Communications and Tracking
Expert Systems Study
Preface

This research was conducted under the auspices of the Research Institute for Computing and Information Systems by T.F. Leibfried, Associate Professor of Computer Science, Terry Feagin, Professor of Computer Science, and David Overland, Research Associate, all at the University of Houston-Clear Lake.

Funding has been provided by the Tracking and Communications Division, within the Engineering Directorate, NASA/JSC through Cooperative Agreement NCC 9-16 between NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA Technical Monitor for this activity is Oron Schmidt, Systems Techniques Section, Communications Performance and Integration Branch, NASA/JSC.

The views and conclusions contained in this report are those of the authors and should not be interpreted as representative of the official policies, either express or implied, of NASA or the United States Government.
INTERIM REPORT
UNIVERSITY OF HOUSTON-CLEAR LAKE
RESEARCH INSTITUTE FOR THE COMPUTING
AND
INFORMATION SCIENCES
RESEARCH ACTIVITY AI-1
COMMUNICATIONS AND TRACKING EXPERT SYSTEMS STUDY

Prepared by:
T. F. Leibfried, Jr., Ph.D. -- Principal Investigator
T. Feagin, Ph.D. -- Research Area Director
D. Overland, B.S.M.E. -- Research Associate

In cooperation with:
O. Schmidt -- Technical Monitor/NASA-JSC
30 January 1987
EXECUTIVE SUMMARY

The original objectives of the study consisted of five broad areas of investigation:

1. Criteria and issues for explanation of C & T system anomaly detection, isolation, and recovery;
2. Data storage simplification issues for fault detection expert systems;
3. Data selection procedures for decision tree pruning and optimization to enhance the abstraction of pertinent information for clear explanations;
4. Criteria for establishing levels of explanation suited to needs;
5. Analysis of expert system interaction and modularization.

Progress was made in areas 1, 2, 3 and 5, but to a lesser extent in area 4 during Phase I.

Among the types of expert systems studied were those related to anomaly or fault detection, isolation and recovery. Specifically, the interim results of Harris and TRW T&C expert system studies were examined and work supplementing them with explanation facilities was initiated.

An expert system which is rule based may be thought of as a sequence of if-(condition)-then-(action and fact(s)) statements in an endless repeating loop. A given statement or rule, when its condition has been satisfied and it executes, is said to have "fired". The rule usually asserts "facts" which may satisfy the condition(s) of other if-statements or rules which in turn can assert actions and/or other facts and so on. The beauty of these systems is that they are flexible, easily expanded or modified, succinct, and readily understood by humans. Their problems are a general lack of structure, modularity for groups of related rules, and therefore poor maintainability. Also, in complex rule-based systems there are problems controlling the order in which the rules are applied. That is, a "resolution strategy" must be provided. Most expert systems of this type are each essentially one big program where object-oriented design and information hiding are relatively absent. These latter concepts are essential for data integrity of software systems. This is a major issue for any project as big as the Space Station, where the total software system size may be measured in millions of lines of code. The potential benefits of expert systems are too great to reject them out of hand, especially since the modularity issue is probably tractable. Modularity in such systems is being investigated as part of area number 5 above.
Results to date indicate that modularity is possible especially in the case of predictable expert systems (i.e., systems where rules cannot make new rules). In fact, virtually any predictable expert system is capable of being rewritten in a procedural language. In some case that may be even desirable for part, if not all, of a given system. This was done in this study for a simple subsystem based upon a system simulator written by TRW. The simulator, written in C, was rewritten in Ada. Then, fault diagnosis and explanation facilities were implemented to investigate area number 1 (explanation facility) and area number 5 (interaction and modularization) described above. The source code for this system is in Appendix B as MAIN_SIM_MOD3.ADA. Results of this portion of the study are shown in item D of the body of this report. Briefly, the results indicate that, an explanation facility is best structured as an integral part of the system rather than as an appendage. Object-oriented modularization promotes data integrity, and that the capability of retrieving and using old data is probably best achieved through a procedural language. Object-oriented design was first suggested by Parnas and implies that each object (e.g., variable) should be controlled by one module and the resources necessary to modify or change any of its characteristics should reside only within that module.

The areas number 1 (expert explanation), number 2 (storage simplification) and number 3 (decision tree issues) were investigated by the implementation of a simple engine diagnosis program. The source code and concomitant comments for DIAG4.ART are in Appendix A. The results are shown in item B of the body of the report. Briefly, the results show that rules have a taxonomy (e.g., explanation rules, bookkeeping rules, action rules), and that the time stamping of facts is necessary for explanations of any past expert system actions. The need for functional or domain partitioning was one of the discoveries of this investigation.

Not much implementation of techniques for pruning decision trees of irrelevant decision nodes (area number 3) was accomplished; however, these and other data compression concepts were discussed extensively and some paths of investigation and experimentation are indicated. So-called spine optimization of the decision tree may provide a capability for explaining in retrospect how and why a decision has been reached or even perhaps why a particular decision was not reached. (A spine is defined as the conjunction of decisions for which the retrieval is a single conclusion.)

In the relatively short time this study has been in progress, what were originally nebulous issues have become more clear. This is not to say that they have become more tractable, but at least some of the issues are better defined than they were four months ago. We expect that this trend will continue.
A. Introduction

The original statement of activity objectives consisted of several items summarized as follows:

1. Examination of existing C&T systems configuration and monitoring problems--
The activity started with an examination of fault detection expert systems and historical database storage, which is one facet of the subject area of C&T system anomalous behaviour. An anomalous event may be defined as the situation when normal systems software has been unable to operate according to plan. Some tentative conclusions for structuring these activities have been determined. This activity continues.

2. Audit trail simplification policies--
Some experimental work has been accomplished for this activity. In particular, audit trails for fault detection expert systems have been analyzed. An hypothesis has been formulated, based on preliminary results which indicates that shallow explanations may be able to use a limited audit database but deeper explanation facilities may require a parallel expert system, otherwise the audit trail database might become too cumbersome. The question of the effect of utilizing distributed expert systems on practical explanation facilities has not yet been considered, but that is a subject which will need to be addressed once the resources needed by such facilities have been identified.

3. Data selection procedures--
This activity is closely related to item 2 but has been modified to focus upon decision tree pruning and storage depth of unchosen paths. For explanation purposes one hypothesis to be explored is to store chosen path nodes and one node level below each chosen path node for all unchosen paths. This would, at a minimum, double the storage required for a chosen decision tree but it would obviate the necessity for reexercising a duplicate of parts of the original expert system for shallow explanations. No general rule with specifics has yet been established but some simple subsystem simulations, namely, DIAG4.ART and MAIN SIM MOD3.ADA, have given some insight into this problem. This activity continues.

4. Criteria development for user-expert system interfaces--
This activity has been modified to focus upon identifying levels of explanation suited to various user's requirements. It will be assumed that user requirements will have already been identified. It will be assumed that short explanations will be supplied initially and that levels of depth may be requested at any point in the process. This activity has not seen much progress to date but will continue in the revised direction.
5. Expert system interaction—
Intercommunication and hierarchical decision priorities have been discussed but no definitive assertions have yet been determined. At first sight this issue seems related to partitioning and modularizing expert systems. This latter issue is one which is to be explored in the next phase of this study.

B. Explanation Facilities and Decision Trees

An explanation facility will be an important part of any system that is used to provide diagnostics for problems that may develop in the tracking and communication systems on the space station. Also, once the most probable source of a problem has been identified and corrective measures are begun, an explanation of why and how such actions have been taken will be necessary. It would appear that there are several levels of explanation that can be provided. At one extreme, a complete explanation with all the intricacies and details of all of the inferences used could be provided, such that the explanation would be effectively equivalent to providing a complete decision tree (in which only one path has been followed). At the other extreme, a brief, superficial explanation could be provided that would only indicate the most immediate reason why a particular path was selected. This might be of use in the case when a systems human monitor might have selected a different alternative than did the expert system and he seeks the reasons for the selection made by said expert system. In this latter case, it would be practical to store the path and the collection of decisions made along the path. However, simply quoting the path to the given conclusion will not, in general, provide a satisfactory explanation because there may be irrelevant decisions on the path and there may be a need for deeper explanations. It appears that the need for deeper explanations can only be satisfied if the complete reasoning process for reaching a given conclusion is available (i.e., either we provide access to all the rules that were used or we provide the complete decision tree). Such might be needed when a large scale manual intervention is planned for direct control of C&T systems and in that case irrelevant decisions would effectively be "noise" in the system. A procedure is available that would permit the removal of irrelevant decisions. It is called spine optimization, which consists of pruning the decision tree to obtain a "prime spine", and which prime spine is formed by delaying or removing irrelevant decision nodes in the tree. Most of the work so far on spines has been theoretical, so it is not clear that the benefits of such removal would justify the costs.

It may be, in some cases, that the option of providing the complete decision tree would not be practical in light of the large amount of storage that this would require for each of many points in the past. It is possible, however, that a few of the most recent decision processes could be stored in their entirety and that a number of older decision processes could be saved in an abbreviated form (perhaps with just a simple rundown on the decision path actually taken with only immediate explanations for taking a particular path provided).
If more elaborate explanations were required for older decisions, then it would still be possible, albeit time-consuming, to reload another knowledge base with the facts that were true at the time of interest and then begin to execute some of the explanation rules over again, only this time with the user interrupting that expert system and asking questions about the decisions which were made.

One of the dilemmas encountered with after-the-fact explanation facilities is that the explanation facility itself must provide some constructive filtering. Indeed, in order to remove irrelevant decision information the explanation program must prune and perhaps redo, albeit with a different perspective, some of what the diagnostic facility did in the first place. The simplest thing to do would be to supply the complete decision tree, but this would be barely one cut above the Automate Reasoning Tool (ART) dribble file as far as user utility is concerned.

An hypothesis or question which probably is worth considering is, "Should the explanation facility 'anticipate' queries so that it would effectively run in parallel to the diagnostic expert facility but without encumbering it?" Another possibility is to initiate the building of a more extensive event data base whenever an anomalous condition occurs.

At this point, without making a definite statement, let us offer a conjecture. We are of the opinion that storing the entire decision tree may be a viable solution, after all. It's not really as bad as storing the entire state at several points in time -- which might be needed since some problems will be building up over time -- in order to be able to provide explanations. The decision tree information would be kept active for a minimal time, presumably until time had passed when an explanation might be required, and then archived. We think that the decision tree also has just those rules used/needed to provide the explanation. If the other approach is used, we would have to ferret out those rules that were relevant or reload the entire expert system.

C. The ART Environment and the Attempt to Implement an Elementary Simulation, Diagnostic and Explanation System

The ART Environment

There are many advantages and some deficiencies to the Automated Reasoning Tool (ART) environment. The flexibility allows both forward and backward chaining with schema (which provides memory slots similar to frame-based languages), and viewpoints. The viewpoints may provide a way of recalling how a particular expert system operated at a given time in the past. This is something which would be useful for providing an explanation of a past event. Unfortunately, the first diagnostic and explanation example program did not use viewpoints and the lack of the availability of file input-output limited the scope of the program. Useful knowledge was obtained, nevertheless, and that should help the development of future explanation experiments.
Recommendations on Expert Systems Languages

A few observations have resulted from the first diagnostic-explanation system. The ART language coupled with the Symbolics system is a very versatile, albeit sometimes clumsy, approach to expert system development. The feasibility of developing an in-house (Ada-based) rule-based language should be investigated. This would allow data structures and I/O requirements to be tailored for the application.

This could also allow the development of subprocedure calls for both subprocedures written in sequential languages and those written in the development language (in other words, calling other expert system programs).

The DIAG4.ART Diagnostic-Explanation Program

Objectives

The goal of this program was to demonstrate at least a rudimentary explanation facility on a limited domain, mainly as a means of exploring the concepts involved, and also as a means of learning the ART language and Symbolics system.

Description

The program, written in ART, simulates the operation of a four-stroke, two-valve, single-piston internal-combustion engine. It also diagnoses failures in the ignition phase of the engine operation, and implements corrective action. It is then capable of explaining the diagnosis and the corrective action taken.

The program was also written so as to reflect some logical organization: The explanation rules are at the beginning of the program, followed by the bookkeeping rules (those responsible for updating the current parental and subgoals), followed by the initializing "split" rules, followed by the rules for action at each stroke of the cycle. The "split" rules were made necessary because the condition portion of a rule in ART does not have a provision to match on two OR'ed schema slots.

Program Outline

The engine state is modeled by a schema named CURRENT which is modified by the action of the stroke and ignition rules to reflect the operation of the engine. Each relevant component of the engine state is stored in a slot of the schema. At the same time, each slot in CURRENT is compared to a similar slot of the IDEAL engine state. Discrepancies, such as the spark plug not firing, cause error flags to be set which allow the diagnostic rules to fire. In this case, each flag triggers one diagnosis, but combinations of flags could also do this.

Diagnostic rules printout specific error messages and take corrective action (replacing the spark plug). They also query if an explanation is required. If one is, another flag is set.

The combination of the error and explanation flags allow the firing of explanation rules, of which there are two in this program.
Conclusions

The program served as a learning process, but ended up with serious deficiencies:

The fault tree was never more than two levels deep (both spark plugs fail) and did not demonstrate any combinatorial failures nor did it ever have to "guess". There couldn't be a wrong diagnosis.

All explanations were developed as the program "sequenced" through the simulation. No explanations could be given after the cycle continued: the program has no memory of what has already happened; it can only respond to the current state. This could be remedied by time-stamping all facts (saving the telemetry stream), but this would also require a new set of rules to react with past data in addition to those already existing which react with current data, effectively doubling the size of the program (at least).

All explanations either exist from the beginning or they could not be given. The drawback with programming in this style is that all possible faults and fault combinations must be figured in advance and coded into the program. This is trivial for a program this size, but is not practical for a large program. This increases the size of the program exponentially with the number of components.

It also requires being able to simulate the "correct" functioning of the system at all times in order to detect discrepancies.

The source code for this program is shown in Appendix A.

Proposed Further Avenues of Exploration

The concept employed with this program would only allow explanations to be given "on the fly", that is, only after each diagnosis and/or fault correction. Explanation cannot be generated past that point in time; there is no memory of the transaction.

To avoid this fault then in order to generate explanations of past events either:

1. Save all explanations for possible future recall, or;
2. It must be possible to generate explanations from saved data.

There are a number of unexplored avenues here, such as the optimum way of generating explanations, the best way of storing data, etc.

Another concept to be explored is that of partitioning the expert system. The program, as currently written, does not support this, but the idea of breaking the system into modules, either functionally, where a module would perform certain tasks, or by domain, where each module would service a system or subsystem of the overall domain. The ability to do so would have quite an effect on the memory requirements and speed of the system.
D. Ada Implementation of a C&T Subsystem Simulator (TRW), Fault Detection and Explanation Facility with User Interaction

Motivation (objective)
This demonstration program was initiated for two reasons. First to demonstrate how a sequential software system could effectively duplicate the results of a simulation written in C and an expert system written in a specialized rule-based language such as ART (Automated Reasoning Tool by Inference Corp.). The second reason was to demonstrate that in contrast to conventional rule-based systems a sequential system can more easily store facts in a database and access them for an explanation facility. An ancillary reason was to examine how modularization could be used when implementing both diagnostic and explanation facilities.

Structure (program outline)
The program, written in Ada, consists of five modules called packages.

1. MAIN SIM MOD3
   This is really not a module but rather the driver program for the demonstration system. It essentially structures the system by calling subprograms. A simplified algorithm for this program is given in the following enumerated steps.
   (1) Initialize the communications system database;
   (2) Check for inconsistencies in the data, and if there are any, call a subprogram to ask the human monitor to correct the data;
   (3) Call the equipment emulator program;
   (4) Based upon the measured equipment output and the status of the switches, determine the condition of the equipment (i.e., diagnose the probable cause of failure, if any);
   (5) Call a subprogram to display the status of the equipment to the human monitor;
   (6) Call subprograms to provide an explanation of the diagnosis if requested;
   (7) Call a subprogram to interact with the human monitor to see if another simulation is to be run;
   (8) If the human monitor wishes to stop the program then terminate else call a subprogram to ask the monitor to update the simulation parameters and then go to step (2).
2. **Y OUT MOD**
   This package contains the equipment emulator subprogram. It effectively simulates the action of the hardware given the parameters in the equipment status database.

3. **FAULT-ANALYZE**
   This package contains the subprograms which measure the observable parameters and determines the probable fault, if any.

4. **EXPLAIN DIAGNOSIS**
   This package presents an analysis of the reasons behind the fault diagnosis when requested by the human monitor.

5. **IO SIM MOD**
   This package is the one which accesses and updates the simulated equipment parameters.

The source code for this system is shown in Appendix B.

**Observations Based Upon Results**

Not too much can be asserted with certainty but there are a few points which the work suggests. Among those are:

1. To explain even a moderately complex fault analysis decision it may be simpler to parallel a part of the decision process rather than try to filter the information from information written into the data base. Again, a trade-off exists between the classical performance parameters of execution time versus memory (primary and secondary storage). For example, in the diagnostic package called FAULT ANALYZE the principal program DIAGNOSIS tests the output power EQUIP Y OUT LEVEL and if it is less than -145 dbm it calls a "low level" program to examine the on-off switches in SWITCH STATUS. If the main power switch is "ON" then the oscillator switches are tested. If the selected oscillator is "ON" then the program "reasons" that the selected oscillator is inoperative. Now when an explanation of this event is requested the EXPLAIN DIAGNOSIS package is activated. Examining the code for procedure EXPLAIN DIAGNOSIS we see that it parallels the reasoning of procedures DIAGNOSIS and SWITCH STATUS in package FAULT ANALYZE, that is to say, it has the same nested "if" structure. The only thing it adds is the diagnostic message, "We found the selected oscillator switch to be ON, the power output was in the noise level, so potentially we had a catastrophic oscillator failure." If the DIAGNOSIS and SWITCH STATUS programs had been more cooperative they could have selected the appropriate literal value for an enumeration variable and stored this "hook" in the data base for access by the EXPLAIN DIAGNOSIS package. Then all the EXPLAIN DIAGNOSIS procedure would have to do is examine this variable by virtue of a simple "case" statement and supply the quoted explanation, "We found...etc."
Such a "type" definition for the desired variable might be:

```plaintext
type Switch_Permutation_Type is
  (POWER_OFF,
   POWER_ON_OSC_OFF,
   POWER_ON_OSC_ON);
```

A variable of this type could be set equal to one of the enumeration literals by procedure SWITCH_STATUS in package FAULT_ANALYZE.

2. Communicating with a user and writing facts to files and/or ephemeral data storage which facts are useful for explanation facilities may require interfacing a given expert system to procedural language I/O routines.

3. It seems that it should be possible to modularize a given expert system to some extent. For example, a FAULT_ANALYZE subsystem could in large measure be separate from an EXPLAIN_DIAGNOSIS subsystem. There would probably be some shared data and possibly even some shared utility routines, but the main thread of each subsystem could be separate.

4. One of the deficiencies of this implementation is that explanations are done with the same database as the fault analysis system, and before any recovery corrections are made. Thus, the database is unchanged when the explanations are made. The programming system could easily be altered to allow changes in the data and still allow an explanation after the fact. The technique employed would be to create two variables for each entity, one for the old value and one for the current value. This is easily done in Ada or in any procedural language but is more difficult in a so-called expert systems language. This may indicate that any production system would require its expert systems language to provide an interface to procedural language subprograms.

Future Direction

The next task which may be proposed is to expand this demonstration program to include more realistic fault detection with more realistic "hook" data generation. Then the explanation subsystem could be restructured to use these improvements. A parallel system in ART will also be implemented (if feasible).
SUMMARY OF OVERALL RESULTS AND RECOMMENDATIONS

A. The issue of storing the complete decision tree versus only storing the knowledge base for an explanation facility has not been resolved, but a few alternatives to be explored have been identified.

1. Store the decision tree in its entirety with perhaps one branch node for paths not taken at each node in the decision tree.

2. Build and store a complete knowledge data base whenever an anomaly occurs. (This could be any undesirable outcome such as the software system displaying unanticipated behavior. This could be signalled by the astronaut monitor or the expert system itself such as when a system failure is detected by the low level systems.)

3. Store only the facts with appropriate time stamps, and when an explanation is required, load a system containing rules similar to the original expert system so as to effectively parallel the operation of the original expert system but this time allowing the user to interrupt this parallel system to ask for appropriate explanations about paths not taken.

These alternatives are not mutually exclusive but all should be examined in future work.

B. The issue of the features of an expert systems language which language is appropriate for development and perhaps implementation of software systems which meet C & T functional requirements needs to be addressed. This recommendation is based upon the experience acquired by the structuring of a simple simulation and diagnostic-explanation program written in ART, (DIAG4.ART in Appendix A), the program had to be all in one module. In addition, there were the file I/O deficiencies of ART and the difficulty of storing more than one value for a given parameter. As a minimum any such language should be capable of interfacing with a compiler language program in a straightforward manner.

C. It is recommended that the issues of object-oriented design for expert systems be raised and investigated. In addition, the concept of supplying adequate "hooks" for explanation should be addressed early. Explanation facilities are best implemented when they are built in and not just "added" as a separate entity. If this is not done the explanation facility requires additional resources and must parallel some previously "paths" already trodden by other systems. The issues of object-oriented design, modularization, and so-called information hiding may not be just academic but a necessity for implementation of any large and perhaps distributed control and monitoring system, be it a procedural and/or rule-based software system. The topic of object-oriented design for expert systems addressing anomaly detection, recovery and explanation was examined by structuring a procedural software system in Ada. The system consisted of a driver and hardware status simulator, fault diagnosis, and fault explanation modules for a small radio frequency communication subsystem. Implementing expert system functions in Ada indicated that modularization and object-oriented design indeed are feasible without compromising effectiveness.
APPENDIX A

Source Code for DIAG4.ART

an Engine Diagnostic/Explanation Program
(defschema engine-state "state of engine - beginning of intake-stroke"
  (carburation good)
  (piston-direction descending)
  (sparkplug1 good)
  (sparkplug2 standby))

(defschema current (instance-of engine-state))

(defschema ideal (instance-of engine-state))

(deffacts state (state-name intake))

(defrule compare-ok "compares current and ideal if same"
  (schema current (carburation ?state3))
  (schema ideal (carburation ?state3))
  (schema current (piston-direction ?state4))
  (schema ideal (piston-direction ?state4))
  =>
  (printout t t "compared ok" t t))

(defrule compare-not-carburation
  (schema current (carburation ?state3))
  (schema ideal (carburation ~?state3))
  =>
  (printout t t "carburation compared not ok" t t)
  (assert (error-trap carburation)))

(defrule compare-not-direction
  (schema current (piston-direction ?state4))
  (schema ideal (piston-direction ~?state4))
  =>
  (printout t t "piston-direction compared not ok" t t)
  (assert (error-trap direction)))

(defrule no-ignition
  (state-name power)
  (?ignition <- (ignition fail))
  =>
  (printout t t "sparkplug did not fire" t t)
  (printout t t ?ignition t t)
  (assert (error-trap ignition))
  (retract ?ignition))

(defrule intake-stroke
  (?state-name <- (state-name intake))
  (schema current (carburation ?state3))
  (schema ideal (carburation ?state3))
  (schema current (piston-direction ?state4))
  (schema ideal (piston-direction ?state4))
  =>
  (printout t t "1st Stroke - Intake Completed" t t)
  (modify (schema current (piston-direction ascending)))
  (modify (schema ideal (piston-direction ascending)))
  (retract ?state-name)
  (assert (state-name compression)))

(defrule compression-stroke
  (?state-name <- (state-name compression))
  (schema current (carburation ?state3))
  (schema ideal (carburation ?state3))
  (schema current (piston-direction ?state4))
  (schema ideal (piston-direction ?state4))
  =>
  (printout t t "2nd Stroke - Compression" t t)
  (modify (schema current (piston-direction descending)))
  (modify (schema ideal (piston-direction descending)))
  (assert (state-name compression)))
(defrule power-stroke
  (?state-name <- (state-name power)
  (?ignition <- (ignition fire)
  (schema current (carburation ?state3))
  (schema ideal (carburation ?state3))
  (schema current (piston-direction ?state4))
  (schema ideal (piston-direction ?state4))
  =>
  (printout t t "3rd Stroke - Power" t t)
  (modify (schema current (piston-direction ascending)))
  (modify (schema ideal (piston-direction ascending)))
  (retract ?state-name)
  (assert (state-name exhaust))
  (retract ?ignition))

(defrule exhaust-stroke
  (?state-name <- (state-name exhaust)
  (schema current (carburation ?state3))
  (schema ideal (carburation ?state3))
  (schema current (piston-direction ?state4))
  (schema ideal (piston-direction ?state4))
  =>
  (printout t t "4th Stroke - Exhaust" t t)
  (modify (schema current (piston-direction descending)))
  (modify (schema ideal (piston-direction descending)))
  (retract ?state-name)
  (assert (state-name intake))
)

(defrule switch-plugs
  (?error-flag <- (error-trap ignition)
  (?state-name <- (state-name ?name)
  (schema current (sparkplug2 standby))
  =>
  (modify (schema current (sparkplug1 fail))
            (sparkplug2 good))
  (retract ?error-flag)
  (retract ?state-name)
  (assert (state-name compression))
  (printout t t "Sparkplug1 set to Failed" t t)
  (printout t t "Sparkplug2 reset from standby to good" t t)
  (printout t t "Is an explanation desired? yes or no" t t)
  (assert (explanation-flag = (read))))

(defrule no-plugs
  (?error-flag <- (error-trap ignition)
  (?state-name <- (state-name ?name)
  (schema current (sparkplug2 good))
  (schema current (sparkplug1 fail))
  =>
  (modify (schema current (sparkplug2 fail)))
  (retract ?error-flag)
  (retract ?name)
  (assert (state-name compression))
  (printout t t "Sparkplug2 set to Failed" t t)
  (printout t t "Is an explanation desired? yes or no" t t)
  (assert (explanation-flag = (read))))
(deffrule ignition-fire
  (state-name power)
  (split ((sparkplug1 current good) =>
            ((sparkplug2 current good) =>)))

  =>
  (printout t t "Did sparkplug fire or fail?" t t)
  (assert (ignition -(read))))

(deffrule explanation-1
  ?error-flag <- (error-trap ignition)
  ?ignition <- (ignition ?fire)
  (schema current (sparkplug1 fail))
  (schema current (sparkplug2 good))
  ?expl-flag <- (explanation-flag yes)

  =>
  (printout t t "Sparkplug 1 did not fire. Therefore it was replaced by the backup." t t)
  (retract ?error-flag)
  (retract ?expl-flag))

(deffrule explanation-2
  ?error-flag <- (error-trap ignition)
  ?ignition <- (ignition ?fire)
  (schema current (sparkplug1 fail))
  (schema current (sparkplug2 fail))
  ?expl-flag <- (explanation-flag yes)

  =>
  (printout t t "Sparkplug 2 did not fire. Since it is the backup sparkplug,
      (sparkplug 1 has already been considered failed) there is no remedy.
      However, since it is unusual to have both sparkplugs failed, the problem may reside elsewhere." t t)
  (retract ?error-flag)
  (retract ?expl-flag))
APPENDIX B

Source Code for System MAIN_SIM_MOD3.ADA

a Communication Amplifier Diagnostic/Explanation Software System
package TYP_DEF_SIM is
    type SWITCH is (ON, OFF);
    type FAULT is (ACTIVE, INACTIVE);
    type SOURCE is (A, B);
    type CONTINUE is (YES, NO);

end TYP_DEF_SIM;

PSECT MAP

<table>
<thead>
<tr>
<th>Psect</th>
<th>Hex Size</th>
<th>Dec Size</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00000037</td>
<td>55</td>
<td>TYP_DEF_SIM__$CODE</td>
</tr>
<tr>
<td>1</td>
<td>0000000C</td>
<td>12</td>
<td>TYP_DEF_SIM__$CONSTANT</td>
</tr>
</tbody>
</table>

%ADAC-I-CL_ADDED, Package specification TYP_DEF_SIM added to library
Replaces older version compiled 17-Nov-1986 00:53

PORTABILITY SUMMARY

There are no uses of potentially non-portable constructs
-- Main program for Equipment Simulator/Diagnose

-- Function
-- To simulate operation and measurements of a radio frequency
-- power source and analyze any anomalies. An explanation facility
-- is in place.

-- Input
-- Initial rf source simulation parameters are read from a file
-- by procedure I_O_INIT in package I_O_SIM; thereafter revised
-- values of these parameters are entered interactively via the
-- keyboard.

-- Processing
-- The input parameter oscillator switch positions are checked for validity
-- then, when set properly, the input parameters are fed to the simulator
-- package Y_OUT_MOD which at present only consists of procedure EQUIPMENT_Y_O.
-- Upon return from this procedure the output power and switch positions
-- are monitored. Base upon the readings a tentative diagnosis is made.
-- If an explanation is requested, same is supplied via EXPLAIN_DIAGNOSIS.

-- Output
-- The results of the diagnosis made by procedures PWR_SWITCH_STATUS and
-- OUT_LEVEL in package I_O_SIM_MOD is communicated by them to the CRT
-- output device screen.

-- Algorithm MAIN_SIM_MOD

-- The algorithm used for the driver program is:

-- 1. Initialize the simulator by reading the equipment parameters from
--    a file.

-- 2. While the human monitor desires to continue
--    repeat 2 thru 9;

-- 3. Test if both A & B sources are switched on and if so force a
--    choice;

-- 4. Run the simulator in package Y_OUT_MOD;

-- 5. Test the results of the simulator (package FAULT_ANALYZE):
--    .1 If the output is noise (i.e. -145.0 dbm) then check switches and
--        notify the monitor if the master switch is off or if on if both
--        the A & B switches are in the off position;

--    .2 If all switches are proper and output is greater than noise but less
--        than the output caused by an output amplifier failure then presumably the
--        selected oscillator output is below the threshold needed to excite
--        the amplifier so advise the monitor to switch to the alternate oscillator;

--    .3 If output is greater than that caused by oscillator substandard
--        output but less than that for normal operation then the failure
--        of the output amplifier may have occurred;

--    .4 If the output is at normal level then report normal operation
58         -- 6. Write out the equipment status to the CRT (procedure OUT_PARAMS
59         -- in package I_O_SIM_MOD).
60         --
61         -- 7. Interactively ask the monitor whether an explanation of the diagnosis
62         -- is desired. If so then supply same.
63         --
64         -- 8. Interactively ask the monitor if the simulation is to be continued.
65         --
66         -- 9. If the monitor chooses not to continue then exit.
67         --
68         -- 10. If the choice is to run another simulation the ask what parameters
69         -- are to be changed (i.e. call NEW_PARAMS in package I_O_SIM_MOD).
70         --
71         -- 11. Go back to item 2.
72         --
73         -- end algorithm MAIN_SIM_MOD
74         --
75         --
76         --
77         -- with TYP_DEF_SIM; with Y_OUT_MOD; with I_O_SIM_MOD; with FAULT_ANALYZE;
78         with EXPLAIN_DIAGNOSIS;       --
79         use TYP_DEF_SIM; use Y_OUT_MOD;
80
81         Procedure MAIN_SIM_MOD3 is
82         --
83         -- AGAIN : CONTINUE:= YES;
84         -- EXPLAIN : CONTINUE:= YES;
85         -- EQPT_Y,
86         -- FREQ_SRC_A,
87         -- FREQ_SRC_B : SWITCH;
88         -- REF_FREQ_LEVEL_A,
89         -- REF_FREQ_LEVEL_B : FLOAT;
90         -- EQPT_Y_OUT_LEVEL_FAIL : FAULT;
91         -- EQUIP_Y_OUT_LEVEL,
92         -- EQUIP_Y_OUT_LEVEL_SIM,
93         -- EQUIP_X_OUT_LEVEL : FLOAT;
94
95         begin
96             I_O_SIM_MOD.I_O_INIT(EQPT_Y,FREQ_SRC_A,FREQ_SRC_B,EQPT_Y_OUT_LEVEL_FAIL,
97              REF_FREQ_LEVEL_A,REF_FREQ_LEVEL_B,
98              EQUIP_Y_OUT_LEVEL_SIM,EQUIP_X_OUT_LEVEL);
99         while AGAIN = YES
100             loop
101                 if FREQ_SRC_A=ON and FREQ_SRC_B=ON
102                     then I_O_SIM_MOD.I_O_SWITCH(FREQ_SRC_A,FREQ_SRC_B);
103                         end if;    -- If both "ON" force a choice between A or B "ON"
104         end loop;  -- If either "OFF"...
105         --
106         -- I_O_SIM_MOD.NEWPAGE:
EQUIPMENT_Y_O(EQPT_Y,FREQ_SRC_A,FREQ_SRC_B,
FREQ_LEVL_A,FREQ_LEVL_B,EQPT_Y_OUT_LEVEL_FAIL,
EQUIP_Y_OUT_LEVEL,EQUIP_Y_OUT_LEVEL_FAIL,
EQUIP_Y_OUT_LEVEL); -- Run the Simulator

FAULT_ANALYZE_DIAONOSIS(EQPT_Y,FREQ_SRC_A,FREQ_SRC_B,
EQUIP_Y_OUT_LEVEL_SIM,
EQUIP_Y_OUT_LEVEL,EQUIP_Y_OUT_LEVEL); -- Diagnose fault

I_O_SIM_MOD.OUT_PARAMS(EQPT_Y,FREQ_SRC_A,FREQ_SRC_B,EQPT_Y_OUT_LEVEL_FAIL,
FREQ_LEVL_A,FREQ_LEVL_B,
EQUIP_Y_OUT_LEVEL_SIM,EQUIP_Y_OUT_LEVEL,EQUIP_Y_OUT_LEVEL);

I_O_SIM_MOD.QUERY_EXPLAIN(EXPLAIN); -- Does monitor want an explanation

if EXPLAIN = YES
then
EXPLAIN_DIAGNOSIS.EXPLAIN_DIAGNOSIS(EQPT_Y,FREQ_SRC_A,FREQ_SRC_B,FREQ_LEVL_A,FREQ_LEVL_B,
EQUIP_Y_OUT_LEVEL_SIM,EQUIP_Y_OUT_LEVEL_SIM,
EQUIP_Y_OUT_LEVEL_SIM); -- If explanation desired
end if;

I_O_SIM_MOD.AGAIN_I_O(AGAIN); -- Ask human if wants to repeat simulation

exit when AGAIN = NO;

I_O_SIM_MOD.NEW_PARAMS(EQPT_Y,FREQ_SRC_A,FREQ_SRC_B,EQPT_Y_OUT_LEVEL_FAIL,
FREQ_LEVL_A,FREQ_LEVL_B,
EQUIP_Y_OUT_LEVEL_SIM,EQUIP_Y_OUT_LEVEL_SIM);

end loop;
end MAIN_SIM_MOD3;

PSECT MAP

Psect Hex Size Dec Size Name
0 0000001F3 499 MAIN_SIM_MOD3.§CODE

%ADAC-I-CL_ADDED, Procedure body MAIN_SIM_MOD3 added to library

PORTABILITY SUMMARY

There are no uses of potentially non-portable constructs

LIBRARY SUMMARY

USER2:[TFL.ADADIR.ADALIB]

Unit name Nodes Percent Blocks read read Unit kind
with TYP_DEF_SIM; use TYP_DEF_SIM;

Package I_O_SIM_MOD is

   -- Procedure to initialize the simulator
   procedure I_O_INIT(EQPT_Y, FREQ_SRC_A, FREQ_SRC_B : out SWITCH ; -- Equipmt sw's
     EQPT_Y_OUT_LEVEL_FAIL : out FAULT ; -- HW failure
     REF_FREQ_LEVEL_A, REF_FREQ_LEVEL_B, -- Osc Output
     EQPT_Y_OUT_LEVEL_SIM, -- HW failure output level
     EQUIP_X_OUT_LEVEL : out FLOAT); -- Normal Out

   -- Procedure to test if both or none of ref oscillators are on
   procedure I_O_SWITCH( FREQ_SRC_A, FREQ_SRC_B : in out SWITCH); -- SW off B or A

   -- Obvious what this does
   procedure NEWPAGE;

   -- Test and report
   procedure FAULT_MSG_LIST(MESSAGE_NO : in INTEGER);

   -- Ask if explanation desired
   procedure QUERY_EXPLAIN(EXPLAIN : out CONTINUE);

   -- Present Explanation if requested
   procedure EXPLAIN_MSG_LIST(MESSAGE_NO : in INTEGER);

   -- Present output parameters all at once
   procedure OUT_PARAMS(EQPT_Y, FREQ_SRC_A, FREQ_SRC_B : in SWITCH ; -- Equipmt sw's
     EQPT_Y_OUT_LEVEL_FAIL : out FAULT ; -- HW failure
     REF_FREQ_LEVEL_A, REF_FREQ_LEVEL_B, -- Osc Output
     EQPT_Y_OUT_LEVEL_SIM, -- HW failure output level
     EQUIP_X_OUT_LEVEL, -- Normal Output Power
     EQUIP_Y_OUT_LEVEL : in FLOAT); -- Actual Output

   procedure PARAM_OUT(PWR_LEVEL : in FLOAT);

   -- Read from the keyboard whether eqpt monitor wants to continue the simulation or not
   procedure AGAIN_I_O(AGAIN : out CONTINUE);

   -- If the simulation is to run again this procedure is called to read in new values
   procedure NEW_PARAMS(EQPT_Y, FREQ_SRC_A, FREQ_SRC_B : in out SWITCH ; -- Equipmt sw's
     EQPT_Y_OUT_LEVEL_FAIL : in out FAULT ; -- HW failure
     REF_FREQ_LEVEL_A, REF_FREQ_LEVEL_B, -- Osc Output
     EQUIP_Y_OUT_LEVEL_SIM, -- HW failure output level
     EQUIP_X_OUT_LEVEL : in out FLOAT); -- Normal Out

end I_O_SIM_MOD;
with TEXT_IO; use TEXT_IO;

Package body I_O_SIM_MOD is

package SWC_ENUM_IO is new ENUMERATION_IO SWITCH;
package SRC_ENUM_IO is new ENUMERATION_IO SOURCE;
package FAULT_ENUM_IO is new ENUMERATION_IO FAULT;
package AGAIN_IO is new ENUMERATION_IO CONTINUE;
package PARAM_IO is new FLOAT_IO FLOAT;

-- Initial : FILE_TYPE; -- File of initial parameters for the eqpt simulator

-- This package contains initialization and I/O programs

procedure I_O_INIT(EQPT Y, FREQ_SRC A, FREQ_SRC B : out SWITCH; -- Equipat sw's
EQPT Y OUT LEVEL FAIL : out FAULT; -- HW failure
REF_FREQ_LEVEL A, REF_FREQ_LEVEL B, -- Osc Output
EQUIP Y OUT_LEVEL SIM, -- HW failure output level
EQUIP X OUT_LEVEL : out FLOAT) is -- Normal Output

begin -- Get initial parameter values for the simulator

OPEN(INITIAL.IN FILE, "INITIAL.SIM.DAT");
SWC_ENUM_IO.GET(INITIAL, EQPT Y); SKIP_LINE(INITIAL);
SWC_ENUM_IO.GET(INITIAL, FREQ_SRC A); SKIP_LINE(INITIAL);
SWC_ENUM_IO.GET(INITIAL, FREQ_SRC B); SKIP_LINE(INITIAL);
FAULT_ENUM_IO.GET(INITIAL, EQPT Y OUT LEVEL FAIL); SKIP_LINE(INITIAL);
PARAM_IO.GET(INITIAL, REF_FREQ_LEVEL A); SKIP_LINE(INITIAL);
PARAM_IO.GET(INITIAL, REF_FREQ_LEVEL B); SKIP_LINE(INITIAL);
PARAM_IO.GET(INITIAL, EQUIP Y OUT LEVEL SIM); SKIP_LINE(INITIAL);
PARAM_IO.GET(INITIAL, EQUIP X OUT_LEVEL);
CLOSE(INITIAL);

end I_O_INIT;

procedure I_O_SWITCH(FREQ_SRC A, FREQ_SRC B : in out SWITCH) is

FREQ_SRC_SEL : SOURCE; -- A or B

begin -- This procedure is called from MAIN_SIM

-- This procedure only called when both A & B switches are ON

PUT(" Both frequency sources A & B are 'ON' & inconsistent");
NEW LINE; PUT(" Which one do you want 'ON'? (type 'A' or 'B')");
SRC_ENUM_IO.GET(FREQ_SRC_SEL); SKIP_LINE; NEW_LINE;
if FREQ_SRC_SEL = A
then FREQ_SRC_B := OFF;
else FREQ_SRC_A := OFF;
end if;

end I_O_SWITCH; -- This procedure shouldn't have logic but pass

FREQ_SRC_SEL to another package supporting the simulation. May change this later

procedure NEWPAGE is

begin
NEW PAGE;
end NEWPAGE;
procedure FAULT_MSG_LIST(MESSAGE_NO : in INTEGER) is
   -- Test all on/off switch states, then decide upon
   -- what diagnosis to transmit to equipment monitor
   case MESSAGE_NO is
   when 1 =>
      NEW_LINE; PUT(" Equipment output power is down in the noise level."); -- i.e. -125 dbm
      NEW_LINE;
    when 2 =>
      NEW_LINE; PUT(" The main switch is 'ON' so an internal failure occurred");
      NEW_LINE;
    when 3 =>
      NEW_LINE; PUT("Both oscillators A and B are OFF so no wonder !!");
      NEW_LINE; PUT(" Next pass try turning on either oscillator.");
      NEW_LINE;
    when 4 =>
      NEW_LINE; PUT("The selected oscillator may have completely failed");
      NEW_LINE; PUT(" Next pass try turning on the other oscillator");
      NEW_LINE;
    when 5 =>
      NEW_LINE; PUT(" The Main switch is 'OFF' so no wonder the output is noise");
      NEW_LINE;
    when 6 =>
      NEW_LINE; PUT("The OPERATIONAL DIAGNOSIS is");
      NEW_LINE; PUT("Since the power output is ");
      NEW_LINE;
    when 7 =>
      NEW_LINE; PUT(" The selected oscillator level is probably below threshold ");
      NEW_LINE; PUT(" so next pass switch to the other oscillator.");
      NEW_LINE;
    when 8 =>
      NEW_LINE; PUT(" A degradation of output has occurred ");
      NEW_LINE; PUT(" probably NOT caused by an oscillator failure ");
      NEW_LINE; PUT(" switch to backup unit if it's available");
      NEW_LINE;
    when 9 =>
      NEW_LINE; PUT("Normal operation with normal output. ");
      NEW_LINE;
    when others =>
      NEW_LINE; PUT("ERROR in value of MESSAGE_NO"); NEW_LINE;
   end case;
end FAULT_MSG_LIST;
procedure QUERY_EXPLAIN(EXPLAIN : out CONTINUE) is
  begin -- This communicates with the human monitor asking if an
  -- explanation desired
  PUT(" Do you want to know the reasoning behind the diagnostic decision? (YES/NO));
  AGAIN_IO.GET(EXIT); NEW_LINE(2);
  end QUERY_EXPLAIN;

procedure EXPLAIN_MSG_LIST(MESSAGE_NO : in INTEGER) is
  begin -- This procedure contains some of the diagnostic logic
  -- for analyzing a substandard power output
  case MESSAGE_NO is
  when 10 =>
    NEW_LINE; PUT(" The operational diagnosis explanation is:" );
    NEW_LINE;
  when 11 =>
    NEW_LINE; PUT(" The output measured was very low and noisy");
    NEW_LINE; PUT(" yet the master switch was 'ON',");
    NEW_LINE; PUT(" so the failure was something else.");
  when 12 =>
    NEW_LINE; PUT(" We then found that both the oscillator switches were 'OFF',");
    NEW_LINE; PUT(" so the malfunction was a set-up error.");
    NEW_LINE;
  when 13 =>
    NEW_LINE; PUT(" We found the selected oscillator switch to be 'ON',");
    NEW_LINE; PUT(" so, since output is noise, potentially we had a catastrophic oscillator failure");
    NEW_LINE;
  when 14 =>
    NEW_LINE; PUT(" We found the master switch in the 'OFF' position so ");
    NEW_LINE; PUT(" the malfunction was a set-up error. ");
    NEW_LINE;
  when 15 =>
    NEW_LINE; PUT(" The output was between -145.0 dbm and -90.0 dbm. Thus the output");
    NEW_LINE; PUT(" device was probably OK but the oscillator was below threshold.");
    NEW_LINE; PUT(" The oscillator output was insufficient to drive the output device.");
    NEW_LINE; PUT(" The logical action is to select the other oscillator on the next pass.");
    NEW_LINE;
procedure OUT_PARAMS(EQPT_Y, FREQ_SRC_A, FREQ_SRC_B : in SWITCH; -- Equipment sw's
EQPT_Y_OUT_LEVEL_FAIL : in FAULT; -- HW failure
REF_FREQ_LEVEL_A, REF_FREQ_LEVEL_B, -- Osc output
EQUIP_Y_OUT_LEVEL_SIM, -- HW failure output level
EQUIP_Y_OUT_LEVEL, -- Normal Output Power
EQUIP_Y_OUT_LEVEL : in FLOAT) is
begin -- This procedure outputs the status of all simulation variable
-- database for the current pass of the simulator code
NEW_LINE;
PUT("The Status of the system is ":NEW_LINE[2]);
PUT("Equipment Switch is ":SWC_ENUM_IO.PUT(EQPT_Y);NEW_LINE;
PUT("Frequency Source A is ":SWC_ENUM_IO.PUT(FREQ_SRC_A); -- Switches
PUT("; Frequency Source B is ":SWC_ENUM_IO.PUT(FREQ_SRC_B);NEW_LINE;
PUT("The actual power output of the system is ");
PARAM_IO.PUT(EQUIP_Y_OUT_LEVEL);NEW_LINE;
NEW_LINE[2];
PUT("The status of the equipment and simulator data is: ");
NEW_LINE;PUT(" (these data normally unavailable to observer) ");
NEW_LINE[2];
PUT("The GREMLIN is ");FAULT_ENUM_IO.PUT(EQPT_Y_OUT_LEVEL_FAIL);NEW_LINE;
PUT("The GREMLIN, if active causes system output of < 3 "); -- power lpower levels
PARAM_IO.PUT(EQUIP_Y_OUT_LEVEL_SIM);
NEW_LINE;
PUT("The Oscillator A normal output is ");PARAM_IO.PUT(REF_FREQ_LEVEL_A);
NEW_LINE;
PUT("The Oscillator B normal output is ");PARAM_IO.PUT(REF_FREQ_LEVEL_B);
NEW_LINE;
PUT("The normal system output is ");PARAM_IO.PUT(EQUIP_Y_OUT_LEVEL);
NEW_LINE[2];
end OUT_PARAMS;
procedure PARAM_OUT(PWR_LEVEL : in FLOAT) is
begin
  PARAM_IO.PUT(PWR_LEVEL);
  end PARAM_OUT;

procedure AGAIN_I_O(AGAIN: out CONTINUE) is
begin
  -- Ask the equipment monitor whether or not to continue
  NEW_LINE;
  PUT(" Do you wish to change some parameters and try again ? ");
  PUT("(type in YES/NO "");AGAIN_IO.GET(AGAIN); SKIP_LINE;
  NEW_LINE(2);
  end AGAIN_I_O;

procedure NEW_PARAMS(EQT_Y, FREQ_SRC_A, FREQ_SRC_B : in out SWITCH; -- Equipmt sw's
  EQUIP_Y_OUT_LEVEL_FAIL : in out FAULT; -- HW failure
  REF_FREQ_LEVEL_A, REF_FREQ_LEVEL_B, -- Osc output
  EQUIP_Y_OUT_LEVEL_SIM, -- HW failure output level
  EQUIP_Y_OUT_LEVEL : in out FLOAT) is -- Normal out

  SubType INPUT_STRING is STRING(1..9);
  STRLEN : NATURAL; -- Length of the string read from the keyboard buffer
  LOCAL_STRING: INPUT_STRING; -- Declaration of max length to be read
  CONVLEN : POSITIVE; -- Dummy local var containing len of enum attrib
begin
  -- Display present parameter values and solicit new ones
  PUT("Enter new values and <CR> or just <CR> to let value stand as is");NEW_LINE(2);

  -- Show master switch position (ON/OFF) and ask for new value
  PUT("Equipment switch is ");SWC_ENUM_IO.PUT(EQT_Y);NEW_LINE;
  PUT("Enter new value (ON/OFF) ");
  GET_LINE(LOCAL_STRING,STRLEN); -- Get new input
  if STRLEN /= INPUT_STRING'FIRST-1 then
    then SWC_ENUM_IO.GET(LOCAL_STRING,EQT_Y,CONVLEN); -- then enter into DBMS
  end if;

  -- Show Frequency sources switch positions and ask for new values
  PUT("Freq source A is ");SWC_ENUM_IO.PUT(FREQ_SRC_A);NEW_LINE;
  PUT("Enter new value (ON/OFF) ");GET_LINE(LOCAL_STRING,STRLEN);
  if STRLEN /= INPUT_STRING'FIRST-1 then
    then SWC_ENUM_IO.GET(LOCAL_STRING,FREQ_SRC_A,CONVLEN); -- convert string to its enumeration type
  end if;

  PUT("Freq source B is ");SWC_ENUM_IO.PUT(FREQ_SRC_B);NEW_LINE;
  PUT("Enter new value (ON/OFF) ");GET_LINE(LOCAL_STRING,STRLEN);
  if STRLEN /= INPUT_STRING'FIRST-1 then
    then SWC_ENUM_IO.GET(LOCAL_STRING,FREQ_SRC_B,CONVLEN);
  end if;

  -- Present the simulated failure status (ACTIVE/INACTIVE) ask which?
  PUT("The GREMLIN is ");FAULT_ENUM_IO.PUT(EQT_Y_OUT_LEVEL_FAIL);
  NEW_LINE;
PUT( "Enter new value (ACTIVE/INACTIVE) "); GET_LINE(LOCAL_STRING,STLEN);
if STLEN /= INPUT_STRING'FIRST-1
then FAULT_ENUM_IO.GET(LOCAL_STRING,EQPT_Y_OUT_LEVEL_FAIL,CONVLEN);
end if; LOCAL_STRING := " "; NEW_LINE;

-- Present the output power levels for each oscillator and ask for updates

PUT("The Oscillator A output is "); PARAM_IO.PUT(REF_FREQ_LEVEL_A);
NEW_LINE;
PUT(" Enter a new value for Oscillator A power output in dbm "); NEW_LINE;
PUT(" (must be > -145.0 ) "); GET_LINE(LOCAL_STRING,STLEN);
if STLEN /= INPUT_STRING'FIRST-1
then PARAM_IO.GET(LOCAL_STRING,REF_FREQ_LEVEL_A,CONVLEN);
end if; LOCAL_STRING := " "; NEW_LINE;

PUT("The Oscillator B output is "); PARAM_IO.PUT(REF_FREQ_LEVEL_B);
NEW_LINE;
PUT(" Enter a new value for Oscillator B power output in dbm "); NEW_LINE;
PUT(" (must be > -145.0 ) "); GET_LINE(LOCAL_STRING,STLEN);
if STLEN /= INPUT_STRING'FIRST-1
then PARAM_IO.GET(LOCAL_STRING,REF_FREQ_LEVEL_B,CONVLEN);
end if; LOCAL_STRING := " "; NEW_LINE;

-- Present the normal operation power and ask for parameter change

PUT("The normal system power output is "); PARAM_IO.PUT(EQUIP_X_OUT_LEVEL);
NEW_LINE;
PUT(" Enter a new value for system output if desired, ");
GET_LINE(LOCAL_STRING,STLEN);
if STLEN /= INPUT_STRING'FIRST-1
then PARAM_IO.GET(LOCAL_STRING,EQUIP_X_OUT_LEVEL,CONVLEN);
end if; LOCAL_STRING := " "; NEW_LINE;

-- Present the output power level when an output device fails (dbm) and inquire

PUT("The output power when output device has failed, "); NEW_LINE;
PUT(" i.e. ACTIVE GREMLIN is "); PARAM_IO.PUT(EQUIP_Y_OUT_LEVEL_SIM);
NEW_LINE;
PUT(" Enter a new value for this 'GREMLIN' power out if desired, ");
NEW_LINE; PUT("note that it must be less than normal power out, ");
NEW_LINE; PUT(" but must also be > -90.0 "); GET_LINE(LOCAL_STRING,STLEN);
if STLEN /= INPUT_STRING'FIRST-1
then PARAM_IO.GET(LOCAL_STRING,EQUIP_Y_OUT_LEVEL_SIM,CONVLEN);
end if; LOCAL_STRING := " "; NEW_LINE(2);

-- Ready to loop back and do it again

PUT_LINE(" Ready or not; Here we go again !! ");
NEW_LINE;
end NEW_PARAMS;
end I_O_SIM_MOD;
with TYP_DEF_SIM; use TYP_DEF_SIM;

package Y_OUT_MOD is
  procedure EQUIPMENT_Y_S (EQPT_Y : in SWITCH; -- Equipment Switch
                           FREQ_SRC_A,FREQ_SRC_B : inwitch; -- Osc Sw's
                           REF_FREQ_LEVEL_A,
                           REF_FREQ_LEVEL_B : in FLOAT; -- Osc Outputs in dbm
                           EQPT_Y_OUT_LEVEL_FAIL : in FAULT; -- Fail Flag
                           EQPT_Y_OUT_LEVEL_NORMAL : -- Normal output > 20.0
                           EQPT_Y_OUT_LEVEL_SIM : in FLOAT; -- Fail out Pwr < 20.0
                           EQPT_Y_OUT_LEVEL_OUT : out FLOAT; -- Equip Pwr Out (actual)
  end Y_OUT_MOD;

  0000 Y_OUTMOD $ELAB:
       0000                     .entry          Y_OUTMOD $ELAB, `mv(dv,iv)
       0002                     subl2          $12, ap
      0005                     moval          $CONSTANT, ap
      000C                     moval          (ap), ap
      000F                     movl           ap, ap
      0012                     movl           ap,-12(fp)
      0016                     movaq          ADA$HANDLER, ap
      001D                     movl           ap, (fp)
      0020                     moval          $CONSTANT, ap
      0027                     movab          4(ap), ap
      002B                     movl           ap, ap
      002E                     movl           ap,-12(fp)

  0032                     movl           #0,-12(fp)
  0036 Y_OUTMOD $ELAB$RTN $LBL25:
        04 0036 ret

PSECT MAP

Psect  Hex Size  Dec Size Name
       0  00000037  55  Y_OUTMOD $.CODE
       1  0000000C  12  Y_OUTMOD $.CONSTANT

%ADAC-I-CL_ ADDED, Package specification Y_OUT_MOD added to library
Replaces older version compiled 27-Nov-1986 00:15

PORTABILITY SUMMARY

There are no uses of potentially non-portable constructs

LIBRARY SUMMARY

USER2:(TFL.ADIR.ADALIB)
Package body Y_OUT_MOD is

procedure EQUIPMENT_Y_O (EQPT Y : in SWITCH; -- Equipment Switch
  FREQ SRC_A, FREQ SRC_B: in SWITCH; -- Osc Sw's
  REF_FREQ_LEVEL A,
  REF_FREQ_LEVEL B : in FLOAT; -- Osc Output in dbm
  EQPT Y OUT LEVEL FAIL: in FAULT; -- Fail Flag
  EQUIP Y OUT LEVEL: -- Normal output > 20.0
  EQUIP Y OUT_LEVEL_SIM : in FLOAT; -- Fail Out Pwr
  EQUIP Y OUT_LEVEL : out FLOAT) is -- Equip Pwr Out (actual)

  Ref_freq_level, Equip_Y_output_internal: FLOAT; -- Internal variables

begin
  if EQPT Y = ON
    then if FREQ_SRC_A = ON
      then Ref_freq_level := REF_FREQ_LEVEL_A;
      else if FREQ_SRC_B = ON
        then Ref_freq_level := REF_FREQ_LEVEL_B;
        else Ref_freq_level := -145.0;
      end if;
    end if;
  if (FREQ_SRC_A = ON) and (REF_FREQ_LEVEL_A <= -120.0)
    then Ref_freq_level := -145.0;
  else if (FREQ_SRC_B = ON) and (REF_FREQ_LEVEL_B <= -120.0)
    then Ref_freq_level := -145.0;
  end if;
  if Ref_freq_level <= -145.0
    then Equip_Y_output_internal := -145.0;
  else if Ref_freq_level < -5.0
    then Equip_Y_output_internal := -90.0;
  else if EQPT Y_OUT LEVEL FAIL = ACTIVE
    then Equip_Y_output_internal := EQUIP Y_OUT_LEVEL_SIM;
    else Equip_Y_output_internal := EQUIP X OUT_LEVEL;
  end if;
  end if;
  if EQUIP Y_OUT LEVEL := Equip_Y_output_internal:
  else EQUIP Y OUT_LEVEL := -145.0;
  end if;
end EQUIPMENT Y_O;
end Y_OUT_MOD;

PSECT MAP

<table>
<thead>
<tr>
<th>Psect</th>
<th>Hex Size</th>
<th>Dec Size</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0000010B</td>
<td>267</td>
<td>Y_OUT_MOD.$C</td>
</tr>
<tr>
<td></td>
<td>0000000C</td>
<td>12</td>
<td>Y_OUT_MOD.$C</td>
</tr>
<tr>
<td></td>
<td>00000001</td>
<td>1</td>
<td>Y_OUT_MOD.$D</td>
</tr>
</tbody>
</table>

%ADAC-I-CL_ADDED, Package body Y_OUT_MOD added to library
Corresponds to package specification Y_OUT_MOD compiled 27-Nov-1986 00:25
with TYP_DEF_SIM; use TYP_DEF_SIM;
with I_O_SIM_MOD;
package FAULT_ANALYZE is
procedure SWITCH_STATUS(EQPT_Y, FREQ_SRC_A, FREQ_SRC_B : in SWITCH);
procedure OUT_LEVEL(EQUIP_Y_OUT_LEVEL_SIM, EQUIP_Y_OUT_LEVEL, EQUIP_X_OUT_LEVEL : in FLOAT);
procedure DIAGNOSIS(EQPT_Y, FREQ_SRC_A, FREQ_SRC_B : in SWITCH; EQUIP_Y_OUT_LEVEL_SIM, EQUIP_Y_OUT_LEVEL, EQUIP_X_OUT_LEVEL : in FLOAT);
end FAULT_ANALYZE;

Pselect MAP

<table>
<thead>
<tr>
<th>Psect</th>
<th>Hex Size</th>
<th>Dec Size</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 00000020</td>
<td>32</td>
<td>FAULT_ANALYZE_$CODE</td>
</tr>
<tr>
<td></td>
<td>1 0000000C</td>
<td>12</td>
<td>FAULT_ANALYZE_$CONSTANT</td>
</tr>
</tbody>
</table>

%ADAC-I-CL ADDED, Package specification FAULT_ANALYZE added to library
Replaces older version compiled 18-Jan-1987 21:21

PORTABILITY SUMMARY
There are no uses of potentially non-portable constructs

LIBRARY SUMMARY

USER2:[TFL.ADADIR.ADALIB]

<table>
<thead>
<tr>
<th>Unit name</th>
<th>Nodes</th>
<th>Percent</th>
<th>Blocks</th>
<th>Unit kind</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYP_DEF_SIM</td>
<td>7</td>
<td>41</td>
<td>7</td>
<td>Package specification</td>
</tr>
<tr>
<td>I_O_SIM_MOD</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>Package specification</td>
</tr>
</tbody>
</table>
package body FAULT_ANALYZE is

MESSAGE_NO : INTEGER;

procedure SWITCH_STATUS(EQPT_Y,FREQ_SRC_A,FREQ_SRC_B : in SWITCH) is

-- Called by MAIN_SIM
-- Test all on/off switch stati, then decide upon
begin -- what diagnosis to transmit to equipment monitor

MESSAGE_NO := 1; -- PUT(" Equipment output power is down in the noise level. ")
I_O_SIM_MOD.FAULT_MSG_LIST(MESSAGE_NO);
if EQPT_Y = ON
then
  MESSAGE_NO := 2 ; -- Main switch is on so must be other failure
  I_O_SIM_MOD.FAULT_MSG_LIST(MESSAGE_NO);
  if (FREQ_SRC_A=OFF) and (FREQ_SRC_B=OFF)
  then
    MESSAGE_NO := 3;
    -- PUT("Both oscillators A and B are OFF so no wonder !!!");
    -- PUT(" Next pass try turning on either oscillator.");
    I_O_SIM_MOD.FAULT_MSG_LIST(MESSAGE_NO);
  else
    MESSAGE_NO := 4; -- Selected oscillator must be inoperative so
    -- Next time select the other oscillator
    I_O_SIM_MOD.FAULT_MSG_LIST(MESSAGE_NO);
  end if;
else
  MESSAGE_NO := 5; -- Main Switch is 'OFF' that explains the poor output
  I_O_SIM_MOD.FAULT_MSG_LIST(MESSAGE_NO);
  end if;

end;

procedure OUT_LEVEL(EQUIP_Y_OUT_LEVEL_SIM, EQUIP_Y_OUT_LEVEL,
                       EQUIP_X_OUT_LEVEL : in FLOAT) is

begin -- This procedure contains some of the diagnostic logic
-- for analyzing a substandard power output
MESSAGE_NO := 6;
-- PUT(" The OPERATIONAL DIAGNOSIS is:");
-- PUT(" The power output is ");
I_O_SIM_MOD.FAULT_MSG_LIST(MESSAGE_NO);
I_O_SIM_MOD.PARAM_OUT(EQUIP_Y_OUT_LEVEL);
if EQUIP_Y_OUT_LEVEL <= -90.0
then
  MESSAGE_NO := 7;
  -- PUT(" The reference oscillator level is below threshold ")
  -- PUT(" so next pass choose the other oscillator. ");
  I_O_SIM_MOD.FAULT_MSG_LIST(MESSAGE_NO);
else
  if EQUIP_Y_OUT_LEVEL <= EQUIP_Y_OUT_LEVEL_SIM
  then
    MESSAGE_NO := 8;
    -- PUT(" A degradation of output has occurred.");
    -- PUT(" caused by an output device Failure.");
  else
    MESSAGE_NO := 9;
    -- PUT(" output level is normal so forget ");
    -- PUT(" as second pass is normal ");
    I_O_SIM_MOD.FAULT_MSG_LIST(MESSAGE_NO);
  end if;
end if;

end;

end FAULT_ANALYZE;
I_O_SIM_MOD.FAULT.MSG_LIST(MESSAGE_NO);
else
if EQUIP_Y_OUT_LEVEL <= EQUIP_X_OUT_LEVEL
then
MESSAGE_NO := 9;
--PUT(" Normal operation with normal output ");
I_O_SIM_MOD.FAULT.MSG_LIST(MESSAGE_NO);
end if;
end if;
end OUT_LEVEL;

procedure DIAGNOSIS(EQPT_Y,FREQ_SRC_A,FREQ_SRC_B : in SWITCH;
EQUIP_Y_OUT_LEVEL_SIM,
EQUIP_Y_OUT_LEVEL,
EQUIP_X_OUT_LEVEL : in FLOAT) is
begin
if EQUIP_Y_OUT_LEVEL <= -145.0
then SWITCH_STATUS(EQPT_Y,FREQ_SRC_A,FREQ_SRC_B);
else OUT_LEVEL(EQUIP_Y_OUT_LEVEL_SIM, EQUIP_Y_OUT_LEVEL,
EQUIP_X_OUT_LEVEL);
end if;
end DIAGNOSIS;
end FAULT_ANALYZE;

PSECT MAP

Psect Hex Size Dec Size Name
0 000000269 617 FAULT_ANALYZE.$CODE
1 0000000C 12 FAULT_ANALYZE.$CONSTANT
2 00000008 8 FAULT_ANALYZE.$DATA

%ADAC-I-CL ADDED, Package body FAULT_ANALYZE added to library
% PORTABILITY SUMMARY
There are no uses of potentially non-portable constructs

LIBRARY SUMMARY
USER2: [TFL.ADA_DIR.ADALIB]
Program body Explain_Diagnosis is

procedure EXPLAIN_DIAGNOSIS(EQPT Y, FREQ_SRC_A,FREQ_SRC_B : in SWITCH; -- Equipm sw's
   REF_FREQ_LEVEL_A, REF_FREQ_LEVEL_B, -- Osc Out
   EQUIP_Y_OUT_LEVEL_SIM, -- HW failure out level
   EQUIP_X_OUT_LEVEL , -- Normal Out Power
   EQUIP_Y_OUT_LEVEL : in FLOAT;
   EXPLAIN : in out CONTINUE) is

   MESSAGE_NO : INTEGER; -- Local variable to contain number of msg
   -- for the message print procedure
   -- This procedure provides an explanation of the reasoning behind the
   -- diagnosis of the equipment's ills.
   -- It is called from MAIN_SIM.

   begin

   MESSAGE_NO := 10;
   I_O_SIM_MOD.EXPLAIN_MSG_LIST(MESSAGE_NO);-- Print Header
   if EQUIP_Y_OUT_LEVEL <= -145.0
      then
         if EQPT_Y = ON
            then
               MESSAGE_NO := 11;
               -- The output measured was very low and noisy.;
               -- yet the master switch was ON.;
               -- Obviously was some other failure.);
               I_O_SIM_MOD.EXPLAIN_MSG_LIST(MESSAGE_NO);
               if (FREQ_SRC_A=OFF) and (FREQ_SRC_B=OFF)
                  then
                     MESSAGE_NO := 12;
                     -- We then found both the oscillator switches were OFF, );
                     -- so the malfunction was a set up error.);
                     I_O_SIM_MOD.EXPLAIN_MSG_LIST(MESSAGE_NO);
               else
                  MESSAGE_NO := 13;
                  -- [We found the selected oscillator switch to be ON,]);
                  -- [so potentially we had a catastrophic oscillator failure.];
                  I_O_SIM_MOD.EXPLAIN_MSG_LIST(MESSAGE_NO);
               end if;
            else
               MESSAGE_NO := 14;
               -- (The master switch was found to be OFF so the malfunction);
               -- (was a set up error.);
               I_O_SIM_MOD.EXPLAIN_MSG_LIST(MESSAGE_NO);
            end if;
         else
            if EQUIP_Y_OUT_LEVEL <= -90.0
               then
                  MESSAGE_NO := 15;
                  -- [The out was between -145.0 dbm and -90.0 dbm. Thus the out ];
                  -- [device was probably OK but the oscillator was below threshold.];
                  -- (The oscillator out was insufficient to drive the out device);
                  -- (Obviously on the next pass one should choose the other oscillator. )
                  I_O_SIM_MOD.EXPLAIN_MSG_LIST(MESSAGE_NO);
            end if;
         end if;
   end if;
end if;
else
    if EQUIP_Y_OUT_LEVEL <= EQUIP_Y_OUT_LEVEL_SIM
        then
            MESSAGE_NO := 16;
            -- (A degradation of out had occurred, the out was above);
            --( -90.0 dbm but was still less than or equal to the out failure dbm.);
            --(This indicates that the selected oscillator was OK.);
            --( since some signal was 'leaking' through the circuitry);)
            --( but there was probably an out device Failure.);
            I_O_SIM_MOD.EXPLAIN_MSG_LIST(MESSAGE_NO);
        end if;
    end if;
else
    if EQUIP_Y_OUT_LEVEL <= EQUIP_X_OUT_LEVEL
        then
            MESSAGE_NO := 17;
            --( Normal operation with normal or near normal out );
            --( since the out is above the out failure level.)
            I_O_SIM_MOD.EXPLAIN_MSG_LIST(MESSAGE_NO);
        end if;
    end if;
    end EXPLAIN_DIAGNOSIS;
end if;
end EXPLAIN_DIAGNOSIS;

PSELECT MAP

<table>
<thead>
<tr>
<th>Psect</th>
<th>Hex Size</th>
<th>Dec Size</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000000F9</td>
<td>249</td>
<td>Explain_Diagnosis.SCODE</td>
</tr>
<tr>
<td>1</td>
<td>0000000C</td>
<td>12</td>
<td>Explain_Diagnosis.SCONSTANT</td>
</tr>
<tr>
<td>2</td>
<td>00000001</td>
<td>1</td>
<td>Explain_Diagnosis.SDATA</td>
</tr>
</tbody>
</table>

%ADAC-I-CL_ADDED, Package body EXPLAIN_DIAGNOSIS added to library
Replaces older version compiled 11-Jan-1987 02:22
Corresponds to package specification EXPLAIN_DIAGNOSIS compiled 11-Jan-1987 02:29

PORTABILITY SUMMARY

There are no uses of potentially non-portable constructs

LIBRARY SUMMARY

USER2:[TFL.ADA_DIR.ADALIB]

<table>
<thead>
<tr>
<th>Unit name</th>
<th>Nodes</th>
<th>Percent</th>
<th>Blocks</th>
<th>Unit kind</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPLAIN_DIAGNOSIS</td>
<td>16</td>
<td>100</td>
<td></td>
<td>Package specification</td>
</tr>
<tr>
<td>TYP_DEF_SIM</td>
<td>16</td>
<td>94</td>
<td>9</td>
<td>Package specification</td>
</tr>
<tr>
<td>I_O_SIM_MOD</td>
<td>7</td>
<td>12</td>
<td>8</td>
<td>Package specification</td>
</tr>
<tr>
<td>OFF</td>
<td>ACTIVE</td>
<td>3.0</td>
<td>7.0</td>
<td>-60.0</td>
</tr>
<tr>
<td>-----</td>
<td>--------</td>
<td>-----</td>
<td>-----</td>
<td>-------</td>
</tr>
</tbody>
</table>

INITIAL_SIM.DAT
<table>
<thead>
<tr>
<th>Section Name</th>
<th>Module Name</th>
<th>Base</th>
<th>End</th>
<th>Length</th>
<th>Align</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CODE</td>
<td>ADA$ELAB_MAIN_SIM_MOD3</td>
<td>0000160D 00000480D 00000320E</td>
<td>12814</td>
<td>LONG 2</td>
<td>PIC, USR, CON, REL, LCL, SHR, EXE, RD, NOWRT, NOVEC</td>
<td></td>
</tr>
<tr>
<td>TYP_DEF_SIM</td>
<td>00001600 0000161D 0000001E</td>
<td>30</td>
<td>LONG 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO_EXCEPTIONS</td>
<td>00001655 0000165A 00000006</td>
<td>55</td>
<td>BYTE 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y_OUT_MOD</td>
<td>00001658 00001691 00000037</td>
<td>55</td>
<td>BYTE 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y_OUT_MOD</td>
<td>00001692 0000179C 0000010B</td>
<td>267</td>
<td>BYTE 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEXT_TO</td>
<td>0000179D 000017B3 00000017</td>
<td>23</td>
<td>BYTE 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEXT_TO</td>
<td>00001784 00001865 000000B2</td>
<td>178</td>
<td>BYTE 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I_O_SIM_MOD</td>
<td>00001666 00001885 00000020</td>
<td>32</td>
<td>BYTE 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I_O_SIM_MOD</td>
<td>00001886 00004278 000029F3</td>
<td>10739</td>
<td>BYTE 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXPLAIN_DIAGNOSIS</td>
<td>00004279 00004298 00000020</td>
<td>32</td>
<td>BYTE 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXPLAIN_DIAGNOSIS</td>
<td>00004299 00004391 000000F9</td>
<td>249</td>
<td>BYTE 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FAULT_ANALYZE</td>
<td>00004392 000043B1 00000020</td>
<td>32</td>
<td>BYTE 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FAULT_ANALYZE</td>
<td>000043B2 0000461A 00000269</td>
<td>617</td>
<td>BYTE 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAIN_SIM_MOD3</td>
<td>0000461B 0000480D 000001F3</td>
<td>499</td>
<td>BYTE 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Key for special characters above:

+-----------------+
| ! * - Undefined |
| ! U - Universal  |
| ! R - Relocatable |
| ! X - External  |
| ! V - Vectored  |
+-----------------+
Virtual memory allocated:
Stack size:
Image header virtual block limits:
Image binary virtual block limits:
Image name and identification:
Number of files:
Number of modules:
Number of program sections:
Number of global symbols:
Number of image sections:
User transfer address:
Debugger transfer address:
Number of code references to shareable images:
OTS transfer address - LIB$INITIALIZE:
Image type:
Map format:
Estimated map length:

Performance Indicators

<table>
<thead>
<tr>
<th>Command processing:</th>
<th>Page Faults</th>
<th>CPU Time</th>
<th>Elapsed Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command processing:</td>
<td>145</td>
<td>00:00:00.16</td>
<td>00:00:00.18</td>
</tr>
<tr>
<td>Allocation/Relocation:</td>
<td>417</td>
<td>00:00:02.33</td>
<td>00:00:04.05</td>
</tr>
<tr>
<td>Pass 2:</td>
<td>31</td>
<td>00:00:00.09</td>
<td>00:00:00.46</td>
</tr>
<tr>
<td>Map data after object module synopsis:</td>
<td>63</td>
<td>00:00:01.21</td>
<td>00:00:02.59</td>
</tr>
<tr>
<td>Symbol table output:</td>
<td>4</td>
<td>00:00:00.10</td>
<td>00:00:00.10</td>
</tr>
<tr>
<td>Total run values:</td>
<td>660</td>
<td>00:00:00.02</td>
<td>00:00:00.14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>00:00:03.91</td>
<td>00:00:07.52</td>
</tr>
</tbody>
</table>

Using a working set limited to 1350 pages and 686 pages of data storage (excluding image)

Total number object records read (both passes): 944
of which 41 were in libraries and 211 were DEBUG data records containing 26139 bytes
2835 bytes of DEBUG data were written, starting at VBN 38 with 6 blocks allocated

Number of modules extracted explicitly
with 2 extracted to resolve undefined symbols

0 library searches were for symbols not in the library searched

A total of 0 global symbol table records was written

LINK/MAP=[]MAIN_SIM_MOD3/EXEC=[]MAIN_SIM_MOD3 SYSSINPUT:/OPTIONS