January 1989

UILU-ENG-89-2210
CSG-99

COORDINATED SCIENCE LABORATORY
College of Engineering

IMPACT OF DEVICE LEVEL FAULTS IN A DIGITAL AVIONIC PROCESSOR

Suk Ho Kim

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-89-2210 (CSG-99)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION NASA
6c. ADDRESS (City, State, and ZIP Code) 1101 W. Springfield Avenue Urbana, IL 61801	7b. ADDRESS (City, State, and ZIP Code) NASA Ames Research Center Moffett Field, CA 94035
8a. NAME OF FUNDING/SPONSORING ORGANIZATION NASA 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER NASA: NAG 1-602
8c. ADDRESS (City, State, and ZIP Code) (see 7b.)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Impact of Device Level Faults in a Digital Avionic Processor

12. PERSONAL AUTHOR(S)
Suk Ho Kim

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	December 1988	52

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	fault injection, fault propagation, device level fault, mixed mode simulation, Mean Error Durations, Mean Time Between Errors, near-coincident errors.

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This study describes an experimental analysis of the impact of gate and device-level faults in the processor of a Bendix BDX-930, flight control system. Via mixed mode simulation, faults were injected both at the gate (stuck-at) and at the transistor levels and, their propagation through the chip to the output pins was measured. The results show that there is little correspondence between a stuck-at and a device-level fault model, as far as error activity or detection within a functional unit is concerned.

(continued on reverse)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

In so far as error activity outside the injected unit and at the output pins are concerned, the stuck-at and device models track each other. The stuck-at model, however, overestimates, by over one hundred percent, the probability of fault propagation to the output pins. An evaluation of the Mean Error Durations and the Mean Time Between Errors at the output pins shows that the stuck-at model significantly underestimates (by 62%) the impact of an internal chip fault on the output pins. Finally, the study also quantifies the impact of device fault by location, both internally and at the output pins.

IMPACT OF DEVICE LEVEL FAULTS IN A DIGITAL AVIONIC PROCESSOR

BY

SUK HO KIM

B.S., University of Illinois, 1986

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1988

Urbana, Illinois

Acknowledgment: This research was supported by the National Aeronautics and Space Administration (NASA) under Contract NASA NAG 1-602.

ABSTRACT

This study describes an experimental analysis of the impact of gate and device-level faults in the processor of a Bendix BDX-930, flight control system. Via mixed mode simulation, faults were injected both at the gate (stuck-at) and at the transistor levels and, their propagation through the chip to the output pins was measured. The results show that there is little correspondence between a stuck-at and a device-level fault model, as far as error activity or detection within a functional unit is concerned. In so far as error activity outside the injected unit and at the output pins are concerned, the stuck-at and device models track each other. The stuck-at model, however, overestimates, by over one hundred percent, the probability of fault propagation to the output pins. An evaluation of the Mean Error Durations and the Mean Time Between Errors at the output pins shows that the stuck-at model significantly underestimates (by 62%) the impact of an internal chip fault on the output pins. Finally, the study also quantifies the impact of device fault by location, both internally and at the output pins.

ACKNOWLEDGMENTS

I wish to thank my advisor, Professor Ravi Iyer, for his guidance and encouragement. His support will always be appreciated. I also thank the researchers at the NASA Langley Research Center (AIRLAB), for many useful discussions. In particular I thank Bernice Becker for providing insight into the BDX-930 simulator. Thanks are also due to G. Choi, J. Singh, R. Llames, Luke Young and Jenny Marcinkiewic for their careful reading of an early draft of this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

A study of fault propagation and its impact is important for effective design of reliable and fault tolerant systems. Such a study, however, is difficult because the mechanisms involved are complex and hence not easily amenable to analytical modeling. In these circumstances an experimental study can not only provide valuable insight into the issues of fault occurrence and propagation, but also help develop a structured basis for future analytical analysis.

This thesis describes an experimental analysis of fault propagation and fault sensitivity in the processor of a Bendix BDX-930, flight control system. The processor was simulated using an event-driven, gate-level logic simulator, developed at NASA Langley Research Center, interfaced with a device-level circuit simulator (SPICE) [1]. Via mixed-mode simulation faults were injected both at the gate (stuck-at) and at the transistor levels, and their propagation through the chip to the output pins was measured. The nature and extent of the dependency of fault propagation on the type of instruction/microinstruction executed were also measured.

The results showed that, for device-level faults, in 5.1% of the cases, errors were detected within the injected unit, and in 20.9% of the cases errors were detected outside the unit (including 12.7% at the output pins); 74% remained undetected. For the stuck-at model, in 1.5% of the cases, errors were detected within the injected unit, in 41.8% of the cases errors were detected outside the unit (26.9% at the output pins); 56.7% remained undetected. The results also showed that there was little correspondence between a stuck-at and a device-level fault model in so far as error activity or detection within a functional unit is concerned. As far as error activity outside the injected unit and at the output pins are concerned, the stuck-at and device models tracked each other, although the stuck-at model overestimated, by over one hundred percent, the probability of fault propagation to the output pins. An evaluation of the

Mean Error Durations and the Mean Time Between Error at the output pins showed that the stuck-at model will significantly underestimate (by 62%) the impact of an internal chip fault on the output pins.

Measurement of error activity at the output pins showed that faults in different functional units affect the output pins to varying degrees, and that each unit had a distinct probability of affecting the output pins. This result suggests that by injecting pin errors with the measured probabilities we can easily emulate with-in chip faults for integrated system testing.

Chapters 2 and 3 contain a detailed description of the experimental procedure and measurements. Chapters 4, 5, 6, and 7 show the experimental results and their analysis. Chapter 4 compares the results from the gate-level and the device-level simulations. In chapter 5, the effect of fault placement in the AMD 2901 chip is defined and quantified. Chapter 6 shows the error characteristics at output pins for device-level faults. Chapter 7 describes the analysis of fault propagation according to instructions and microinstructions executed at the gate-level fault and the device-level fault. The final chapter highlights the important results and makes suggestions for future research.

1.1. Related Research

In recent years, there has been considerable research in the area of error and failure analysis of computer systems. In [2,3,4], automatically collected error data from several general-purpose computers are analyzed. By analyzing jointly, the performance and error data on several machines, valuable insight into error manifestation and discovery in large systems is provided. A series of experiments focusing on error analysis through fault insertion was conducted by several investigators at the NASA AIRLAB test-bed facility. A summary of these experiments is given in [5]. In [6,7,8], the evaluation and modeling of fault latency in digital avionic systems is investigated by determining the degree of fault latency in a redundant flight control system. In [9,10], further experiments to study fault and error latency distributions under varying workload conditions are discussed.

A detailed simulation experiment to study error propagation within a chip is discussed in [11]. The study develops a systematic experimental methodology to quantify error propagation via gate level

CHAPTER 1

INTRODUCTION

A study of fault propagation and its impact is important for effective design of reliable and fault tolerant systems. Such a study, however, is difficult because the mechanisms involved are complex and hence not easily amenable to analytical modeling. In these circumstances an experimental study can not only provide valuable insight into the issues of fault occurrence and propagation, but also help develop a structured basis for future analytical analysis.

This thesis describes an experimental analysis of fault propagation and fault sensitivity in the processor of a Bendix BDX-930, flight control system. The processor was simulated using an event-driven, gate-level logic simulator, developed at NASA Langley Research Center, interfaced with a device-level circuit simulator (SPICE) [1]. Via mixed-mode simulation faults were injected both at the gate (stuck-at) and at the transistor levels, and their propagation through the chip to the output pins was measured. The nature and extent of the dependency of fault propagation on the type of instruction/microinstruction executed were also measured.

The results showed that, for device-level faults, in 5.1% of the cases, errors were detected within the injected unit, and in 20.9% of the cases errors were detected outside the unit (including 12.7% at the output pins); 74% remained undetected. For the stuck-at model, in 1.5% of the cases, errors were detected within the injected unit, in 41.8% of the cases errors were detected outside the unit (26.9% at the output pins); 56.7% remained undetected. The results also showed that there was little correspondence between a stuck-at and a device-level fault model in so far as error activity or detection within a functional unit is concerned. As far as error activity outside the injected unit and at the output pins are concerned, the stuck-at and device models tracked each other, although the stuck-at model overestimated, by over one hundred percent, the probability of fault propagation to the output pins. An evaluation of the

simulation. To characterize the error propagation within the chip, distributions of error activity within the chip and at the output pins are generated. Based on these distributions, measures of error propagation and severity are defined. The analysis quantifies the dependency of the measured error propagation on the location of the fault. The study also shows the nature and extent of the dependency of error propagation upon the type of microinstruction and assembly level instruction executed.

Our experience with large circuits has shown that there are only certain sections or paths that require simulation with the highest level of detail, while the simulation accuracy for the rest of the circuit is less critical. To optimize the cost-accuracy tradeoff, one should be able to specify the level of detail required by selecting the simulation mode for each module. The current effort based on mixed-mode simulation is initiated to meet this need. To date, there has been no research to investigate fault propagation from the device to the pin level. This information is crucial for flight-critical digital systems, and additionally would allow the determination of the effect of placement on fault propagation.

CHAPTER 2

THE EXPERIMENT

The system targeted for this study is the CPU in the Bendix BDX-930, which is a digital avionic *miniprocessor*. The BDX-930 is used in a number of flight control avionic systems, e.g., in SIFT [12], and in AFTI F-16 [6]. Fault tolerance is achieved by replication of the processing and voting in software. The BDX-930 consists of 86 microcircuits printed on one circuit board [13]. The processor is designed around the AMD 2901 four-bit microprocessor slice [6]. In our experiments, the processor was simulated using an event-driven, gate-level logic simulator (developed at NASA Langley) interfaced with a device-level circuit simulator (SPICE). Since the AMD 2901 is the most complex chip in the BDX-930, it was used for fault injection and error data collection. In the simulations, fault propagation data were collected at device and gate levels as well as at the output pins. From the data provided by simulations, issues relating to fault propagation and fault sensitivity of the chip architecture were addressed.

2.1. Mixed Mode Simulation

The simulator [14], designed at NASA AIRLAB, is an experimental tool to simulate fault and reliability checking for Bendix BDX-930. This simulator is an event-driven, gate-level, unit delay logic simulator, and includes the CPU with its instruction set, the memory, and sections of the program memory containing six application programs and a self-test program. The simulation model is based on the circuit schematic of the AMD2901 and includes all of the devices identified in those schematics. Each device is represented by a gate-level equivalent circuit supplied by the chip manufacturer. Six gate types are used to represent devices, i.e., NAND, AND, OR, NOT, NOR, Exclusive OR. Unit delay is assumed between logic gates.

Although this simulator was quite accurate for gate-level simulation, it could not simulate faults occurring at the transistor level. By interfacing the gate-level simulation with a circuit-level simulator, SPICE2 [1], a method that permitted the injection of transistor level fault and the observation of fault propagation at the gate or module level was implemented. Thus, by using a combination of circuit and gate-level simulation, we could achieve the accuracy of circuit-level fault injection and the speed of gate-level simulations.

An important issue in any mixed-mode simulation is accurate analog-to-digital signal conversion. In our simulator, once the gate in which the fault is going to be injected is chosen, the SPICE simulator runs only for the faulty gate according to the fault model inside the gate. The SPICE generates an analog output which ranges from zero to five volts. Because logic values (one or zero) are required for the gate-level simulator, a subroutine is needed to convert the analog voltages to logic values. In order to get proper logic values, the analog voltages are sampled and averaged in the scanning window through the time axis. The averaged voltages are evaluated by assuming higher than 4.2 volts as a logic one and lower than 0.8 volt as a logic zero to determine the corresponding logic values. After acquiring the values from the SPICE run, the rest of the simulation occurs at the gate-level.

2.2. Fault Injection

Based on previously published results [15], physical failures may generally be divided into two categories, device failures and interconnection failures. In this study we consider only device failures. In [16], it is reported that of the device faults the most likely are oxide level faults and metal faults. Typically, according to [16], 68% are oxide faults and the remaining 32% are metal faults. We used these percentages for determining the types of fault to inject in the simulations.

Two-hundred forty-five gates corresponding to a Bendix circuit diagram for the AMD 2901 were selected for fault injection. In order to have consistent statistical results, 700 device faults and 300 gate-level faults were injected. The target gates were randomly selected[1] among the twelve functional units of

the AMD 2901 except for the Q Register. The Q register was excluded to avoid effects of the fault latency which have been studied elsewhere [6,8]. The units into which faults were injected were RAM Shift, Q Shift, Multiplexor, Arithmetic Logic Unit, Ram Control, Output, Output Data Select Unit, Destination Control, ALU Control, and Source Control. Table 1 shows the number of gates and transistors in these units.

TABLE 1: Number of Gates and Transistors in AMD 2901

SEC NUM	SEC NAME	NUM OF GATE	NUM OF TRANS
1	Q Register	0	0
2	Ram	12	64
3	Q Shift	16	100
4	Source Contl	4	8
5	ALU Control	6	24
6	Output	6	38
7	Output Select	9	50
8	Ram Shift	16	100
9	Ram Control	83	572
10	Dest Control	10	36
11	ALU	51	326
12	MUX	32	224

[1]While sequential injection such as used in [11] is exhaustive, it is not practical in modeling the behavior of physical faults since the numbers can be very large.

CHAPTER 3

MEASUREMENTS

The simulator and the fault injection facility programs were written in Fortran 77 and in VAX/VMS system language (Digital Command Language) [17]. Based on the random fault injections, a total of 700 device-level and 300 stuck-at fault simulations were performed to obtain experimental data on the fault propagation characteristics. Each simulation corresponded to executing the sequences of microinstructions of a CPU-test program[2]. First, a "gold" or unfaulted simulation run was performed. Next, one-thousand simulation runs, each containing an injected fault, were performed. For each faulted simulation, a comparison was made with the gold simulation to generate the error data for subsequent fault propagation analysis. An error was defined as follows:

1) A gate activated in the gold simulation but not activated in the faulted simulation.

2) A gate activated in the faulted simulation but not activated in the gold simulation.

3) A gate activated in both simulations for the same time slice but with different logic values.

The results of these measurements enabled us to analyze the dependency of the error propagation on the location of the fault and the type of instruction/microinstruction executed and to characterize the error activity at the output pins.

3.1. Data Collection

After each simulation run and prior to the next run, the output consisting of time stamps, a trace of gate activity and logic values was appended to the existing output file. The output file created by each run was then compared with fault-free data from the gold simulation to generate an "Error Data File." Table 2 shows a sample of an Error Data File. To provide detailed information to the data set, many

[2]There are four individual subsets within the self-test program, i.e., the cyclic RAM test, the CPU test, the ALU test, and the memory address processor test.

independent variables were included in the raw data as shown in the table. The first column indicates the simulation number. Columns 2, 3 and 4 show the gate name, the unit name and the unit type where the fault injections were made respectively. Column 5 specifies the gate type into which the fault was injected. The type of the injected faults is shown in columns 6 and 7. For example, in simulation 1, a fault corresponding to an oxide breakdown was injected at device-level. Column 8 shows whether or not an error was detected in that simulation run. Column 9 shows the time slice during which an error resulting from the injected fault was detected. The remaining columns show the number of gates affected by the injected fault in each unit (for brevity, not all the units are shown in the table). For example, in simulation 5, at time 887, eight gates in the RAM, one gate in the Q-Shift and four gates in the MUX were affected due to the fault in the gate GAMULT6CPU32.

To identify the fault propagation through the chip as well as the output pins, information about time, number of faults and name of affected unit during the propagation were traced from the fault injection point to all other units in the chip and output pins. These measurements were used for determining the percentage of the faults which propagated out of the unit and the percentage of the other units affected given that the fault did propagate out of the unit.

Because study of fault distribution to the output pins is of great practical significance, obtaining precise output data from the simulation was crucial for analyzing and evaluating a system at the output pin level. The output pin data were collected in order to observe how each pin behaved in the event of a fault condition in the system. In particular, simultaneous occurrences of errors and the probability of near-coincidence of pin errors were investigated. Also, these data were used to characterize each pin based on the mean time between errors and mean error durations.

TABLE 2: Sample of Error Data File

1 SIM. NUM	2 GATE NAME	3 SECT NAME	4 SECT TYPE	5 GATE TYPE	6 FLT- TYP1	7 FLT- TYP2	8 ERROR	9 TIME	10 Q REG	11 RAM	12 Q SHIFT	13 MUX
1	GA1LCPUIC32	RAM	Memory	Or	Device	Oxide	0	-	0	0	0	0
2	GATE7CPUIC32	RAM CNTL	Control	And	Device	Oxide	0	-	0	0	0	0
3	GR0B1CPUIC32	ALU	Logic	Nxor	Device	Oxide	0	-	0	0	0	0
4	GATE10CPUIC32	RAM CNTL	Control	And	Device	Metal	0	-	0	0	0	0
5	GAMULT6CPUIC32	RAM CNTL	Control	And	Device	Oxide	1	396	0	0	0	0
5	GAMULT6CPUIC32	RAM CNTL	Control	And	Device	Oxide	1	397	0	4	0	0
5	GAMULT6CPUIC32	RAM CNTL	Control	And	Device	Oxide	1	398	0	8	1	0
5	GAMULT6CPUIC32	RAM CNTL	Control	And	Device	Oxide	1	887	0	8	1	4
5	GAMULT6CPUIC32	RAM CNTL	Control	And	Device	Oxide	1	888	0	8	2	8
5	GAMULT6CPUIC32	RAM CNTL	Control	And	Device	Oxide	1	889	0	8	2	8
5	GAMULT6CPUIC32	RAM CNTL	Control	And	Device	Oxide	1	890	0	8	2	8
5	GAMULT6CPUIC32	RAM CNTL	Control	And	Device	Oxide	1	891	0	8	2	8
5	GAMULT6CPUIC32	RAM CNTL	Control	And	Device	Oxide	1	892	0	8	2	8
5	GAMULT6CPUIC32	RAM CNTL	Control	And	Device	Oxide	1	893	0	8	3	8
5	GAMULT6CPUIC32	RAM CNTL	Control	And	Device	Oxide	1	894	0	8	3	8
5	GAMULT6CPUIC32	RAM CNTL	Control	And	Device	Oxide	1	895	0	8	3	8
5	GAMULT6CPUIC32	RAM CNTL	Control	And	Device	Oxide	1	896	0	8	3	8

Information concerning the instruction and microcode activity was collected concurrently with the gate activity data. By knowing which microaddress had been accessed, the executed microinstruction was uniquely identified. Finally, by examining the sequence of microinstructions, the macro (or assembly) level instruction that was executed was determined.

CHAPTER 4

COMPARISON OF PHYSICAL AND STUCK-AT FAULT INJECTIONS

With the increasing complexity of VLSI circuits, there is a growing concern that fault simulation based on stuck-at faults is not adequate. In this section, the similarities and differences between the results of gate-level fault injections and device-level fault injections are discussed. The comparisons are based on the measured error activity resulting from gate-level and device-level fault injections into the same functional unit.

Four different types of comparisons are made. The first comparison is based on the detectability of the injected faults inside the injected units. The second is based on the percentages of faults that were detected outside the injected unit (i.e., the measured fault propagation). The third comparison is based on the extent of error activity outside the injected unit. Finally, the pin-level error activity resulting from gate and device-level fault injections are compared.

Figure 1 shows the percentages of injected faults detected within the injected unit. Percentages for both stuck-at and device-level fault injections are shown. The vertical axis indicates the location of the fault, and the horizontal axis indicates the percentage of the injected faults detected within the unit. A number of observations can be made from this figure. First, device faults have higher percentages of being detected within the unit as compared to gate-level faults. This is reasonable because there are fewer levels of signal transitions at the gate-level than at the device-level. For example, a given device-level fault may propagate through 25 transistors before getting outside the unit while a gate-level fault may only have two or three logic levels to go through. In comparing the relative behavior of stuck-at and device-level faults across the functional units in Figure 1, we see that they do not track each other. Thus, the results show that there is little correspondence between the behaviors of device and stuck-at faults.

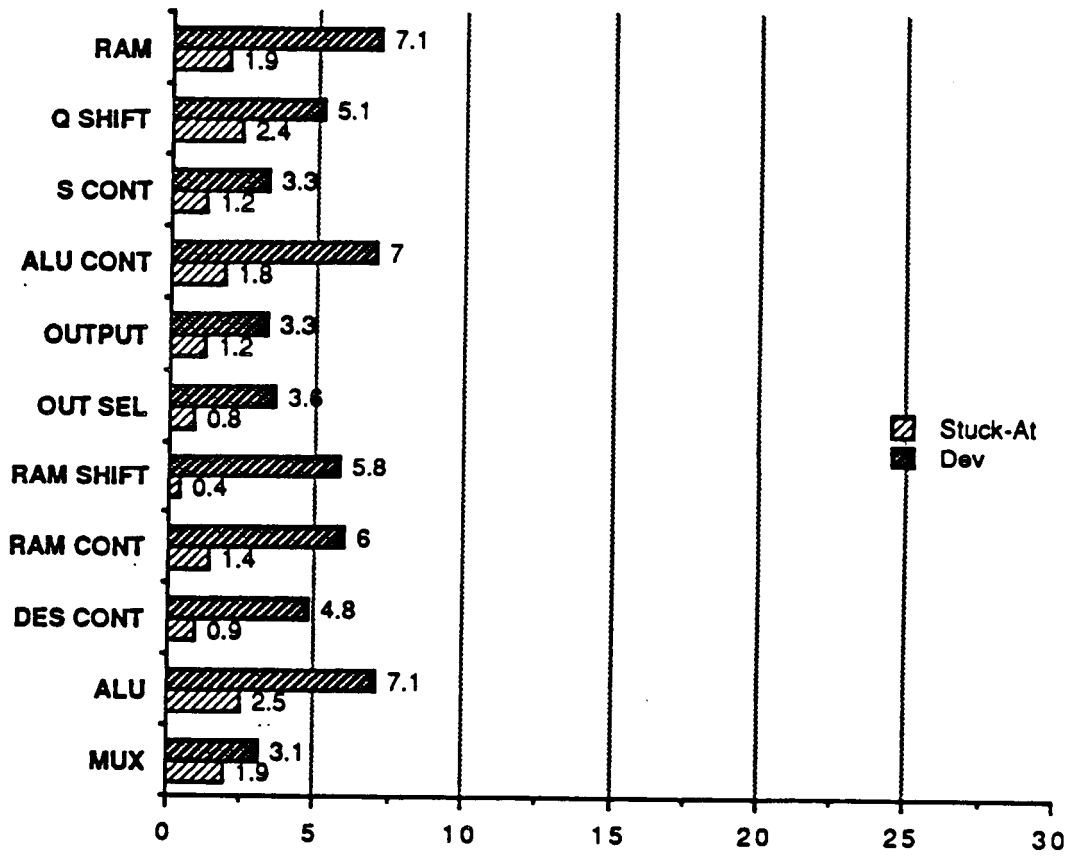

Figure 1: Percentages of Faults Detected That Remained in the Unit

Figure 2 shows a comparison of the percentages of faults detected outside of the injected unit, for stuck-at and device faults. Again, the vertical axis indicates the location of the fault, and the horizontal axis is the percentage. The figure shows that although the stuck-at and the device-level fault behaviors track each other, their percentages are quite different. On the average, the stuck-at faults tend to propagate approximately twice as frequently outside the unit as compared to device faults. Thus, by assuming a stuck-at fault model, although the relative impact of a fault on other units reasonably may reflect the physical failures, the results are likely to be considerably pessimistic.

In the above-mentioned case, we were concerned with whether or not error activity due to a fault is detected outside of the unit in which a fault is injected. The next comparison is based on the "extent" of measured error activity outside the injected unit. For example, if a fault injected in the ALU unit is detected in four other functional units, the impact of the fault may be quadrupled due to propagation. To quantify this effect, a new measure "the propagation factor" is defined. The propagation factor is defined as the average number of external functional units affected due to the fault in a specified functional unit. This factor can be calculated by dividing the sum of error activity outside the injected unit by the measured error activity inside the injected unit. Table 3 shows the propagation factors for the stuck-at and the device-level fault injections in each unit.

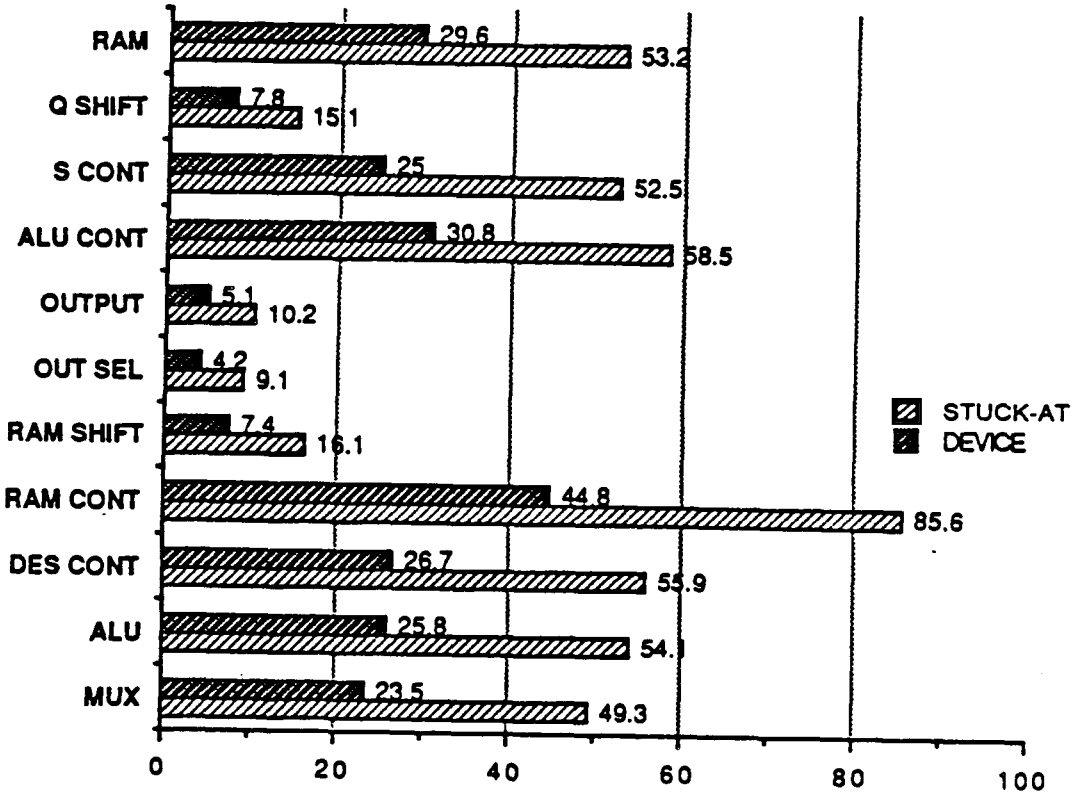

Figure 2: Comparison of Fault Propagation Going Outside of the Unit

TABLE 3: The Propagation Factors

Faulted Unit	Pro.Factor for Stuck-at	Pro.Factor for Device
RAM	6.6	7.9
Q SHFT	3.0	5.4
S CONT	8.3	6.8
ALU CON	8.4	7.3
OUTPUT	0.5	2.0
OUT SEL	10.0	10.0
RAM SHFT	3.4	7.0
RAM CONT	2.9	5.0
DES CONT	7.7	7.5
ALU	6.0	6.1
MUX	6.4	6.8

In Table 3, the first column shows the units in which the fault injections were made. The next two columns are the propagation factors for stuck-at faults and device faults, respectively. Here again there is no clear correlation across all units between device and stuck-at faults. There is also a higher variability (2.92 vs. 2.01) in fault propagation with stuck-at faults. An examination of the table shows that for most of the units (except the microinstruction decode units which are the Source Control, the ALU Control, and the Destination Control) the stuck-at faults had a smaller propagation factor than device faults, i.e., functional units are more sensitive to the device faults than to the stuck-at faults. The only exceptions are the faults in the microinstruction decode unit which have the opposite effect.

Finally, the impact of gate and device-level faults on the output pins was compared. A comparison of the percentages of faults which affected the pins (similar to Fig. 2) and the propagation factors (similar to Table 3) showed that the stuck-at and the device faults did track each other as shown in Appendix A. Comparisons were also performed based on the Mean Time Between Errors and Mean Error Durations at the pins. The Mean Time Between Errors is obtained by computing the average time interval between two consecutive errors on the pin. The Mean Error Durations indicates the holding time of the error at the specified pins. The MED is calculated by averaging the time between the instance of error occurrence

and disappearance. By examining these values, the impact of an internal fault on the external environment can be estimated. The Mean Time Between Errors and the Mean Error Durations on the pins for both simulations are shown in Table 4.

TABLE 4: MTBE and MED for Stuck-at and Device Faults

PIN NUM.	STUCK-AT		DEVICE	
	MTBE	MED	MTBE	MED
PIN1	345.1	45.9	303.8	47.8
PIN2	241.0	26.6	189.1	27.2
PIN3	218.0	57.1	189.6	58.2
PIN4	408.3	40.8	296.7	46.4
PIN5	215.0	31.9	185.6	41.6
PIN6	465.6	16.9	369.4	17.5
PIN7	184.7	67.1	147.6	67.7
PIN8	166.8	62.5	141.1	66.9
PIN9	169.8	59.8	144.4	61.7
PIN10	176.6	60.2	148.1	60.6

The first column shows the pin numbers. The next two columns are the MTBE and the MED for the gate-level simulation and the device-level simulation. The MTBEs for the stuck-at faults are longer, and the MEDs for the stuck-at faults are shorter than those for the device faults. Typically, the MTBEs of Pin 4 and Pin 6 for the stuck-at faults were 100 time steps longer than those for the device faults. The shorter the MTBE, the more likely it was that the error would propagate although it was also easier to detect the error outside the chip. The shorter the MED, the less likely it was that the error would propagate outside the chip. Note that the MTBE for the stuck-at faults was larger (72% - 88%) and the MED was shorter (1% - 30%) than the corresponding values for device faults. Thus, stuck-at faults were less likely to propagate outside the chip. Since device faults had the longer duration, they were more likely to exert an impact external to the chip. Thus, assuming a stuck-at model for failures may underestimate the fault propagation characteristics external to the chip.

In summary, results of the measurements show that 5.1% of the device faults are detected within the injected unit, and 20.9% of faults are detected outside the unit (which include 12.7% at the output pins) and 74% remain undetected. In the stuck-at case, 1.5% of faults are detected within the unit, and 41.8% of faults are seen outside the unit (26.9% are at the output pins), and 56.7% are not detected. Results show that there is little correspondence between stuck-at and device-level fault models as long as error activity or detection within a functional unit is concerned. In so far as error activity outside the injected unit and at the output pins are concerned, the stuck-at and device models closely track each other although the stuck-at model overestimates by approximately one hundred percent fault propagation of the chip. At the pin level, although the percentages of errors for stuck-at and device faults do track each other, an evaluation of the Mean Error Durations and the Mean Time Between Error shows that the stuck-at model will significantly underestimate (by 62%) the impact of an internal chip fault on the external environment.

The comparisons between the gate-level simulation and the device-level simulation based on the instruction/microinstruction executed are shown in Appendix B. Since it is clear that the device-level injection is more accurate and realistic, we consider only the device-level fault in the remainder of this study.

CHAPTER 5

EFFECT OF FAULT PLACEMENT

This section discusses the effect of fault location on propagation through the chip. Recall that Figure 2 shows the percentages of the faults which propagate outside the injected unit. For example, among all the faults injected in the ALU unit, only 25.8% of those faults were detected outside the ALU. The functional unit with the highest percentage of external propagation was the RAM Control (44.8%), and the units with the lowest level of propagation were the output units which include the Output and the Output Select. The results for the RAM Control are not surprising because this unit has a large fan-in and fan-out (it controls many input and output paths around the RAM unit), and is the most complex unit in the system. Therefore, more faults in this unit tend to propagate to other functional units. The results also show that the Output Select and the Output units are least likely to propagate to other functional units. This result is intuitive since there is little feedback from these units to the other units. However, faults in these units will almost certainly affect the output pins as will be shown in Chapter 6. A relatively high percentage of faults in the microinstruction decode units (Source Control, ALU Control and Destination Control) tend to travel out of that unit. These results also seem reasonable because the microinstruction decode units are extremely important in the correct operation of the processor. Further, the faults in the ALU behave somewhat similarly to those in the MUX because the MUX outputs feed directly into the ALU. A very low percentage of faults in the RAM Shift and the Q Shift units propagate outside. One explanation is that the RAM Shift and Q Shift units are used primarily for the multiplication and division instructions. These instructions are not highly used in the self-test program employed in this study.

Given that a fault propagates out of the injected units, the probability that the injected fault affects the other units is shown in Table 5. The first column shows the units in which the faults were injected. The remaining columns show the other units in which the faults may be detected. Each entry shows the probability with which the injected fault affects each unit.

By examining Figure 2 and Table 5 together, we can get a clear picture of fault propagation in the chip. For example, in the ALU 25.8% of injected faults propagated outside of the ALU (refer to Fig. 2), and, as shown in Table 5, 32.5% of these faults affected the RAM unit, and 90% affected the Q-Register and so on. As expected, the ALU is strongly affected by faults in other units. The converse, however, is not true, e.g., faults in the ALU do not affect the microinstruction decode units (Source Control, ALU Control, and Destination Control). They do, however, propagate to the output. Although the Output Select is not very likely to impact the other units (Fig. 2), when it does, several other units are uniformly affected. As explained earlier, this is most likely due to the fact that the Output Select unit has several data paths which feed back to many other functional units. Because the ALU and MUX are closely located, the faults from these units act similarly. Faults in the microinstruction decode units (Source Control, ALU Control and Destination Control) have a high probability of fault propagation, and given that propagation occurs the other units are uniformly affected. The table also shows the Output unit is severely affected by faults originating in most of the functional units. Thus, given that a fault propagates outside the injected unit, it is very likely to affect the output pins.

TABLE 5: Probability of Units Affected

faulted Section	RAM	Q-SHF	S-CNT	ALU-C	OUT	OUT-SL	RAM-SHF	RAM-CNT	DS-CNT	ALU	MUX
RAM	--	0.875	0.500	0.375	0.875	0.875	0.875	0.875	0.250	0.875	0.875
Q SHFT	0	--	0	0	0.778	0.778	0.778	0.778	0	0.778	0.778
S CONT	0.310	1.0	--	0	1.0	1.0	1.0	1.0	0	1.0	0.500
ALU CON	0.250	1.0	0.250	--	1.0	1.0	1.0	1.0	0.250	1.0	0.500
OUTPUT	0.250	0.250	0	0	--	0.250	0.250	0.250	0	0.500	0.250
OUT SEL	1.0	1.0	1.0	1.0	1.0	--	1.0	1.0	1.0	1.0	1.0
RAM SHFT	1.0	1.0	0	0	1.0	1.0	--	1.0	0	1.0	1.0
RAM CONT	0.632	0.444	0.248	0.137	0.444	0.444	0.444	--	0.017	0.444	0.444
DES CONT	0.500	1.0	0	0	1.0	1.0	1.0	1.0	--	1.0	1.0
ALU	0.325	0.900	0.225	0.225	0.950	0.900	0.900	0.900	0.225	--	0.425
MUX	0.311	0.878	0.389	0.389	0.878	0.878	0.878	0.878	0.389	0.833	--

In summary, the results demonstrate that many faults, about 25% of the injected faults in the ALU and the MUX, tend to propagate and also that these units are strongly affected by faults in other units. (Around 60% of faults in other units affected these units.) Faults from the microinstruction decode units have uniform impact on the system overall. Given that faults propagate outside the injected unit, the faults severely affect the Output unit, i.e., the output pins are most likely to be affected by the faults.

5.1. Comparison of Fault Distributions of Different Material Failures

Figure 3 shows the frequency distributions of faults occurring based on the different materials (oxide or metal). The vertical axis represents the frequencies of faults detected at a specific time over the entire simulation. The horizontal axis has the time steps in one clock cycle (with one clock cycle consisting of 70 time steps). In the plot, the solid line depicts the overall fault distribution, the dashed line is the distribution of the oxide fault, and the dotted line represents the distribution of the metal fault. The plot is generated by overlaying the 50 clock cycles in the sample and plotting the frequency of upset in the system for each of the 70 time steps. Mostly, there was little activity beyond 30 and below 5 time steps per clock cycle.

Because the device-level fault injection was performed in this study, two materials,e.g., the oxide and the metal, were involved in the fault injection. In the previous chapter, 68% of the faults were injected into the oxide, and the remaining 32% of the faults were injected into the metal. Numerically, more than twice the fault injections were performed in the oxide material. In Figure 3, the fault distribution of the oxide is closer to the overall distribution than that of the metal. Surprisingly, the frequency distribution of the oxide was not twice that of the metal as we would have expected. From this result it appears that faults in the metal can more actively affect the system than faults in the oxide material.

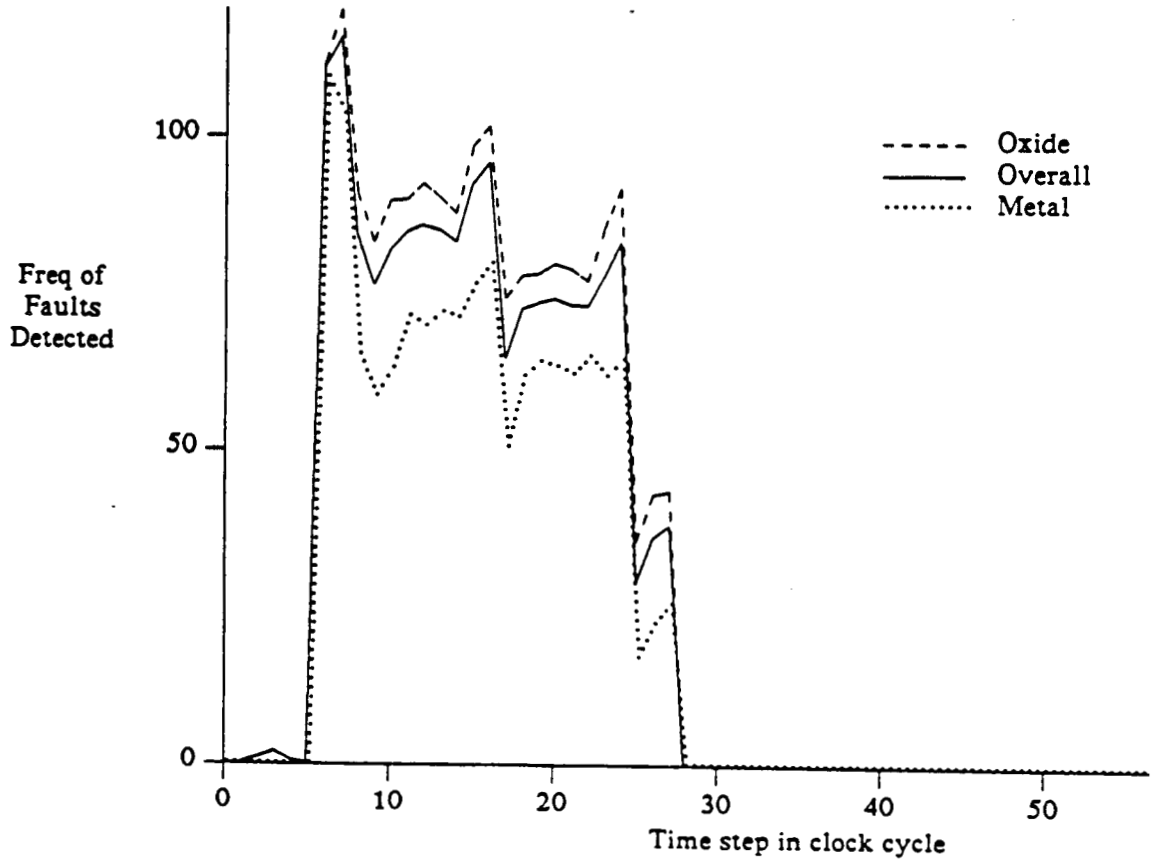

Figure 3: Activity Comparisons Between Materials

Table 6 shows the impact of the metal and oxide faults as a functions of the unit into which the fault was injected.

TABLE 6: The Effect of Fault in Oxide and Metal

SEC	Fault in Metal	Fault in Oxide
RAM	46.4 %	58.4 %
Q SHFT	66.7 %	59.6 %
S CONT	24.6 %	25.5 %
ALU CON	18.8 %	19.7 %
OUTPUT	63.8 %	63.5 %
OUT SEL	63.8 %	59.8 %
RAM SHFT	63.8 %	59.8 %
RAM CONT	63.8 %	59.8 %
DES CONT	7.2 %	13.9 %
ALU	65.2 %	63.5 %
MUX	40.6 %	59.1 %

The first column shows the unit into which fault injection occurred. The percentages of the injected faults which resulted in some error activity are shown in the next columns. These two columns of numbers show a distinct similarity. Even though fewer faults were injected into the metal, the percentages of faults that affected the unit are about same; consequently, we can conclude that the faults in the metal actively affected the units.

CHAPTER 6

CHARACTERIZATION OF ERRORS ON OUTPUT PINS

Previous sections of this paper have defined and measured fault propagation throughout the AMD 2901 bit slice processor. In this section, characteristics of the error activity at the output pins are studied. Our previous results show that the output unit is very sensitive to the location of the fault within the chip. The output pins are also the place at which a fault can affect the external environment. Therefore, appropriate measurements and accurate evaluations of the output pins are absolutely necessary for evaluating the performance of a system. The AMD 2901 has 40 I/O pins around the body of the chip. Of these pins, ten are used for the output lines of the chip, and the others are used for inputs, power lines, tri-states, etc. Because individual pin data are obtained from the simulation and analyzed for the characteristics of each pin, the functions of each pin are worth considering carefully. Instead of using real pin numbers, ten numbers from one to ten are used for convenience. Table 7 shows the names of ten pins and short descriptions of them.

TABLE 7: Pins of AMD 2901

PIN NUM	PIN NAME	DESCRIPTION
PIN1	GCN4CPUIC32	Carry out
PIN2	GF3BCPUIC32	Bit7 ALU out
PIN3	GFEOCPUIC32	ALU Zero out
PIN4	GGBARCPUIC32	Carry prop.(P)
PIN5	GOVRCPUIC32	Overflow output
PIN6	GPBARCPUIC32	Carry gen.(G)
PIN7	TSY0CPUIC32	Y00 output
PIN8	TSY1CPUIC32	Y01 output
PIN9	TSY2CPUIC32	Y02 output
PIN10	TSY3CPUIC32	Y03 output

The first column contains the pin numbers used in this study, and the second column shows the pin names which are used for the simulation. Pin 1, Pin 4 and Pin 6 are all used for the carryouts of the arithmetic functional result from the ALU. The differences are that Pin 1 is a carryout line of the usual full adder, and Pin 4 and Pin 6 are the carry propagate and generate outputs of the internal ALU (used in the carry lookahead). Pin 2 is the most significant ALU output bit. Pin 3 indicates whether the result of an ALU operation is zero or not. The overflow signal is shown at Pin 5. Pins 7,8,9 and 10 are the four outputs of the ALU or the data of the register stack, as determined by the destination decoder[18].

6.1. Probability of Pin Errors

As shown in Chapter 4, on the average, 12.7% of the device-level faults are detected at the output pins (while 5.1% are detected within the unit and 20.9% are outside the unit). Table 8 shows the impact of faults in the specified functional units on the output pins. The first column shows the units in which the faults were injected. The remaining columns identify the specific output pins in which the faults may be detected. Each entry shows the probability with which an injected fault affects the specified pin.

TABLE 8: Probability of Pin Errors

faulted Section	PIN1	PIN2	PIN3	PIN4	PIN5	PIN6	PIN7	PIN8	PIN9	PIN10
RAM	0.650	0.875	0.875	0.700	0.700	0.650	0.875	0.875	0.600	0.600
Q SHFT	0.400	0.500	0.700	0.500	0.400	0.700	0.500	0.500	0.400	0.500
S CONT	0	0	0	0	0	0	0	0	0	0
ALU CON	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
OUT SEL	0.950	0.950	0.950	0.950	0.950	0.950	0.950	0.950	0.950	1.0
RAM SHFT	0.800	0.500	1.0	0.800	0.800	0.800	0.750	0.750	0.600	0.500
RAM CONT	0.411	0.436	0.444	0.436	0.410	0.205	0.410	0.410	0.423	0.444
DES CONT	0	0	0	0	0	0	0	0	0	0
ALU	0.700	0.800	0.700	0.750	0.700	0.850	0.750	0.750	0.650	0.600
MUX	0.556	0.611	0.778	0.870	0.556	0.444	0.611	0.611	0.722	0.556

Recall that in the previous chapter the faults in most units have a high probability of affecting the Output unit (which includes the output pins). Table 8 not only confirms this observation but also shows that the impact is rather uniform across all output pins. Different units, however, affect the pins to varying degrees. This is due to the fact that the characteristics of functional units differ due to the combined effect resulting from their different functional operations, locations and structural complexities. For example while 100% of the faults in the ALU Control affect all pins, faults in the Source and Destination Control units do not affect the output pins at all. As expected, most of the faults in the Output Select readily affect all pins. The results in this table also show that each unit has a distinct probability of affecting the output pins. This result is significant for integrated system testing because it suggests that by injecting pin errors with the measured distinct probabilities, we can emulate within-chip faults.

6.2. Mean Time Between Errors (MTBE) and Mean Error Durations (MED)

Recall that in Chapter 4 we calculated the MTBE and the MED of pin errors. Clearly, the longer the MTBE, the less the impact of the error on the external environment. It also means, however, that it is harder to detect the errors. Similarly, the shorter the error duration, the less the impact on the external to the chip. Tables 9 and 10 show the MTBE and the MED for the different functional units into which faults are injected.

TABLE 9: Mean Time Between Errors

faulted Section	PIN1	PIN2	PIN3	PIN4	PIN5	PIN6	PIN7	PIN8	PIN9	PIN10
RAM	.	631.9	586.9	502.9	.	.	136.0	111.3	135.8	491.8
Q SHFT	.	574.0	574.0	.	.	.	478.3	.	.	717.5
S CONT	.	.	.	.	.	.	.	.	.	.
ALU CON	281.3	137.1	156.7	205.5	170.4	305.8	159.6	146.3	146.3	143.6
OUTPUT	211.4	121.7	113.5	186.9	144.2	298.9	102.2	98.1	108.0	103.7
OUT SEL	217.9	133.9	128.4	192.4	150.7	302.1	116.6	102.9	111.7	117.2
RAM SHFT	.	816.0	816.0	.	.	.	.	.	816.0	816.0
RAM CONT	304.8	205.5	163.8	278.6	156.4	317.8	151.4	134.8	161.6	154.9
DES CONT	.	.	.	.	.	.	.	.	.	.
ALU	356.2	166.9	177.0	345.3	203.8	367.2	122.9	141.5	128.5	120.6
MUX	347.4	180.6	201.4	329.9	206.4	505.8	164.2	158.8	144.1	130.5
in chip	303.8	189.1	189.6	296.7	185.6	369.4	147.6	141.1	144.4	148.1

In Table 9, the first column shows the units in which the fault injections were initiated. The figures in the next ten columns are the MTBE on each output pin. The dots in the table indicate that computing values on that unit are not possible due to insufficient data. For instance, if there is only one error detected during the simulation, it is impossible to obtain the time interval to the next error.

Referring to the table, faults in the ALU and the MUX resulted in very similar distributions of the MTBE across all the output pins as discussed in the previous chapter. The shortest MTBEs are shown when faults are in output units, i.e., there is high external detectability of these fault. Faults in the Q-Shift and the Ram Shift units have long MTBE to the pins. This is due to low utilization of these units resulting in fewer faults propagating to the pins. Since Pins 7, 8, 9 and 10 are fed out from the same unit and operated by the same functions, the values of the MTBE of these pins are almost identical. Pins for carryout (Pin 1, Pin 4 and Pin 6) had relatively long MTBEs. This is due to the fact that the carryout operations are used only for the event of generating carries during or after arithmetic operation in the ALU. These pins, therefore, are not utilized as often as the other pins.

Table 10 shows the Mean Error Durations. The units of fault injection are shown in the first column. The rest of the columns show the MED for the individual pins. The dots in the entries indicate

that the data are not sufficient to perform proper computation for the corresponding unit.

TABLE 10: Mean Error Durations

faulted Section	PIN1	PIN2	PIN3	PIN4	PIN5	PIN6	PIN7	PIN8	PIN9	PIN10
RAM	.	84.3.	73.7	86.2	.	.	38.3	71.1	20.0	63.2
Q SHFT	.	1.8	1.6	.	.	.	1.7	.	.	2.0
S CONT	.	.	.	.	.	.	.	.	.	.
ALU CON	36.7	47.5	43.2	45.6	69.8	18.6	56.5	43.5	48.8	49.4
OUTPUT	58.1	88.7	78.3	91.6	71.8	23.2	89.0	81.6	83.3	89.1
OUT SEL	56.2	85.7	74.2	88.5	70.0	21.7	87.9	79.8	81.5	88.7
RAM SHFT	.	68.3	33.5	.	.	.	.	.	68.3	68.3
RAM CONT	46.5	23.7	46.5	46.6	30.4	16.7	58.4	40.3	49.2	45.4
DES CONT	.	.	.	.	.	.	.	.	.	.
ALU	52.8	29.8	57.3	44.1	33.8	19.3	74.1	61.6	68.2	70.9
MUX	49.6	15.2	82.1	42.9	35.2	13.3	85.1	77.1	81.3	86.5
in chip	47.8	27.2	58.2	46.4	41.6	17.5	67.7	66.9	61.7	60.6

By examining values of the MTBE and the MED together, the impact of the output pins is more clearly shown than by either approach separately. From Table 10, faults in the Q-Shift unit have the shortest duration. Recall that they also have a long MTBE. Thus, these faults are likely to be hard to detect external to the chip. As expected, the faults in the output units have a long duration. Thus, both from the error frequency and error duration perspectives these faults are easily detectable. Faults in the Q-Shift and the Ram Shift have behaved very similarly during their propagation in the system due to their functional similarity. The MEDs of these units, however, exhibited large differences. One possible explanation for this phenomenon is that the time duration of holding a error on the pin is determined not only by the functional operation of the unit, but also determined by multiple effects of the data propagated from several different signal paths. Because their effects by functional operation or the geographical location are almost identical, Pins 7, 8, 9, and 10 have similar MEDs. These pins tend to have longer errors than any of the other pins. The shortest MED is shown on Pin 6, showing that this pin clears errors very quickly.

Table 9 and Table 10 also show that Pin 6 (carry generate) is least affected by the faults regardless of the injected unit (the longest MTBE and the shortest MED). The impact of faults on the data pins (Pins 7, 8, 9 and 10) is considerable as shown by the short MTBE and long MED. Faults in the output units severely affect the output pins. Thus, data faults are expected to be easily detected outside while carry faults appear to be more insidious.

6.3. Near-Coincident Errors

It is well known that fault tolerant systems are highly vulnerable to near-coincident faults [19,20]. In this section we investigate the likelihood of near coincident errors at the output pins resulting from an injected device fault. Generally, an injected fault may sensitize many other data paths simultaneously during the propagation. Additionally, the output pins may have multiple errors at the same time or in a short span of time due to a propagated fault. Near-coincident errors are defined as the errors which are observed within a short time span. In order to properly measure the number of near-coincident errors, an appropriate size of time window is chosen; then, the window is moved over the total simulation time. Each time, the number of errors discovered within each time window is observed and recorded as near-coincident errors. In this study, the number and probability of near-coincident errors was averaged over the entire simulation measurement according to given window size.

Figure 4 shows the effect of varying the time-window size on the mean number of near-coincident errors. As expected the mean number of near-coincident errors increases steadily as a function of the window size. The rate of increase in the mean number of near-coincident errors is seen to be lower both for large (more than 2700 time steps) and medium (500-1300 time steps) window sizes. The reason for this finding is that for the smaller window size, fewer errors are detected until the window size is large enough to hold these errors actively. For the larger window sizes, since a large number of faults have already been observed, the results are not greatly changed by further increasing the size of the window.

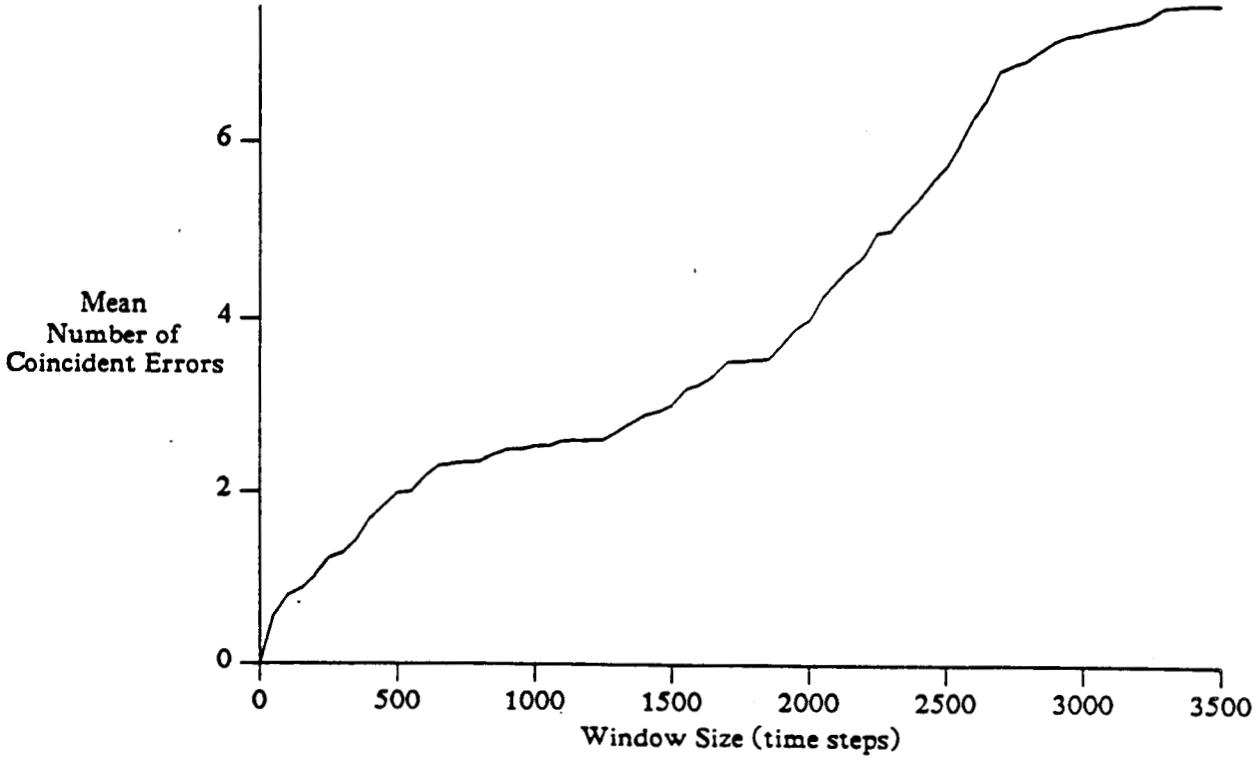

Figure 4: Mean Number of Coincident Errors

Figure 5 shows the probability of near-coincident errors. The probability is obtained from the ratio of the total number of errors occurring in that time window to the total number of errors injected. The probability increased rapidly up to a window size of 1700 time steps as a function of the window size. The rate of increase, however, is much less for a window size greater than 1700 time steps.

From a practical viewpoint, however, it can be seen that given a fault, there is a relatively high likelihood of encountering two output pin errors with less than seven clock cycles (500 time steps). This is explicitly shown in Fig. 5 where the probability of near-coincident errors is plotted as a function of window size. The figure also shows that, given a fault, there is approximately a 15 percent chance of a multiple error with 7 clock cycles.

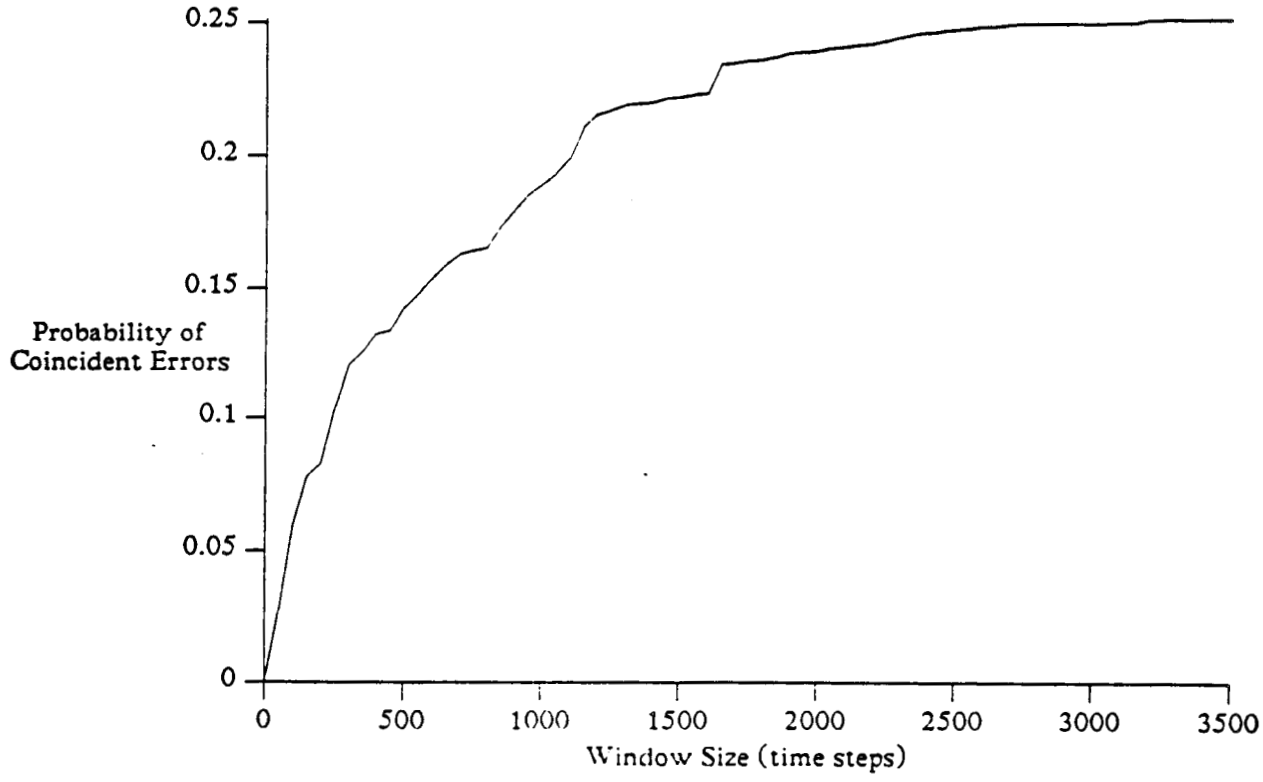

Figure 5: Probability of Near-Coincident Errors

CHAPTER 7

INSTRUCTION/MICROINSTRUCTION ANALYSIS

Fault propagation in the chip is highly dependent upon the assembly-level instruction and microinstruction under execution. That is to say, that fault propagation is influenced by not only the amount of nonfaulted gate activity but also by the interaction between gate activity and type of instruction. In a previous work [11], error propagation influenced by similar and dissimilar instructions was studied. The work also examined the influence of the type of microinstruction executed on error propagation. In this study, analysis for the device-level simulation was performed based on the instruction/microinstruction executed.

Figure 6 shows the total gate activity as a function of the time (in clock cycles) for a device-level fault. The vertical axis is the sum of the gate activities over all the injected faults. The horizontal axis indicates the time in clock cycles. The instruction executed is also labeled across the horizontal axis. In the graph, the peak gate activity occurs during an instruction prefetch or some other memory access because of the concurrent activity in the processor. The Store (sto), the Load (ldm) and the Subtract instruction (pre-sub) show similarities in their gate activities. Low gate activities are seen when the jump instruction (ju-ind) is executed. The instruction for store multiple registers from memory (stm) shows the highest gate activity because of the high frequency of register transfers.

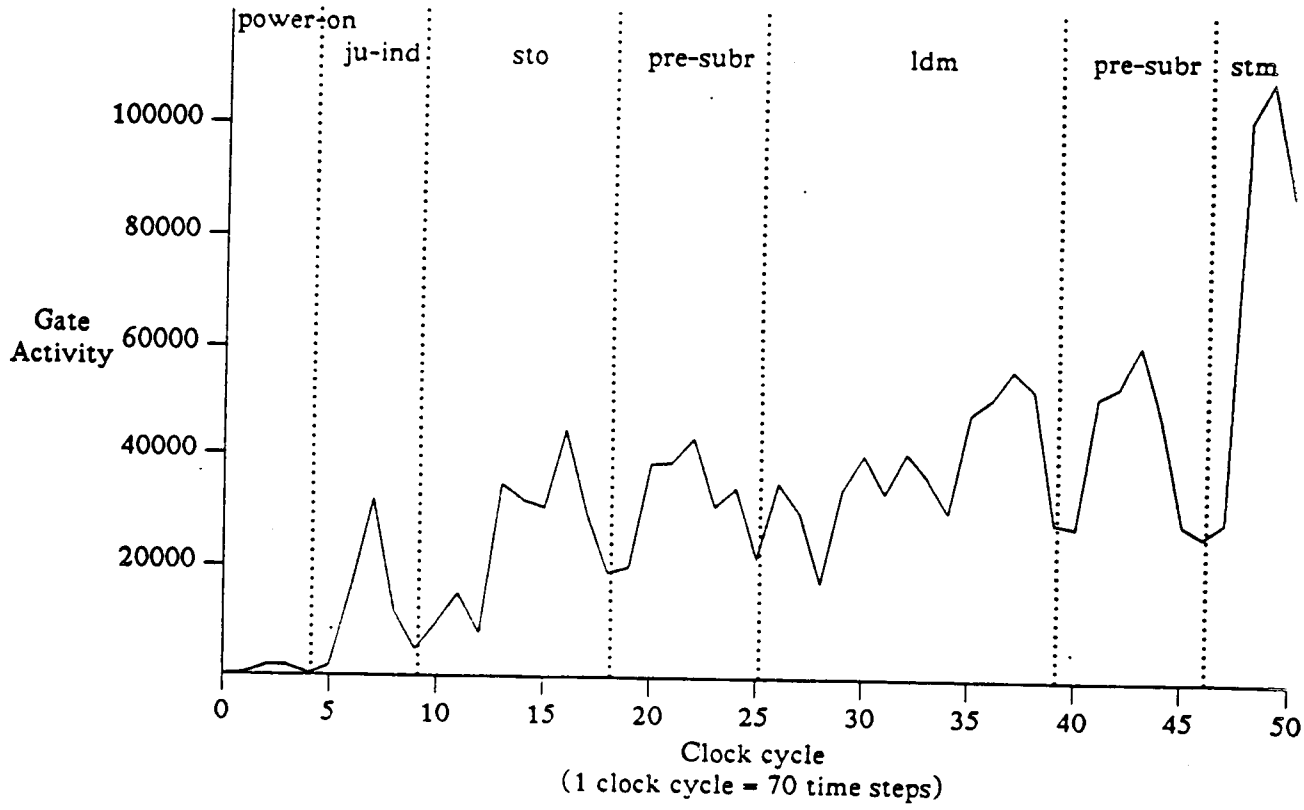

Figure 6: Gate Activity for Device Level Fault

Table 11 shows the error probabilities by different instruction types. These results are the averages over the entire fault set. The highest probability of detection is shown in jump instruction and the lowest is in store instruction. Referring to Fig 6, it appears that there is little relationship between the amount of gate activity and the probability of error occurrence. For example, while the number of gate activity is the highest during the stm instruction execution and the lowest during the jump instruction, the stm instruction has the lowest error probability and the jump instruction has the highest. The reason for the differences in the measured error probabilities between the different instruction types can be explained by investigating the relationship between the error activity and the microinstructions.

TABLE 11: Probability of Fault Detection for Instruction Executed

INSTRUCTION	ERROR PROB.
ju-ind	0.26
sto	0.14
pro-subr	0.21
ldm	0.16
stm	0.19

Toward this end, these microinstructions were classified according to the type of activity contained in each microinstruction. The classifications include the register transfer, the memory access, logic computation, arithmetic computation and conditional/unconditional branch. Due to parallelism, one microinstruction may involve more than one classified function. The fault activity determined at the microinstruction level can be used to explain the fault propagation at the instruction level, because the microinstruction is the building block of the assembly instruction. Table 12 shows the probabilities of fault detection in the device-level simulation according to the microinstruction executed. The function of each bit of microcode is indicated as follows:

if bit4 = 1, then a register transfer,
if bit3 = 1, then a memory access,
if bit2 = 1, then a logical computation,
if bit1 = 1, then a arithmetic computation,
if bit0 = 1, then a conditional branch.

TABLE 12: Probability of Fault Detection for Microinstruction

MICRO-INSTRUCTION	PROB. OF DET. in DEVICE LEVEL
00000	0.11
00001	0.04
01000	0.18
01001	0.21
10000	0.18
10001	0.23
11000	0.17
11001	0.20

As shown in the table the probabilities of detecting a fault when a conditional branch operation (bit0=1) is involved, is generally increased. As expected, the microinstruction for branch operation that is used for jump instruction has high probability of detection, while the microinstruction for store and load instruction, which include the register transfer operations, has low probability of detection.

CHAPTER 8

CONCLUSIONS

This thesis has described a systematic experimental study of fault propagation in the Bendix BDX-930, a digital avionic miniprocessor. Error activity was investigated by comparing the gold (unfaulted) simulation run with each faulted simulation run. The simulations were performed only on the bit-slice processor, AMD 2901. In the simulations, fault propagation data were collected at device and gate-levels, as well as at the output pins. The results provided by these data allowed us to not only analyze the dependency of error propagation on the location of the fault and by the type of instruction and microinstructions executed, but also to compare the accuracy of the stuck-at fault model with the more realistic physical failure model for permanent faults.

Results show that assuming a stuck-at model can overestimate the probability of fault propagation to the output pins by over one hundred percent. The Mean Time Between Errors for the stuck-at faults were longer, and the Mean Error Durations shorter, than those for the device faults. Thus, assuming a stuck-at model for physical failures may overestimate the fault propagation characteristics within the chip and underestimate the impact on the external to the chip.

Measurement of error activity at the output pins showed that faults in different functional units affect the output pins to varying degrees and that each unit has a distinct probability of affecting the output pins. This result suggests that by injecting pin errors with the measured distinct probabilities we can easily emulate with-in chip faults for integrated system testing.

The Mean Time Between Errors and the Mean Error Duration at the output pins were also evaluated. Among the ten output pins, the carry generate pin had the longest MTBE and the shortest MED. The Data pins (Pins 7, 8, 9 and 10) had relatively short time between errors and relatively long error durations.

Thus, the current work has shown that a wide variety of fault propagation behavior can result from device failures. Further research is in progress to use the results of such analyses in identifying the "weak" links in a system, from a fault tolerance viewpoint, in the design stage itself, so as to make design improvements in a cost-effective manner.

APPENDIX A

COMPARISONS BETWEEN STUCK-AT AND DEVICE

A.1. Comparison of Percentage of Fault Detected

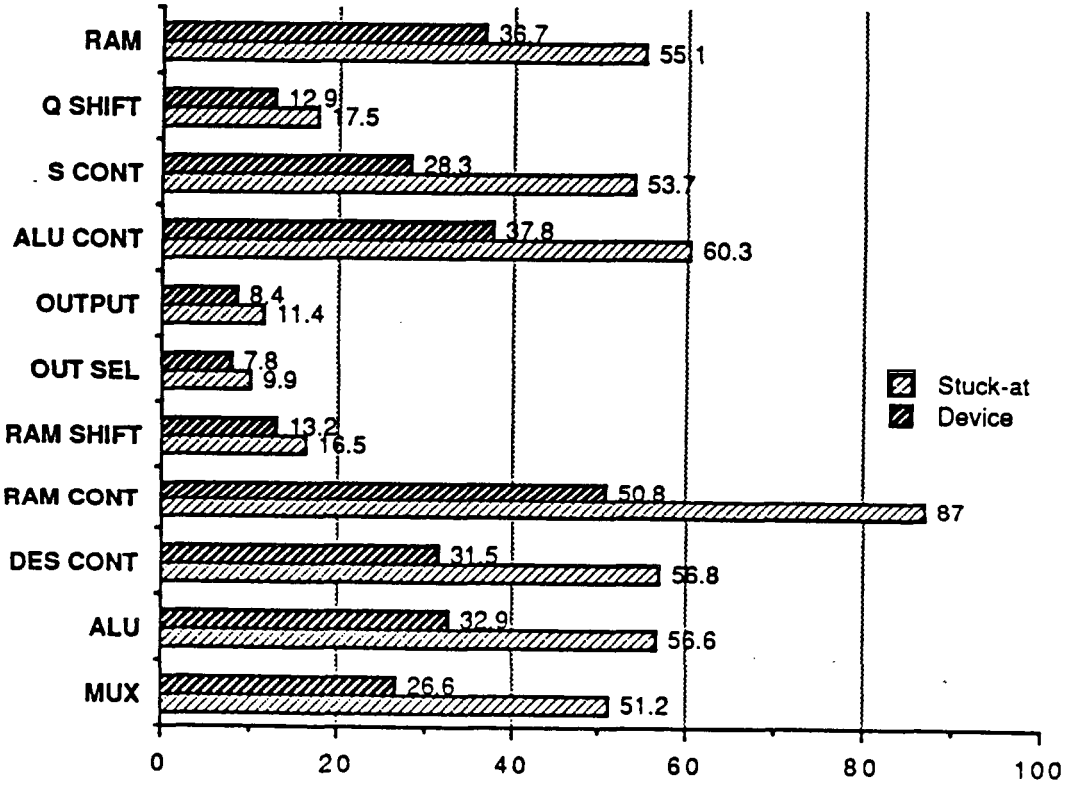

Figure A.1: Comparison of Percentages of Faults Detected

A.2. Comparison of Percentage of Faults Detected at Output Pins

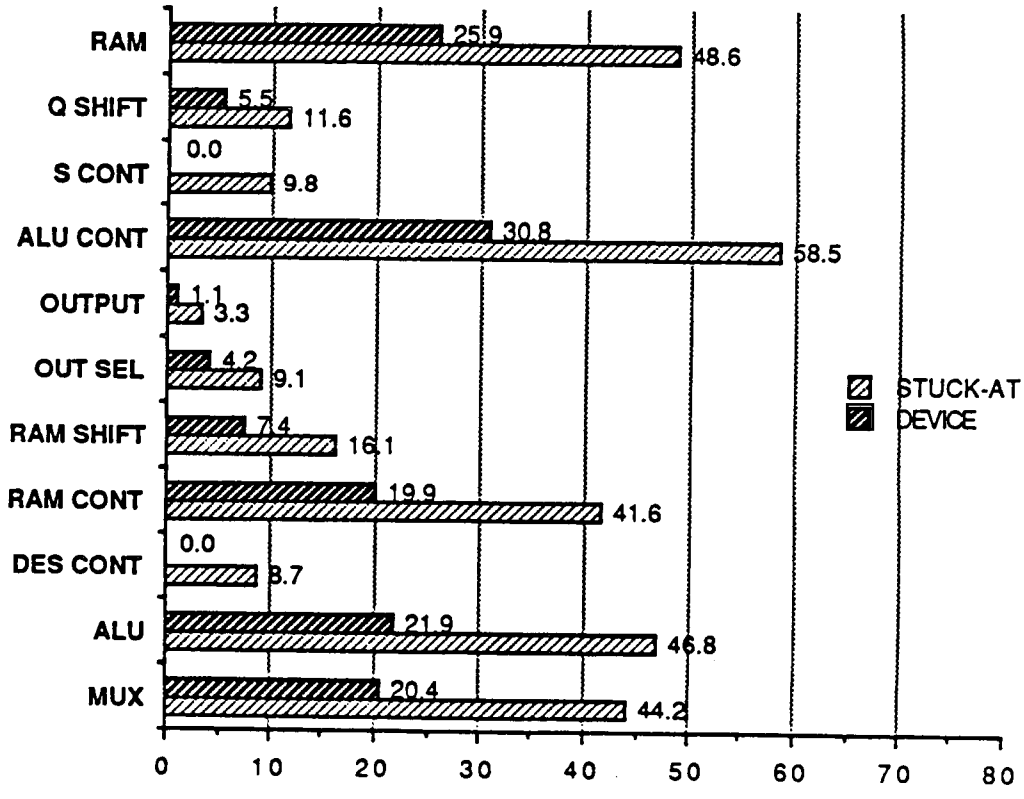

Figure A.2: Comparison of Percentages of Faults Detected at Output Pins

A.3. Comparison of Propagation Factor at Output Pins

TABLE A.1: The Propagation Factors to Output Pins

faulted Section	Pro.Factor for Stuck-at	Pro.Factor for Device
RAM	4.8	5.9
Q SHFT	6.5	7.2
S CONT	1.8	2.7
ALU CON	9.3	8.6
OUTPUT	9.1	10.3
OUT SEL	9.9	10.5
RAM SHFT	4.1	8.2
RAM CONT	2.2	4.9
DES CONT	1.7	2.1
ALU	8.0	9.1
MUX	7.2	8.3

APPENDIX B

INSTRUCTION/MICROINSTRUCTION COMPARISONS

B.1. Comparison of Gate Activity

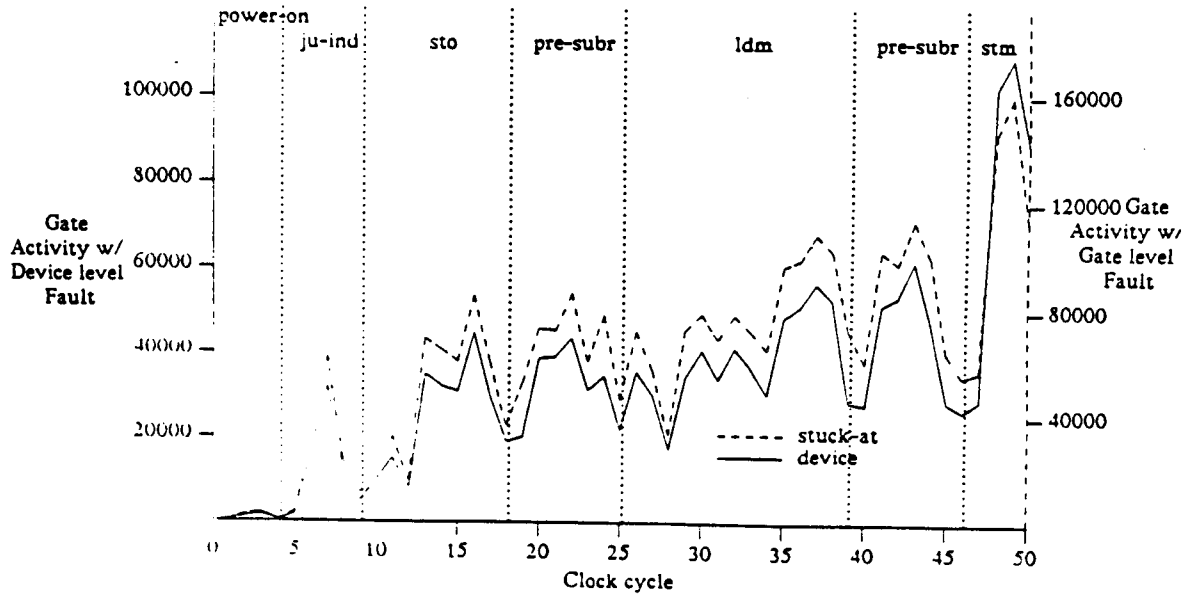

Figure B.1: Comparison between Device and Gate Level Faults

B.2. Comparison of Error Probability based on Instruction/microinstruction

TABLE B.1: Comparison of Error Probability for Instructions

INSTRUCTION	PROB. OF DET. in GATE LEVEL	PROB. OF DET. in DEVICE LEVEL
ju-ind	0.33	0.26
sto	0.19	0.14
pro-subr	0.25	0.21
ldm	0.22	0.16
stm	0.24	0.19

TABLE B.2: Comparison of Error Probability for Microinstructions

INSTRUCTION	PROB. OF DET. in GATE LEVEL	PROB. OF DET. in DEVICE LEVEL
power-on	0.15	0.11
ju-ind	0.09	0.04
sto	0.23	0.18
pro-subr	0.29	0.21
ldm	0.25	0.18
pre-subr	0.32	0.23
stm	0.22	0.17

REFERENCES

[1] A. Vladimirescu, A.R. Newton, and D.O. Pederson, *SPICE Version 2G.1 User's Guide*. Berkeley, CA: EECS Dept., UC Berkeley, 1980.

[2] R.K. Iyer and D.J. Rossetti, *A Statistical Load Dependency of CPU Errors at SLAC*. Santa Monica, California: Digest FTCS-12, June 1982.

[3] R.K. Iyer, D.J. Rossetti, and M.C. Hsueh, "Computer System Reliability and System Activity: Measurement and Modeling," *ACM Trans. on Comp. Sys.*, August 1986.

[4] X. Castillo and D.P. Siewiorek, *Workload, Performance and Reliability of Digital Computing Systems*. Portland, Maine: Digest FTCS-11, June 1981.

[5] K.G. Shin, "Measurements of Fault Latency: Methodology and Experimental Results," Tech. Report CRL-TR-45-84, Computing Research Lab, Univ. of Mich., 1984.

[6] J.G. McGough and F.L. Swern, "Measurement of Fault Latency in a Digital Avionic Mini Proc.(Parts I & II)," NASA Contractor 3651, NASA Langley Research Center, Oct. 1981, Jan. 1983.

[7] J.G. McGough, F.L. Swern, and S. Bavuso, "New results in fault latency modeling," *Eascon*, vol. b, 1983.

[8] J.H. Lala, "Fault Detection, Isolation, and Reconfiguration in FTMP: Methods and Experimental Results," *Proc. 5th DASC*, 1983.

[9] R. Chillarege and R.K. Iyer, "Measurement-Based Analysis of Error Latency," *IEEE Trans. Comp.*, vol. C-36 , May 1987.

[10] R. Chillarege, "Fault and Error Latency Under Real Workload-An Experimental Study," Ph.D. dissertation, Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, 1986.

[11] D.L. Lomelino, "Error Propagation in a Digital Avionic Mini Processor," M.S. Thesis, ECE Dept. Univ. of Illinois at Urbana-Champaign, 1986.

[12] J. Wensley et al., "SIFT: Design and Analysis of a Fault Tolerant Computer for Aircraft Control," *Proc. IEEE*, vol. 66, pp. 1240-1254, October 1978.

[13] P. Forman and K. Moses, "SIFT: Multiprocessor Architecture for Software Implemented Fault Tolerance Flight Control and Avionics Computers," *Third Digital Avionics Systems Conference*, pp. 325-329, November 1979.

[14] D. Migneault, "The Diagnostic Emulation Technique in the Airlab," Internal Report, NASA Langley Research Center, 1985.

[15] R.L. Wadsack, "Fault modeling and logic simulation of CMOS and NMOS integrated circuits," *The Bell Sys. Tech. J.*, vol. 57, May 1978.

[16] J.P. Shen, W. Maly, and F.J. Ferguson, "Inductive Fault Analysis of nMOS and CMOS Integrated Circuits," Research Report CMUCAD-85-51, ECE Dept. CMU, Pittsburgh, PA, August 1985.

[17] Digital Equipment Corporation, *VAX/VMS Primer*. Maynard, MA: Digital Equipment Corporation, May 1982.

[18] Advanced Micro Devices, *Bipolar Microprocessor Logic and Interface Book*. Sunnyvale, CA: Advanced Micro Devices, 1981.

[19] J. McGough, "Effects of near-coincident faults in multiprocessor systems," *Proc. IEEE/AIAA Fifth Digital Avionics Systems Conf.*, pp. 16.6.1-16.6.7, 1983.

[20] S.G. Mitra, "Near-Coincident Fault Discovery in a Shared Memory Multiprocessor," M.S. Thesis, University of Illinois at Urbana-Champaign, 1988.