A Low Maintenance and Highly Reliable Fly-By-Light Architecture

Dan Palumbo
NASA Langley Research Center

ABSTRACT
This paper proposes a highly reliable distributed architecture for use in a commercial transport. Fly-By-Light characteristics of the architecture are a serial fiber optic backplane and redundant fiber optic data bus. The fiber optic bus is fault tolerant and Byzantine resilient. A distributed architecture (both physically and logically) can be realized by utilizing several processing clusters on the data bus. A processing cluster is composed of one or more processing units. The processing units consist of several modules. The modules of the processing unit are connected via a serial fiber optic backplane. Fault tolerance is implemented at the backplane to increase reliability and availability.

INTRODUCTION
The NASA Langley and Lewis Research Centers are advocating a Fly-By-Light/Power-By-Wire (FBL/PBW) program which will develop and demonstrate optical device technologies (both sensor and communication), an advanced power management and distribution system (PMAD), and electrical actuators (EAs) for use on commercial transport aircraft. An element in this program is an FBL architecture design, development and integration. The rationale for investigating optical devices for use in flight control systems is to decrease the system's susceptibility to upset caused by High Intensity Radiated Fields (HIRF) and lightning strikes. While HIRF immunity may be the primary motivating factor, there are several interesting architectural concepts which are made possible with the greater bandwidth provided by optical data paths. These architectures may provide additional cost savings through reduced maintenance requirements.

DESIGN GOALS
The high level design goals are Design for Validation, Integrated/Distributed System Design and Fault Tolerance/Low Maintenance.

Design for Validation
The FBL Architecture will comply with the tenets of Design for Validation. In simplest terms, to validate a system one observes the system's behavior and verifies that the behavior conforms to the intended function. For example, aircraft's flight characteristics (e.g., drag), as predicted by theory and simulation, are validated by flight test. Unfortunately, many characteristics of a digital flight control system are often unobservable. The most frequently used example is the high reliability requirement of a probability of failure less than 10^-9 at 10 hours. It is not economically feasible to test a system to this specification. Instead, mathematical models are made which predict the system's reliability. To complete the validation, the model must be shown to accurately represent the system's behavior. Here is where validation is often thwarted by design. The design of a system could be such that the corresponding reliability model will contain state transitions whose rates are difficult to obtain.

To illustrate this point, consider a quad redundant system which does not reconfigure and, thus, will be vulnerable to voter failure after two channel failures. To increase reliability, the system could be designed to reconfigure system components to protect its voter from multiple errors and, therefore, increase reliability. To compute the reliability using the reconfiguring model, the rate of reconfiguration must be measured. Past validation methods have relied on fault injection experiments to create errors which then stimulate the reconfiguration process so that the rate of recovery could be measured. Upon closer examination, this method may not be sufficient. The reconfiguration process may be described as a three step process, the first step being fault detection. This is followed by fault diagnosis and finally reconfiguration. The experimenter will learn that some of the injected faults do not produce errors. These are termed latent faults. The experimenter will now need to measure the fault coverage\(^1\) of his system.

If the system relies solely on voting to detect faults, both the fault coverage and the detection rate for covered faults will depend on the software application executing on the system and the data stream it is processing. This makes it difficult to characterize the reconfiguration parameters. Additionally, the analyst may find that the system is implemented in VLSI logic precluding his attempt to conduct an adequate fault injection test. He may entertain the notion of large scale simulations only to be dissuaded by the cost of such an effort. This system is then not validatable.

Using Design for Validation, validatability becomes a requirement of the design process. To do this, the validation process is defined at the same time as the design is developed. If it is determined that the design cannot achieve the validation goals, the design is changed. In the above example, the designers may decide to use built-in-test and on-line diagnosis to detect faults instead of relying on the voter. Or they may decide to add an additional channel to eliminate the need for reconfiguration. (This underscores the need for cost modeling as part of the validation toolset. Additional hardware can, at most times, be added to increase reliability.)

\(^1\) There are many definitions for fault coverage. Here, fault coverage refers to the percentage of device failures that are detectable.

U.S. Government work not protected by U.S. copyright.
The Validation Process: The validation process, of necessity, involves evaluating the system's ability to perform as intended in its operating environment. As we have seen, testing is not an adequate evaluator in many instances. Alternatively, Formal Methods can be used in those areas where testing does not apply (such as design verification). In particular, a formal specification will be written and elaborated so that those tests that remain can be done with confidence. Using the reliability model as an example, the model structure may be formally verified to represent the system's behavior. Parameters of the model (e.g., recovery rate) will have to be measured by laboratory experimentation.

The validation process is defined as follows.

1. Use Formal Methods to demonstrate correctness and consistency of system specification.
2. Perform tests:
   a. Validate system function, as defined by formal specification, in operating environment.
   b. Validate any theory derived in formal process.
   c. Validate any assumptions used in formal process.
   d. Validate that system performance and reliability models are consistent with observed system behavior.
   e. Obtain performance and reliability model parameters.
3. Show that system performance, reliability and cost models predict that the system meets requirements.

The Design for Validation Process: To accomplish the above validation goals, the Design for Validation process is defined as follows.

1. Create a Formal Specification of system function from System Requirements.
2. Design the system so that complete and accurate performance, reliability and cost models can be constructed. Those aspects of the system which cannot be tested must be amenable to Formal Methods. The models and all parameters of the models which cannot be deduced from the logical design must be measured and validated. All such parameters must be measurable in a feasible amount of time.
3. Assess the validatability of the system design against the specification.
   a. Show that approximated reliability, performance and cost models predict that the system will meet requirements.
   b. Define all tests and proofs that must be done to complete validation process.
   c. Confirm feasibility of performing required tests and proofs.
4. Elaborate (using Formal Methods) invalidatable areas of the specification if necessary. Repeat (2).

Integrated/Distributed System Design

One of the requirements on the FBL system will be that it be physically distributed over the aircraft. There have been studies of the trade-offs between centralized and distributed systems [1] with the conclusion that distributed designs are less efficient than centralized designs. However, a distributed system architecture is thought to be necessary if the digital signal controllers associated with electrical actuators (EAs) and electrical power management and distribution (PMAD) are integrated into the fault tolerant and upset secure environment of the FBL Architecture.

To achieve the most efficient distributed design, all system functions (flight critical and others) are integrated into the design. To achieve this level of integration without jeopardizing the flight critical functions, the data path which forms the communication links across the distributed system must be highly reliable and secure in the presence of the failure of non-critical subfunctions.

A Distributed System Data Path: There are several design goals which are significant in the design of the Distributed System Data Path (DSDP).

1. All inter-process system data is available on the Distributed System Data Path. Processes can then be distributed on the system without precluding functional integration.
2. All transactions on the DSDP can be deterministically scheduled. This requirement aids validatability.
3. The DSDP meets the reliability requirements of the most critical application. Flight critical functions can then utilize the data path.
4. The DSDP path is secure from corruption by non-critical (e.g., simplex) processes. Non-critical functions can then use the DSDP without threatening flight critical functions.

Fault Tolerance/Low Maintenance

The mission reliability requirements for the flight control system of a commercial transport has been given as a probability of failure of 10^-9 at 10 hours. It is desirable to achieve an availability, stated as the probability of a repair action over a ten year period, of 0.1. A design goal is to utilize the redundant components to increase system availability. Several additional goals are necessary to support the validatability of the architecture's fault tolerance.

1. Module Level Redundancy. Redundancy should be organized at the module level to reduce the failure rate of the reconfiigurable component, thus increasing reliability and availability.
2. Consistent Data Distribution. All data should pass through a process which ensures that Byzantine failures will not corrupt the system.
3. Formal Clock Synchronization. A formally verified clock synchronization algorithm should maintain a known degree of synchrony across the system. A Global Time Base can be established based on the synchronized clocks.
4. Deterministic Scheduling. Task schedules and device utilization/partitioning should be derived deterministically (e.g., static schedules). Events and processes should be able to be scheduled according to the Global Time Base.
5. HIRF/Lightning Upset Immunity. The architecture should be designed to be 'immune to' (as opposed to 'recover from') the type of system wide upset that is likely to result from HIRF incidents. This requirement is due to the infeasibility of validating a recovery process from an undefined, upset induced, state.
6. SEU Recoverable. The architecture should be able to recover from transient Single Event Upsets (SEU) of the kind that affect only single components.
7. Slow On-Line Reconfiguration. It would be optimal if reconfiguration could be limited to off-line maintenance activities. This would greatly enhance
validatability. However, it may be desirable to implement a slow on-line reconfiguration to provide high reliability and dispatch with failure.

8. On-line Fault Detection. If on-line reconfiguration is used then fault detection must be provided by on-line diagnostics and built-in-test. Vote functions are limited to error masking.

9. Fault Secure Operating System. The operating system must be capable of containing the failure effects of non-critical functions in a fault secure processing environment.

ARCHITECTURAL CONCEPTS

The proposed FBL architecture is addressing the issue of integrating non-critical tasks with critical tasks in a validatable, reduced maintenance, distributed system design. Central to the design are the concepts of a highly reliable and fault secure communications platform. Two elements of this platform are the Distributed System Data Path (DSDP) and the Fiber Optic Serial Backplane (FOSB). The DSDP and FOSB rely on the Redundancy Management Unit (RMU) to provide the basic fault tolerance functions of clock synchronization, voting and reconfiguration. Together, these components will provide a processing and communications platform which will meet the requirements of flight critical tasks. Surplus throughput and bandwidth can be utilized by non-critical tasks providing that the operating system can enable a fault secure environment. Non-critical hardware (e.g., I/O modules) can be integrated through the fault secure communications platform.

Redundancy Management Unit

Previous studies of fault tolerant systems have shown that hardware support is necessary to support fault tolerance functions. The Software Implemented Fault Tolerance (SIFT) [2] computer favored software functions over hardware implementation with the results that the voting was tremendously inefficient and the synchronization relatively loose [3]. The Advanced Information Processing System (AIPS) which was used in the Integrated Airframe/Propulsion control System Architecture (IAPSA) [4] relied upon hardware support for both clock synchronization and voting. The AIPS processors were tightly synchronized at the instruction level and provided a memory mapped hardware register into which pre-vote values are written and post-vote values are read. The Redundancy Management Unit addresses the limitations encountered in these efforts.

Consistent Data Distribution. The largest load created by fault tolerance functions are those which guarantee the consistency of the data distributed across the system. The data consistency algorithms are designed to protect against Byzantine failures [5], i.e., a single failure which could cause different, non-failed, processors to interpret the same transmission differently. The data consistency algorithms are used to ensure that the redundant processes in a fault tolerant computer operate on identical sets of what was originally simplex data (e.g., sensor data).

It is argued that Byzantine failures are too rare to be of concern. However, all the fault tolerant computers in NASA's Avionics Integration Research Laboratory (AIRLAB) have exhibited Byzantine behavior either spontaneously or under fault injection stimulation. These computers were heavily instrumented during these tests. It is suspected that the perceived rarity of Byzantine phenomena is due to poor system internals observability.

The RMU integrates a data consistency algorithm into its data transfer protocol. The algorithm is meant to both ensure the integrity of critical data and to isolate the possibly malicious behavior of non-critical components.

Block Structured/Message Based Data Transfer. Although the IAPSA/AIPS design relied on hardware for voting and synchronization, voting still created a bottleneck in the system. This was due in part to the design. Data, destined for a voter, originates at redundant sites at different times. The skew between the slowest and fastest member of the set is bounded by the clock synchronization algorithm. The data is delayed first by the communication path, but then must be delayed at the voter to deskew the data streams before the vote. In other words, a voting device must wait the maximum transmission time plus the maximum expected clock skew before voting the data. In the design of the IAPSA/AIPS voter, the unit of transfer was 1 word. It was necessary to tolerate the deskew overhead for each word of I/O traffic.

To reduce this overhead, the RMU is designed around block structured/message based transmissions on both the Distributed System Data Path and Fiber Optic Serial Backplane.

Clock Synchronization/Global Time. As can be inferred from the above discussion, achieving tight synchronization reduces voting overhead and, therefore, improves communications performance. A local clock will be maintained in each RMU. The RMU will perform a clock synchronization algorithm which will guarantee that all local clocks are within a defined maximum skew of each other. This will establish a Global Time Base on the system. The Global Time Base will be used to order events on the system, thus increasing efficiency and enabling validation.

---

![RMU Block Diagram](image)

**Figure 1. RMU Block Diagram**

RMU Block Diagram. The RMU block diagram is pictured in Figure 1. The RMU is accessed through serial or parallel I/O ports. The input data stream is buffered locally and transmitted to three other RMUs. Any of the four data streams can be masked. Timing data on the relative skews between the data streams is gathered as the data streams are synchronized. The timing data is used by the clock to maintain synchrony with the other channels. The synchronized data streams are fed to the
The voted data is then available as serial or parallel output.

The Distributed System Data Path

The Distributed System Data Path (DSDP) will support all inter-process system data (critical and non-critical). The DSDP must then be designed to the reliability requirements of the most critical functions and protect these functions from possible failure of less critical tasks. To accomplish this, the DSDP supports 4 point-to-point fiber optic links between each processing site (see Figure 2). Each processing site contains a DSDP controller (DSDPC) which is based on an RMU. The RMU maintains a time base which is used to time multiplex access to the DSDPC. The RMU also provides cross link voting so that a failed link will be regenerated at the next processing site (as illustrated in Figure 2).

Figure 2. Distributed System Data Path

Global Time Base. Timing statistics gathered at each DSDPC will be used to maintain global clock synchronization. A global time base will then be established based on the synchronized clocks. The global time base will be used to control access to the DSDP. The global time will be made available to the local processing site so that local tasks can be scheduled in cooperation with remote tasks.

Self Repair. Each processing site is quad redundant. All four channels of the processing site place data on the DSDP simultaneously. The quad redundant data is exchanged, voted, and retransmitted at each processing site. In this way, errors present on any input link are masked at the next processing site. Thus, the bus has a self-repairing attribute. The fine degree of fault containment also aids diagnosis of fault location.

Fiber Optic Serial Backplane

The Fiber Optic Serial Backplane (FOSB) consists of a read and write bus (see Figure 3). Output from each module is placed on the write bus where it is collected and voted by the RMU. Voted data is then transmitted by the RMU over the read bus. The read bus is a multi-drop bus with the RMU having sole write access. Thus, all modules will be able to phase lock to the RMU output. The time base in the RMU is used to time division multiplex the write bus. The module's write bus transmitters will get their timing from the module's receivers which are phase locked to the RMU transmitter clock.

The RMU receiver should then see minimal shift in the timing as the modules multiplex their data onto the write bus. Increased fault isolation on the write bus can be achieved by using a different wavelength transmitter for each module and a broadband receiver in the RMU.

Module Level Redundancy. The RMU establishes a fault containment region at the write bus boundary. The FOSB and RMU work together to give working processors access to I/O resources on channels with failed processors. For example, in Figure 3, if the processor in the top channel failed (PROC1) then its I/O resources (I/O1a and I/O1b) would still be accessible by the processor in the lower channel, PROC2.

Interactive Consistency. I/O modules or simplex processes on the FOSB which generate single source data will be required to pass their output data through the Interactive Consistency algorithm supported by the RMU. When simplex data is to be passed through the RMU, all for channels must command the RMU to open that channel for a finite time for the single source distribution. At the end of this time, the RMU will return to quad operation, redistribute and vote the data, thus completing the Interactive Consistency algorithm exchange.

DSDP/FOSB Gateway

The components on the Fiber Optic Serial Backplane (FOSB) need a path to the Distributed System Data Path (DSDP). The DSDP/FOSB Gateway serves this function. Since the FOSB and the DSDP must access the Redundancy Management Unit (RMU) fault tolerance functions, it makes sense to embed the RMU in the the DSDP/FOSB Gateway (See Figure 4). The RMU, and thus the FOSB read and write buses, must have the bandwidth to pass all local and global traffic.
As of this writing, a reliability analysis of a complete integrated/distributed system has not been done. However, much can be learned from the preliminary analysis given below.

This analysis focuses on a quad redundant node. Each channel in the node holds 10 modules including one DSDP/FOSB gateway. Six analyses were done including three different initial conditions and two failure/recovery models. The first initial condition is a full-up system, with no failures. The second initial condition is a system with one module failure. The third initial condition is a system with one RMU failure and, therefore, one failed channel. For each of these three initial conditions analyses were done for two failure/recovery models. One model is conservative with a module failure rate a $10^{-5}$ per hour. The second model is optimistic with a module failure rate of $10^{-6}$ per hour. The RMU failure rate is assumed to be one tenth that of the module failure rate. The recovery time is 1 minute for the conservative model and 15 minutes for the optimistic model. Fault detection coverage is assumed to be 100%. Table I summarizes the results.

As can be seen from Table I, a full-up system meets flight critical reliability requirements. The conservative model has a marginal reliability with one bad module and an unsafe reliability (for flight critical functions) with a failed RMU. The optimistic model meets the reliability requirements across the board, but relies on modules achieving million hour mean time between failures. An analysis of a 10 node, quad redundant DSDP produces identical results.

### Availability Analysis

Availability analyses were done on both a single processing node with 5 quad redundant modules (20 total) and a distributed system with 100 modules which are distributed across 5 processing nodes on the DSDP. It was assumed that it would be necessary for two modules of the same redundant set to fail before repair was necessary. The probabilities of a repair action for several operating times were derived for the conservative and optimistic systems (see Table II below). Only a single node, based on the optimistic model, meets the target probability of repair action after 10 years of 0.10.

### REFERENCES