
TSP_NR_001.TC	1/13/95	1/13/95	12/21/94	N	
TSP_NR_002.TC				N	
TSP_NR_003.TC				N	
TSP_RO_004.TC				N	
TSP_RO_005.TC				N	
TSP_NR_006.TC				N	
TSP_NR_007.TC				N	
TSP_RO_008.TC				N	
TSP_RO_009.TC				N	
TSP_RO_010.TC				N	
TSP_RO_011.TC				Y*	
TSP_NR_001.TC	4/7/95	4/6/95	4/6/95	N	
TSP_NR_002.TC				N	
TSP_NR_003.TC				N	
TSP_RO_004.TC				N	
TSP_RO_005.TC				N	
TSP_NR_006.TC				N	
TSP_NR_007.TC				N	
TSP_RO_008.TC				N	
TSP_RO_009.TC				N	
TSP_RO_010.TC				N	
TSP_RO_011.TC				Y*	

*: For this robustness test case, the difference flagged by the ANA file is in the 14th digit of ATMOSPHERIC_TEMP. This amounts to a relative error less than that required by the simulator.

D.12 Pluto Test Case Results Log for SP Subframe

TEST CASE NAME	EXECUTION DATE	DATE CODE FETCHED	DATE TEST CASE FETCHED	RESULTS (was .ANA file generated Y or N?)	PR #
SP_001	3/6/95	1/13/95	3/2/95	N	
SP_001	4/7/95	4/6/95	4/7/95	N	

D.13 Pluto Test Case Results Log for GP Subframe

TEST CASE NAME	EXECUTION DATE	DATE CODE FETCHED	DATE TEST CASE FETCHED	RESULTS (was .ANA file generated Y or N?)	PR #
GPSF_001.	3/6/95	1/13/95	3/2/95	N	
GPSF_002.				N	
GPSF_003.				N	
GPSF_004.				N	
GPSF_005				N	
GPSF_006				N	
GPSF_007				N	
GPSF_008.				N	
GPSF_001.	4/7/95	4/6/95	4/7/95	N	
GPSF_002.				N	
GPSF_003.				N	
GPSF_004.				N	
GPSF_005				N	
GPSF_006				N	
GPSF_007				N	
GPSF_008.				N	

D.14 Pluto Test Case Results Log for CLP Subframe

TEST CASE NAME	EXECUTION DATE	DATE CODE FETCHED	DATE TEST CASE FETCHED	RESULTS (was .ANA file generated Y or N?)	PR #
CLP_001	3/6/95	1/13/95	3/2/95	N	
CLP_002				N	
CLP_003				N	
CLP_004				N	
CLP_005				N	
CLP_006				N	
CLP_007				N	
CLP_008				N	
CLP_009				N	
CLP_010				N	
CLP_011				N	
CLP_012				N	
CLP_013				N	
CLP_014				N	
CLP_001	4/7/95	4/6/95	4/7/95	N	
CLP_002				N	
CLP_003				N	
CLP_004				N	
CLP_005				N	
CLP_006				N	
CLP_007				N	
CLP_008				N	
CLP_009				N	
CLP_010				N	
CLP_011				N	
CLP_012				N	
CLP_013				N	
CLP_014				N	

D.15 Pluto Test Case Results Log for FRAME

TEST CASE NAME	EXECUTION DATE	DATE CODE FETCHED	DATE TEST CASE FETCHED	RESULTS (was .ANA file generated Y or N?)	PR #
FRAME_001	3/6/95	1/13/95	3/2/95	N	
FRAME_002					
FRAME_003					
FRAME_004					
FRAME_005					
FRAME_006					
FRAME_007					
FRAME_008					
FRAME_009					
FRAME_001	4/7/95	4/6/95	4/7/95	N	
FRAME_002					
FRAME_003					
FRAME_004					
FRAME_005					
FRAME_006					
FRAME_007					
FRAME_008					
FRAME_009					
FRAME_009					

D.16 Pluto Test Case Results Log for Trajectory

TEST CASE NAME	EXECUTION DATE	DATE CODE FETCHED	DATE TEST CASE FETCHED	RESULTS MATCHED EXPECTED FRAMES	GP_PHASE = 5	PR #
TRAJ_ATM_UD/IC_001	3/6/95	3/6/95	3/6/95	Y	Y	
TRAJ_ATM_UD/IC_002				Y	Y	
TRAJ_ATM_UD/IC_003				Y	Y	
TRAJ_ATM_UD/IC_004				Y	Y	
TRAJ_ATM_UD/IC_005				Y	Y	
TRAJ_ATM_UD/IC_006				Y	Y	
TRAJ_ATM_UD/IC_007				Y	Y	
TRAJ_ATM_UD/IC_008				Y	Y	
TRAJ_ATM_UD/IC_009				Y	Y	
TRAJ_ATM_UD/IC_010				Y	Y	
TRAJ_ATM_UD/IC_011				Y	Y	
TRAJ_ATM_UD/IC_012				Y	Y	
TRAJ_TD_UD/IC_013				Y	Y	
TRAJ_TD_UD/IC_014				Y	Y	
TRAJ_TD_UD/IC_015				Y	Y	
TRAJ_TD_UD/IC_016				Y	Y	
TRAJ_TD_UD/IC_017				Y	Y	
TRAJ_TD_UD/IC_018				Y	Y	
TRAJ_TD_UD/IC_019				N	Y	27
TRAJ_TD_UD/IC_020				Y	Y	
TRAJ_TD_UD/IC_021				Y	3	27
TRAJ_TD_UD/IC_022				Y	Y	
TRAJ_TD_UD/IC_023				Y	Y	
TRAJ_TD_UD/IC_024				Y	Y	
TRAJ_TD_UD/IC_025				Y	Y	
TRAJ_TD_UD/IC_026				Y	Y	
TRAJ_TD_UD/IC_027				Y	Y	
TRAJ_TD_UD/IC_028				Y	Y	
TRAJ_TD_UD/IC_029				Y	Y	
TRAJ_TD_UD/IC_030				Y	Y	
TRAJ_TD_UD/IC_031				Y	Y	
TRAJ_TD_UD/IC_032				Y	Y	
TRAJ_TD_UD/IC_033				Y	Y	
TRAJ_TD_UD/IC_034				Y	Y	
TRAJ_ATM_UD/IC_001	4/7/95	4/6/95	4/7/95	Y	Y	
TRAJ_ATM_UD/IC_002				Y	Y	
TRAJ_ATM_UD/IC_003				Y	Y	
TRAJ_ATM_UD/IC_004				Y	Y	
TRAJ_ATM_UD/IC_005				Y	Y	
TRAJ_ATM_UD/IC_006				Y	Y	
TRAJ_ATM_UD/IC_007				Y	Y	

TRAJ_ATM_UD/IC_008				Y	Y	
TRAJ_ATM_UD/IC_009				Y	Y	
TRAJ_ATM_UD/IC_010				Y	Y	
TRAJ_ATM_UD/IC_011				Y	Y	
TRAJ_ATM_UD/IC_012				Y	Y	
TRAJ_TD_UD/IC_013				Y	Y	
TRAJ_TD_UD/IC_014				Y	Y	
TRAJ_TD_UD/IC_015				Y	Y	
TRAJ_TD_UD/IC_016				Y	Y	
TRAJ_TD_UD/IC_017				Y	Y	
TRAJ_TD_UD/IC_018				Y	Y	
TRAJ_TD_UD/IC_019				Y	Y	
TRAJ_TD_UD/IC_020				Y	Y	
TRAJ_TD_UD/IC_021				Y	Y	
TRAJ_TD_UD/IC_022				Y	Y	
TRAJ_TD_UD/IC_023				Y	Y	
TRAJ_TD_UD/IC_024				Y	Y	
TRAJ_TD_UD/IC_025				Y	Y	
TRAJ_TD_UD/IC_026				Y	Y	
TRAJ_TD_UD/IC_027				Y	Y	
TRAJ_TD_UD/IC_028				Y	Y	
TRAJ_TD_UD/IC_029				Y	Y	
TRAJ_TD_UD/IC_030				Y	Y	
TRAJ_TD_UD/IC_031				Y	Y	
TRAJ_TD_UD/IC_032				Y	Y	
TRAJ_TD_UD/IC_033				Y	Y	
TRAJ_TD_UD/IC_034				Y	Y	

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
01- 12 - 2008	Technical Memorandum	

4. TITLE AND SUBTITLE	**5a. CONTRACT NUMBER**
Guidance and Control Software Project Data - Volume 3: Verification Documents	
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	**5d. PROJECT NUMBER**
Hayhurst, Kelly J. (Editor)	
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
	457280.02.07.07.06.02

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	**8. PERFORMING ORGANIZATION REPORT NUMBER**
NASA Langley Research Center Hampton, VA 23681-2199	L-19550
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	**10. SPONSOR/MONITOR'S ACRONYM(S)**
National Aeronautics and Space Administration Washington, DC 20546-0001	NASA
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)
	NASA/TM-2008-215552

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified - Unlimited
Subject Category 61
Availability: NASA CASI (301) 621-0390

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The Guidance and Control Software (GCS) project was the last in a series of software reliability studies conducted at Langley Research Center between 1977 and 1994. The technical results of the GCS project were recorded after the experiment was completed. Some of the support documentation produced as part of the experiment, however, is serving an unexpected role far beyond its original project context. Some of the software used as part of the GCS project was developed to conform to the RTCA/DO-178B software standard, "Software Considerations in Airborne Systems and Equipment Certification," used in the civil aviation industry. That standard requires extensive documentation throughout the software development life cycle, including plans, software requirements, design and source code, verification cases and results, and configuration management and quality control data. The project documentation that includes this information is open for public scrutiny without the legal or safety implications associated with comparable data from an avionics manufacturer. This public availability has afforded an opportunity to use the GCS project documents for DO-178B training. This report provides a brief overview of the GCS project, describes the 4-volume set of documents and the role they are playing in training, and includes the verification documents from the GCS project.

15. SUBJECT TERMS
Software engineering; Computer programming; Software reliability; DO-178B

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			STI Help Desk (email: help@sti.nasa.gov)
					19b. TELEPHONE NUMBER ***(Include area code)***
U	U	U	UU	355	(301) 621-0390

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18