TEST ENGINEERING LANGUAGE FOR AVIONIC SYSTEMS

M.Mainini and G.P. Mariani
Agusta S.p.A.
Tradate, Italy

Abstract

To-day, the test of complex avionic systems is a task that requires tools with high level and a user friendly approach to write and perform test sequences regardless of the complexity of the software and hardware of the system under test. This approach allows the test engineer to concentrate his attention verifying the system performance against requirements instead of losing time writing complicated low level test sequences. The scope of this paper is to present the Test language developed and employed by Agusta during the development and testing of the EH101 and A129 integrated avionic systems. The structure, formal syntax and examples of the language will be presented together with the specific environment where the tool is used for both the host computer with its facilities and the hardware architecture of the rig.

The conclusions will outline the importance and benefits of utilizing a tool like TELAS in a project of an avionic system.

Introduction

A digital avionic computer is basically interfaced to aircraft sensors and equipment by means of two digital buses: MIL-STD 1553 bus or Arinc 429 bus.
Moreover, for the cases in which the sensors have no bus interfaces, a sensor interface unit provides interfacing between sensors and on board computers.
In such a way we can summarize that a general test environment provide stimulation and monitoring of the unit under test by means of the following equipments, as shown in fig 1 :

- MIL-STD 1553 Bus Station
- Arinc 429 Bus Station
- Aircraft Sensor Emulator

The 1553 Bus Station may perform multiple remote terminals, bus controller or bus monitor function depending on the test set-up, and are connected to the test computer directly on the internal bus of the computer.
The Arinc 429 Bus Station may perform the function of equipment simulation or bus monitor and are connected to test computer by means of a dedicate line.
All the sensors of the aircraft sub-systems (Elctrical,Hydraulic,Transmission ect.) may be emulated by a set of jigs interfaced to test computer by means of a line.

On this ground considerations, Agusta has developed for its own purpouse a test language, called TELAS (Test Engineering Language for Avionic Systems), that interfaces and drives the above mentioned equipment in an easy and flexible way.
Telas is a tool for automatic testing oriented to avionic computer validation and qualification at system level.
The tool has been developed on VAX Digital computers with VMS operating system, using Fortran language for application programs and DCL (Digital Command Language) VAX/VMS for the definition of Telas statements.
Starting directly from system requirements and test plan it is possible to write the test procedures by means of the Telas statements.
The test procedures become Telas command files that the test computer executes automatically and produces the test results report.

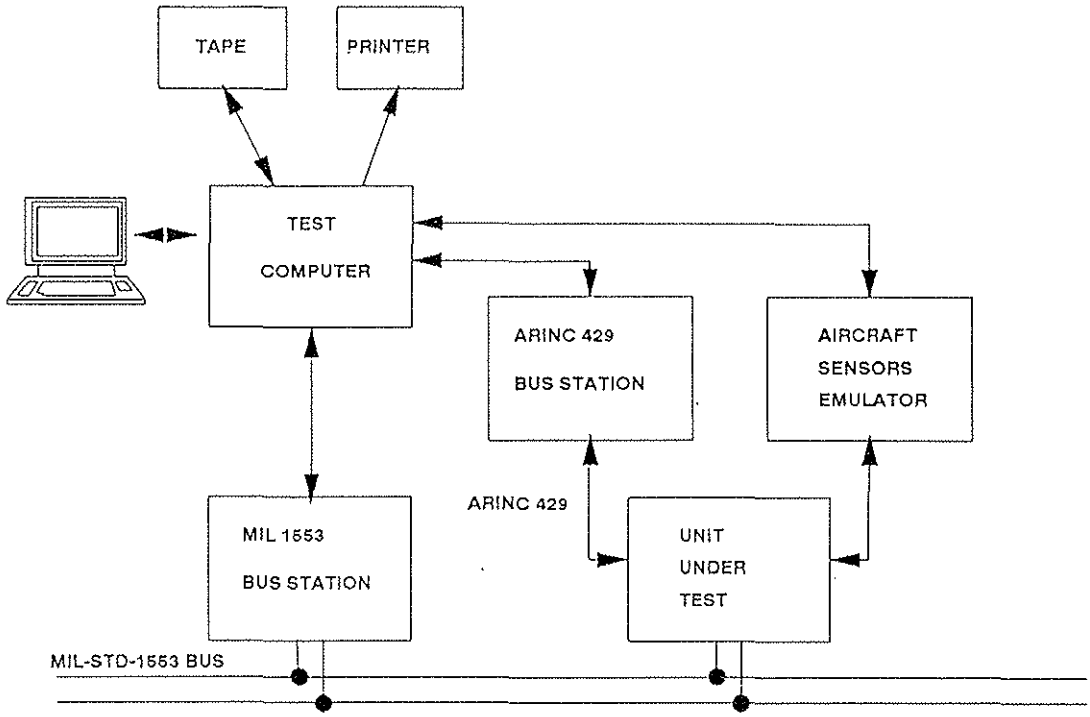

fig. 1

Telas Architecture

The Telas architecture is based on a data base that contains all the parameters definitions and their initialization values related to Mil 1553 bus station ,Arinc 429 bus station, aircraft sensors emulators (fig. 2).
These parameters characterize all the information that the Unit Under Test exchanges with emulated aircraft equipment/sensors.
The Parameters definition is performed by a test engineer by means of an utility called TCF (Telas Configuration File) which allows the following:

- assign a mnemonic name to each parameter
- describe the meaninig of the mnemonic name
- specify the parameter characteristic in terms of format,LSB value,measure unit,range etc.
- specify the 1553, Arinc 429 and sensors emulators address of the parameter

These definitions allow the test system to be configured and to analyze the acquired data.
The initialization of the parameters defined with the TCF utility is performed by the test engineer using an utility called TSU (Telas Set-Up) that allows values to be assigned to each parameter in order to create particular test conditions.
A shared memory provides interface between Telas commands and the real time emulation programs that manage the avionic bus stations and sensors emulators.
Moreover, real time programs allow temporization of the emulated sensors and drives the bus stations and the sensors emulator.

User Interface

In order to allow the test engineer to interact with TELAS a user friendly interface based on video menu driven forms with on line help facility has been developed.
The operator interface allows the test engineer to:

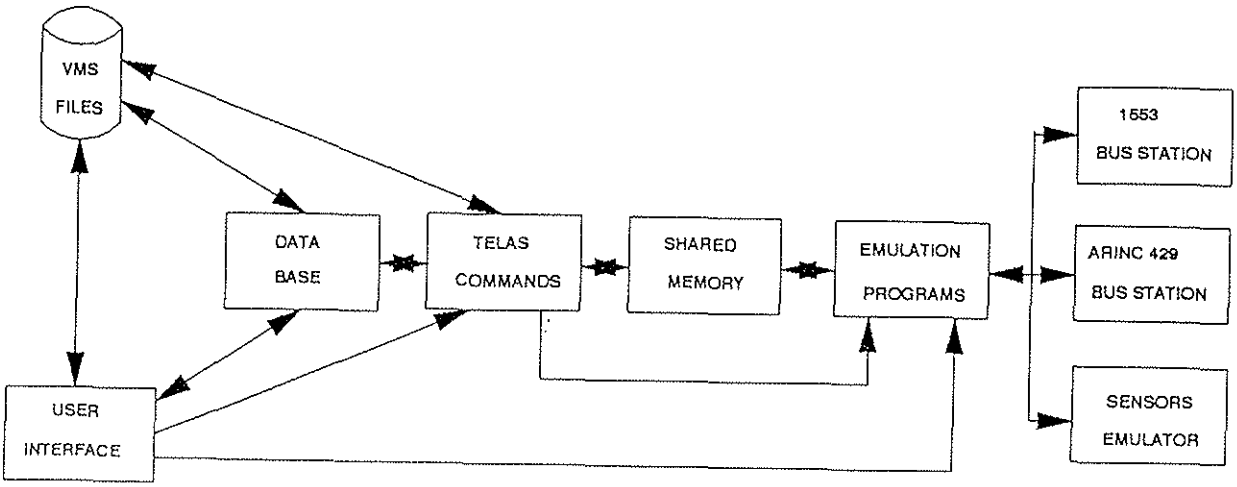

fig. 2

- Configure the test system by means of TCF utility

- Initialize parameters by means of TSU utility

- Write test sequences

- Run the automatic test and control the execution on the display.
  The operator can see on the display the Telas statement that the system is executing, the result of the test case

- Execute test in interactive mode during the debug phase.
  The operator digits Telas statement and control test result on the display

- Print test results

- Manage data files

For each facility described above error control and diagnostic messages are provided.

Test development cycle

The test development cycle begins from the analysis of system requirements and test plan document (fig. 3).
The documents mentioned above constitute the starting point to write test procedures.
Test procedures verify a group of system requirements items related to sub-systems or homogeneous parts of sub-systems as specified by the test plan document.
Moreover, a test procedure is split into a certain number of test sequences.
First of all, the test engineer has to configure and initialize Telas environment by means of TCF and TSU utilities.
At this point each test sequence is written in Telas language by means of a text editor, following the system requirements step by step.
The result of this process is the production of the test instructions that Telas will execute automatically and, at the same time, the production of Automatic Test Procedure document.
The output of the Telas processing is the test result reports which can qualify the system or starts corrective action process in case of non-compliance with the system requirements.

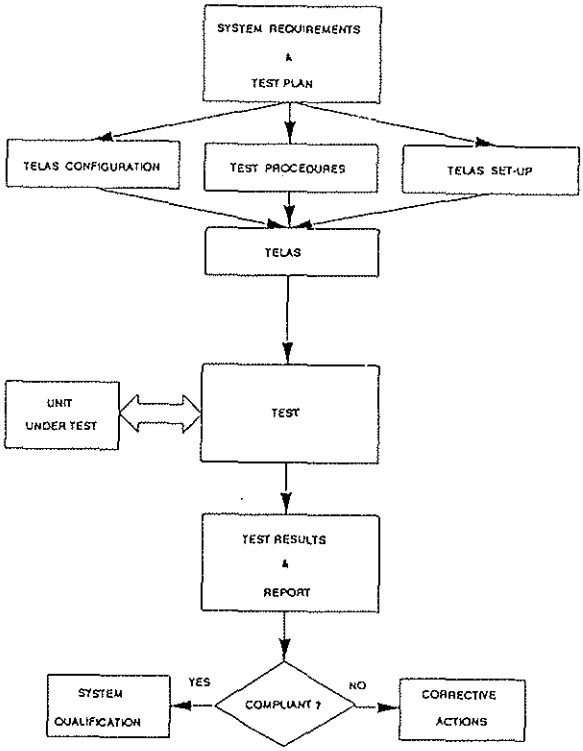

fig. 3

Telas program structure

A TELAS program structure provides control of the overall test execution sequence.
Execution of a complete test program proceeds sequentially by statement order except where modified by control statements.
A Telas program structure consists of three distinct parts:

- preamble statements

- main procedural statements

- terminate statements

Preamble statements - The preamble statements precede the procedural statements. Preamble statements do not cause any tests to be executed,but include configuration and set-up information that are referenced in the procedural section.

There may be only one program preamble structure within an entire program structure.

The preamble statements are:

INCLUDE < configuration file >
INCLUDE < setup file >

These statements include the configuration and set-up files prepared during the **"configuration phase"**.
The following structure of statements can be considered part of the preamble and allows the test sequence to be identified or to introduce general comments.

```
C"
! 5 lines reserved for
! test case title or comments
!
!
!
C&
```

The five test lines will be printed in the test results document.
These statements will also cause the printing of the header file immediately before the C" statement.

Main procedural statements - The main procedural statements are a series of statements, each of which describes a portion of the required test which must be completed prior to the next statement.
Each statement implies an instruction to proceed to the following statement after completion of the present one unless directed otherwise in a branching statement.
The procedural statements are listed below:

FILL
START EMUL
STOP EMUL
START ACQ
MONITOR
READ
COMPARE
FIND
IF
GO TO

FILL Statement - The FILL statement set a specified parameter to a value specified in the statement.
The parameters are related to bus stations(1553 and Arinc 429) and sensors emulator.
The syntax of the statement is the following:

FILL <mnemonic> <format> <value> <value1>

where:

<mnemonic>: parameter name to be set

<format>: parameter data format (real,integer,logical,hex,octal, character)

<value> : data value

START EMUL statement - The statement schedules the emulation tasks related to 1553 remote terminals,bus controller, Arinc equipment or sensors emulators.
The syntax of the statement is:

START_EMUL <option1> <option2> .....<option8>

<optionN=1,8> : name of remote terminal, bus controller,Arinc equipment or sensor to be emulated; up to eight selections may be specified.

STOP EMUL statement - The statement permits to deactivate the emulation of the 1553 units activated with the START_EMUL statement
The syntax of the statement is :

STOP_EMUL <option1> <option2>.....<option8>

<optionN=1,8> : name of emulator to be stopped.

START ACQ MBT statement - The statement allows 1553 data to be acquired on the military bus by means of the bus 1553 bus station operating as bus monitor.
The syntax is:

START_ACQ_MBT <buffer> < Major frame number> <sync option>

<buffer> : name of the data buffer in which the acquired data are stored

<Major frame number> : number of 1553 major frames to be acquired

<sync option>: data acquisition is synchronized with the step by step behavior of bus controller of the unit under test

START ACQ ARINC statement - The statement allows Arinc 429 data to be acquired on the bus by means of the Arinc bus station operating as bus monitor.
The syntax is:

START_ACQ_ARINC <buffer> <sync option>

<buffer> : name of the data buffer in which the acquire data are stored

<sync option>: data acquisition is synchronized with the step by step behavior of the unit under test

MONITOR statement - The statement allows to display/print the value of the parameters from the acquired buffer with the START_ACQ statements.

The syntax is:

MONITOR <buffer> <mnemonic1>.....<mnemonic7>

<buffer> : name of data buffer where the parameters value are stored

<mnemonic1,7) : parameter name; maximum of seven parameters are allowed.

READ DATA statement - The statement reads the value of the acquired parameters stored in the

data buffer and performs comparison with the expected values.
The syntax is:

READ_DATA <buffer> <mnemonic> <format> <value> <option> <option1>

<buffer> : data buffer

<mnemonic>: parameter name

<format> : parameter data format

<value> : expected value

<option> : specifies the kind of algorithm (average,in-limits,out - of -limits) to be performed on the acquired parameter values

<option1> : specifies the kind of the comparison limits according to algorithm type

COMPARE statemenet - The statement performs comparison between two acquired data buffers.

COMPARE <buffer1> <buffer2> <option>

<buffer1,2> : acquired data buffers to be compared

<option> : specifies if the average is to be performed on the acquired values

FIND statement - The statement makes a comparison of all the values related to a specified parameter contained in the acquired buffer with a reference value.
If the condition is satisfactory the value and its buffer position are printed.
Moreover, a Boolean variable is created (TRUE,FALSE) and it can be used in the IF statement.

FIND <buffer> <mnemonic> <log.op> <format> <value>

<buffer> : buffer name

<mnemonic>: parametr name

<log.op.> : logical operator (EQ.NE.GE. .....)

<format> : parameter data format

<value> : reference value

PAUSE statement - The statement suspends the test execution and asks the operator if the test has to continue or not.
It can be used in each point of the test.
the sintax is :

PAUSE

IF statement - The statement tests the value of an expression containing the variable created by the FIND statement and depending on the syntax specified, executes TELAS statements:

IF expression
THEN [TELAS statement]
TELAS statement
.
.
.
[ELSE] [TELAS statement]
TELAS statement
.
.
.
ENDIF

GO TO statement - The statement transfers control to a label in a Telas program.

GO TO label

Terminate statement - The terminate statement completes the test program and must be the last statement of the entire test.

END

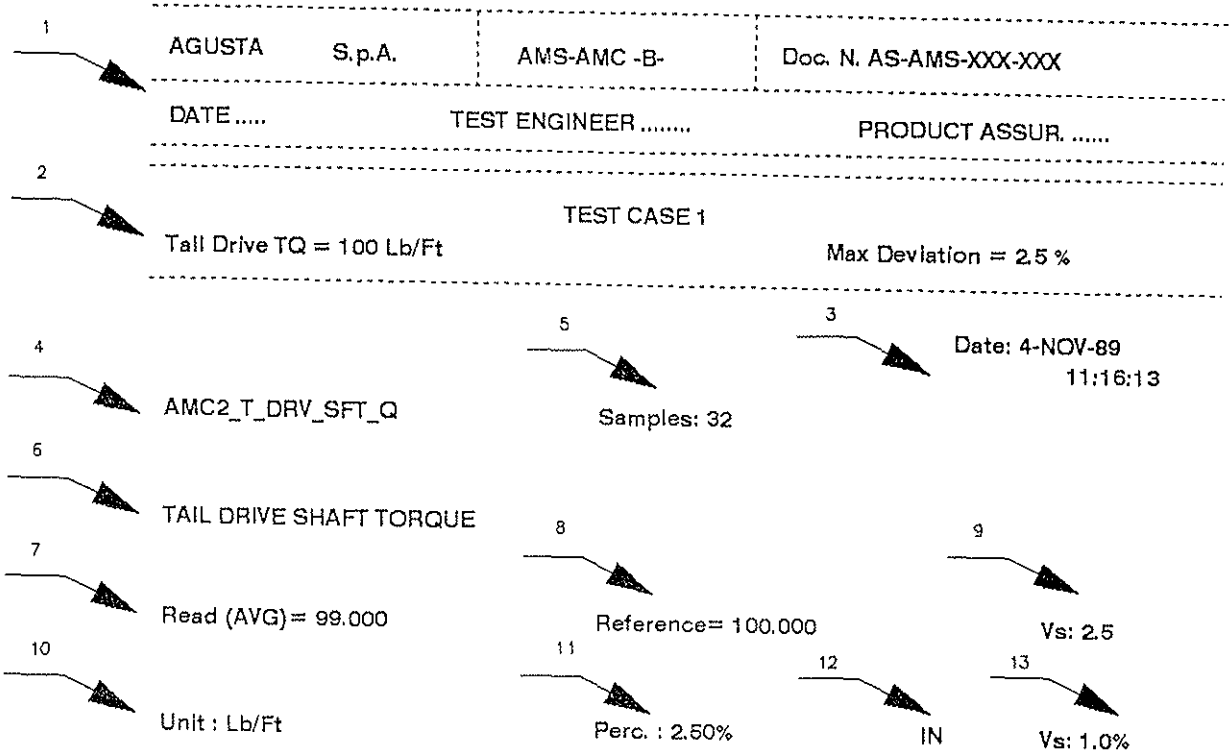

fig. 4

Telas Result

As previously mentioned Telas automatically produces test result reports.
Telas provides results with different format types depending on the options specified in the statements that foresee results printing.
In general all the result formats contain the following information:

1 Header on each test result page
2 Test case title
3 Execution time and data
4 Parameter mnemonic name
5 Acquired samples number
6 Parameter description
7 Average of acquired parameter values
8 Expected parameter value
9 Allowed parameter deviation
10 Parameter measure unit
11 Maximum percentage of deviation allowed
12 Key word that specifies if the test is passed or not
13 Percentage deviation

A typical example is shown in fig. 4

Test example

A simple example of a test case related to helicopter computer qualification is illustrated here below.
In particular, the requirement related to an alarm management is verified.

The relative part of the system requirement document is:

```
        (NGH is managed as above).

        If ENG1-NG 'OR' ENG1-OIL-TEMP 'OR' ENG1-OIL-PRESS = NOT
        VALID, then manage the alarm reset.


ENA2.21.6     If:
              ENG1-NG >= 62.6 for < 5 min,          'AND'
              ENG1-NG <= NGH                        'AND'
              ENG1-OIL-TEMP <= 38,                  then:

                 If ENG1-OIL-PRESS < 2.412 'OR' > 4.324 BAR, then manage
                    the ENG1-OIL-PRESSURE CRT ON alarm.
                    (NGH is managed as above).

        If ENG1-NG 'OR' ENG1-OIL-TEMP 'OR' ENG1-OIL-PRESS = NOT
        VALID, then manage the alarm reset.


ENA2.21.7     If ENG1-NG >= 62.6%, then:

                 If ENG1-OIL-TEMP > 120, then manage the ENG1-OIL-TEMP CRT
                 ON alarm.
                 (NGH is managed as above).

        If ENG1-NG 'OR' ENG1-OIL-TEMP 'OR' ENG1-OIL-PRESS = NOT
        VALID, then manage the alarm reset.
```

The requirement is verified performing the test sequence shown below:

```
$!
$!------ TEST PROCEDURE: TP07 - TEST SEQUENCE : TS31 ------------------
$!------ TP FILE NAME  : AMC_R2_EXE_TP07.COM ---------------------------
$!------ TS FILE NAME  : AMC_R2_TP07_TS31.COM --------------------------
$CC
$!
$!        In this TS the AMC will operate in Single_Frame mode.
$!
$!        Scope of this TS to test the Requirement ENA2.21.6 for the
$!        ENG OIL PRESS 1,2,3 CRT ON alarm.
$!
$!
$!        The TS 31 is structured as shown in the following table:
$!
$!-------------------------------------------------------------------------
$!SIUs  PAL  PAL  NG>=62.6  NG<=    OT     OP<2.412   CRT    TC
$!Stat  SO   GI   for<5min  NGH     <=38   or>4.824   ON
$!-------------------------------------------------------------------------
$!good  0    0    N(62)     T       T(37)  T(1.0)     N      1
$!good  0    0    T(66)     T       T(37)  T(1.0)     T      2
$!good  0    0    T(66)     T       T(37)  T(6.0)     T      3
$!good  0    0    T(66)     T       T(37)  N(3.5)     N      4
$!good  0    0    T(68)     N       T(37)  T(6.0)     N      5
$!good  0    0    T(66)     T       N(39)  T(6.0)     N      6
$!good  0    0    T(SF)     T       T(37)  T(6.0)     N      7
$!good  0    0    T(66)     T       T(SF)  T(6.0)     N      8
$!good  0    0    T(66)     T       T(37)  T(SF)      N      9
$!good  0    1    T(66)     T       T(37)  T(6.0)     T      10
$!good  1    1    T(66)     T       T(37)  T(6.0)     N      11
$!good  1    0    T(66)     T       T(37)  T(6.0)     T      12
$!good  0    0    N(>5min)  T       T(37)  T(6.0)     N      13
$!-------------------------------------------------------------------------
$!
$!- Set SIU1 TIT 1,2,3 = 140   and   SIU1 NG 1,2,3 = 4.0
$!     -Execute TC 14
$!-------------------------------------------------------------------------
$!SIUs  PAL  PAL  NG>=62.6  NG<=    OT     OP<2.412   CRT    TC
$!Stat  SO   GI   for<5min  NGH     <=38   or>4.824   ON
$!-------------------------------------------------------------------------
$!good  0    0    N(4.0)    T       T(37)  T(1.0)     N      14
$!-------------------------------------------------------------------------
$!
$!- Restore SIU1 TIT 1,2,3 = 700   and   SIU1 NG 1,2,3 = 66.0
$!     -Execute TC 15
$!-------------------------------------------------------------------------
$!SIUs  PAL  PAL  NG>=62.6  NG<=    OT     OP<2.412   CRT    TC
$!Stat  SO   GI   for<5min  NGH     <=38   or>4.824   ON
$!-------------------------------------------------------------------------
$!fail  0    0    T(66.0)   T       T(37)  T(1.0)     N      15
```

For semplicity we consider the test case number 1.
We can see the part of the test statements related to the test case in which the FILL statements set the test case condition and the READ statements read and compare the parameters values with the expected ones:

```
$!
$!-------- TP07 - TS31 ----------------------------------------------------
$!
$!-------- TEST STEP 1 ----------------------------------------------------
$!
$CC
$!
$       FILL siui_ng_1   R 62.0
$       FILL siui_oil_temp_1     R 37.0
$       FILL siui_oil_press_1    R 1.0
$       START_AMC        150
$       START_ACQ_ARINC          buffer_a429 SYNC
$       START_AMC        8 SYNC
$       READ_DATA        buffer_a429 sgx_s_eng_o_pr_1 I 0 TYP
$       READ_DATA        buffer_a429 sgx_s_crton_eng X 00 TYP
$       READ_DATA        buffer_a429 sgx_s_crton_h_g X 0000 TYP
$!
$C-
$!
$!-------- TP07 - TS31 ----------------------------------------------------
$!
$!------- TEST STEP 2 -----------------------------------------------------
$!
$CC
$!
$       FILL siui_ng_1   R 66.0
$       START_AMC        150
$       START_ACQ_ARINC          buffer_a429 SYNC
$       START_AMC        8 SYNC
```

In conclusion, we can see the test case results according to the format

```
-------- TP07 - TS31 ----------------------------------------------------

-------- TEST STEP 1 ----------------------------------------------------

                                                          Date: 16-APR-90
                                                                10:47:17

SGX_S_ENG_O_PR_1      Read : F    Reference : F    Samples :    2

27Z AMC ENGINE DISCRETE 1 - ENG OIL PRESS 1

XInt : 0080.0      SDI :  0      SSM :  0

                 11.................29
Read (TYP):    00000000000000000000    Bit_pos. : 11          EQU


SGX_S_CRTON_ENG       Samples :    2    Counter :    2

272   PRESS AND TEMP OF ENGINES CRT_ON

XInt : 0080.0      SDI :  0      SSM :  0

Bit_start : 11      Bit_end : 16

                 11.................29
Read (TYP):    000000                          EQU
Reference :    000000


SGX_S_CRTON_H_G       Samples :    2    Counter :    2

271   PRESS AND TEMP OF HYDR AND GEARBOX  CRT_ON

XInt : 0080.0      SDI :  0      SSM :  0

Bit_start : 11      Bit_end : 25

                 11.................29
Read (TYP):    0000000000000000                EQU
Reference :    0000000000000000
```

Conclusion

Telas, even though it may not be a very sophisticated test language like the famous ATLAS, it has proved to be a very useful and efficient tool for the integration and qualification phase.
It has been developed with the main aim to obtain a product oriented to avionic computer integration and qualification test at box level. These aspects are important to be able to understand more about the specific environment where the tool is utilized and the type of Telas statements in comparison with other commercial test languages.
As a consequence of these considerations we can summarize the advantages of using Telas:

- user friendly approach
- flexibility in system configuration
- reduction of uman errors
- test reproducibility
- automatic test results documentation