Generic Faults and Architecture Design Considerations in Flight-Critical Systems

Stephen S. Osder*
Sperry Flight Systems, Phoenix, Arizona

Specific examples of generic faults involving interaction of hardware and software in flight-critical systems are described. Such faults can cause an avalanche shutdown of all redundant computer channels. The popular technique of redundant computer frame synchronization is shown to be particularly vulnerable. Architecture solutions that allow dissimilar redundancy and incorporate "brick-wall" isolation are described. Practical techniques of coping with time-skew effects in unsynchronized computer channels are given, and it is shown that they offer many simplicity advantages.

Introduction

THE term "generic fault" became part of the technical language to describe design defects that elude the test and analysis procedures used to validate a redundant control system design. Although the existence of such defects can be postulated in any type of system, the generic fault concept is especially significant in the flight-critical system application because it defeats the massive redundancy strategies that designers rely on to meet safety or reliability objectives. A vision of such a fault toppling each channel of a flight-critical redundant system is a nightmare that haunts the designer and the "certifier" communities.

The first part of this paper examines various classes of such faults, acknowledging that there can be no 100% certainty that a system is free of them. Specific illustrations show why their detection and correction are difficult.

It is contended that redundancy architecture is a key factor in generic fault vulnerability, with redundant channel synchronization as a major aggravator of error mechanisms. Well-known solutions to such vulnerabilities involve concepts of channel decoupling, including the "brick wall" approaches that were popular in early analog mechanizations.[1,2] The term "brick wall" refers to extraordinary measures taken to insure the physical and electrical separation of redundant channels. Dissimilar redundancy also has its advocates,[3] but both solutions preclude the advantages provided by frame (or loose) synchronization of computer channels. Such synchronization allows an inherent simplification of monitoring algorithms. Various degrees of synchronization tightness have been used. The Space Shuttle computers synchronize to sublevels of frames or tasks,[4] while some systems under development use complete microclock synchronization in massively redundant central processing unit (CPU) and memory structures.[5,6] When channel decoupling and isolation (including dissimilar hardware and software mechanisms) are used to avoid generic-fault vulnerabilities, channel synchronization must then be abandoned.

A recurring theme of this paper is that the synchronization of redundant computers opens pathways for generic faults to cause multiple channel shutdowns. In some instances these faults reside in incomplete logic designs that might not anticipate the multiplicity of computer resynchronization interactions following transient events in the power distribution system. In other instances, the generic fault could also exist in the nonsynchronized systems, but an event which triggers the fault would cause only one channel to shut down, whereas the same fault in synchronous architectures would cause multiple channel shutdowns. The most prevalent computer architectures found in digital flight control systems that are currently operational or undergoing development testing are of the synchronized type. Hence, it is apparent that such systems can be implemented in practical designs and obviously have attractive attributes. It is not the intent of this paper to judge the superiority or inferiority of any system architecture. Design decisions involve many complex requirements and compromises that are beyond the scope of this paper. The subject of the paper involves phenomena whose probability of occurrence may be in the 10^{-6} to 10^{-9} per hour category (related to frequency of power and EMI disturbances). Depending on specific application requirements, these low probability phenomena can influence the system design decisions.

In the description of various types of generic faults, this paper stresses the vulnerability of the synchronized architectures. The alternative unsynchronized architectures have been avoided in the past because they seem, at first glance, to involve performance or complexity penalties. It is shown that some of the commonly viewed disadvantages of unsynchronized systems can readily be overcome.

The second part of this paper describes straightforward solutions to cope with time skews, integrator divergences, and other potential tolerance buildup problems that are considered to be the liabilities of unsynchronized autonomous channels.

The information presented here is derived from more than a decade of experimental work in a digital systems laboratory that was originally set up to integrate the triplex digital flight-control system being developed for the Boeing/U.S. Government SST. This laboratory work was supplemented by several flight demonstrations of the principles,[7] and systems using some of these principles have become operational.[8,9] However, most of the discussions related to triplex and quad architectures are derived from laboratory experiments accomplished with hardware in the loop simulations.

Types of Generic Faults and Specific Illustrations

The technical folklore on this subject is filled with endless accounts of generic faults that range from benign and often humorous phenomena to catastrophic events. This information rarely finds its way into print, but the events are often sufficiently newsworthy to become widely known in the technical community. The F-16's simultaneous loss of all quad redundant channels[10] must be classified as a generic fault due to a system architecture/logic inadequacy in coping with transients resulting from faulty power-generation equipment. The abort of the Space Shuttle's first launch[11] is another example that could be classified as a timing design

Presented as Paper 82-1595 at the AIAA Guidance and Control Conference, Gatlinberg, Tenn., Aug. 15-17, 1982; submitted Sept. 7, 1982; revision received Dec. 6, 1982.

*Director, Research and Development. Associate Fellow AIAA.

defect that was intimately related to the system's computer synchronization architecture.

The following is an attempt to classify the types of generic faults, but even more insidious effects can result from combinations of these different categories.

Computation and Scaling

This is the best known or more publicized defect, which manifests as a computation overflow. Such errors in scaling are usually easy to expose by thorough testing.

Timing

This is the other well-recognized defect which manifests itself by a program running out of its allocated time. In testing for nominal system operation, this type of problem is easily exposed. However, conditions associated with the diversion of nominal program flow into fault isolation and reconfiguration routines are considered to be most vulnerable to inadequate time margins. It is possible to design robust, multirate executives that cope with bursts of added time consumption, but many of the more rigorously structured timing executives do not tolerate programs that push to the limit of timing margins.

Logical Errors

This category dominates the concern with generic faults. The permutations of a system's sequential states reach enormous proportions when off-nominal situations are considered. The timing of events within a computation cycle, the previous history of faults and reconfiguration responses, the specific engaged mode and, in frame synchronized architectures, the instantaneous alignment of the comptuation timing frames all conspire to produce subtle, event-sensitive logic problems.

Before considering some of the subtle logic error vulnerabilities, a simple example should clarify how logic errors can devastate a flight-critical system. The example is the classic case of the triplex voter (Fig. 1). Its application was widespread in the analog era,[12] but its mechanization is now being translated into software and one can still observe the defects. These defects involve inadequate treatment of signal tolerances. Two problems are illustrated. First, the threshold of fault detection is set at 1.0 unit and, in the initial condition illustrated by Fig. 1, x_1 and x_3 are at the opposite ends of their tolerance extremes. If a fault is defined as absolute value of $\hat{x}-x_i$ greater than 1.0 unit, where $\hat{x}$ is the mid-value, then the condition of two quantities at their tolerance extreme is masked by the mid-value, which is x_2. The figure illustrates the case where x_2 fails toward hardover. Its failure trajectory carries the mid-value through an intercept with x_1, at which time x_3 is declared "failed." (A failed channel switches to zero so that a second failure will be fail safe.) The new mid-value is x_1 so that the next event will be $|\hat{x}-x_2|>1.0$, causing x_2 to be switched to zero. The sequential shutdown of the entire system has now occurred because of one true failure and perhaps an inadequate setting of the fault threshold.

The second defect in this logic affects performance. If x_1 and x_3 were at ± 0.4, then the failure shown in Fig. 1 would have properly shut down channel 2. Now with channel 2 set to zero, zero will become the mid-value, thereby forcing a deadzone in the system equal to $|x_1-x_3|$ or 0.8 units. If x represented an automatic landing sensor, such as a localizer receiver, deadzones of such magnitudes can result in unsafe performance.

If a triplex voter is used, solutions to the above type of problems include:

1) Channel equalization in which x_i is slowly converged toward $\hat{x}$;

2) The use of computed fault thresholds based on bias and scale-factor error models[8]; and

3) The use of combined averaging and mid-value selection algorithms, or hardware mechanizations.

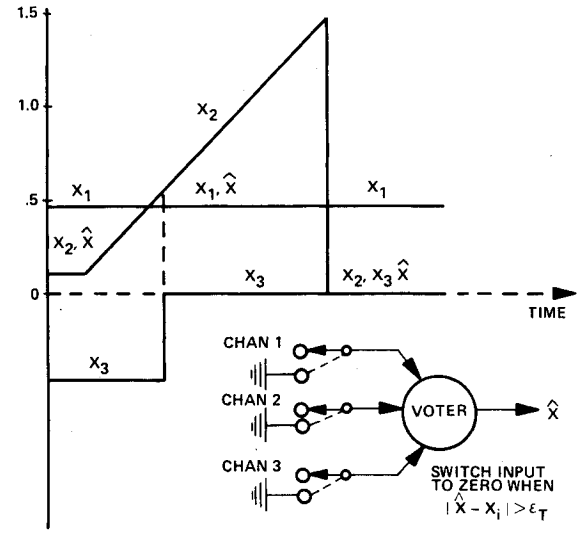

Fig. 1 Triplex voter avalanche failure mechanism.

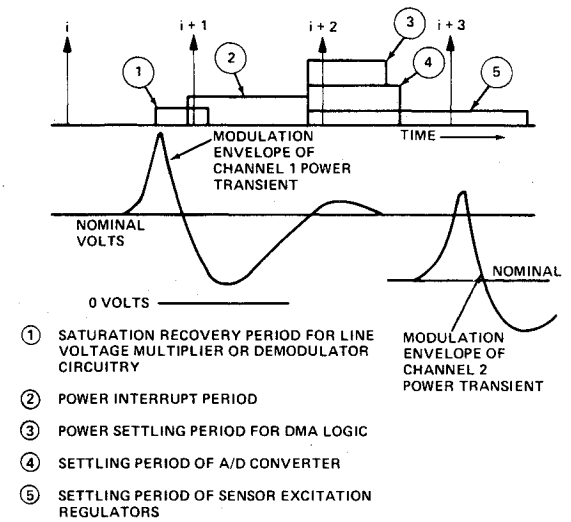

Fig. 2 Power transient stress to system logic.

Perhaps the greatest challenge to the system logic designer is the disruptive impact of power transients. Figure 2 illustrates a power transient sequence that occurs first in channel 1 of a redundant system and then occurs an instant later in another channel. The characteristics of the transient are consistent with MIL-STD-704 (aircraft power systems). The sequence of transients of this type could occur when large loads are switched from one bus to another. The figure identifies the major computation frame timing events as i, $i+1, \ldots$. The transient duration is about one timing period, but it overlaps into two timing frames. Also shown on this figure are periods (not necessarily to scale) that are associated with equipment responses to the power transient. The phenomena associated with these responses must be accommodated in the system logic implemented in software. All five illustrated categories have been experienced in operational applications. Logic to cope with three of the five was implemented in one system,[8] but initial flight experience indicated that the actual power transients in the aircraft environment were more severe than the worst-case condition used to test the original design. To avoid nuisance disengagement, system logic changes were needed to stretch monitor inhibition periods. Testing system hardware and software in a laboratory validation facility environment would require duplicating an actual aircraft's power distribution system and associated loads to accurately reproduce power transient phenomena. This capability is rarely available, and component testing is usually substituted.

Component testing with power transients will not produce the pattern of disruptions that involves the computer and the sensor data.

Consider the five phenomena shown on Fig. 2 as events associated with the power transient. Item 1 represents a hardware design defect that can be corrected by software logic. It is encountered in systems that use line-voltage excitation of sensors (such as synchros). Signal processing prior to analog-to-digital (A/D) conversion involves compensation for line-voltage variations. Circuitry associated with this processing often saturates when the line voltage exceeds the nominal value by more than about 30%. While there are possible hardware protections, this expense can be avoided by substituting software logic. This logic is based on detecting excess line voltages and inhibiting the use of information affected by the saturated elements.

Item 2 represents the power-interrupt logic. A transient power-down must be distinguished from a true system turn-off to permit automatic recovery. That recovery requires storage of all CPU working registers in a memory that is nonvolatile. If the line voltage returns within the specified transient recovery period, the power-up logic to the CPU commands restoration of the machine state that existed at the instant of the power interrupt. On recovery, the CPU picks up where it left off as though the disruption had not occurred. This would normally be a straightforward operation, but the CPU may not be the only element that has to recover from the power transient. Item 3 represents an example of another device that may have to recover from the power transient before normal computation sequences can occur. It is a direct memory access (DMA) device that reads and writes into memory but which may receive its logic voltage from a source other than the CPU. If the CPU starts to access memory before the DMA has recovered, erroneous data can be written or read. A similar phenomenon existed in many early systems that used magnetic core memories equipped with a data-guard circuit that locked out read/write operations when logic voltages were too low. If the CPU depended on the data-guard logic to control its own startup, it could find itself trying to execute a program before adequate logic voltages are achieved. The AND combination of the CPU's own voltage monitor and the data guard would insure safe operation, but this was not always incorporated in designs and memory wipe-out problems were once commonplace. Some of these occurrences were occasionally reported in published literature.[13]

Item 4 indicates that A/D converters also need time to settle from power interrupt transients. Not only has sensor data been corrupted by the signal processing transients already discussed, but the A/D converter usually requires some time to settle after voltage is restored. Attempts to use data from the converter prematurely could lead to the detection of nonexisting faults by wraparound monitors and by monitors that compare sensor data.

Item 5 illustrates a problem with a sensor excitation power supply that must generate and regulate transducer excitations. When these excitations must be held to precise tolerances, the regulators must be given some additional time to settle. If data from the associated transducers are used prematurely, the various monitors would detect erroneous faults.

What have these hardware-type considerations to do with logic? The system logic must be properly responsive to the effects illustrated in Fig. 2. That logic must recognize when power disruptions have occurred and computation strategies must be altered as a result. Synchronized computers tend to be more vulnerable to logical defects in coping with power-transient disruptions. Some typical frame synchronization algorithms in current use have been analyzed, and scenerios were postulated to produce avalanche shutdown of the total system. These scenarios involve four frame-synchronized control channels that encounter a sequence of power transients successively passing through each computer's power supply. Each computer would drop out of synchronization lock, and the dropout would be recognized by the unaffected channels. Simultaneously the sensor data associated with a disrupted channel can be recognized as faulty and declared invalid. As a working computer tries to apply a "smart" algorithm that recognizes the phenomenon as a power interrupt, it is itself disrupted. When that computer finally recovers, it encounters a different situation regarding where the faults are indicated. It is possible to provide sufficient logic protection for some combinations of power-transient sequences, but it is also possible to postulate a sequence that will defeat this logic. The tighter the coupling of systems and the more dependent they are on computers 1, 2, and 3 voting out computer 4, the easier it is to postulate an avalanche type of power-transient event. One might dismiss such events as highly improbable, but many of these improbable sequences could be associated with lightning-strike phenomena.

Hardware and Firmware Defects

The computer folklore contains many accounts of microcode defects emerging at most inopportune moments—the instruction worked fine until certain improbable conditions came together. It is instructive to examine a case history of such a defect that occurred in a flight-critical application, and to note its etiology and how it was exposed by exhaustive hardware and closed-loop validation testing. This fault could have resulted in a total system shutdown in a frame-synchronized architecture. However, in the unsynchronized autonomous redundancy architectures, the fault would have been contained within a single channel.

The problem involves implementation of a checksum instruction in microcode (Fig. 3). The checksum instruction verifies the integrity of the flight-critical program (ROM). In a typical application, the stored program for the high-speed control loops are checksummed at high rates (complete check about every 1.0-2.0 s), while the less time-critical memory is checked in slower loops (2.0-20 s). The instruction sums the content of memory from a specified starting address to a specified terminal address. The starting address is initialized in register B (R_B), and the terminal address is stored in register C (R_C). The operation performed by the microcode may be described by

$$R_A = R_A + \sum_{(R_B)}^{(R_C)} \mathrm{MEM}(R_i) \quad \text{where} \quad i = B \text{ to } C \qquad (1)$$

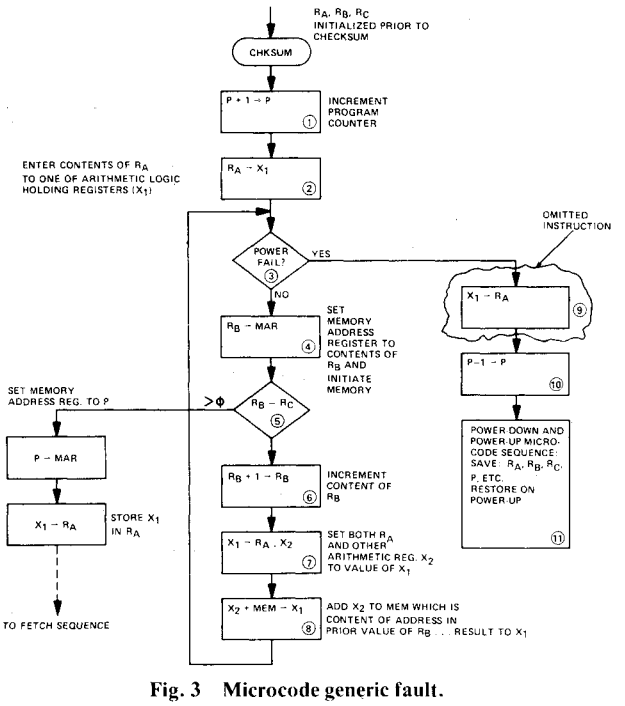

Fig. 3 Microcode generic fault.

It adds into register A the sum of the contents from the address contained in R_B to the address contained in R_C.

The microprogram flowchart for this instruction, as implemented in a specific processor, is illustrated in simplified form in Fig. 3. The power-fail interrupt check must be made in each pass in order to store the needed information to restart if power recovers within the specified transient period. Other instructions are completed in a short enough time to allow ignoring the power interrupt until the next instruction sequence must be fetched.

In studying the flowchart, observe that storage of the latest cumulative sum (x_1) in register R_A was omitted during the power-interrupt condition. In the original design, step 8 preceded step 7, and this would have precluded the need for the omitted $x_1 \rightarrow R_A$ sequence. The step 7,8 sequence was reversed by the CPU designers to allow an additional microclock cycle for memory response (initiated in Step 4); hence the seed for the error was planted.

This defect was discovered concurrent with the start of flight tests. It was not detected during system integration tests because the memory checksum subroutines are always disabled until the program memory is finalized. If memory checksums consume about 2.0% of the total flight program, then the probability of encountering the problem was 0.02 for each power-interrupt transient. If such interrupts occurred with a frequency of one per hour, then it would require about 50-100 h of operation before we could expect to see the fault occur. The failure would have been a computer shutdown, with the automated diagnostic being "failure of the program memory." If redundant computers were operating in frame synchronization, then all computers would be performing checksums simultaneously. A power transient, such as one that could occur during a lightning discharge through the aircraft's ground structure, would probably have caused simultaneous faults in all channels for the synchronized system architectures.

Fortunately, this defect was found during black-box testing where power interrupts were inserted at 1-s intervals while the computer was being exercised through a repertoire test. However, the story does not end here. After the microcode was corrected, continued power-interrupt testing would show an occasional checksum failure. Another, but more subtle, generic fault remained. If the power interrupt occurred during the microcycle that contains steps 3 and 4 of Fig. 3, there was a 5.0-ns vulnerability window where the program executed step 4 as well as branched to step 9. This would appear to be a harmless error, except that upsetting the memory address register interfereed with the memory addressing that occurred during the steps 9-11 sequence. If the probability of a fault was 0.02 for a power-interrupt rate of one per hour, then for this same power-interrupt rate, the probability of the fault became

$$P_F = (0.02)(5/1100) = 90 \times 10^{-6} \text{ per hour} \qquad (2)$$

where 5 ns is the vulnerability during checksum and 1100 ns is the time to perform each checksum loop. Another design change synchronized the power-fail logic to the microclock to eliminate this vulnerability.

Latent Failure Effects

These failure effects have always been a key consideration in the failure mode and effects analyses (FMEAs) used in validating flight-critical designs. In this case we are referring to failures in hardware that are either masked by other circuits or which remain latent until the circuit is activated. Digital mechanizations have largely eliminated the latter latent category. Those remaining, such as memory failures, are easily accommodated by the current monitoring technology, although microprogram bit failures provide some significant challenges.[8] Thus, a system design must provide for the conditional probability that a previously undetected hardware fault could interact with a set of vulnerable events.

Low failure rates for susceptible devices do not help much in the probability computations because the exposure time can be as large as the lifetime of the equipment. System designers acknowledge that this problem category presents the most difficult challenge to providing system safety. Let us examine a specific example.

The real-time clock used to control the allocation and sequencing of tasks is typically derived from the processor's microclock by a counter device. FMEAs of such counters indicate that individual gate failures can cause errors of a few percent as well as the severe errors that are easily detected by system monitors. The so-called heartbeat monitors or watchdog timers must contain precise, independent clocks to detect faults that produce small errors, but these monitors are usually quite crude. In an unsynchronized system, a small degradation in the acuracy of the real-time clock is analogous to a capacitance tolerance error in an analog computer. In frame-synchronized systems, the error can either propagate to all other channels or be masked by the good channels. Consider the following scenario for a latent real-time clock failure in a synchronized system.

Real-Time Clock Failure Speeds Up Clock

Channel 1 runs faster than the other channels. When there is an adequate time margin, the faster clock controls the process. (Each computer's synchronization algorithm requires a "wait for real-time clock interrupt" *or* a cross-channel clock interrupt.) Assume that a condition occurs where the timing for a task is marginal. Channel 1 receives its real-time interrupt before the task of formatting the sensor data for cross-strapping to the other channels is complete. If a task completion monitor is not used, the last sensor quantity is transmitted erroneously. Sync is lost for this one cycle, but computers 2, 3, and 4 recognize a faulty sensor. This condition recurs intermittently, recovering after one cycle of inadequte time. Now consider the condition where channel 1 had completed its task. It issues a sync interrupt to the other channels and all but channel 2 are prepared for that interrupt. Channel 2 will drop out of sync but it is a good computer. Is a good sensor declared failed? Does dropping of sync for one cycle warrant declaring the computer faulty? How does the monitoring logic respond to intermittent loss of sync?

Solutions can be devised for such situations, but they add complexity and the interchannel interactions may leave subtle defects. The insidious nature of this type of problem is that the validation testing cannot accommodate all of the postulated potential problems. A robust design is achieved in a frame-synchronized system if significant timing margins can always be maintained. If this luxury cannot be attained, then the unsynchronized autonomous systems become more attractive.

Solutions Using Decoupled, Nonsynchronous Architectures

Synchronous vs Nonsynchronous Tradeoff

Frame synchronization is a popular approach to redundant computer system architectures because it appears to offer monitoring simplifications. If time skew is eliminated in the sampling of sensor data and all computers receive identical values of redundant sensor information, then all computers must produce identical output commands. Thus, a one-bit discrepancy in a control command issued to surface actuators by redundant computers would represent a computer fault. If conditions were ideal, the computer output values (control commands, logic states, etc.) could be compared to the least significant bit, and any computer that produced a one-bit error could be declared faulty by the other machines. The problem of insuring identical sensor data can be solved in two ways. One approach uses separate processing and multiplexing subsystems for transmitting the sensor data. The multiplexer/demultiplexer (MDM) units of the Space Shuttle are an example of this approach.[14] The other approach allows

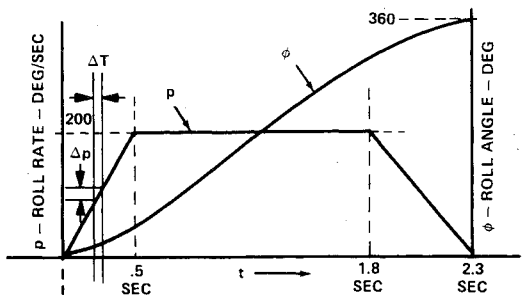

Fig. 4 A 400 deg/s^2 roll maneuver challenge to unsynchronized computers.

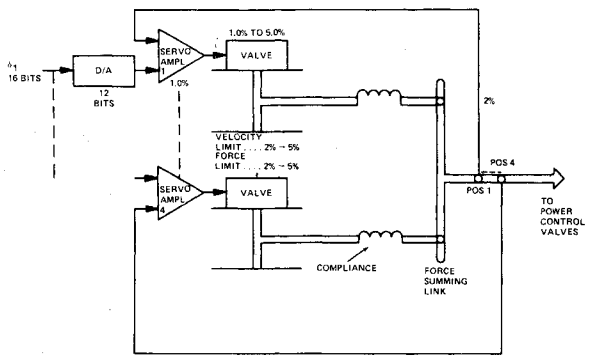

Fig. 5 Block diagram of tolerance deterioration from computation to actuation in typical quad force summing configuration.

each computer channel to process sensor data associated with that channel, and then synchronously transmit the values of its sensors to the other computers via a cross-channel digital data link.[15] Both methods are susceptible to disruption of synchronism in the data transmission as a result of power transients. Hence, in realistic environments, the bit-by-bit identity of each computer output is not a sufficient condition for declaring a computer (or sensor) as "failed."

If computers are not synchronized, then time skews between channels can cause significant differences in output commands during high dynamic conditions. Figure 4 illustrates a 360-deg roll maneuver accomplished in an idealized sequence of 400 deg/s^2 accelerations and a 200 deg/s fixed roll rate. If the iteration rate is 100 per s, then the maximum time skew can be one iteration cycle (ΔT) of 0.01 s. During the acceleration phase of the maneuver, the roll-rate discrepancy (Δp) between computers can be

$$\Delta p = \dot{p}\Delta t = 400 \text{ deg/s}^2\ (0.01)s = 4 \text{ deg/s} \tag{3}$$

The discrepancy in the commanded rolling surface (δ_A) will be dominated by the following term:

$$\Delta\delta_A = k\Delta p \tag{4}$$

For a typical control law, the value of k is about 2, so we can expect different channel outputs to differ by as much as 8 deg of surface command. Typical failure thresholds would normally be expected to detect an 8-deg surface command discrepancy. Hence, designers have accepted computer synchronization as the way to avoid the apparent tolerance problem. The conclusion that synchronization is needed would be valid if fixed monitoring thresholds were used, as they were in the era of redundant analog mechanizations. When monitoring thresholds are computed from models of scale factor and bias tolerances,[8] then the 8-deg discrepancy obtained with the Fig. 4 maneuver can easily be accommodated as a normal condition. This will be demonstrated later in the description of the unsynchronized computer approach to comparison monitoring.

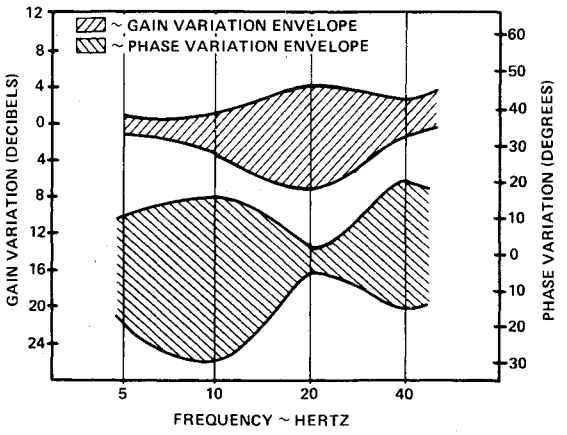

Fig. 6 Rate gyro gain and phase variation resulting from typical variations of rate gyro damping ratio.

Under normal conditions, synchronized computers provide identical outputs. If the computers are 16-bit machines, then a quad system will produce four outputs, each matched to the least significant bit of the computer's 16-bit word length. This accuracy is not needed for downstream control. Actually, the 16-bit accuracy is rapidly dissipated as the computed numbers are converted to the physical phenomena associated with aircraft control.

Figure 5 illustrates how the 16-bit accuracy is rapidly lost. It degrades to 12-bit resolution, with perhaps 10 bits of accuracy (because of noise effects) at the digital-to-analog (D/A) converter. The fidelity of the actuator in achieving the required position command is statically dependent upon the position sensor accuracy (about 2%), and dynamically dependent upon actuator and valve tolerances. Figure 5 shows a force-summing actuator in which dynamic and static discrepancies between channels will appear as a force fight. The 2% position feedback tolerance converts the original 16-bit system to one that is now less than 6 bits from an accuracy viewpoint. In maneuvers such as those illustrated in Fig. 4, the actuators would tend to be driven against their velocity and position limits, thereby commanding very large force fights during short dynamic intervals. The 8-deg discrepancy caused by the time skew could easily be accommodated by monitors designed to cope with the comparatively larger tolerances found in the dynamics of the redundant actuator responses.

Johnson[16] discusses the performance corruption in redundant asynchronous systems. He concluded that the penalties of asynchronous operation encountered in the USAF DAIS experiments were significant in that they caused excessive expansion of the tolerances that had to be accommodated by monitoring thresholds. He demonstrated how channel mismatch grows as the input signal bandwidth increases. This analysis highlights the problem of wide bandwidth monitoring rather than the issue of synchronous vs asynchronous computer operation. Reference 16 defines a typical input signal as a sinusoid of frequency f_i:

$$x = A_0 \sin 2\pi f_i t \tag{5}$$

Two asynchronous computers skewed by half a sample period will read the input x as $f_1(t)$ and $f_2(t)$:

$$f_1(t) = A_0 \sin 2\pi f_i K T_s \tag{6}$$

$$f_2(t) = A_0 \sin 2\pi f_i (K - \tfrac{1}{2}) T_s \tag{7}$$

where K is the sample number and T_s is the sample

period $= 1/f_s$. It was shown that the difference between $f_1(t)$ and $f_2(t)$ is

$$d(t) = f_1(t) - f_2(t)$$

$$= 2A_0 \sin[2\pi(f_i/4f_s)]\cos[2\pi(2K - \tfrac{1}{2})f_i/2f_s] \qquad (8)$$

At the sample frequency it is apparent that the discrepancy can be as high as $2A_0$. Reference 16 suggests a statistical threshold criteria, based on the percentage of time that a signal exceeds a threshold, and concludes that a tolerance of $A_0/\sqrt{2}$ is required. However, this type of analysis neglects the design experience that has been used to treat the high-frequency content of sensor information. Both the sampling frequency and the sensor bandwidths must be selected to match the process. When signal frequency content approaches the sampling rate, then noise and aliasing phenomena must be considered. Thus, appropriate prefiltering and postfiltering must be used to alleviate potential problems.

The apparent channel mismatch indicated by $d(t)$ actually will occur in a similar manner at the sensor voting plane in a synchronized system. This results because of the large dynamic tolerance in sensors, such as flight-control rate gyros. Figure 6, derived from Ref. 17, shows the tolerance band in phase and amplitude needed to cope with only the damping ratio tolerance of typical rate gyros that have been used in fly-by-wire systems. (The natural frequency tolerance band is zero, and scale-factor errors are also assumed to be zero.) A phase spread of 40 deg at 10 Hz will cause cycling of the mid-value selection, resulting in an error similar to $d(t)$ above. The practical solutions to this type of problem include filtering of the comparison algorithm and roll-off filtering on the output commands.

Unsynchronized Architectures

Figure 7 illustrates how redundant channels are managed in unsynchronized but loosely coupled architectures. The main method of cross-channel communications is via a broadcast bus output for each computer in which control commands, mode status, fault monitoring status, and important internal state information are transitted to the other channels. This information is broadcast without complex protocols or the need for a bus controller because transmission is continuous and only outward. Relatively high-impedance receivers accept this information so that failure effects at the receive end cannot disrupt bus operation the way they can in the multiplex bus systems of the 1553 type. Data rates of 100-200 kHz are more than adequate to support the most complex flight-control systems, using 100 to 200 iterations per second computation rates.

The key element in managing the time-skew effects discussed previously is the equalization loop which generates a channel-balancing correction identified as δ_{EQ1} in Fig. 7. It should be emphasized that this is not a complexity penalty because most of the functions performed in this loop are needed for the conventional cross-channel comparison functions. Channel 1's output command to a given control surface actuator is

$$\delta_1 = \delta_1' + \delta_{EQ1} \qquad (9)$$

where δ_1' is its control law solution and δ_{EQ1} is the computed channel equalization correction. The equalization is

$$\delta_{EQ1} = K_{EQ} \int F_1 F_2 (\hat{\delta} - \delta_1)\, dt \qquad (10)$$

where $\hat{\delta}$ is a middle-value selection obtained from δ_1 and the serial words δ_2, δ_3, and δ_4 received from channels 2, 3, and 4, respectively, K_{EQ} is the equalization gain, F_1 is a deadzone function computed from the magnitude $\hat{\delta}$, and F_2 is a switch

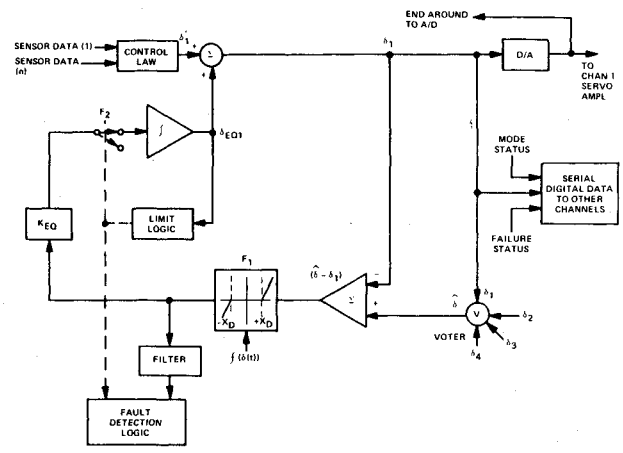

Fig. 7 Monitoring and equalization of loosely coupled channels.

(one or zero) which effectively limits the value of δ_{EQ}'. The intent of the δ_{EQ} correction is to prevent integral control law divergence (the integrator within the control law that generates δ_1'). All channels compute their control laws by using voted redundant sensor data. This is illustrated in Fig. 7 by showing sensor data (1) to (n). Although all computer channels used sensor data (1) to (n), their individual samples of that data may be time-skewed. Also, if the data are analog, then each channel's A/D converter will introduce its own error. Hence, a typical system will have static errors of about 0.1-0.5% full scale (1σ) for sensor conversion, with dynamic errors resulting from time skew rising to as high as 5% (1σ) during short time bursts within very high maneuver situations.

The intent of the equalization equation (10) is to correct static or long-term differences so that the channels track to about 8- to 10-bit accuracy. Thus, if 16-bit computations are made, the basic deadzone in $F_1(X_D)$ can be set to about 6 bits (or 63 least significant bits of the 16-bit computation). When large values of δ are commanded, one can expect larger contributions to time-skew errors, but these errors are voted or averaged out in the downstream actuator redundancy mechanism. There is no need to attempt equalization of these mismatch errors if they are short-lived. Hence, the deadzone X_D is increased as $\hat{\delta}$ is increased. However, if a large trim $\hat{\delta}$ is an expected equilibrium condition, then X_D must be reduced back to acceptable levels. The technique for accomplishing this is a washout on the $f(\hat{\delta})$ function

$$|X_D| = |X_{DO}| + |k_1\hat{\delta}(TS/TS+1)| + |k_2\hat{\delta}| \qquad (11)$$

The value $(\hat{\delta} - \delta_1)$ provides the basis for cross-channel comparison monitoring. The expansion and contraction of the deadzone is the method of computing the fault-detection threshold. Propagation of faults from one channel to another is prevented by the δ voting and by constraining the equalization integrator to a low value (F_2 on Fig. 7).

Mode Status Synchronization

The unsynchronized systems must be able to accommodate time skew in achieving mode transitions. Even synchronized computers can get into trouble with mode transitions if noise/power transients coincide with the transitions. The solution for asynchronous designs requires arming sequences in the logic and short-monitor inhibitions. If an event requiring mode transitions (such as a switch operation or penetration of a reference altitude) occurs at time t, then four asynchronous computers will recognize that event in specific routines that are time-skewed with respect to each other. The recommended approach is to have each computer generate a mode-enable state as soon as it recognizes the transition requirement. The enable state (contained within the mode status words) is then broadcast on the intercomptuer serial

bus, and enters the memories of the other computers via the direct memory access (DMA) channel which manages the serial data. The existence of an enable state for a majority of computers permits mode engagement by each computer. Concurrent with mode engagement, cross-channel comparisons and equalizations are inhibited for about two frames. Inadvertent mode engagements are prevented. Actual engagement will be time-skewed, but the effects of the different control computations are isolated from the monitors for an instant. In a 100-iteration-per-second system, the maximum time skew can be held to 10 ms, with an occasional overflow into 20 ms when the cross-channel data transfer is accomplished via the DMA channels (which do not require any software overhead).

Autonomous Architectures and Dissimilar Redundancy

The unsynchronized systems allow us to implement autonomous and dissimilar computers that have reduced vulnerability to a domino-type collapse as a result of generic faults. The intercomputer communication illustrated by Fig. 7 is not dependent upon cross-cannel comparisons to achieve fault detection. Individual computer channels are intended to be autonomous regarding monitoring, using concepts such as those discussed in Refs. 7 and 8, although a persistent comparison error can be used to declare another computer faulty. When a channel detects its own fault, existene of the fault is transmitted to the other computers by the absence of a health status code on the faulty computer's cross-channel data broadcast. Absence of this code causes the other computers to drop all data from the faulty computer in the equalization algorithm. With a return to acceptable health status, the computer's data are again accepted.

A variety of redundancy structures that exploit dissimilar mechanisms can be defined, but they are largely dependent upon aircraft control surface concepts, including the use of multiple combinations of surfaces with reversionary, degraded combinations to provide safety. This subject is beyond the scope of the present paper. An expanded version of the present paper[18] demonstrated a self-healing feature that can be added to an autonomous channel. The advantage of the autonomy or isolation from the other channels is that the normal channels require essentially zero software to accommodate the failure of a channel and its subsequent recovery.

Conclusions

Dissimilar redundancy and brick-wall separation strategies are viable approaches to overcoming generic-fault vulnerabilities. Their implementation requires unsynchronized channel operation.

As we build logical defenses against generic defects that can cause multichannel avalanche faults, we effectively increase each channel's self-monitoring capability, giving that channel more inherent autonomy. As a channel's autonomy improves, the need for multichannel synchronization tends to disappear.

References

[1] Tomlinson, L.R., "Control System Design Considerations for a Longitudinally Unstable Supersonic Aircraft," *Journal of Aircraft,* Vol. 10, Oct. 1973.

[2] Flapper, J.A. and Throndsen, E.O., "Integrity in Electronic Flight Control Systems," L-1011 Flight-Control System, AGARD-AG-224: April 1977.

[3] Eccles, E.S., "Softwre in Digital System: An Engineer's Approach," *IEEE Transactions on Aerospace and Electronic Systems,* Vol. AES-11, Sept. 1975.

[4] Caulfield, J.T., "Application of Redundant Processing to Space Shuttle," Presentation to SAE Controls and Guidance Committee, Oct. 28-30, 1981.

[5] Murray, N.D., Hopkins, A.L., and Wensley, J.H., "Highly Reliable Multiprocessors," AGARD-AG-224.

[6] Goldberg, J., "The SIFT Computer and Its Development," AIAA Paper 81-2278, Fourth Digital Avionics System Conference, Nov. 17-19, 1981.

[7] Osder, S.S., Mossman, D.C., and Devlin, B.T., "Flight Test of a Digital Guidance and Control System in a DC-10 Aircraft," *Journal of Aircraft,* Vol. 13, Sept. 1976.

[8] Osder, S.S., "DC-9-80 Digital Flight-Guidance System Monitoring Techniques," *Journal of Guidance and Control,* Vol. 4, Jan.-Feb. 1981.

[9] Van't Riet, R. and Thomas, F.R., "KC-10A Refueling Boom-Control System," *NAECON Proceedings,* 1980, p. 354.

[10] "USAF/General Dynamics Work on F-16 Electrical System Fixes," *Aviation Week and Space Technology,* Aug. 17, 1981, p. 29.

[11] "Columbia Exceeds Flight Goals – Quick Fix Leads to Faultless Performance," *Aviation Week and Space Technology,* April 20, 1981, p. 21.

[12] Osder, S.S., "Chronological Overview of Past Avionic Flight-Control System Reliability in Military and Commercial Operations," AGARDograph No. 224.

[13] Damman, L., et al., "Flight Test Development and Evaluation of a Multi-Mode Digital Flight-Control System Implemented in an A-7D (DIGITAC)," AFFTC-TR-76-15, June 1976.

[14] Rubenstein, S.Z. and Shroyer, L.O., "Digital Processing Subsystem for the Space Shuttle," *NAECON Proceedings,* 1974, p. 100.

[15] Hall, J., "Rockwell-Collins FCS-240 Digital Flight-Control System for the Lockheed L-1011 Tristar," Paper prepared for SAE Controls and Guidance Systems Committee, Feb. 1982.

[16] Johnson, T.W., "A Qualitative Analysis of Redundant Asynchronous Operation," *NAECON Proceedings,* 1978.

[17] Emfinger, J.E., "ACT System Design for Reliabiility, Maintainability, and Redundancy Management," SAE Paper 751052, National Aerospace Engineering and Manufacturing Meeting, Nov. 17-20, 1975.

[18] Osder, S.S., "Generic Faults and Design Solutions for Flight Critical Systems," Paper 82-11595 presented at the AIAA Guidance and Control Conference, Aug. 1982.