AIAA 81-2281R

# CPU Coverage Evaluation Using Automatic Fault Injection

Raymond P. Kurlak* and J. Robert Chobot†
*General Electric Company, Binghamton, New York*

**A method has been developed for evaluating central processing unit (CPU) coverage by automatically injecting faults into actual CPU hardware while it is executing relevant test software. A special hardware test fixture is used which contains the CPU under evaluation, program and data memory, and a terminal interface. The test fixture is connected to a microcomputer controller and an in-circuit read only memory (ROM) emulator. The faults injected are "stuck-at-1," "stuck-at-0," and open on the appropriate integrated circuit pins, plus the altered state of every microprogram memory bit. The effect of each fault is determined by observing the status of the watchdog monitor. Data reduction is performed after each run on a separate host computer, and summary results are tabulated.**

## I. Introduction

IN digital avionics systems the central processing unit (CPU) is obviously the dominant functional element, upon which many other elements are dependent in order to perform correctly related interface and control tasks. Normally, there are arrays of system monitors, in both hardware and software, which have an inherently overlapping capability to detect CPU malfunctions. The combined effectiveness of these monitors determines the total CPU fault coverage. However, since many of these monitors are themselves dependent upon CPU operation, there is a critical "top-down" relationship between the monitors which must be considered. In particular, a software monitor is of little subsequent value in the presence of a CPU fault unless one of the two following conditions is present.

1) It is actually executed, and it produces the planned fault detection results, and the fault detection is properly communicated to and acted upon by the ultimate hardware control elements in the system.

2) It causes fault detection elsewhere when it is not executed, and the fault detection is properly communicated to and acted upon by the ultimate hardware control elements in the system.

In simplest terms, software monitors depend upon a "rational" CPU; that is, a CPU which is at least able to follow the normal program flow without making addressing or branching errors. If this is not the case, then how can one guarantee that the software monitor is executed? If it is possible to detect the fault of concern without executing the software monitor, then why is the monitor present? The point is that it is critically necessary to have a CPU monitoring scheme which establishes the minimum CPU fault coverage that is required to support all other monitoring activity. Such a monitoring scheme must straddle the hardware-software boundary so that external communication is absolutely guaranteed. In the simple terms introduced in the preceding, such a monitor must be able to detect all faults resulting in "irrational" CPU operation.

In most modern digital avionic systems, the monitor which does these things is the so-called "watchdog" monitor (WDM) (or alternately the "heartbeat" or "deadman" timer monitor). The WDM usually operates in relation to the timer interrupt and the definition of minor and major computational frame times. The CPU is required to output a discrete pulse once per minor or major frame which meets specific time requirements for period and duration. If the WDM is not pulsed properly, it will "time-out" and indicate CPU failure.

This simple hardware device is able to comprehensively monitor CPU and program execution only if the program is properly constructed. Specifically, there must be an intricate sequence of software interlocks placed between the timer interrupt and the decision to pulse the WDM discrete. By using task completion flags, foreground/background checks, memory parity and checksum tests, and CPU instruction set execution tests, in series with the WDM pulse decision, a very effective monitor can be created. In this way the monitor is not only able to detect program execution flow errors, but also more subtle faults that only affect computed results (other than simple addressing arithmetic).

The focal point of this paper, then, is the evaluation of the minimum CPU fault coverage achieved via the WDM. The emphasis is on CPU operation from the point at which a timer interrupt strikes until the WDM discrete pulse is issued. The only software interlock that will be considered directly is a two-word flag that is conditioned during the successful execution of the CPU self-test routine. All normal operational program software related to this activity is included in the evaluation.

## II. The Evaluation Approach

In terms of its definition, fault coverage is a deceptively simple parameter; viz., coverage = the percentage of the significant failure rate which corresponds to those single faults that are properly detected by the applicable monitor(s). That is,

$$C = \frac{\lambda_{\text{detected}}}{\lambda_{\text{detected}} + \lambda_{\text{undetected}}} \times 100\%$$

In fault tolerant system reliability models this parameter has meaning as a conditional probability of (continued) system success given a particular fault condition. Notice that coverage is a single, not a multiple, fault parameter, and that it is not directly a ratio of the number of detected faults to the number of possible faults. Also, as used here, fault "detection" includes the process of acting upon the fault information to prevent the fault from affecting ultimate system operation.

The CPU coverage evaluation problem is to partition the hardware failure rate according to the effect of each fault as seen by the WDM. Since a typical avionics CPU contains

Presented as Paper 81-2281 at the AIAA/IEEE 4th Digital Avionics Systems Conference, St. Louis, Mo., Nov. 17-19, 1981; submitted Nov. 12, 1981; revision received July 2, 1982. 

*Manager, Computer Engineering. Aerospace Controls Systems Department, Aircraft Equipment Division. Member AIAA.

†Senior Engineer, Computer Engineering, Aerospace Controls Systems Department, Aircraft Equipment Division. Member AIAA.

30,000-40,000 fault possibilities, this is an enormous task. To address this task requires that three basic evaluation tools be defined.

1) A method is needed to evaluate accurately the effect of each selected fault under realistic operating conditions.

2) A criterion must be set for selecting the faults to be evaluated that balances the ideal of considering "all possible" faults, with the practical need to define a manageable task with a timely completion date and reasonably convincing results.

3) An analysis procedure is required for reducing the data to a usable form based upon failure rates which also includes treatment of any "unevaluated" faults, whether due to random sampling techniques or otherwise.

In developing the method that was used here, the choices considered in each area were as follows.

For fault effect evaluation, logic simulation, fault injection on actual hardware (either manually or automatically), and paper and pencil failure mode and effect analysis (FMEA) were considered.

For fault level criteria, gate level failure modes (i.e., internal as well as external device faults), device and connector "pin fault" level failure modes, functional level failure modes based upon the I/O functions of groups of devices, and combinations of the various levels just given were appraised.

For the analysis procedure stastical estimation based upon a small ($<1000$) random sample of faults, stastical estimation based upon a large ($>10{,}000$) sample of faults that are relatively more easy to evaluate, and an engineering estimate based upon elements of statistics (relative to failure rates, and the concept of "worst case" analysis) were examined.

The central questions involved with these choices concern the level and amount of faults to be considered, and the extent of the operational software that is to be included in the evaluation. The most rigorous and comprehensive approach would be to consider all possible gate level faults plus all relevant portions of the operational software. This leads to a logic simulation requirement that exceeds the known commercially available capabilities. The available logic simulators (TESTAID, DLASAR, etc.) are oriented to "test vector" generation for circuit card testing, and not to simulated execution of a program resident in memory. Even when artifices are used to simulate program execution, the size of the simulation is unwieldy and practically unmanageable in terms of run time. Over the years the authors have found only a few instances[1-3] where total logic simulation has been reported as successful. In these cases the logic simulators were in-house programs that were modified for the CPU coverage evaluation task. Even then, all possible gate level faults were not simulated but, rather, only a limited sample of faults was evaluated.

The key goal of this effort was to develop an evaluation method that does not require an extensive parallel software activity to develop or enhance a logic simulation program. Further, it was considered highly desirable that the evaluation method be as straightforward and understandable as possible, because the results were subject to review by both airframe manufacturers and concerned government agencies. Finally, it was a goal that the method be repeatable easily as often as necessary, because design iteration of the self-test software would allow performance improvement.

Generally speaking, there are areas of a digital computer where gate or functional level fault evaluation using manual FMEA is possible. The CPU is not one of these areas. Functional level fault analysis does not provide adequate resolution to sort out "irrational" CPU operation; and manual FMEA at the gate level is simply not able to trace all possible consequences of thousands of internal CPU faults when combined with complex software. Therefore, considering the aforementioned goals and the limitations of manual FMEA, we focused on fault injection as a method of attack.

The use of either manual or automatic fault injection on actual hardware, while it is executing relevant software, has the distinct advantage that there should be no doubt that one has truly determined the effect of the injected fault. The only disadvantages are that not all faults can be practically injected, and the time required to inject large numbers of faults could be excessive. For example, gate level (or internal device logic) faults cannot be handled by fault injection on actual hardware, unless each device is replaced with its NAND-gate hardware equivalent. This is not practical for most CPU designs. Depending upon the mix of small-, medium-, large-scale integrated. (SSI), (MSI), and (LSI), devices in the design, a NAND-gate equivalent model might contain 3000-10,000 gates.

In spite of the limitations with respect to gate level faults, we concluded that fault injection was a worthwhile approach to pursue. We decided that we could consider exhaustively all possible device pin faults plus all PROM/ROM bit faults in the microprogram memory. This fault set would constitute a large systematic sample of all possible faults on the CPU. Using the data from this evaluation we would then be in an excellent position to infer total CPU coverage.

To this end we developed a totally automatic means for injecting faults on actual hardware. We combined this effort with a manual FMEA treatment of discrete components (resistors, capacitors, and clock oscillator) plus the multilayer circuit card runs, solder joints, and connector pins. The resultant data provided a comprehensive basis for making an engineering estimate of total CPU coverage.

## III. Test Care Data

The CPU considered in this evaluation is the General Electric MCP-701B, which is in production of several current programs. The key characteristics of this CPU are given in Table 1.

The CPU component breakdown and the associated failure rates are summarized in Table 2. The failure rates are from MIL-HDBK-217C, notice 1, for an airborne-inhabited environment at 47.5°C average component case temperature. As will be seen later (in Table 3) not all of this failure rate constitutes a significant CPU failure. The percentage failure rate apportionment based upon failure mode for each component type is given in Table 4.

The integrated circuit (IC) device apportionment is based upon the assumption that, for SSI ($<20$ gates) and MSI ($<100$ gates) devices, it is reasonable to attribute all failure rates to I/O pin fault effects. For LSI, "electrical" and "mechanical" portions of the MIL-HDBK-217C model were used to apportion external (pin fault) and internal (gate fault logic) failure rates.

When all failure modes are considered, there are 40,921 possible faults in the CPU hardware as shown in line a of Table 3. This corresponds to the total failure rate of 12.014 failures per million hours. When all of the not used/spare hardware and the indistinguishable failures are subtracted out, there are 24,174 significant faults to be evaluated per line b in Table 3. Thus, only 59% of all possible faults and 77% of the total failure rate are significant to the CPU operation. For example, the 2901-bit slice microprocessor contains 16 general

**Table 1 Characteristics of the General Electric MCP-701B**

| | |
|---|---|
| Type | 16 bit, fixed point, parallel |
| Instructions | 120 (microprogrammable) |
| Size and power | Single multilayer card (7 × 9 in.) and 13 W |
| Throughout | 500 KOPS |
| Circuit technology | Low power Schottky (SSI and MSI) and 2901 bit slice microprocessor (LSI) |
| Microprogram | 512 × 56 = 28,672 bits |
| Memory microcycle clock | 4 MHz |

**Table 2 CPU failure rate (FR) summary**

| Component type | Quantity | FR (per $10^6$ h) |
|---|---|---|
| RLR Resistors | 14 | 0.00325 |
| RNC Resistors | 1 | 0.00008 |
| DIP Resistors (16 per) | 1 | 0.08600 |
| CSR Capacitors | 1 | 0.04620 |
| CKR Capacitors | 13 | 0.14365 |
| 20 MHz oscillator | 1 | 0.50000 |
| Multilayer board (1628 solder joints) | 1 | 3.41880 |
| Board connector (152 pins used) | 1 | 0.12008 |
| Bipolar SSI IC devices | 44 | 1.48587 |
| Bipolar MSI IC devices | 24 | 2.83175 |
| 2901A Bipolar LSI | 4 | 2.37068 |
| 5341 (512 × 8) PROM | 7 | 0.97856 |
| Test connectors | 4 | 0.02888 |
| Total | 116 | 12.01380 |

registers; the MCP-701B instruction set utilizes only 8 of these. The extra 8 registers and associated logic account for 2592 of the 2857 not used/spare hardware subtraction under the all other faults column. Examples of indistinguishable faults are pull-up resistors that fail open, de-coupling capacitors that fail open, and normally "high" or "low" signals that fail to their normal state.

## IV. Test Setup

The technique that was selected for automatically injecting pin faults is to insert electronic switching between each integrated circuit device pin and its associated connection on the CPU multilayer board. This was accomplished by replacing each dual in-line package (DIP) on the CPU with a socket, and then plugging a specially constructed fault injector board into each socket. The replaced DIP was then plugged back into the top end of the fault injector board. With this arrangement any single device pin may be switched to a stuck-at-1, stuck-at-0, or an open position. Shift register logic on each fault injector board is interconnected in a serial string

**Table 3 Data summary**

| | | IC Pin faults | | PROM Bit faults | | All other faults | | Total | | Average FR[a] per fault |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Qty | FR | Qty | FR | Qty | FR | Qty | FR | |
| a) | Total possible faults | 1679 | 6.200 | 2867 | 0.377 | 10570 | 5.437 | 40921 | 12.014 | 0.000294 |
| | Faults not significant: | | | | | | | | | |
| | Not used/spare hardware | −242 | −0.583 | −12843 | −0.169 | −2857 | −0.749 | −15942 | −1.501 | 0.000094 |
| | Failed state is indistinguishable from normal state by the hardware circuitry or software program | −143 | −0.0469 | −265 | 0.00348 | −397 | −0.792 | −805 | −1.264 | 0.000157 |
| b) | Total significant faults | 1294 | 5.148 | 15564 | 0.205 | 7316 | 3.896 | 24174 | 9.249 | 0.000383 |
| | Faults not evaluated | −27 | −0.109 | −1969 | −0.0259 | −6079 | −0.819 | −8075 | −0.954 | 0.000118 |
| c) | Total evaluted faults | 1267 | 5.039 | 13595 | 0.179 | 1237 | 3.077 | 16099 | 8.295 | 0.000515 |
| | Confirmed undetected faults | −2 | −0.00289 | −3999 | −0.0525 | −6 | −0.010 | 4007 | −0.654 | 0.000163 |
| d) | Total detected faults | 1265 | 5.036 | 9596 | 0.126 | 1231 | 3.067 | 12092 | 8.230 | 0.000681 |
| e) | Percentage line d with respect to line c | 99.8% | 99.9% | 70.6% | 70.6% | 99.5% | 99.7% | 75.1% | 99.2% | 132.2% |
| f) | Calculated estimate for total coverage | Coverage = $\frac{8.230 + (0.75)(0.9540)}{9.249} \times 100\% = 96.7\%$ | | | | | | | | |

[a] FR = Failure rate per million hours.

**Table 4 Failure rate apportionment**

| | | |
|---|---|---|
| Film resistors (RLR, RNC) | Open 88% | Short 12% |
| DIP resistors | Open 88% | Short 12% |
| Ceramic (CKR) capacitors | Open 36% | Short 64% |
| Tantalum (CSR) capacitors | Open 28% | Short 72% |
| 20 MHz oscillator | No output or incorrect frequency | 100% |
| Multilayer board runs and solder joints | Open 100% | |
| Connector pins | Open 100% | |
| SSI and MSI IC devices | Output stuck at 1 or 0, output open, or input open100%<br>Apportionment based upon pin count and type | |
| 2901 device (540 gates and 40 pins) | | |
| | Pin faults (as above) | 46% |
| | Internal logic faults | 54% |
| 5341 PROM (½096 bits and 24 pins) | | |
| | Pin faults (as above) | 62.5% |
| | Bit faults (state change) | 38.5% |

fashion so that the entire array of fault injectors is controlled by one serially shifted control word.

Figure 1 shows a block diagram of the entire CPU coverage evaluation test setup. The setup consists of a test chassis, PROM emulator, an HP-85 microcomputer, a fault system controller, and three actual circuit cards: the CPU with fault injectors installed, a RAM memory card, and a test console interface (TCI) card to provide an RS-232 interface for loading the memory. The MCP-701B CPU under test required 79 fault injector boards in five generic sizes: 14, 16, 20, 24, and 40 pin DIPS.

Each microprogram PROM device was replaced with a PROM emulator plug-in cable connected to a Step-II emulator. The entire microprogram was then loaded into the Step-II RAM so that each microprogram bit could be set to the opposite from normal state.

The fault system controller card is essentially an interface card between the IEEE 488 bus of the HP-85, the RS-232 and keyboard buses to/from the PROM emulator, and the control bus to the fault injector cards. Additionally, the fault system controller provides CPU clock control to start and stop a test sequence, and to time scale the clock for lower speed operation (necessary because of the increased distributed capacitance added by the fault injector boards). Finally, the controller monitors the status of the fault detection lines from the CPU, and in particular, the WDM.

The HP-85 provides overall control and data processing for the fault coverage evaluation. The computer provides a cathode ray tube (CRT) and keyboard, a printer, a mag-tape cassette, and IEEE 488 bus interface. A BASIC language program controls the automatic faulting of each device pin and PROM bit. For each fault, the status of all CPU fault indicators is recorded. At the end of a run this set of data is transmitted to a VAX 11/780 for postprocessing. The postprocessing is required in order to sort out "don't care" and insignificant faults from significant faults.

## V. CPU Self-Test Software

The CPU self-test routine is scheduled to be executed once per minor frame in the foreground computation in the actual system. When it is executed, all interrupts are disabled so that the test produces a fixed data flow sequence without pauses for interrupt servicing. Two 16-bit flag words are used to indicate a pass condition. The two words are generated by a series of partial sums distributed throughout the self-test routine. Two words are used in order to minimize the probability of random errors causing an erroneous pass indication. (If only one 16-bit pattern out of 65,536 possible patterns is required in each word, then the probability of two correct patterns occurring in sequence due to random errors is on the order of $2.3 \times 10^{-10}$ per double error.)

During the timer interrupt servicing both flag words are checked before the pulse is generated to satisfy the WDM. If the flag words do not indicate a pass condition, the pulse is not generated; and the WDM will time out and indicate a CPU failure. After checking the flag words, the timer interrupt service routine clears the words to zero, and then verifies that they are clear. If this check fails, the pulse to the WDM is also skipped. If, during execution of the CPU self-test, a fault is detected, the routine will execute a "jump-to-self" instruction which effectively stops the CPU, and thus causes the WDM to time out and indicate failure.

The CPU self-test routine was designed to take advantage of the fact that the minimum software code required to service the timer interrupt, and pulse the WDM, inherently yields about 69% coverage of CPU faults. Thus, the self-test routine is oriented to address the 31% not already covered by the time the self-test is executed. Generally, this reduces to instruction execution tests which exercise as many microprogram control states as practical. The entire routine consists of 589 program memory words and executes in 1.13 ms.

## VI. Test Results

A complete summary of the CPU coverage evaluation test data is given in Table 3. As already indicated, there are a total of 24,174 significant faults to consider in line b of Table 3. A total of 8075 of these were not able to be evaluated. The breakdown of this category of faults is as follows.

1) 27 pin faults were not testable in the test fixture because they require additional I/O interface signals not provided.

2) 1969 PROM bit faults were not testable because the state changes have no effect in the test fixture unless more than one failure exists (i.e., some cause fault indicators not to indicate), or unless additional I/O interfaces are present.

3) 6079 other faults were not testable, including which includes 31 connector pins and multilayer board hole connections that could not be tested as "opens," and 6048 internal logic faults for the 2901 devices (assuming 4 faults per each of 378 significant gates in 4 devices).

The total not evaluated was 8075. Even though these faults were not evaluated they have been classified entirely as "significant," which is conservatively pessimistic (i.e., the "worst case" condition). The treatment of these faults in the final coverage estimate will be described in later paragraphs.

Line c in Table 3 shows that a total of 16,099 faults were individually evaluated using the automatic fault injection test setup. The breakdown of the confirmed undetected faults is as follows.

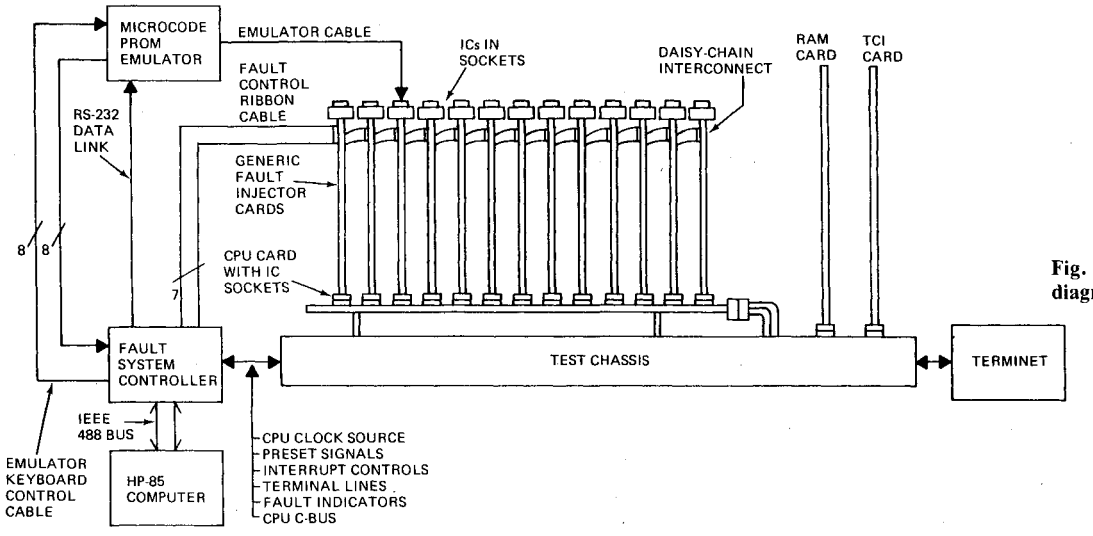


**Fig. 1 Test setup block diagram.**

1) There were 2 undetected pin faults [0.03% of the significant failure rate (FR)].

2) There were 3999 undetected microprogram PROM bit faults (0.57% of the significant FR).

3) There were 6 undetected "open" connector pins and multilayer board hole connections (0.11% of the significant FR).

The total number of undetected faults was 4007.

Examining line e in Table 3 shows that 99.9% of the evaluated pin faults (failure rate) were detected. On the other hand, only 70.6% of the total PROM bit faults (failure rate) were detected. In combination, because of the low relative failure rate of the PROM bits, a total of 99.2% of the evaluated faults (failure rate) were detected by the WDM and CPU self-test.

This coverage result for the evaluated faults is quite good and is more than acceptable for our intended applications. However, there remains the question of how to treat the unevaluated faults. The extreme worst case approach would be to consider them all undetected. A more optimistic, and perhaps more reasonable, approach would be to consider them to be detected in the same "proportion" as the large evaluated sample. Since we will be concerned here with the proportions based upon both failure rate and fault number ratios, it is necessary to define precise terminology. Let,

$\lambda_S$ = total significant FR
$\lambda_E$ = total evaluated (sample) FR
$\lambda_D$ = total confirmed detected FR (within the sample)
$\lambda_U$ = total unevaluated FR
$N_S, N_E, N_D, N_U$ = total number of faults corresponding to the respective failure rates already given

The total coverage estimate we seek is of the form

$$\hat{C} = \frac{\lambda_D + K\lambda_U}{\lambda_S}$$

where $K$ is an unknown proportionality factor to be determined. Ideally, one would like to have a complete statistical estimation theory that leads to a value for $K$ with an associated confidence level. Such a theory has been developed in preliminary form[4] but is not yet applied here. However, it is easily shown[4] that the best unbiased estimator for $\hat{C}$ is the coverage achieved in the evaluated sample

$$\hat{C} = \lambda_D/\lambda_E = 99.2\%$$

which leads to

$$K = \lambda_D/\lambda_E = 0.992\%$$

Since the precise confidence limit theory is not yet established to apply here, an engineering limit will be developed. Using the superscript bar to denote mean failure rate for each category, we have

$$K = \frac{N_D\bar{\lambda}_D}{N_E\bar{\lambda}_E} > \frac{N_D}{N_E} \text{ for } \bar{\lambda}_D > \bar{\lambda}_E$$

which means that,

$$K > \frac{12{,}092}{16{,}099} = 0.75$$

if $\bar{\lambda}_D = 6.81 \times 10^{-10}$ is greater than $\bar{\lambda}_D = 5.15 \times 10^{-10}$; which is true. Thus, 0.75 is a conservative lower limit for $K$ as long as we expect that in the total fault set population the mean detected failure rate is greater than the population mean failure rate. This assumption has been used to calculate the 96.7% coverage estimate shown in line f of Table 3.

An alternate statistical estimation theory is available[5] to apply here. It was not used because the relatively large number of undetected PROM bit failures conservatively biases the result, way out of proportion to the actual failure rate significance. The alternate method[5] is based upon estimating the single parameter of a binomial distribution, and is best applied when the number of undetected faults in a sample is small.

## VII. Conclusions

Subsequent manual analysis performed on the 27 unevaluated pin faults proved that all 27 faults were, in fact, detected in an actual system environment, which supports the contention that the 0.75 estimate for $K$ is a conservative lower limit.

Automatic fault injection of a large number of faults has been shown to be feasible for CPU coverage evaluation. Over 16,000 faults were evaluated, with a resultant coverage estimate of 96.7%. The test procedure is fairly easy to set up, is conveniently repeatable, and produces quick results on actual hardware. More work is required if internal logic faults (gate level) are to be treated in any way other than with statistical inference.

## References

[1] Hardie, F.H. and Suhocki, R.J., "Design and Use of Fault Simulation for Saturn Computer Design," IEEE Trans. Electrical Compt., Vol. EC-16, No. 4, Aug. 1967, pp. 412-429.

[2] Leaman, R.J., Lloyd, M.H., and Repton, C.S., "The Development and Testing of a Processor Self-Test Program," *The Computer Journal*, Vol. 16, No. 4, Nov. 1973, pp. 308-314.

[3] Yount, L.J. and Winkler, G.W., "Digital Computer Self-Test Confidence Level Validation Methodology," IEEE Paper CH1518-0/79/0000-0335, 1979.

[4] Nelson, W., "Estimation of Fault Coverage by Stastical Sampling of Faults," General Electric Corporate Research and Development Center, Jan. 1981.

[5] Bjurman, B. et al, "Airborne Advanced Reconfigurable Computer System," NASA CR-145024, 1976.