Broad Range of Programmable Fault Tolerance in Transition Machine Computer Architectures

R. Fred Vaughan
Principal Engineer

The Boeing Aerospace Company
Seattle, Washington

Abstract

Transition Machine computer architectures have been investigated by the Boeing Aerospace Company over the last several years primarily as a means of obtaining large amounts of throughput at low cost and secondarily as a means for reducing software development costs for aerospace systems (1,2,3,4,5). The coupling of pluralities of identical components in Transition Machine architectures has suggested fault tolerance as a readily achievable goal. Recent research has confirmed this expectation by defining classes of redundant component organizations endowed with superior reliabilities. A broad range of source programmable fault tolerance has been defined for these organizations, comprising a spectrum from simple fault detection up through triplicated processing pipes, each with its own failure correction capability. The approach encompasses HOL commands to effect a user-specified degree of fault tolerance. These programmed designations can be based upon the criticality of the individual computations to the overall avionics mission success. The automated translation of these sections of the source program into an executable task organization proceeds according to one of several algorithms appropriate to the specified degree of fault tolerance.

Introduction

Transition Machines are a class of computer architectures implementing a situation/response computation structure. They embody separate system control elements designed to maintain and evaluate the status of activating situations defined for each response, these being performed by conventional processors configured in the system.

Previous research has shown that system control elements can be designed for very efficient operation. This feature supports extensive multiprocessing even where very small tasks (the responses) are defined (2). (See Figure 1.) This in itself suggests the possibility of exploiting the dynamic reconfigurability of multiprocessors, but as recent studies have shown, even more fault tolerance advantages accrue with this unique architecture.

A software development support system has been developed which translates conventional higher order languare (HOL) programs into the situation/repsonse structure of the Transition Machine (6).

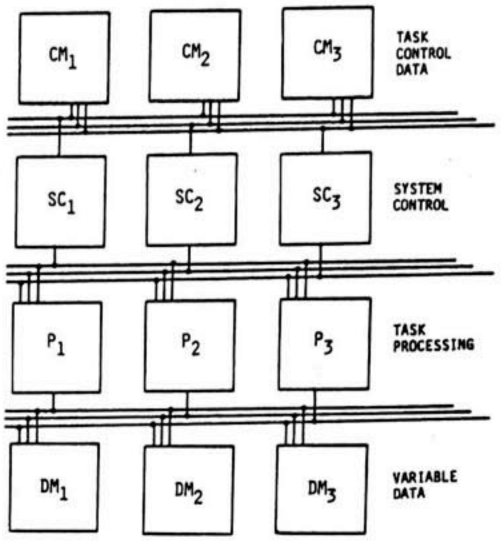

Figure 1: Typical Transition Machine Organization

(See Figure 2.) This translation capability provides a basis for incorporating standardized programmer-keyed permutations to the translation algorithm. The output of the translator can then

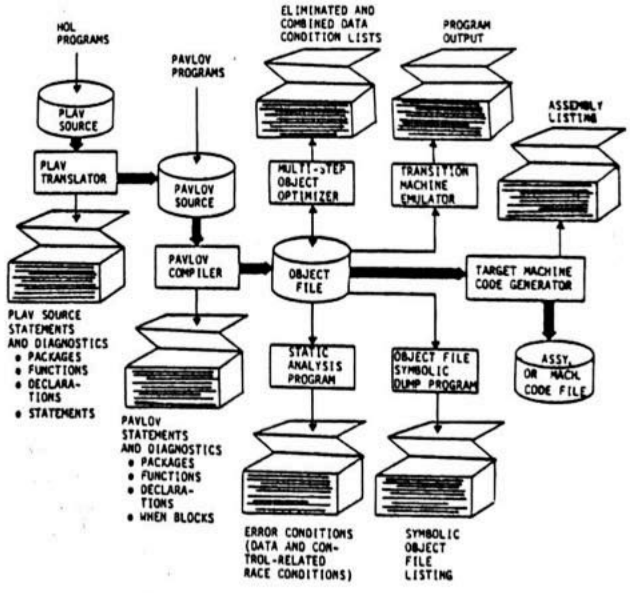
HOL PROGRAMS
PLAV SOURCE
PLAV TRANSLATOR
PAVLOV PROGRAMS
PAVLOV SOURCE
PAVLOV COMPILER
ELIMINATED AND COMBINED DATA CONDITION LISTS
MULTI-STEP OBJECT OPTIMIZER
PROGRAM OUTPUT
TRANSITION MACHINE EMULATOR
ASSEMBLY LISTING
OBJECT FILE
TARGET MACHINE CODE GENERATOR
ASSY. OR MACH. CODE FILE
STATIC ANALYSIS PROGRAM
OBJECT FILE SYMBOLIC DUMP PROGRAM
PLAV SOURCE STATEMENTS AND DIAGNOSTICS
• PACKAGES
• FUNCTIONS
• DECLARATIONS
• STATEMENTS
PAVLOV STATEMENTS AND DIAGNOSTICS
• PACKAGES
• FUNCTIONS
• DECLARATIONS
• WHEN BLOCKS
ERROR CONDITIONS (DATA AND CONTROL-RELATED RACE CONDITIONS)
SYMBOLIC OBJECT FILE LISTING

Figure 2: Transition Machine Support Software

provide low level redundancy, eror detection, recovery correction and even the allocation of dedicated redundant processing pipes, each federating unique segments of the tightly coupled multiprocessor system as resources in an isolated stream of computations. These features, although implemented to a large extent as software on Transition Machines, provide fault coverage which is more typical of hardware implementations, and at a much lower cost in number of components.

Reliability Considerations

In general where n identical components are coupled in such a way that all are active and anyone of the components can accomplish any of the designated functions, with n-k of the components required to accomplish all functions, the reliabiity of the system is given by:

$$r\ (n,\ k,\ \lambda t) = \sum_{i=0}^{k} \frac{n!(-1)^{n-i}}{i!(n-i)!} e^{-(n-i)\lambda t}(1-e^{-\lambda t})^{i}$$

where λ is the failure rate of each individual component and t is the duration of operation. It is assumed that the reliability of a single component is $e^{-\lambda t}$.

As a concrete example to illustrate reliability differences, consider an avionic system comprised of two critical functions, each requiring a processor and data storage. In figure 3 a configuration is defined which handles the two functions separately, each with a designated backup processor. In figure 4 the same functions are handled by a configuration of processsors for which any processor can satisfy either function.

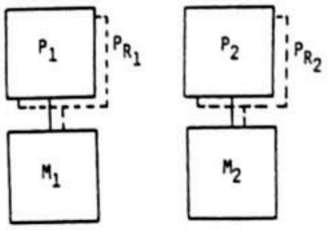

Figure 3: Example Uncoupled Implementation of Two Avionic Functions

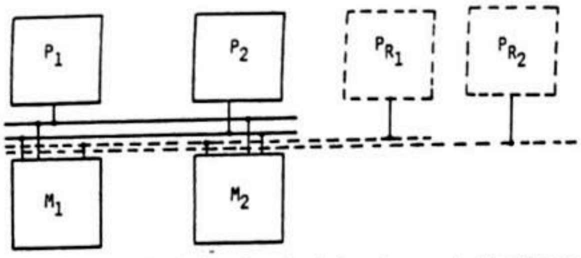

Figure 4: Tightly Coupled Counterpart Configuration

If the functions each require a full processor capability, then n-1 failures would fail the configuration shown in figure 4. But, if each function only required one half (or less) of the capability of the configured processors, or if acceptable degraded modes of operation could be defined, then it would take n processor failures to completely fail the system configured as shown in figure 4. For comparison, n-1 failures would always fail the configuration of figure 3. Certain failure combinations would fail it with n/2 component failures.

For the configuration shown in figure 3, the processor reliability is $R_1 = r^2$ where r is the reliability of each of the two functional halves of the system.

Plotted in figure 5 are reliability curves for the two basic configurations with different numbers of configured processors, n, and success criteria, k.

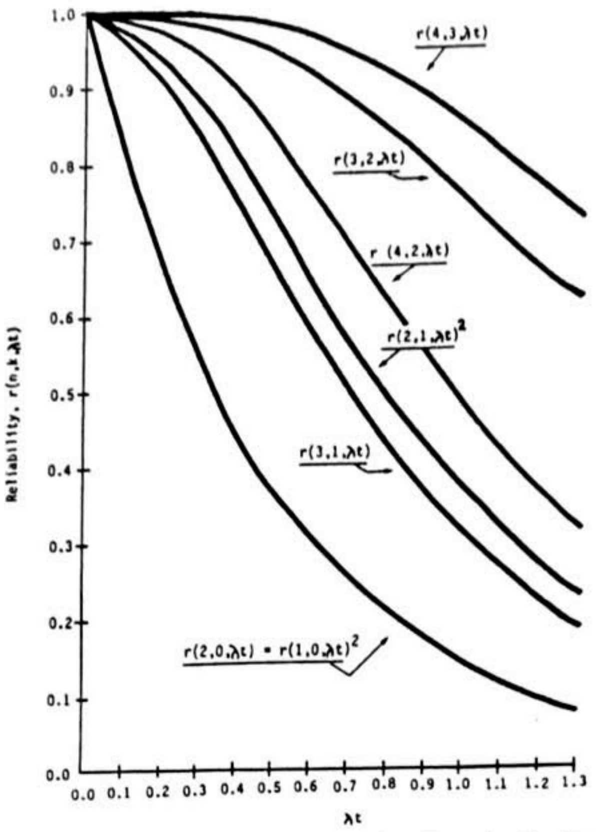

Figure 5: Reliability Curves for Example Configurations

Clearly with no redundancy and no degraded modes defined, the two configurations have equivalent reliabilities. Reliability enhancements are very significant for the tightly coupled configuration when four processors are configured however.

Furthermore, as the number of configured components increases, so does the advantage of tight coupling over functional dedication. This advantage becomes even more pronounced when one includes the parallel access memories, interfaces and system control elements featured in transition machine architectures. Negative aspects of tightly coupled configurations from a reliability point of view are the added complexity of the interconnections and the requirement to detach failed but active components from the system. These aspects are not treated in depth here, but do not in any case nulify the advantages otherwise envisioned. Where very large numbers of components are configured however, reduced coupling can be effected without eliminating reliability advantages as will be shown.

Configuration Variations

The Transition Machine's organization can be adjusted to support variant versions of fault tolerant, reconfigurable systems. A basic configuration was shown in fiqure 1 which supports reconfiguration around any single-point failure with slightly degraded modes of operation. The fault detection and component switching logic must be implemented as a part of the resident system software controlled by the system control element. Specific methods will be discussed further on. For many applications however the flexibility of completely interconnected components is unnecessary. For example the front-end processor responsible for entry of signal data from various sources, where cursory checking and simple protocol are all that is required on each, could be implemented as shown in figure 6. The individual

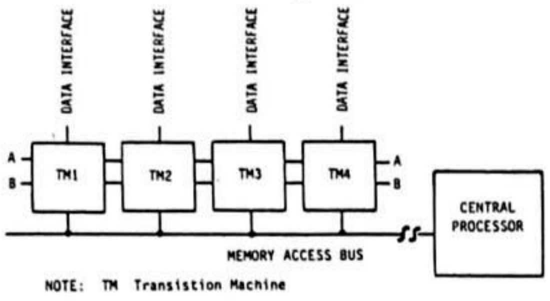

Figure 6: Front-End Signal Processing Configuration

Transition Machine elements are organized as shown in figure 7. In this example, processing loads imposed by a single source could be distributed somewhat but not over the entire system. Likewise any single point failure could be circumvented (whether a processor, a memory, an interface or a System Controller), but this recovery would involve a neighboring component rather than just any similar component in the system as could be done in a completely interconnected multiprocessor system. The modularity and interconnection simplicity make this approach preferable for many applications.

Another configuration investigated for application to airborne C^3 systems with possible application to avionics as well is shown in figure 8. In this configuration, sensor signals are routed redundantly. The signal processors are each assigned responsibility to back up a left-hand neighbor while their right-hand neighbor is assigned to back them up. Similarly all other components and interfaces have a designated recovery support role and a designated backup. These types of configurations are ideal when growth in processing demands can be expected to remain proportionate to increased interface requirements, equipment for which will also increase

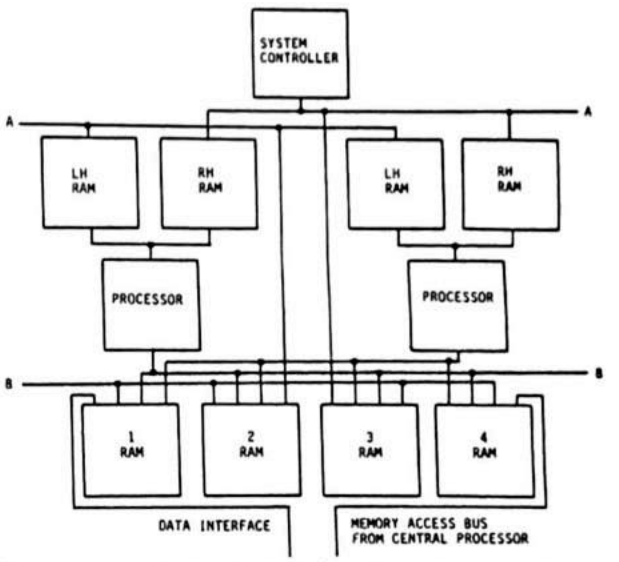

Figure 7: Individual Transition Machine Element Organization

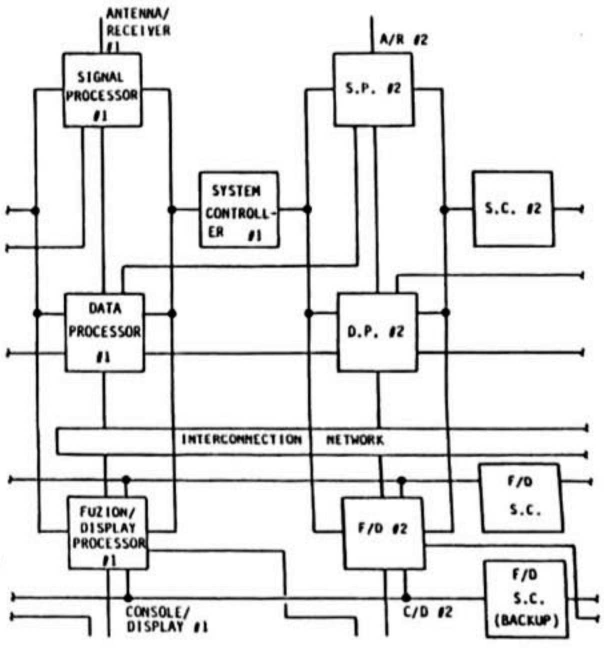

Figure 8: Completely Interlaced Organization

in modular units. It is vulnerable however to certain combinations of failures.

The detection and switching logic can be hardware-supported, but most of these functions can also be performed in software. In this regard Transition Machine archtectures provide significant inherent advantage since process assignment is performed independently by the system control elements and can therefore be maintained even when processors fail. The failures of these independent process management mechanisms must be addressed, but this is quite straight forward since processors select

a system control component to assign them a task rather than being specifically chosen by a system control element.
In figure 1 a typical modular component structure was shown wherein the system control (process management) functions were partitioned among multiple identical components. Failure of any one (or even several) will not result in system failure nor even would it necessarily degrade system performance. Switchover in case of system control component failure does not present inordinate problems. The processor can impose a consistency check of data transmitted from the system control element as a part of the task assignment protocol. Failure of the check will precipitate the processor requesting task assignment from another system control component. In turn system control will time-out the task processing performed by the processor and can implement task roll-back if the processor fails to request exit processing within the allocated time. Details of this implementation will be discussed further on as well.

Real-time Recovery Procedures

Configuring major components cannot in itself produce a fault tolerant system. There must be fault detection and component switching mechanisms as well. Clearly, any component in the layout of figure 1 could fail without incapacitating the system of any of its major functions; however, the recovery procedures must be implemented which accommodate uninterrupted processing. These mechanisms will be described for each of the major functional categories: System control, task processing and variable data or program control storage. I/O interfaces must also be accommodated, but these are frequently implemented with built-in test equipment and automatic redundancy switchover mechanisms in avionic systems, and are therefore not addressed here.

System Control Element Failures

The system control function is comprised of task control data storage and task eligibility determination and activation logic. Methods of redundant storage and switchover mechanisms for the task control storage can be implemented just as it would be for the variable common data storage memories. This is discussed further on. The logic elements on the other hand can be verified with each task assignment such that if consistency checks fail on the task assignment interface, the processor will not accept the assignment and will request an assignment instead from another system control element. Multiple faulty encounters with a given system control element from different processors will effectively result in switching the element out of the system by mutual consent of the processors.

Processor Failures

The function of the processors in a Transition Machine is to perform expression evaluations and variable assignments associated with task processing. Failure of a processor in performing its function can be detected in several ways depending upon the type of failure, its precise phase of occurence and the duration of the fault. Approaches to identifying/correcting failures in the processors are discussed further on and diagrams provided for task organizations to be generated by the support software to effect these approaches. The processor must also of course implement the system control interface to assure appropriate task assignment. Failure of a processor to meet protocol requirements will not only result in ignoring the effect of its processing, but will also result in no more task assignments to the processor if a consensus of system controllers encounter problems with the processor. Hard switching can then be effected.

Memory Failures

Memory failures can be detected by traditional means using parity implemented for verification at the interfaces, checksums, etc.

Programing to Handle Faults

The software development support system shown in figure 2 features a translator which converts conventional HOL source code to a data flow structure at the executable task level. This translator generates tasks at approximately the individual statement level of the HOL, such that logical relation determinations and variable assignments are each incorporated into separate tasks called evaluators and assignments respectively. During execution these tasks will be triggered in an order constrained only so as to preserve the data flow characteristics of the program. (Since reference 6 was published, additional algorithms have been developed which support the translation of recursive procedures, but which require runtime support for dynamic allocation of variables, etc.) This translation process results in separable actions united by the data flow constraints of the program to effect a result which is independent of the number of processing elements incorporated into the system. Communication between tasks is indirect through variables maintained in a common data base with neither the generator nor user task responsible for the transfer, it being effected by a separate system control element, whose control information is also generated by the translator and which activates users only after their respective generators have completed. Thus, by maintaining the status of the data base, system control can effect the proper data flow by activating tasks when their required data conditions are met.

This normal translation implementation can be modified however to include other constaints over and above those preserving the program's data flow. For example, individual tasks can be contrained to be repeatable in sequence (roll-back compatible), doubly redundant, etc. Furthermore, in-line translation mode changes can be accommodated by indications in the source code which effect different translation constraints at various points in the source program to support programmer-specified variations to the degree of fault tolerance to be implemented based upon the criticality of the computation to the overall avionics mission. Programming the degree of fault tolerance is to be supported by the insertion of a key word into the source program which indicates specific failure detection/recovery techniques to be employed in translating the tasks resulting from ensuing source code. A key word (NORMAL) will also indicate at which point in the source program to re-institute normal translation processes to single thread computation. Key words are the following:

NORMAL
DETECT
RECOVER
DEDICATE
ROLLBACK

DETECT. The incorporation of failure detection into the programs that run on Transition Machine processors is fairly straight-forward. Since the normal translator process implements tasking at approximately the statement level of the HOL source code, tasks need merely be duplicated with unique variables allocated for assignment by the duplicate tasks. This allows verification of expression evaluation prior to assignment to variables that are used in the expression being evaluated. The results of the redundant tasks are compared, with a consensus required for proceeding, failure indication resulting otherwise. This is shown in figure 9. For failure detection on evaluations, duplication is employed with contingent tasks requiring truth or falsehood from each of the redundant tasks and failure indication resulting otherwise.

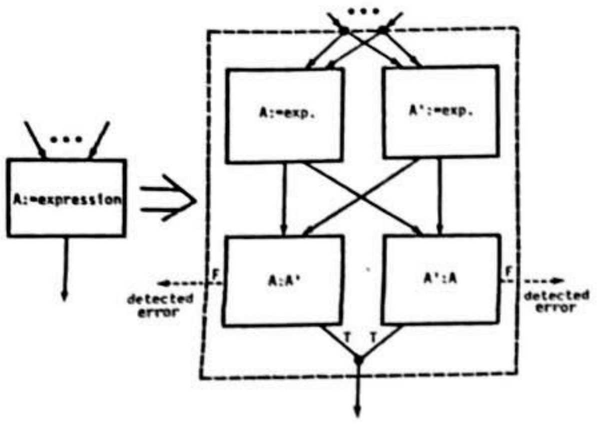

Figure 9: Detection of Any Single Failure

RECOVER. Failure detection/recovery is incoporated into object tasks by the translator's triplication of normally generated variable assignment tasks, with voting comparison tasks being generated in addition. Assigned variables are allocated in triplicate with the resulting values being compared as shown in figure 10. Agreement among all three variables indicates a non-failure situation. Failure of one or more comparisons will result in isolation logic as shown in the diagram with failure identification procedures ensuing.

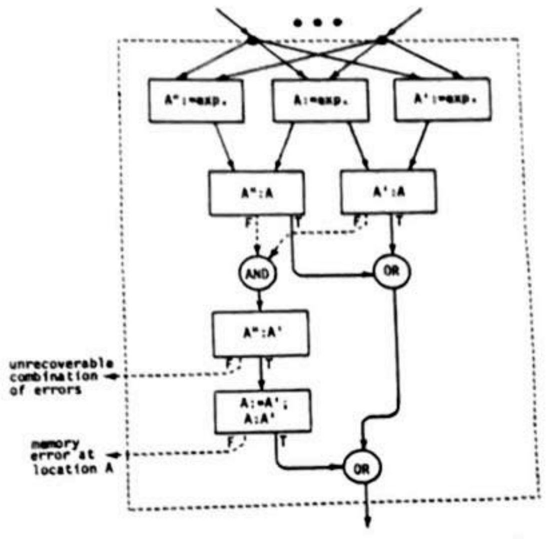
A":=exp.
A:=exp.
A':=exp.
A":A
A':A
F
T
F
T
AND
OR
A":A'
unrecoverable combination of errors
A:=A';
A:A'
memory error at location A
OR

Figure 10: Detection/Recovery of Any Single Failure

The detection/recovery approach for logical tasks is to triplicate the evaluator tasks together with the output combinations. Activation of follow-on tasks will necessitate a concensus of output from any two of these evaluators.

DEDICATE. Dedicated processing pipes are implemented by defining categories of system control elements, processors and memories that can be employed on respective redundant tasks in each computation. These processing definitions must be declared in the source programs, e.g. PIPE1 = SD3, SC1, P2, M1. (In the configuration shown in figure 1 , 81 unique processing pipes could be defined, like the one which is declared above.) Value comparison schemes similar to those previously described are implemented in this case also, but the components performing the various assignments and comparisons are rigidly controlled so that only authorized tasks are assigned within the dedicated pipes. Figure 11 illustrates the resulting organization among the various tasks.

ROLLBACK. To preclude computational errors due to processor halt, or indefinite delay conditions, variable assignment tasks may be generated as two serially redundant units each of which performs part of the task function and both of which are repeatable. Then by implementing a timeout mechanism in the system control element, the tasks can be re-initiated in another processor without indeterminate results. (Notice that certain tasks will not require duplication; the translator will determine which require duplication as a part of its normal data flow analysis.) This is illustrated in figure 12. The approach is also valid when implemented in combination with the approaches described in the preceeding paragraphs.

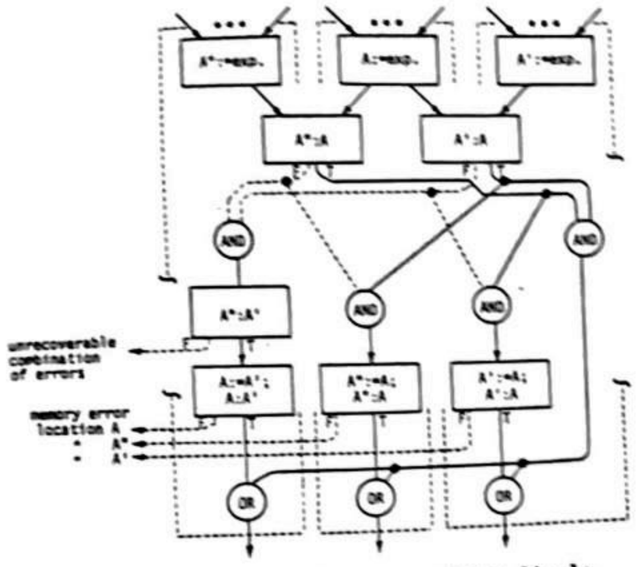

Figure 11: Detection/Recovery of Any Single Failure (Triplicated Channels)

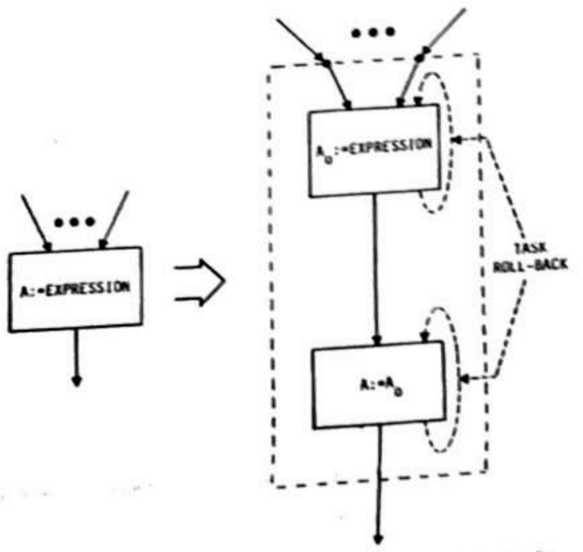

Figure 12: Protection Against Processor Halt Condition

Conclusions

The conclusions of the research that has been conducted to date indicate that Transition Machine computer architectures form an ideal basis for fault tolerant avionic computing systems. The advantages stem from the configurability of components in these architectures, the peculiarities of the system control functions, and the support software's abilities to adapt to task structures according to source progammable constraints. These advantages can be exploited at execution time because of the task control efficiency provided in Transition Machine architectures. Together these capabilities provide a programmable degree of fault tolerance to accomodate uninterrupted real-time avionics processing.

References

(1) Anastas, M.S. and Vaughan R.F., "Parallel Transition Machines." Proc 1979 International Conf. on Parallel Processing. Aug 1979

(2) Vaughan, R.F. and Anastas, M.S. "Microprocessor Based Transition Machines." Proc. of COMPCON FALL 1979. Sept 1979

(3) Vaughan, R.F. and Anastas, M.S. "Limiting Multiprocessor Performance Analysis", Proc. 1979 Int. Conf. on Parallel Processing, (Aug 1979)

(4) Vaughan, R.F. and Anastas, M.S. "An Analysis of Multiprocessor Throughput Performance in the Limit." Journal of Digital Systems. Vol. 4, No. 2, Summer 1980, pp 153-175.

(5) Anastas, M.S. and Vaughan R.F., "A Prototype Parallel Computer Architecture for advanced Avionics Applications." Proc. of NAECON '82. May 1982

(6) Vaughan, R.F. and Anastas, M.S. "Software Development Support System for Advanced Avionics Applications Incorporating a Parallel Machine Architecture." Proc. of NAECON '82. May 1982