Apr 1st, 8:00 AM

# A Checkout Language for Future Space Vehicles

J. W. Meadlock
*President, M&S Computing, Inc. Huntsville, Alabama*

T. T. Schansman
*Vice President, M&S Computing, Inc. Hunts ville, Alabama*

R. E. Thurber
*System Consultant, M&S Computing, Inc. Hunts ville, Alabama*

# A CHECKOUT LANGUAGE FOR FUTURE SPACE VEHICLES

J. W. Meadlock
President
M&S Computing, Inc.
Huntsville, Alabama

T. T. Schansman
Vice President
M&S Computing, Inc.
Huntsville, Alabama

R. E. Thurber
System Consultant
M&S Computing, Inc.
Huntsville, Alabama

## ABSTRACT

To support an increased emphasis on automated checkout of future space vehicles, a procedure-oriented computer language is required. This language needs to be more user-oriented and needs to have a more complete set of capabilities than existing languages. Such a language, named TOTAL, was developed under contract to NASA-KSC. This paper presents an overall view of the language in terms of its major characteristics as derived from the basic design objectives.

## INTRODUCTION

### Why a New Language?

It is anticipated that future space vehicles will require an increased emphasis on automation of various types of testing at the launch site and during the actual mission. One of the major tools required to effectively support this automation is a procedure-oriented language, usable by engineering personnel, to describe test procedures to a computer. Such a language has as major design objectives:

- Easy to learn and use by engineers
- Easy to read by non-writers
- Capable to support future space vehicle checkout

M&S Computing reviewed existing, significant languages (see Bibliography) to determine if such a language was already available. Although several languages fulfilled the objectives to various degrees, none could be considered satisfactory. It was therefore necessary to perform a new language design. This new design is based on existing language designs, with modifications and additions, to fulfill the major design objectives and to include improvements, where experience has shown these to be necessary.

The language thus developed has been named TOTAL (Test engineer Oriented Test Application Language) and is the subject of this paper. The effort was performed under contract to the NASA-Kennedy Space Center.

### Some Language Highlights

There are many ways to characterize a language. The following are a selection of those characteristics considered of greatest interest to a potential user.

- Simple statement format and vocabulary
- Automatic English-like output for simplicity of reading
- Error prevention and detection (verification) aids
- "Built-in" procedure writing standards and conventions
- Language extension capability to adapt to local requirements or special situations
- Hardware configuration independence
- Usable in ground computer as well as on-board computers
- Designed for integrated, systems, subsystems, and component testing
- Simplified management/configuration control
- Not dependent on professional programmers' support
- Applicable to a timesharing environment
- Direct interface with major system services such as "monitoring" and "logging"
- Full man/machine interface capabilities

### Purpose of this Paper

The space in this paper does not allow a detailed description of the language. Our attempt here is to present an overall view of the language design, and to show how this design was derived from the major language design objectives. A language design is only meaningful if the underlying philosophy is understood.

We first discuss the major design objectives and the reasons for these. We then discuss a selection of the major language characteristics and relate these to the design objectives. Reference material and a Bibliography are included at the end of this paper for readers who desire more detailed information.

## MAJOR DESIGN OBJECTIVES

We will first discuss the major design objectives and the reasons for their existence before we discuss their effect on the language design.

### Easy to Learn and Use by Engineers

The automation of an application is necessarily started through the appropriate use of programming personnel. As the automation effort matures, however, it is mandatory that the actual user has a more direct control over the automation effort. That is, for increased and efficient automation, the need for intermediaries between the actual user and his application should be minimized.

In our case this implies that the engineer should be able to directly communicate his test requirements to the checkout complex, rather than provide these to a programmer for implementation. The language should, therefore, be such that engineering personnel, without previous programmer training or experience, are able to familiarize themselves with it.

Note, however, that this does not mean that the user does not have to be aware of "automation concepts". It is neither possible nor desirable to hide the fact that a computer is used as a major tool in the automation process. The user has to understand and appreciate the capabilities and shortcomings of computers to the same degree that he is aware of other support equipment.

### Easy to Read by Non-Users

In the environment under study it is frequently necessary that the test procedures are perused by personnel not directly involved in the implementation of the test procedure.

It is a very costly task to train all potential language readers to the same depth as the procedure writers. In addition, the occasional reader can easily forget his training. It is, therefore, imperative that procedures specified in the language can be understood by the non-writer with an absolute minimum orientation.

### Capable to Support Future Space Vehicle Checkout

This, of course, is the prime objective. A major concern is that a single language can be used to specify most (if not all) test procedures used in a variety of testing situations. The idea is that testing personnel will only have to be familiar with one language. The basic language design should therefore first of all be applicable to most common testing situations. In addition, the language should allow a degree of extendability, by the intended user, to adapt to unexpected or specialized testing situations. Consideration need also be given to expected testing techniques and methods. These are imposed not only by the prime equipment test requirements, but also by system considerations.

## MAJOR LANGUAGE CHARACTERISTICS

The analysis of the major design objectives resulted in various language design characteristics that form the basis of the language design. There are, of course, numerous characteristics that were derived from the major (or minor) design objectives. It is clearly impossible to describe all of these in the space of this paper. We will therefore only describe those characteristics that are considered critical and that provide an overall view of the language design.

### Easy to Learn and Use by Engineers

A wide variety of factors influence the effort required by the procedure writer to accomplish his task. The factors that most significantly influenced the language design are summarized below.

#### Input Statements

Simplification of the effort required to describe the test procedure is a prime concern. To accomplish this, the format of the input statements, as well as the vocabulary used, has to be considered. A statement format, flexible enough to describe the necessary actions, but simple enough to accomplish our objective, is the Fixed-Length, Fixed-Assignment, Field Format shown at the top of Figure 1.

This is a form divided in columns of fixed widths. Each column has a heading, describing the type of information to be entered in that column. Therefore, once a user knows what information is required to describe an action (usually self-evident), he immediately can form the statement. There is, therefore, a minimal need to remember various statement formation rules.

Minimal Vocabulary

By keeping the number of words in the language vocabulary to a minimum, an additional simplification is accomplished. Note that the format previously described aids in this effort. No words are needed to indicate separation between statement entries or to indicate sequencing of statement entries. The main vocabulary needed is therefore the action to be described in the "Operation" field. The most common words used in the "Operation" field are shown in Figure 2 (see Figure 1 for examples). An operation can consist of one to three words. The first word is the "verb", the main action to be performed. The second word is the "object", the class of items to which the verb is to be applied. The third word is a "condition" or "modifier" that is occasionally used to further refine the description of the action to be taken. Abbreviations can be used by simply combining the first two characters of each word to be entered in the "Operation" field.

Design Conventions

Prior to the design and implementation of a test procedure in a specific language, it is necessary to define a set of groundrules that specify how the language is to be applied. For example, it is necessary to define labeling or naming conventions, to partition use of computer memory, to assign use of decision switches, etc., such that the writers use reasonably similar conventions to improve intercommunication and readability.

This is necessary because most programming languages are "generalized" to cover a wide, and frequently unpredictable, variety of applications. However, when a language is designed for a very similar set of applications, it is much more useful to design the most commonly required conventions into the language. This aids both the procedure writers and their management in minimizing the time necessary to define and document the required design conventions.

The language design includes design conventions for the following items:

- Predefined item names/labels
- Procedure structuring rules
- Use of internal/external storage
- Communication between Test Procedure Modules (interface rules)
- Use of decision switches

Minimal Programmer Support

To effectively increase the use of automation, it is mandatory that the personnel involved are minimized. That is, the test procedure writer should be able to specify his requirements directly in a format interpretable by the computer, rather than through a programmer. This, in effect, implies two major factors.

It is first of all necessary to absolutely limit the use of "Assembly Language Programming" to emergencies. That is, all (presently predictable) functions should be able to be specified by the engineer. Assembly Language should only be used for unanticipated requirements and, when used, should be clearly separated from the other procedure statements.

Second, it is necessary to absolutely limit "configuration dependent knowledge" in the system support software; that is, information which is subject to change as test methods change or as prime or support equipment changes. For example, test point dependent data is captured in a central Test Point Characteristics File (described in Figure 3), external to the system software. This information is standard information, so that it should not have to be written down each time that it is required. However, it is subject to changes that should be under control of test engineering personnel, so it should not be an integral part of the compiler and/or operating system. Another item is the provision for User Defined Operators. These allow user-defined extensions to the basic signals spectrum, if required, without modifications to the language compiler. Both of these last items are discussed in more detail later on in this paper.

The above summarizes the main language features provided to make the language as easy to learn and use as possible.

Easy to Read by Non-Writers

The statement format and terminology previously described are very easy to read by personnel that are regularly using the language. However, personnel that are not involved in the writing of the test procedures, but mainly peruse the test procedures for various reasons, would have some difficulty reading the raw input statements. Of course, this is a problem that is common to all existing programming languages.

The most common solution is to have the writer write in some sort of pidgin English that makes the statement somewhat easier to read. That approach has been applied with varying degrees of success. It sometimes favors the writer and sometimes favors the reader. In the environment we are discussing here, the reader has less use for a detailed knowledge of the language than

for most other languages. It was therefore decided that it would be more effective to optimize writeability and readability (for the non-writer) as largely independent items.

To make the raw input statements readable to personnel with no or limited language training, the compiler expands the input statements into brief English sentences. This "Expanded Listing" therefore becomes available as soon as the test procedure starts becoming operational. Note that no special burden is imposed upon any of the users. Figure 1 illustrates some typical input statements and their corresponding Expanded Listing.

Capable to Support Future Space Vehicle Checkout

To provide a rational basis for the definition of language design requirements, it was necessary to identify the major characteristics of the environment to which the checkout language will be applied. From the many vehicles under consideration for future missions, the avionics systems of the Space Shuttle and Space Station were identified as representing the most pertinent checkout language requirements. The definition of the checkout environment was complicated considerably by the wide variety of concepts and groundrules under consideration for these systems. From these different approaches, the major features were abstracted and analyzed for potential impact on a checkout language. From these features, it was possible to establish a baseline of checkout activities and a checkout system configuration. The following discussion explains the major characteristics of the language which were derived from these features.

The baseline checkout system configuration depicted in Figure 4 reflects the major pertinent features of potential vehicle checkout systems which might impact a checkout language. These impacts are primarily through the computer configuration and the data distribution system which transports signals among vehicle components. The baseline checkout system includes a selectively decentralized computer configuration with a central data management computer and additional computers dedicated to some of the subsystems. It also includes a separate ground computer to control ground support equipment and to control or support on-board checkout functions during mission phases on the ground. A multiplexed digital data bus interconnects these computers with other on-board components, and a standard Data Bus Interface (DBI) attaches devices to the data bus.

Signal Spectrum

One of the most fundamental characteristics of a checkout language is the spectrum of signals to and from the system under test (SUT) which can be represented. An integral part of that characteristic is the way in which they are represented. The key to TOTAL's signal spectrum is the spectrum of the DBI's of Figure 4. Across the many approaches to checkout of the Space Shuttle and Space Station, the concept of the multiplexed digital data bus and its standard signal interface is very consistent. The DBI converts both ways between serial digital (data bus) data and:

- o Discrete (On/Off) Signals
- o DC Analog Voltages
- o Digital Words

To generate a particular DBI output to a system under test (SUT) test point, the computer must supply a "data bus address" which identifies a particular output, and a data word which supplies the value of the standard signal to be generated at that output. Conversely, a computer input from an SUT test point consists of an identifying address and the value of the standard signal at the test point. This data bus message, consisting of an address and a value, is the lowest level of communication between the computer and the SUT. It is also the most frequently described level, since one DBI signal generally results in the performance of a complete test function, such as applying a specific stimulus or measuring a specific physical parameter. These standard interface signals, through which all communication with the SUT must pass, form the "primary interface" which is directly described in TOTAL. That is, the engineer may write test statements which explicitly identify discretes, DC analog voltages, and digital words. TOTAL also recognizes the existence of more specialized "intermediate interface" signal types and provides an additional, more specialized, capability to describe them. That capability is presented after the following discussion of the primary interface signals.

Test Point Characteristics File

While the standard DBI signal forms are the means of conveying information to and from the SUT, they are not always the most useful form to the engineer. He is concerned with the temperatures, pressures, and positions which DC analog voltages represent, rather than the voltages themselves. Furthermore, he is familiar with the nomenclature of a measurement test point on a pump or actuator, rather than a data bus address. Concepts such as data bus addresses and

transducer output voltages should not be forced into the language of the engineer. Neither should their corresponding test point identifications and engineering values of physical parameters be programmed into a compiler or operating system. An effective compromise to these two extremes is TOTAL's Test Point Characteristics File, through which translations can be made between the engineer's test point identification nomenclature and the corresponding data bus addresses. The one-to-one correspondence between individual SUT test points and unique data bus addresses is particularly suited to description through such a data file. As indicated in Figure 3, the Test Point Characteristics File also contains calibration and conversion data to translate between Data Bus Interface signal values and the engineering units of the physical parameters they represent. This translation allows the engineer to describe his test steps with the physical units and ranges peculiar to each test point. It also makes the written test procedure independent of transducer replacement and recalibration. Only the Test Point Characteristics File needs to be updated for these changes.

Other pertinent data which is unique to individual test points is contained in this file. These additional items are nominal values for certain signal characteristics, so the test statements need not include them explicitly. In most applications, the engineer will be able to rely on these nominal values for his tests. For the exceptions, TOTAL allows him to override the nominals by specifying other values in the test sequence statements.

The Test Point Characteristics File contains a translation for every test point at the primary signal interface. It is defined so that it can be accessed by the compiler and/or operating system, but still be separate from them so it can be easily and independently maintained.

In a bench test environment, there are potentially three signal interfaces which can be described through a Test Point Characteristics File. These are depicted in Figure 5, which represents a typical bench test complex. The first interface is the Unit Under Test (UTT) interface which relates the internal test point of the UUT to the connector plugs on the test complex. The second is the patchboard interface which relates the connector plug pins to the stimulus generating and response measuring devices in the complex. Some of these interconnections are typically through program controlled routing switches, so test equipment devices can be switched from one UUT test point to another during a test sequence. Therefore, the patchboard interface file must identify the routing switch and switch state required to complete each connection. The third interface is the test equipment interface with the control computer. This file defines the computer commands and data formats required to communicate with the stimulus/measurement devices and the routing switches. These three Test Point Characteristics Files describe the interfaces which change for different UUT's, different tests, and test equipment device replacement or modification. These files will contain somewhat different information from that described for the integrated vehicle checkout system, but they perform the same basic function of isolating one-to-one translations which are specialized and subject to change.

User Defined Operators

For most of the devices attached to the data bus in the integrated vehicle checkout system, each "primary interface" operation is a complete test function. However, there are likely to be some special signal generating and monitoring devices to provide more than discrete, DC voltage, and digital word communication with the SUT. These special devices form an "intermediate interface" between the data bus and the SUT as shown in Figure 6. It is anticipated that the signal characteristics of these intermediate devices will not be standardized, but will be adapted to the unique requirements of the SUT's for which they are provided. Therefore, the intermediate devices will be more specialized to individual SUT's and will require a more complex manipulation at the primary interface to activate a single stimulus or measurement at an intermediate test point. The incorporation of this type of intelligence into a compiler or operating system would put the burden of understanding specialized equipment back on the professional programmer and tends to defeat the purpose of a checkout language.

The intermediate interface concept is exemplified by the digital ramp generators currently used during prelaunch checkout of the Saturn V Flight Control System. These ramp generators are shown in Figure 7, including their interfaces with the Saturn IU Stage and the Saturn Ground Computer Complex (SGCC). The primary interface shown includes discrete (on/off) stimuli out of the SGCC and measurement inputs to the SGCC through the Digital Data Acquisition System (DDAS). The purpose of the ramp generators is to supply simulated vehicle rates to nine rate gyros in the IU Stage. Since the SGCC cannot provide the needed signals directly, intermediate devices are required in the form of three digital ramp generators. The corresponding intermediate interface consists of nine rate simulation signals, three out of each ramp generator and one into each rate gyro. Discrete stimuli at the primary interface control the ramp generators. One group of three discretes is used for ramp generator selection and another group of

three selects vehicle axes. Identification of one intermediate interface signal requires that one discrete in each of these groups be on and the others be off. The direction and ramp rate of the selected intermediate signal are then established with the ramp control discretes. The resulting simulated rate detected by the rate gyro can be read from the DDAS measurement of the rate gyro output. In summary, the generation of a single stimulus value at a selected rate gyro input requires the manipulation of eight discretes and monitoring of one measurement at the primary interface.

While this level of test equipment dependent intelligence should not be programmed into a compiler or operating system, neither should the engineer have to specify such details every time he wishes to stimulate a gyro. To avoid burdening the professional programmer with this type of logic, but still simplify the writing task of the engineer, TOTAL includes the concept of a User Defined Operator. This capability allows the engineer to write the sequence of primary interface operations required to generate an intermediate stimulus or take a measurement, and to assign a single new language operator to that sequence. Such a User Defined Operator can be defined once and can then be used in the same manner as the basic operators of the language.

To apply the User Defined Operator concept to the ramp generators described above, the general logic sequence is first determined. To select a specific rate gyro input signal, two selection discretes must be turned on and four turned off. Then, for example, a RAMP UP operator would require that the "ramp positive" discrete be turned on until the desired simulated vehicle rate, read from one of nine DDAS measurements, is reached. The "ramp positive" discrete is turned off to complete the operation. To apply the RAMP UP operation to any single rate gyro input, the same sequence of logic is performed. The only changes are in identification of the individual primary interface points to be monitored or turned on and off, and the final rate to be achieved.

The preceding sequence is typical of the type of intermediate interface manipulation anticipated for future space vehicles. A User Defined Operator therefore defines a fixed sequence of operations with variable test point identifications and values. The definition includes naming the intermediate interface signals to which it may be applied. Each intermediate interface signal is defined in terms of the primary interface signals which control or monitor it, and these primary signals names are substituted into the defined sequence when it is compiled. This capability to extend and specialize the language has been specifically designed so that an engineer may define his own operators without reliance upon a professional programmer to understand his equipment. It constitutes a user oriented language extension and adaptation capability at the signal interface level, where the most extensive flexibility and change activity in the hardware can be expected.

For vehicle checkout, an individual DBI signal is expected to perform a complete test function, in most cases. Therefore, the application of User Defined Operators is the exception rather than the rule. In a bench test complex, however, it is common that several test equipment interface signals are required to generate a single LRU interface signal. In this environment, the User Defined Operator is an essential tool to avoid a very detailed test writing effort on the part of the test engineer.

System Services

A checkout system contains various hardware and software items that cooperatively perform the testing. The purpose of the language is to communicate with and direct the system. It is therefore necessary to design the language such that it can indeed communicate with the system. At this stage of development, the total checkout system is not well-defined. However, based on current approaches to equipment configurations and checkout activities, there are significant system services which are clearly required. Various characteristics of TOTAL are therefore based on the required interaction with these system services. Some of these are described below.

System Monitor

Performance monitoring is an essential ingredient in present day launch vehicle checkout and it promises to be an even more significant activity in the future. Performance monitoring is accomplished by reading vehicle parameters or combinations of parameters and evaluating them relative to some pre-established standard. This is "passive" testing in the sense that no stimuli are applied to generate parameter values; the natural environment provides the stimuli. In addition to immediate evaluation of parameter values, performance monitoring includes displaying the data and storing it for future reference.

To support performance monitoring, the language provides control over the monitoring of vehicle test points for extended periods of time. This language capability is predicated on the existence of a System Monitor, provided as part of the checkout system to sample and test measurements repeatedly over an extended period of time. The extensive use of performance monitoring

projected for future space vehicles indicates that the following capabilities will be required of a System Monitor:

- Test the values of test points against fixed or sliding limits.
- Logically combine the results of individual tests to generate responses to multiple test point conditions.
- Display the values of test points being monitored and identify out of tolerance values.
- Respond to out of tolerance conditions by one of the following:
  - Display a message.
  - Initiate execution of a defined command/response test sequence to further diagnose or resolve the condition.
  - Modify test limits and sampling rates on other monitored points.
  - Initiate or cancel other monitor activities.

These capabilities and restrictions are based on a general purpose service which is easy to use. Because it will be used so extensively, the service should be provided centrally for ease of use and efficiency in execution. These capabilities are expected to be used concurrently with the execution of command/response test sequences which issue stimuli and verify expected responses. TOTAL includes operators dedicated to the definition of test conditions and responses for the System Monitor, as well as to control their activation and de-activation.

In addition to the concurrent monitoring through the System Monitor, the language also provides for monitoring a measurement at a specified sample rate within a test sequence. Through this "sequential read" capability the test engineer may specify that a certain number of samples of a measurement be taken and stored in computer memory for analysis. Mass Storage Files can be defined in memory to easily store and retrieve these readings. Mass Storage Files can also be defined on peripheral storage devices such as magnetic tapes or disk so that large quantities of data can be stored outside computer memory and still be easily accessed by the program.

Closely associated with the monitoring of test points is the logging of events and test point values for later reference. Using a peripheral Mass Storage File, the test engineer may store any data which his test sequence can access. In addition to this capability, the checkout system itself will automatically log a great deal of data for an activity record and audit trail without any explicit command from a test sequence. Since such fully standardized automatic logging systems frequently record too much or too little information, TOTAL increases the utility of the logging system to the test engineer by providing him some control over what is logged. The following information is logged at the request of a test sequence:

- Individually selected data items which are stored in computer memory by the test sequence
- All measurement values read by the test sequence
- Start of execution of all test sequence modules called in the test sequence

This list is very limited to prevent an individual test sequence from overly restricting the permanent record of checkout activities.

Timesharing

One of the important considerations in the projected Space Shuttle mission time line is its short ground turnaround time. This requirement may force testing and preflight preparation of various vehicle subsystems to be accomplished in parallel. A "system service" that TOTAL has to interface with is the ability of the system to run more than one test sequence in a timeshared mode. Although the fact that the procedure is executed in a timeshared mode is generally transparent to the procedure writer, there are instances that he must be aware of possible difficulties imposed by this mode of execution. The language design, therefore, provides the following required features:

- Procedure structure that allows portions to be executed in a timeshared mode
- The ability to command timeshared execution of test procedures
- The ability to prevent test steps from being timeshared (i.e., interrupted) during time critical portions of a test procedure
- Aids to identify potential conflicts in the use of common test points by concurrently executing test sequences.

CONCLUSION

As should be clear from the preceding discussion,

it was not our purpose to design a novel language. Our intent was to build upon existing, proven, concepts and experience. We added slightly novel features and expanded existing features, to remove presently known shortcomings, and to insure the capability to handle future requirements.

We believe that the language, thus designed, provides a firm basis for effective use, and for possible expansion where actual experience so dictates.

REFERENCES


(1) "Development of a Test and Flight Engineer Oriented Computer Language, Phase I", M&S Computing, Report No. 70-0022, NAS10-7307.

(2) "Development of a Test and Flight Engineer Oriented Computer Language, Phase II", M&S Computing, Report No. 70-0031, NAS10-7307.

(3) "Development of a Test and Flight Engineer Oriented Computer Language, Final Report, Volumes I and II", M&S Computing, Report No. 70-0034, NAS10-7307.


BIBLIOGRAPHY


A. Future Space Vehicle Checkout

(1) "A Two-Stage Fixed Wing Space Transportation System, Vol. II, Preliminary Design", McDonnell Douglas Corporation, MDC E0056.

(2) "Space Shuttle Final Technical Report, Vol. VII, Integrated Electronics", General Dynamics Convair Division, GDC-DCB69-046.

(3) "Study of Integral Launch and Re-entry Vehicle Systems, Vol. IV, Design and Subsystems Analysis", North American Rockwell, SD 69-573-4.

(4) "A Conceptual Design of the Space Shuttle Integrated Avionics System", MSFC, NASA TMX-53987.

(5) "Final Report, Integral Launch and Re-entry Vehicle, Vol. III, Special Studies", Lockheed Missiles and Space Company, LMSC-A959837.

(6) "Information Management Study", The MITRE Corporation, MTR-1524.

(7) "Space Station Definition, Vol. V, Subsystems", McDonnell Douglas Corporation, MDC G0605.

(8) "NASA Space Shuttle - Vehicle Checkout Design Concept", M&S Computing, Report No. 69-0013.

B. Language Descriptions

(1) "Flight Computer and Language Processor Study", Logicon, Inc., NASA-ERC, Contract No. NAS12-2005.

(2) "Vital Programmer's Manual and User's Guide, Volume I", PRD Electronics, Inc., PRD Control No. 181-69-02, Rev. 0.

(3) "The Compiler for the Programming Language for Automatic Checkout Equipment (PLACE), Part I PLACE Language and Compiler", Batelle Memorial Institute, Air Force Aero-Propulsion Laboratory, AF APL-TR-68-27.

(4) "Abbreviated Test Language for Avionics System (ATLAS)", Aeronautical Radio, Inc., 418-1.

(5) "MOLTOL Test Writer's Reference Manual", McDonnell Douglas Corporation, no Report number.

(6) "Appendix of ATOLL References, Specification for the Operating System for the Saturn V Launch Computer Complex", IBM Corporation, 66-232-001.

(7) "Automatic Sequence Execution and Processor (ASEP)", General Electric Corporation, ATM-U002-0.

(8) "Intermediate Language Definition (CVA-VTRAN)", PRD Electronics, Inc., 181-69-08.

(9) "OCS-Test Language Assembler", General Electric Corporation, OCS-03-Rev. 1.

(10) "CAGE Test Language Description", Martin Marietta Corporation.

(11) "Programming Manual for Test Station, Guided Missile System AN/TSM-93, Vol. 2, UTEC Compiler Programming", RCA Corporation, Aerospace System Division.

(12) "TOOL-Test Oriented On-board Language System, Flight Packaged On-board Checkout Systems Development Unit, Overall

Specification-Level 2", Martin Marietta Corporation.

(13) "STOL User's Manual", McDonnell Douglas Corporation, SM-46842.

(14) "Preliminary ATOLL II Spec.", Mesa Scientific Corporation, NASA-MSFC.

ILLUSTRATIONS

INPUT STATEMENTS

| LABEL | OPERATION | TO STORAGE<br>DISP FORMAT | TEST POINT<br>MODULE NAME<br>DISP ADDRESS | VALUE(S) | TIMING | NEXT<br>LAST | FLAGS |
|---|---|---|---|---|---|---|---|
| | NAME | | TP0050 | PUMP TURN-ON SEQUENCE | | | |
| | ISSUE DIS ON | | TURNON-1 | | | | |
| | TEST PRP GT | | VERIFPT-1 | 2500 PSIA | 50 | DONE | |
| | DISP LINE | | C5 | | | | |
| | TURN-ON NOT COMPLETED | | | | | | |
| DONE | END | | TP0050 | | | | |
| | 6 pos | 12 pos | 12 pos | 12 pos | 24 pos | 6 pos | 6 pos | 2 pos |

EXPANDED LISTING

```
        START OF TP0050-PUMP TURN ON SEQUENCE
        SET -PUMP NO 1 START- TO ON
        IF VALUE OF -PRESSURE NO 1- IS GREATER THAN 2500 PSIA, WITHIN
                50 MSEC, GO TO DONE
        DISPLAY ON CONSOLE 5
                TURN ON NOT COMPLETED
DONE    END OF EXECUTION OF TP0050-PUMP TURN ON SEQUENCE
```

SAMPLE INPUT STATEMENTS AND EXPANDED LISTING

Verbs

ADD - add two numbers
CALL - call test procedure
DISP - display information
DIV - divide two numbers
END - end of test procedure
GO TO - go to statement nr _ _
ISSUE - issue stimulus
MPY - multiply two numbers
NAME - name of the Test Procedure is _ _ _
READ - read measurement
SET - set value to _ _
SUB - subtract two numbers
TEST - test measurement/ condition
DELAY - delay execution

Conditions

DA - disable
DL - delete
EN - enable
EQ - equal to
GE - greater or equal
GT - greater than
LE - less than or equal
LT - less than
NE - not equal to
OF(F) - off
ON - on
OL - outside limits
WL - within limits

Objects

CDE - digital code
COND - condition indicator
DATA - data stored in memory
DIS - discrete signal
FORM - display format
LINE - text line
OPTN - option indicator
PRP - proportional signal
TIME - system clock time
VALUE - value of parameter

COMMON OPERATOR TERMS

Test Point Names

- o Input Names
- o Descriptive Phrase Output Name

Data Distribution System Address

Test Point Type

- o Measurement/Stimulus
- o Information Form
  - Discrete
  - Proportional
  - Digital Code

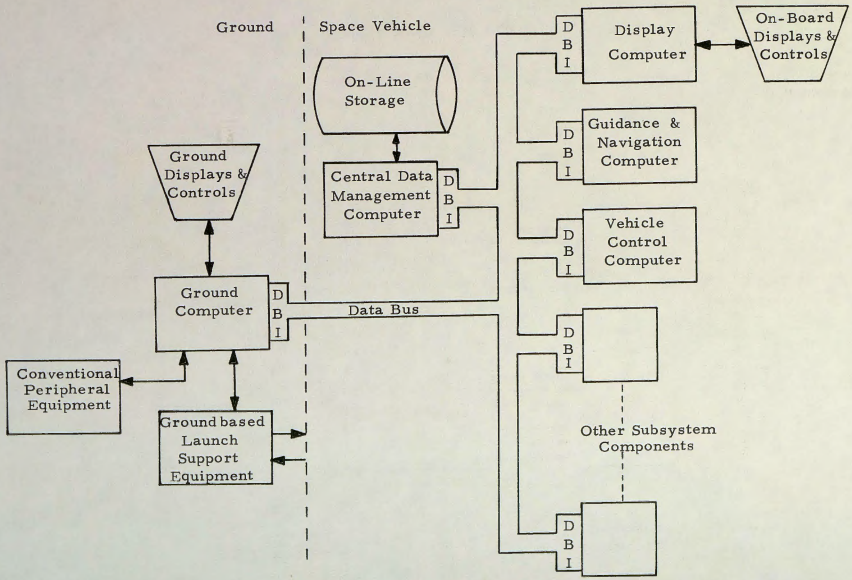
Proportional Signal Conversion Data

- o Calibration/Conversion Points
- o Negative Value Representation
- o Engineering Units Name
- o Nominal Number of Samples in Average
- o Filter Characteristics for Filtered Readings

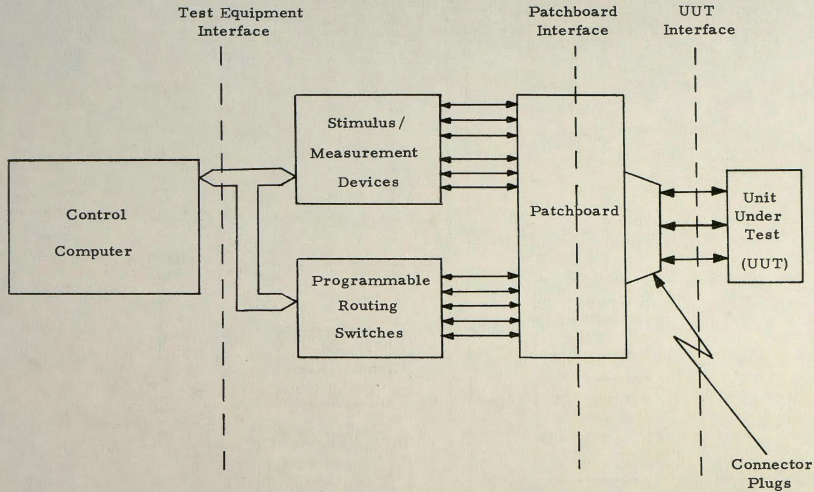
Nominal Sampling Interval

Post-Stimulus Delay

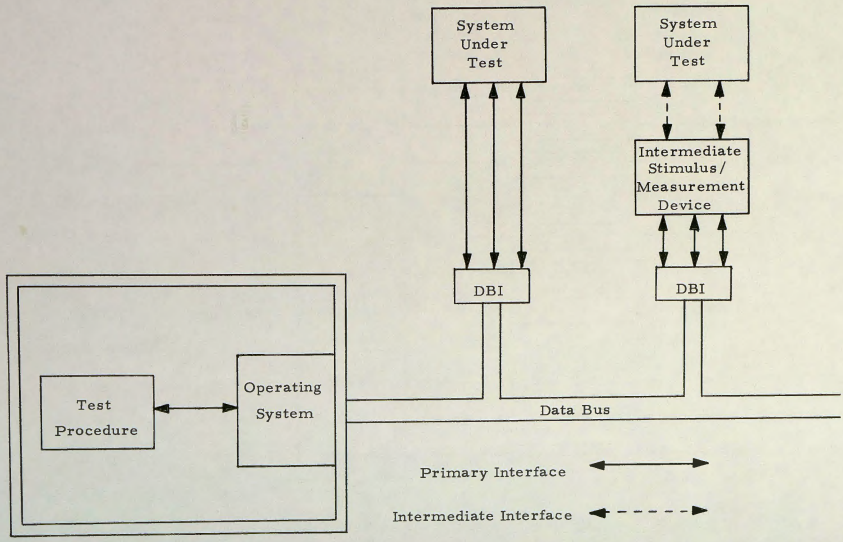
TEST POINT CHARACTERISTICS FILE



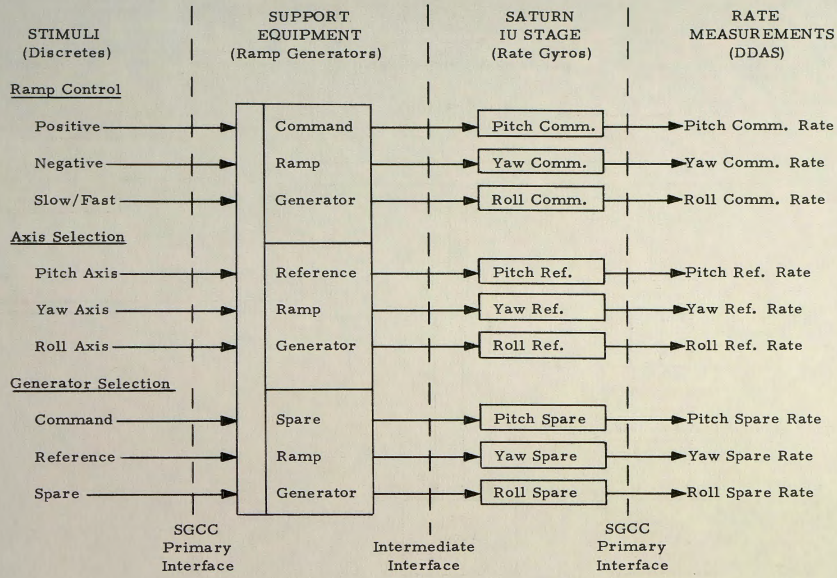

BASELINE VEHICLE CHECKOUT SYSTEM




BENCH TEST COMPLEX




VEHICLE CHECKOUT SYSTEM SIGNAL INTERFACES




DIGITAL RAMP GENERATOR INTERFACES