FLIGHT SOFTWARE FAULT TOLERANCE VIA THE BACKUP FLIGHT SYSTEM

Terry D. Humphrey and Charles R. Price
NASA Lyndon B. Johnson Space Center
Houston, Texas 77058

ABSTRACT

A generic software error in the quadruply redundant primary flight system could result in the catastrophic loss of Space Shuttle vehicle control in the hostile environment of ascent or reentry. The Space Shuttle backup flight system was designed to protect the crew and vehicle in this eventuality. The significant challenges met in the design and development of this state-of-the-art protective system is the subject of this paper.

INTRODUCTION

Implementation of the backup flight system (BFS) for the Space Shuttle is a major advance in state-of-the-art fault-tolerant software in general and in Space Shuttle fault-tolerant flight software in particular. The BFS was chartered to protect against a software fault in the most sophisticated flight software system ever implemented: the Space Shuttle primary flight software (PFS). The PFS is designed to operate a redundant set of general-purpose computers (GPC) to control an innovative, multiple-element, reusable, manned spacecraft through an ensemble of flight regimes never before encountered by a single vehicle. For STS-1, the PFS consisted of more than 400 000 computer words of flight code, a complete test of every possible combination of branch instruction or decision point of which would take more than 10 000 years of computer time on today's fastest computers.

To protect against a latent programing error (software fault) existing in an untried branch combination that would render the Space Shuttle out of control in a critical flight phase, the BFS was chartered to provide a safety alternative. The BFS is designed to operate in critical flight phases (ascent and descent) by monitoring the activities of the Space Shuttle flight subsystems that are under control of the PFS (e.g., navigation, crew interface, propulsion), then, upon manual command by the flightcrew, to assume control of the Space Shuttle and deliver it to a noncritical flight condition (safe orbit or touchdown). Many technical, managerial, and operational challenges were experienced by the development team of NASA and its contractors in bringing the BFS from a concept to a working operational system. The challenges addressed herein are those associated with the selection of the PFS/BFS system architecture, the internal BFS architecture, the fault-tolerant software mechanisms, and the long-term BFS utility.

CHALLENGE 1: A MANAGEABLE SYSTEM ARCHITECTURE

Central to any concept of a reusable spacecraft is the theme of higher system reliability through redundancy of finite-lived components. In the Space Shuttle avionics, the availability of a properly functioning flight computer is assured through the wiring of five identical GPC's in parallel. Since the memory available for state-of-the-art GPC's at the beginning of the Space Shuttle development cycle was much less than the flight software requirements dictated, an early partitioning scheme was established. For each of three flight phases (ascent, on-orbit, descent), a redundant copy of all critical functions was loaded in each of four GPC's and the fifth GPC was loaded with a complement of useful functions the loss of which could be tolerated. This strategy assured protection from multiple, sequential computer hardware failures, but did not address the possibility of a software fault generic to the set of four redundantly programed GPC's causing loss of control of the Space Shuttle. Concern for such a software fault is valid in that regardless of the number of checks and balances that are put in place to find and eliminate specification and coding errors in major software developments, there can be no 100-percent assurance that latent, potentially dangerous software errors do not exist in the delivered product.

The obvious strategy for increasing the software reliability of the Space Shuttle was through software redundancy, but the challenge of the problem was the form of the redundancy to implement within time and cost constraints. Three redundancy alternatives were available: (1) increasing the internal PFS redundancy; (2) duplicating the PFS in another software version programed by a different set of programers completely isolated from the PFS programer, or (3) implementing a reduced-capability backup system by a semi-isolated set of programers. The first alternative was not pursued because it was felt that every practical internal measure was already being pursued by the PFS designers. The second alternative was considered too costly and fraught with duplication of functions not essential for a secondary system. The third alternative was selected since it afforded an additional measure of protection that was achievable within cost and schedule constraints. The reduced capability was set by a single memory load for both ascent and descent in a single GPC. The BFS also assumed the non-flight-critical functions that had been scheduled for the nonredundant fifth GPC for ascent and descent. The semi-isolation of the programers was achieved by having the BFS programed by a contractor geographically separated from the PFS contractor.

THE BFS FAULT-TOLERANT ARCHITECTURE

Significant challenges were faced by the designers to develop a BFS which would closely track the PFS, protect itself and the PFS from data pollution from each other, and also be ready at any point in the ascent, abort, or descent profile to take over control of the vehicle safely when manually engaged by the crew. To provide this capability, a technique had to be developed that would provide for tight synchronization of the BFS to the PFS in order to prevent divergence, but that would also protect both from inducing faults in the other. A more pollution-protective technique than that used for PFS redundant-set synchronization had to be developed. To protect the PFS from faulty BFS data or timing, this technique would permit no transfer of data from the BFS to the PFS.

An innovative technique for synchronization of the BFS and the PFS was developed using flight-critical data bus input profile tracking of the PFS that involves use of the input/output processor (IOP) input data bus listen capability and the transfer of input profile and minor cycle data from the PFS to the BFS. The BFS protects itself from pollution by erroneous input profile data by voting on the redundant data sent to it by the individual PFS GPC's. In addition, the BFS protects itself from input profile timing faults of the PFS by the use of its own data bus timing window thresholds for each of the individual groups of input profile data.

To protect the BFS and PFS from pollution by erroneous data received from one another, an interface design policy was established which allowed no transfer of software-generated data from the BFS to the PFS but did allow data absolutely essential to the proper tracking of the PFS to be transferred and used by the BFS. The absence of data transfer to the PFS prevents any pollution of the redundant PFS by the BFS. The BFS was designed to protect itself from pollution from erroneous PFS data first, by being limited to the use of a small amount of absolutely essential transfer data; second, by performing a vote on the redundant sets of data obtained from the redundant PFS GPC's; and third, by performing reasonableness checks on the voted data.

Another challenge faced in the development of the fault-tolerant BFS was the design of a safe method of taking control of the vehicle at any point in the flight profile without inducing control effector transients which might endanger the crew and the vehicle. The design developed to provide this protection required the input of engage initialization data from the subsystems via the flight-critical data buses immediately after BFS engagement. These data were then used to ensure that subsequent BFS control commands did not overstress or generate significant transients on the control effector subsystem.

One of the foremost innovative techniques used in the BFS fault-tolerant design was developed to provide for recovery from the loss of PFS-generated master events controller (MEC) sequencing as well as for attempting recovery from BFS GPC hardware or software errors. The loss of MEC sequencing commands might occur either as a result of a generic PFS software failure or as a result of the abrupt termination of all PFS-controlled flight-critical data buses at BFS engagement. Recovery from these types of errors is provided by the BFS software restart technique. A software restart is initiated upon BFS engagement, and, in the event that critical MEC command sequences are found not to have been properly performed, the BFS reinitiates the full MEC sequence of commands for the appropriate mission event.

The use of the restart technique to attempt recovery from BFS GPC hardware or software errors was developed to protect the BFS from transient errors and, in the case of hard errors, to continue attempts at recovery in hopes that the error will not persist. This restart recovery involves reinitialization of input/output (I/O) and restarting of BFS application processing at the beginning of a new GPC major processing cycle. The restart recovery technique provides this protection for both the preengaged and engaged modes of BFS operation.

A MOVING TARGET: MAINTAINING TRACK OF SOFTWARE CHANGES

Unlike the Approach and Landing Test (ALT) PFS, the BFS for ALT could not be used as a base upon which to build Orbital Flight Test (OFT) software. The BFS software for ALT was designed and developed by Charles Stark Draper Laboratories and provided backup for flight control functions only, provided no CRT/crew interface, and provided only a very minimal task-list-type executive. Rockwell was selected to develop the BFS for OFT and essentially started anew about 2 years behind the PFS software development. A new operating system had to be designed and developed, all existing PFS requirements and change requests (CR's) had to be reviewed for applicability to the BFS, and an overall BFS software design had to be developed. The BFS not only had to catch up with the PFS level of maturity very quickly, but then had to maintain pace with a very large amount of PFS requirements development and software change activity. A significant amount of effort and manpower was required to accomplish this goal.

POST-OFT UTILITY OF THE BACKUP FLIGHT SYSTEM

The BFS was initially envisioned to be used only through the Shuttle OFT flights. The expectation was that after OFT, the entire Shuttle system design, including PFS software, would be proven safe for operational use and, therefore, the BFS would no longer be needed. Close to the end of OFT, an examination was undertaken to assess the need for continuing the use of the BFS. Assessments of the PFS software discrepancy report (DR) traffic showed it to correlate proportionally to the level of PFS software change traffic but, even in cases in which software change traffic was small, the number of DR's appeared to decay exponentially rather than to drop abruptly. These data indicated that latent software errors had high levels of persistence. This information was used in conjunction with the projections of PFS software change traffic for future flights. It was determined that for a significant time in the future, the PFS software change traffic would continue to remain at significant levels and therefore the risks would remain high for latent PFS software errors. Therefore, it was concluded that for at least a significant time in the future, the BFS would be needed to protect against generic PFS software faults.

BIBLIOGRAPHY

Rockwell International Specification: Backup Flight System Program Requirements Document Overview. Rockwell International MG038100, 1979.

Rockwell International Specification: Primary Avionics Software System (PASS) and Backup Flight System (BFS) Software Interface. Rockwell International ICD-3-0068-03, 1981.

Rockwell International Specification: Backup System Services Program Requirements Document. Rockwell International MG038101, 1982.