PARALLEL HARDWARE ACCELERATED SWITCH LEVEL FAULT SIMULATION

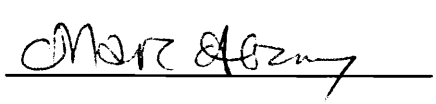
By

Christopher A. Ryan

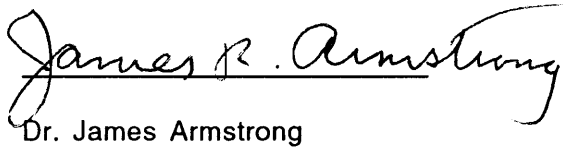
A dissertation submitted to the Graduate Faculty of the Virginia Polytechnic Institute & State University in partial fulfillment of the requirements for the degree of Doctorate of Philosophy in Electrical Engineering.

Approved by:

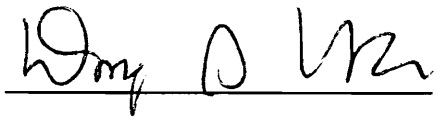
Dr. Joseph Tront (Chairman)



Dr. Marc Abrams



Dr. James Armstrong



Dr. Dong Ha

Dr. Scott Midkiff

June, 1993

Blacksburg, VA

PARALLEL HARDWARE ACCELERATED SWITCH LEVEL FAULT SIMULATION

By

Christopher A. Ryan
Dr. Joseph G. Tront (Chairman)
Electrical Engineering

(Abstract)

Switch level faults, as opposed to traditional gate level faults, can more accurately model physical faults found in an integrated circuit. However, existing fault simulation techniques have a worst-case computational complexity of $O(n^2)$, where n is the number of devices in the circuit. This paper presents a novel switch level extension to parallel fault simulation and the switch level circuit partitioning needed for parallel processing. The parallel switch level fault simulation technique uses 9-valued logic, N and P-type switch state tables, and a minimum operation in order to simulate all faults in parallel for one switch. The circuit partitioning method uses reverse level ordering, grouping, and subgrouping in order to partition transistors for parallel processing. This paper also presents an algorithm and complexity measure for parallel fault simulation as extended to the switch level. For the algorithm, the switch level fault simulation complexity is reduced to $O(L^2)$, where L is the number of levels of switches encountered when traversing from the output to the input. The complexity of the proposed algorithm is much less than that for traditional fault simulation techniques.

Acknowledgments

The author would like to give thanks to his advisor, Dr. Joseph Tront, for his guidance during this endeavor. Special thanks are given to Dr. Marc Abrams, Dr. James Armstrong, Dr. Dong Ha, and Dr. Scott Midkiff for serving on the committee to review this Dissertation research.

The author would like to thank Dr. Krzysztof Kozminski of the Microelectronics Center of North Carolina for his assistance in providing the switch level descriptions of the ISCAS85 benchmark circuits. The author would also like to thank Dr. Dong Ha for his assistance in providing the gate level test vector generation tool ATALANTA, Dr. Scott Midkiff for providing the computer engineering research seminar as a forum for ongoing dissertation research updates, and Dr. James Armstrong for providing additional computer resources.

And lastly, a tremendous word of thanks goes to Virginia Polytechnic Institute & State University and North Carolina Agricultural & Technical State University both of which supported the author financially during this endeavor.

Table of Contents

List of Figures

List of Tables

Chapter 1.

Introduction

1.1 Integrated Circuit Testing

Digital integrated circuit (IC) design and test generally includes both functional simulation and fault simulation. Functional simulation is used to verify the functional operation of the IC. This is achieved by first modeling the device at the behavioral, register transfer, gate, switch, or circuit levels. Inputs are applied to the model and then the simulated output responses are compared with the expected or predicted functional operation of the IC. If the two outputs are different, the design is modified until it achieves the desired functional operation. After the IC is fabricated, the input functional test set can be applied and the resulting outputs compared to the simulated outputs. If these results agree, then the IC is considered functionally correct for the test set. To ensure complete correct functional operation, the functional test set must be exhaustive. However, with the growing complexity of ICs, an exhaustive functional test set can become very large. Therefore, functional simulation is currently used more in the design phase than in the test phase for the IC.

Physical verification using fault simulation is used in the test phase. The fault simulation process is similar to the functional simulation process with the addition of fault models that represent certain physical faults. Here, a faulty circuit comprised of the good circuit modified to include one fault is simulated. The output responses from the faulty circuit are then compared to those of the good circuit. If the outputs disagree, then a test set which detects the presence of that fault has been determined. Furthermore, if an input test set can be found that detects the presence of all such single faults, then this test set can be used to verify the absence of such physical faults in the IC. This test set is generally smaller than an exhaustive functional test set.

At the gate or logic level, three typically accepted fault simulation methods are: parallel fault simulation, deductive fault simulation, and concurrent fault simulation [Fuji85]. New fault simulation methods include differential fault simulation [Wu90], and parallel pattern single fault propagation fault simulation [Waic85]..

By using only one bit of the computer word to simulate the good circuit and one bit of the word for each fault, *parallel fault simulation* simulates multiple faults for one test vector. Thus, if the host computer word size is 16 bits, then approximately 15 faults can be simulated in parallel. *Deductive fault simulation* simulates only the good circuit and then the faulty circuits to produce lists of faults at each line from the primary inputs to the primary outputs. Deductive techniques are then applied to determine which faults have been detected with the test set. In *concurrent fault simulation*, all faults are simulated in one pass for each input test vector. The good circuit is simulated concurrently with the faulty circuit. However, since each fault only affects a small part of the circuit, only that affected part of the circuit need be

simulated. *Differential fault simulation* simulates the good circuit and the faulty one for each test vector separately. This reduces the memory requirements over concurrent fault simulation. *Parallel pattern single fault propagation fault simulation* uses the implication relationship [Light82], as in the D-algorithm [Pradhan90], to propagate faults to the output, thereby reducing simulation time.

One problem with performing fault simulation at the gate level is that it has been shown that physical faults within a metal oxide semiconductor (MOS) IC cannot be modeled by only gate level stuck-at faults [Shen85]. Additional fault models needed are: line stuck-at faults internal to the gate, transistor stuck-at faults, floating line faults, and bridging faults [Shen85]. Switch level simulation has greater accuracy and can model physical faults better. By reducing the fault simulation level from the gate to the switch level, the first three fault models above can be implemented [Hayes82]. By injecting a connection between two nodes in question, the fourth model for bridging faults can also be implemented [Rajs87] . The advantage of fault simulation performed at the switch level is increased accuracy [Shen85], while the disadvantages are longer simulation times and increased memory requirements.

The simulators SLS [Barzi86], FAUST [Shih86], CSASIM [Kawai84], Bose [Bose82], FMOSSIM [Bryant8], COSMOS [Bryant87b] and [Lee91] are all switch level fault simulators implemented in software to be run on general purpose computers. The number of switch level fault simulators is limited possibly due to the computational complexity of the problem and also because of the difficulty in modeling the switch level as compared to the logic level. One problem common to these simulators is that of long simulation times. This problem occurs because of the upper bound on computational complexity of the traditional fault simulation techniques used. If n is the

number of gates, the upper bounds on simulation time are $O(n^3)$, $O(n^2)$, and $O(n^2)$, for parallel, deductive, and concurrent fault simulation, respectively [Goel80]. When switch level simulations are considered, the problem is compounded since each CMOS gate is typically comprised of from two to ten transistors. This makes switch level fault simulation of even an average sized integrated circuit unfeasible.

Fault simulator implementations other than those built on general purpose computers, are implemented on general hardware accelerators, general massively parallel computers, distributed general purpose computers, and on special purpose hardware accelerators. MARS [Agraw87.1], [Agraw87.2] and FACOM [Ishi90] are examples of general purpose hardware accelerators, while the Connection Machine[Hills85] and IPSC2 [Kush89] are examples of parallel computers. All of these have been used to implement gate level functional simulators and gate level fault simulators e.g., refer to [Agraw89], [Agraw90]. MOZART [Gai88a], [Gai88b] is a hierarchical fault simulator implemented to run on general purpose computers. CHIEFS [Duba88] is a hierarchical fault simulator implemented to run on distributed general purpose computers. However, CHIEFS's [Duba88] lowest level of fault simulation is the gate level. Special purpose hardware is used by HALII [Taka89] and HSS [Smith87]. HALII [Taka89] performs gate level fault simulation, while HSS [Smith87] performs simulation at the switch level but does not perform fault simulation. These and other hardware accelerators are summarized in [Blank84]. COSMOS, a switch level simulator which is implemented both in software [Bryant87] and on the Connection Machine [Krav91], shows that the speed up advantage in using the massively parallel computer over general purpose computers for switch level simulation is not cost effective. This is because the communications to execution time ratio is high because of the very fine

grain nature of the switch level computation problem. Comparing the overall performance results of these different switch level fault simulators is not feasible since different circuits and different test vector sets were reported. However, the software switch level fault simulators could not be used to grade even an average sized IC. Conversely, the hierarchical fault simulators do perform faster but they only implement partial fault lists.

1.2 Research Overview

The work presented in this dissertation addresses the complexity and large simulation time for switch level fault simulation by examining the feasibility of two-dimensional parallel fault simulation using a Parallel Hardware Accelerated Fault Simulator (PHAFS). As shown in Figures 1.1 and 1.2, PHAFS is comprised of two components: a software component and a hardware component. The PHAFS software component partitions the circuit by performing a reverse level ordering of it. Reverse level ordering is of particular interest since it extends critical path tracing [Abram84] from the single processor environment to the parallel processing environment. The PHAFS hardware component uses a parallel SIMD architecture based on an Application Specific Integrated Circuit (ASIC) in order to perform two-dimensional parallel switch level fault simulation. The general outline of the architecture for the PHAFS ASIC is shown in Figure 1.3. The architecture includes processing elements (PE's) and interconnect modules (IM's). One PE , consisting of less than 800 transistors, performs parallel fault simulation for one N - type or one P - type switch using the switch level extension to parallel fault simulation as detailed in Chapter 3. Also detailed in Chapter 3 is the IM which consist of less than 200 transistors and

performs the connector operation or fan - in operation. Two-dimensional parallel fault simulation is performed by programming a partition of switches on the PHAFS board and then simulating all PE's in parallel.

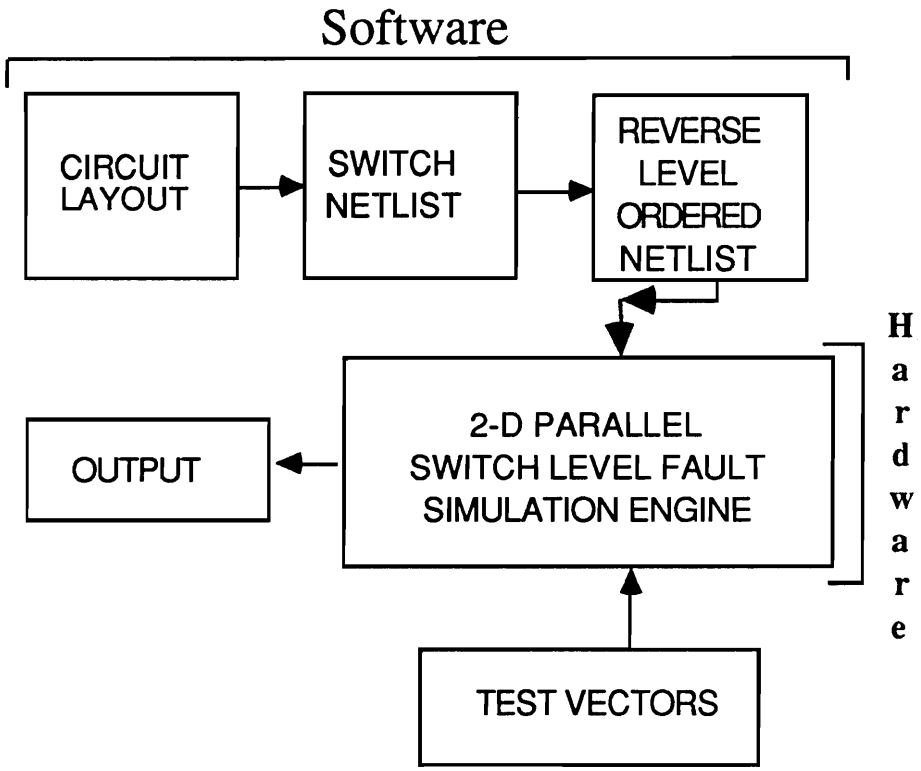


FIGURE 1.1 PHAFS switch level fault simulation system.

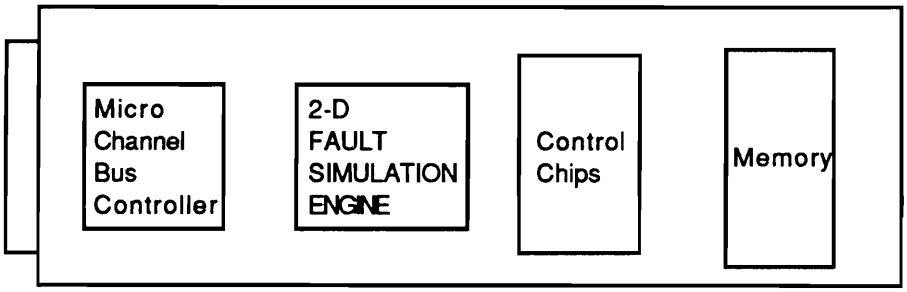


FIGURE 1.2 Fault simulation hardware accelerator board.

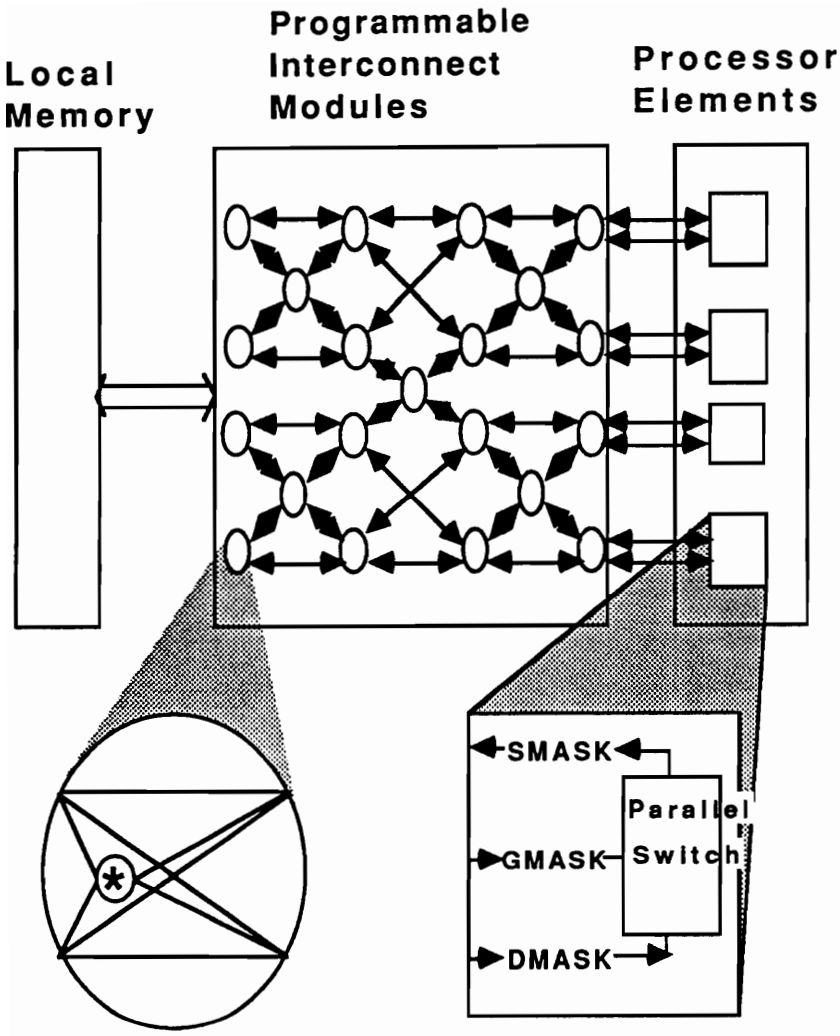


Figure 1.3 PHAFS ASIC architecture.

1.3 Organization of Dissertation

This dissertation describes in detail the theory that forms the basis of the PHAFS system. Chapter 2 gives an overview of the switch level models and the mathematics used for fault simulation. Using nine-valued logic, switch state tables, and a minimum operation, a novel switch level extension to parallel fault simulation is given in Chapter 3. Chapter 4 presents the circuit partitioning required for parallel processing. The circuit partitioning operation includes reverse level ordering, grouping, sub-grouping, and parallel processing partitioning. Chapter 5 describes the two-dimensional parallel fault simulation algorithm. The algorithm complexity, which is on the order of L^2, where L is the number of reverse levels of transistors, is also shown here. In order to determine the number of processor elements required for parallel switch level fault simulation, Chapter 6 studies switch level implementations of the ISCAS85 combinational benchmark circuits for reverse level, group and subgroup sizes [Brglez85]. Chapter 6 also looks at hardware cost and speed issues by examining complexity increase and processor element load versus parallel processing partition size. Chapter 7 describes the compiled code VHDL fault simulator used to verify the algorithm and theoretical complexity presented in Chapter 5. This includes modeling of the Processing Elements (PE), Interconnect Modules (IM) and controller. Experimental results on complexity and speed up for the parallel fault simulation purposed is also given in Chapter 7. Chapter 8 gives some conclusions generated by this research and suggests future work.

Chapter 2

SWITCH LEVEL MODELS AND MATH

2.1 Introduction

This chapter describes the switch level models and mathematics required in order to perform fault simulation. Section 2.2 starts with a survey of the models and math used by recent and widely accepted fault simulation techniques. Section 2.3 gives in detail the models and math used for this work. Finally, Section 2.4 discuses switch level fault models.

2.2 Device Models

Much like gate level fault simulation, most MOS fault simulators use the concept of discrete events and models. The models used range from representing the transistor as a current source, to representing it as a three-valued switch, to mapping the discrete logic effect of the transistor fault to the inputs and outputs of the logic gate. Figures 2.1-2.5 show examples of MOS device models. At the lowest level, FAUST [Shih86] uses transistors, resistors, and capacitors to model a circuit. The transistors, as

shown in Figure 2.1, are modeled as current sources (Ids) controlled by the gate to source voltage (Vgs) and the drain to source voltage (Vds). To speed simulation over that of the circuit level, FAUST simulates the various sized transistors in the network using SPICE first [Nagel75]. This information is then stored in a look-up table to be retrieved when needed later. To model circuit delays, capacitance and resistance are also extracted and placed in the network in a lumped fashion as loads on gates of transistors.

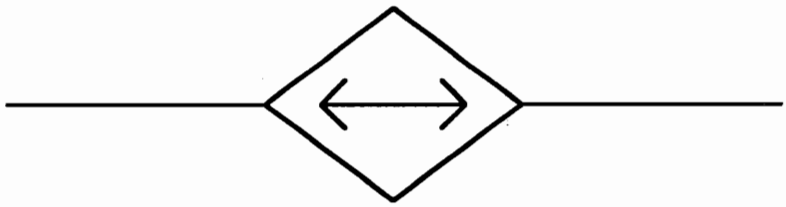


FIGURE 2.1 FAUST transistor model.

At a higher level, CSASIM [Kawai84] uses connectors, switches, and attenuators to model MOS networks as shown in Figure 2.2. The connectors not only link switches and attenuators but also act to resolve differing discrete level values. Similar to this level, FMOSSIM [Bryant85] uses nodes interconnected by transistor switches to represent the MOS network. The two types of nodes used represent stored charge. The input node is used to represent points that are connected to a primary input, power,

or ground. The storage node is used to represent the voltage or charge at the node due to the capacitance on that node. The transistor is modeled as a discrete conducting device with a strength of between 1 and W associated with it. W, used to override all other values, is the strongest value, and is reserved for primary inputs. The switch conductance (T), used to override the strength of node stored charge, has the next strongest values of $K \leq W - 1$. The node stored charge (S) has the lowest values of $1 \leq K$. The node and transistor models are shown in Figure 2.3.

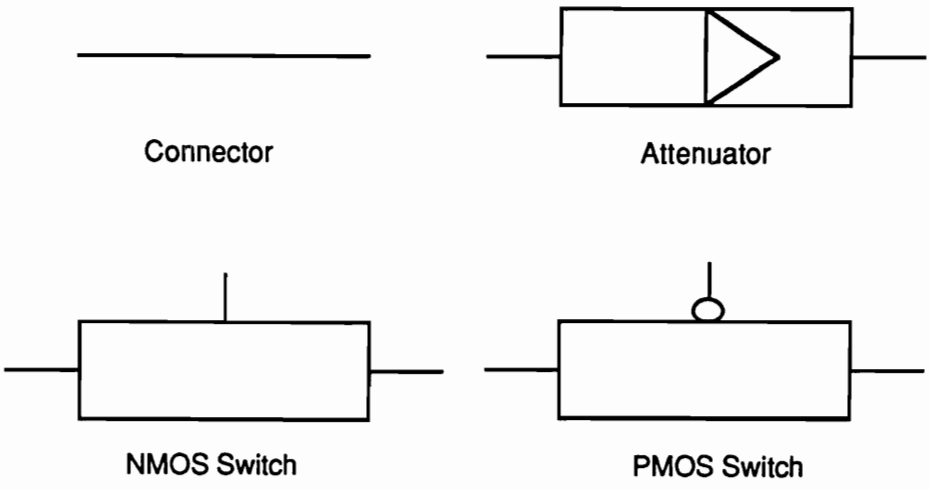


FIGURE 2.2 CSASIM device models.

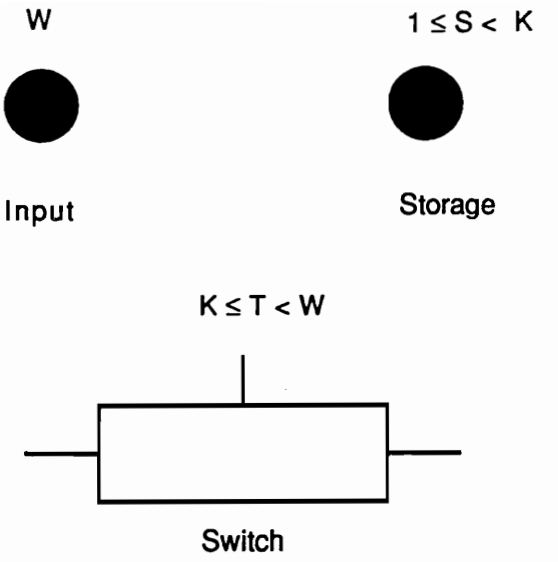


FIGURE 2.3 FMOSSIM device models.

At a still higher level, SLS [Barzi86] represents the network as being connected transistor source-drain pairs. An example is the CMOS inverter as shown in Figure 2.4. Still increasing the level of abstraction, BOSE et al. [Bose82] use logical models to represent the network by first classifying each transistor as either a driver or a load device. The logical effect of these devices are then mapped to the input or output of the logic gate as shown in Figure 2.5. Load devices are those which separate the output and Vcc, while drive devices are those which separate the input and Vss. While primarily used for NMOS fault simulation, it is asserted that this concept can be used for CMOS [Bose82] fault simulation.

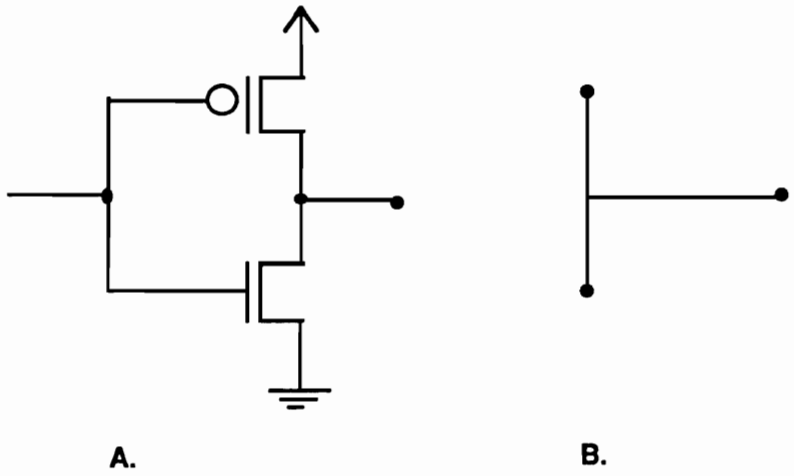


FIGURE 2.4 SLS graph model. A. CMOS inverter circuit diagram. B. The CMOS inverter undirected graph.

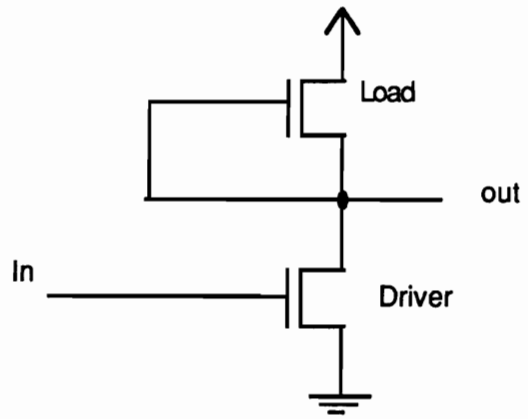


FIGURE 2.5 BOSE logical models.

A sample of the mathematical and conceptual models are shown in Figures 2.6-2.8. Figure 2.6 and Table 2.1 show discrete levels and the CONNECTOR function as used for CSASIM and simplified here. The four discrete level values used are {Z, 0, 1, X}, where Z represents high impedance, X represents unknown, 0 represents logical low, and 1 represents logical high. The order of values shown in Figure 2.6 is determined by resolution in Table 2.1. Any number of strengths can be represented in this fashion [Hayes87]. Figure 2.7 and Table 2.2 show transistor state, three-valued node levels, and the strength concept as used in FMOSSIM. The three types of transistors are N-type (NMOS), P-type (PMOS), and D-type (enhancement mode pull-up). At the gate of the transistor, 0 is logic level low, 1 is high, and X is unknown. For the transistor state, 0 is non-conducting, 1 is conducting, and X is unknown. Strength level values range from 1 to W and three types exist. The strengths with value range $\{1 \leq k \leq S\}$ are the lowest type and are used to model storage nodes. Different values for the storage node are used to model the stored charge with respect to the capacitance at that node. The strengths with value range $\{S < k < W\}$ are a higher strength type and are used for transistor conductance capability. The transistor conductance capability is a function of transistor size, its width to length ratio, and its connect neighbor transistors sizes. The third strength type has the highest value of W and is used for primary inputs, power, and ground.

As mentioned earlier, SLS uses the notion of connected transistors (source/drain pairs). To represent states (voltage assignment), a directed graph is extracted from the network. The graph is based on the state of the transistors along with the relative strengths of the nodes on the connected path. Figure 2.8 shows the resulting graphs of the CMOS inverter relative to the different states of the transistors. Figure 2.8 B shows the logic value flow of the inverter when an input value of zero is applied, while

Figure 2.8 C shows the logic value flow of the inverter when an input value of one is applied.

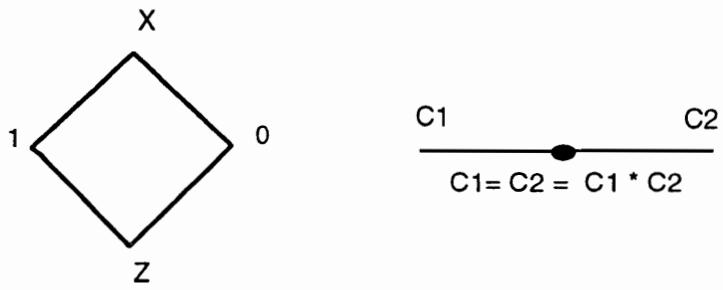


FIGURE 2.6 CSASIM A. Discrete levels. B. Connector operation at a node.

TABLE 2.1 CSASIM CONNECTOR function.

*	Z	0	1	X
Z	Z	0	1	X
0	0	0	X	X
1	1	X	1	X
X	X	X	X	X

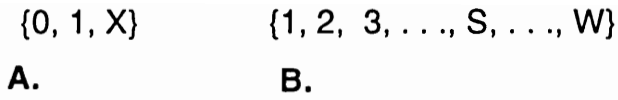


FIGURE 2.7 FMOSSIM strength system.

A. Node levels. B. Strengths.

TABLE 2.2 FMOSSIM transistor states.

Transistor Gate State	N	P	D
0	0	1	1
1	1	0	1
X	X	X	1

Unlike the logic gate representation of the circuit, the basic switch is bidirectional. This bidirectional nature is shown schematically in Figure 2.9. Node strength is used

to determine the transistor conductance direction. For example, a value of 1 or zero at one terminal of the switch would flow into a terminal with a value of Z.

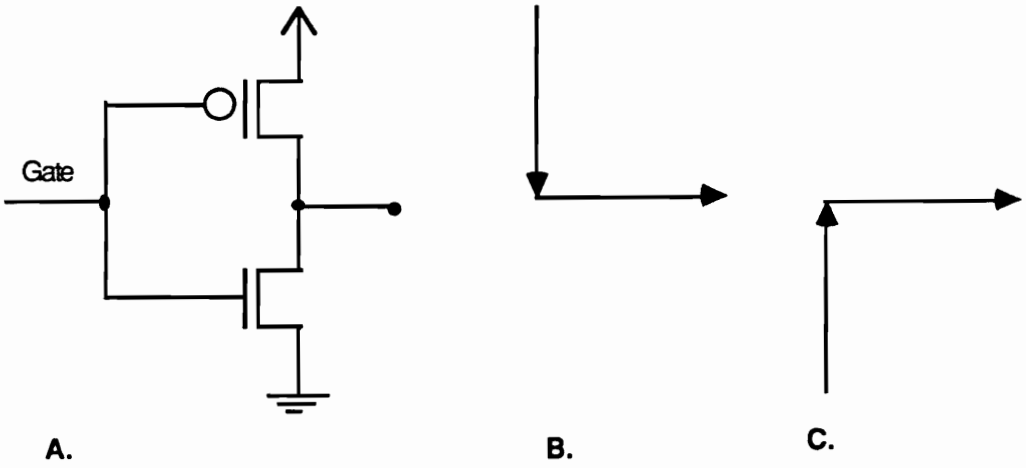
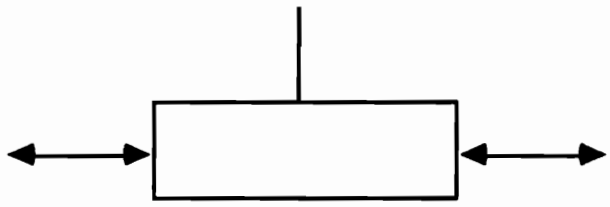


FIGURE 2.8 SLS A. CMOS inverter, and network directed graphs. B. Gate = 0, C. Gate = 1.



Bidirectional Nature

FIGURE 2.9 Bidirectional switch.

2.3 Switch Level Models and Algebra

The device models used in this research are the P - type switch, N - type switch and the connector shown in Figure 2.2. However, a nine-valued logic system is used to represent stored charge.

This research uses the IEEE proposed standard nine-valued logic [Bill91] for switch level simulation, somewhat similar to that used by [Hayes82]. The values, listed here in Figure 2.10 ranging from the weakest to the strongest, are {Z, -, L, H, W, 0, 1, X, U}. The values of 1 and 0 represent logic levels forcing high and low, respectively. The values of H and L represent logic levels weak high and low respectively. The values of X and W represent forcing and weak unknown respectively. The values of U, -, and Z represent uninitialized, don't care, and high impedance, respectively.

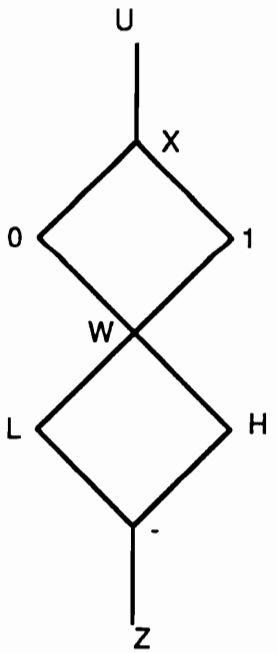
Using nine-valued logic, the N - type MOS switch is modeled by a state table as shown in Table 2.3 where the input is applied to the drain and the output is taken at the source. Shown in Figure 2.11, the transistor switch model has basically three modes; off, on, and unknown.

In the off mode, the N - type switch isolates the output from the input. The next output value is then based on the previous output value and the capacitance at the output node. Since that output node is isolated from both power and ground, it is

assumed that while the transistor is off a previous output value of a forcing 1 degrades to a weak H, a forcing 0 to a weak L, a forcing X (1 or 0) to a weak W (H or L), and all other weak W, H, L, -, Z to high impedance Z.

In the on mode, the N - type switch connects input to output and effectively passes the input value to the output. However, assuming the worst-case, the N - type switch degrades the 1 and H values to H and W, respectively.

In the unknown mode, the N - type switch can be either on or off. Thus in the worst-case, the output becomes forcing unknown X with strong inputs U, X, or 0 present, while the output becomes weak unknown W with weak inputs 1, W, H, L, -, or Z present.



U - Uninitialized

X - Forcing 0 or 1

1 - Forcing 1

0 - Forcing 0

W - Weak 0 or 1

H - Weak 1

L - Weak 0

- - Don't Care

Z - High Impedance

FIGURE 2.10 Nine-valued logic ordering.

Using nine-valued logic, the P - type MOS switch is also modeled by a state table as shown in Table 2.4 where the input is applied to the drain and the output is taken at the source. The transistor switch model has basically three modes; off, on, and unknown.

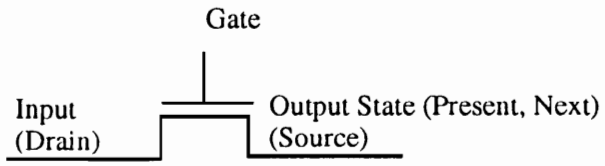


FIGURE 2.11 Switch model (N-type).

TABLE 2.3 N-type switch state table.

Present State	Input	Gate	Next State
-	0	1,H	0
-	1	1,H	H
-	H	1,H	W
-	L	1,H	L
-	U,X,W,-	1,H	U,X,W,-
-	U,X,0	U,X,Z,W,-	X
-	1,W,H,L,-,Z	U,X,W,-,Z	W
-	Z	1,H	W
U	-	0,L	U
X	-	0,L	W
1	-	0,L	H
0	-	0,L	L
W,H,L,-,Z	-	0,L	Z

TABLE 2.4 P-type switch state table.

Present State	Input	Gate	Next State
-	0	0,L	L
-	1	0,L	1
-	H	0,L	W
-	L	0,L	W
-	Z	0,L	W
-	U,X,W,-	0,L	U,X,W,-
-	U,X,1	U,X,Z,W,-	X
-	0,W,H,L,-,Z	U,X,W,-,Z	W
U	-	1,H	U
X	-	1,H	W
1	-	1,H	H
0	-	1,H	L
W	-	1,H	Z
H	-	1,H	Z
L	-	1,H	Z
-	-	1,H	Z
Z	-	1,H	Z

In the off mode, the P-type switch isolates the output from the input. The next output value is then based on the previous output value and the capacitance at the output node. Since that output node is isolated from both power and ground, it is assumed that while the transistor is off a previous output value of a forcing 1 degrades to a weak H, a forcing 0 to a weak L, a forcing X (1 or 0) to a weak W (H or L), and all other weak W, H, L, -, Z to high impedance Z.

TABLE 2.5 Nine-valued connector operation.

*	U	X	0	1	Z	W	L	H	-
U	U	U	U	U	U	U	U	U	U
X	U	X	X	X	X	X	X	X	X
0	U	X	0	X	0	0	0	0	0
1	U	X	X	1	1	1	1	1	1
Z	U	X	0	1	Z	W	L	H	-
W	U	X	0	1	W	W	W	W	W
L	U	X	0	1	L	W	L	W	L
H	U	X	0	1	H	W	W	H	H
-	U	X	0	1	-	W	L	H	-

In the on mode, the P-type switch connects input to output and effectively passes the input value to the output. However, assuming the worst-case, the P-type switch degrades the 0 and L values to L and W, respectively.

In the unknown mode, the P-type switch can be either on or off. Thus in the worst-case, the output becomes forcing unknown X with strong inputs U, X, or 1 present, while the output becomes weak unknown W with weak inputs 0, W, H, L, -, or Z present.

TABLE 2.6 Minimum operation.

Min	U	X	0	1	Z	W	L	H	-
U	U	X	0	1	Z	W	L	H	-
X	X	X	0	1	Z	W	L	H	-
0	0	0	0	0	Z	W	L	H	-
1	1	1	0	1	Z	W	L	H	-
Z	Z	Z	Z	Z	Z	Z	Z	Z	Z
W	W	W	W	W	Z	W	L	H	-
L	L	L	L	L	Z	L	L	L	-
H	H	H	H	H	Z	H	L	H	-
-	-	-	-	-	Z	-	-	-	-

The nine-valued logic uses operators AND, OR, NOT, and RESOLVE as described in [Bill91]. Repeated here in Table 2.5, the RESOLVE operator will be used to implement the nine-valued CONNECTOR (*) operator. In addition, the MINIMUM (MIN) function

as shown in Table 2.6 will be used. These functions will be used for the parallel switch level fault simulation described in Chapter 3.

Unlike gate level fault simulators, switch level fault simulators must take into account the bidirectional nature of transistors. Methods of model implementation for the bidirectional switch vary from table look up of preset voltage conductance values [Shih86], to a type of signal strength resolution [Bryant85], to dividing the circuit into two separate unidirectional parts [Kawai84], [Barzi86]. This research will handle complex combinational logic gates only and not handle the bidirectional switch or sequential circuits. Study into the bidirectional switch as well as sequential circuits is proposed for future work.

2.4 Fault Models

It has been shown that a group of switch level fault models accurately describe most classes of actual physical defects found on the CMOS IC [Shen85]. The fault models are line stuck-at faults, transistor stuck on/off faults, floating line faults, and bridging faults. These fault models are shown schematically in Figure 2.11. Line stuck-at faults are similar to gate stuck-at faults except at the lower switch level. Transistor stuck on/off faults can be modeled by the line stuck-at fault on the gate of the switch. In the presence of the transistor stuck on/off fault, the transistor is either always conducting or never conducting. The floating line fault can be viewed as the split of one node into two non-conducting nodes, while the bridging fault can be viewed as the superposition of two non-conducting nodes into one node. These fault models are shown in Figure 2.11 A, B, C, and D, respectively. Most of the more recent

switch level fault simulators implement two or more of these fault models in one capacity or another.

The nine - valued logic is adequate for the switch level stuck-at faults because the fault effect can be propagated logically through the circuit as will be shown in Chapter 5. The bridging fault is not simulated in the work presented here. One reason that the bridging fault is not considered in this work is that propagating the effects of the bridging fault is generally better left for current I_{DDQ} testing. This is because the sensitization of the of the bridging fault produces a logic value of a strong unknown X. This unknown X is hard to propagate in the logic domain, i.e., the propagation of the X depends on the relative strength of the bridge. However, current I_{DDQ} testing detect the presents of the fault by measuring the circuit current. If excessive current is flowing then a bridge between Vdd and Ground or a strong unknown X is present in the circuit. For more information on bridging fault severity and fault simulation of bridging faults using current testing see [Koe89] and [Lim89], respectively.

The fault set considered in this research is line stuck-at-1/0 on the switch as shown in Figure 2.11 A. Transistor stuck on and off faults are modeled by transistor gate stuck-at-1/0, depending on whether the device is N - type or P - type. The transistor stuck on and off faults are not considered with traditional gate level fault simulation. Also implemented in the fault set, as shown in Figure 2.12, is fan-out line of stem stuck-at-1/0 fault. Only a single fault per circuit is assumed, i.e., the classic single fault model is used.

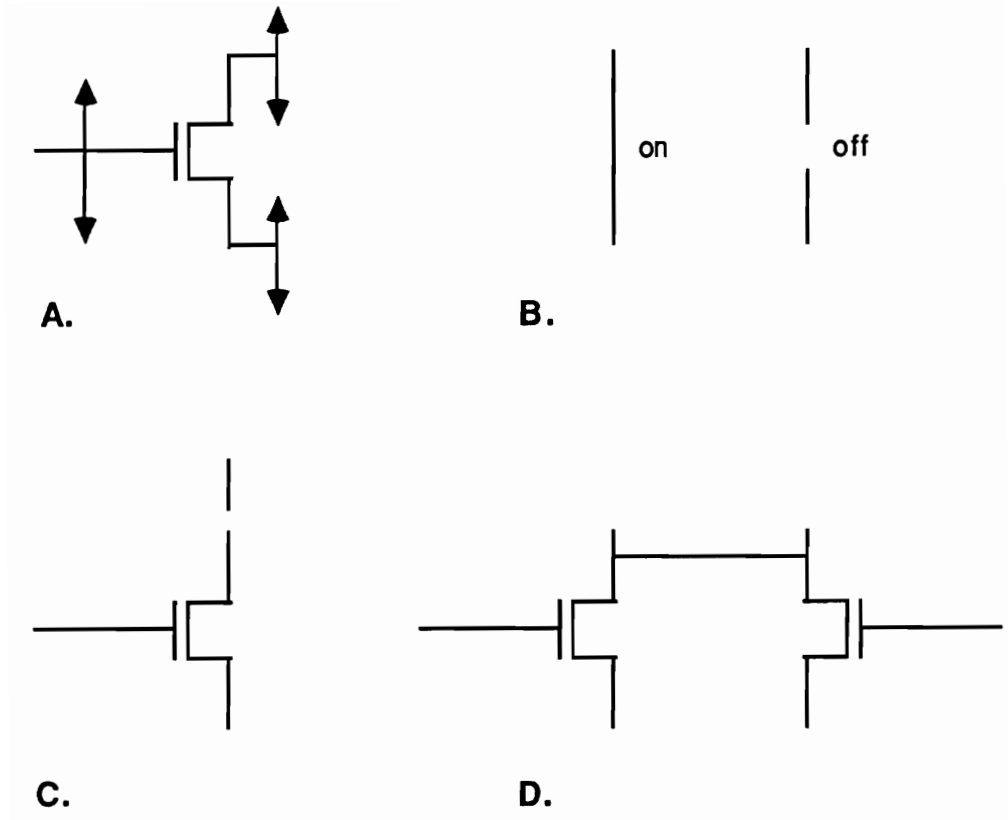


FIGURE 2.12 Fault models: A. Line stuck-at-0 or 1. B. Transistor stuck-on/off. C. Floating line. D. Bridging.

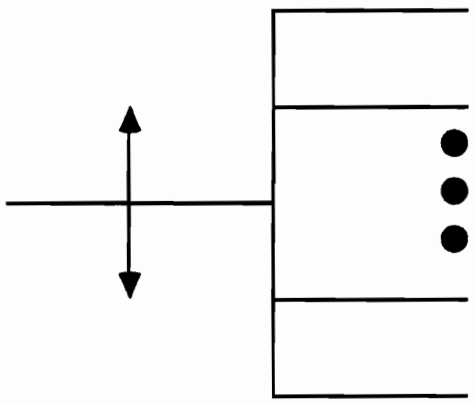


FIGURE 2.13 Line stuck-at (stem) before fan-out.

At the switch level, the fault implementation technique for simulation is fault injection. Fault injection used by SLS, FAUST, CSASIM, BOSE and FMOSSIM is the process of adding the effects of a fault into the circuit by injecting an extra device into that circuit. In this research, fault injection is used for both line stuck-at faults and fan-out line stem faults. The line stuck-at faults are injected via a *mask* in parallel fault simulation, while the stem faults are injected by inverting the stem lines one by one for each test vector, simulating, and comparing the results to the original test for each test vector. The injection of line stuck-at and stem faults will be discussed in Chapters 3 and 5, respectively.

Chapter 3.

Parallel Fault Simulation

3.1 Introduction

Parallel fault simulation, as shown schematically in Figure 3.1, is well-established for the gate level [Fuji85]. By taking advantage of the general purpose computer's built-in logic operations, the idea in parallel fault simulation is to performs bit-wise simulation on parallel faults for a logic gate (AND, OR, XOR). One bit in a word is used for the good value, while all other bits are used for faulty values. Using the parallel fault simulation process multiple faults can be propagated through an arbitrary gate in the simulated circuit in a single clock cycle.

This chapter discusses the parallel fault simulation technique as extended to the switch level. Included in this discussion is an example from the VHDL simulation results of a single N - type switch. In contrast to the gate level, the computational problems encountered at the switch level are: nodes have fan-in, and nodes have a state or stored charge. Furthermore, unlike the AND, OR, NOT, and XOR gate level built-in logic instructions of the general purpose computer, the sequential SWITCH operations as defined in Tables 2.3 and 2.4 are not built-in to the general purpose computer's instruction set.

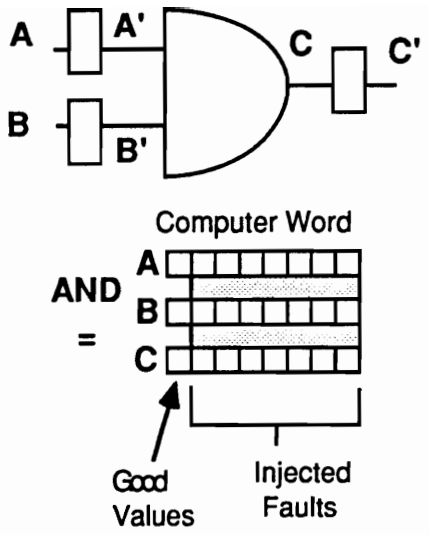


FIGURE 3.1 Gate Level Parallel Fault Simulation.

3.2 Switch Level Parallel Fault Simulation

As shown in Figure 3.2, parallel fault simulation is extended to the switch level by using the nine-valued logic and the N-type and P-type switch state tables which are given in Figures 2.10, and Tables 2.3 and 2.4, respectively. During parallel fault simulation, faults are injected on each signal line by the MASK blocks. Two constant vectors used in the MASK procedure are the *fvalue* vector (GF,DF,SF) and the *mask* vector (GM,DM,SM) as shown in Tables 3.1 and 3.2, respectively.

The *mask* vector is used to inject the faults and has value of Z for faulty places and U at all other positions. Since Z has the lowest strength and U has the highest strength of the values in the nine-value logic, the minimum operation using the *mask* vector and the good signal vectors as inputs will always yield a Z in the faulty positions.

The *fvalue* vector is used to select the fault value and contains a position for the fault free value and for each faulty value. Here, a line with fault stuck-at-1 has value 1, while a line with fault stuck-at-0 has value 0. All other values are Z. For example, the transistor gate *fvalue* vector GF has a value of 1 in the G1 position designating a stuck-at-1 type fault on the gate, while there is a 0 in the G0 position designating a stuck-at-0 type fault. Since all other values are Z, the minimum operation is now used in order to select the faults 1 and 0 in the G1 and G0 positions, while all other positions remain at their good value.

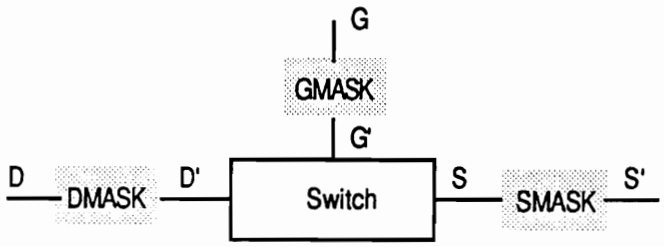


FIGURE 3.2 Switch Level Parallel Fault Simulation.

The Bold face constants in Tables 3.1 and 3.2 are defined as follows:

GF-Gate *fvalue* vector,

DF-Drain *fvalue* vector,

SF-Source *fvalue* vector,

GM-Gate *mask* vector,

DM-Drain *mask* vector,

SM-Source *mask* vector,

FF-Fault Free Position,

G1-Gate Stuck-At-1 Position,

G0-Gate Stuck-At-0 Position,

D1-Drain Stuck-At-1 Position,

D0-Drain Stuck-At-0 Position,

S1-Source Stuck-At-1 Position,

S0-Source Stuck-At-0 Position

TABLE 3.1 Constant fvalue vectors.

	[FF	G1	G0	D1	D0	S1	S0]
GF	[Z	1	0	Z	Z	Z	Z]
DF	[Z	Z	Z	1	0	Z	Z]
SF	[Z	Z	Z	Z	Z	1	0]

TABLE 3.2 Constant mask vectors.

	[FF	G1	G0	D1	D0	S1	S0]
GM	[U	Z	Z	U	U	U	U]
DM	[U	U	U	Z	Z	U	U]
SM	[U	U	U	U	U	Z	Z]

In order to perform vector operations on the switch, the gate, drain, and previous source good values must be expanded to the word length as

$$G <= [GGGGGGG], \tag{3.1}$$

$$D <= [DDDDDDD], \tag{3.2}$$

and

$$S_{-1} <= [S_{-1}S_{-1}S_{-1}S_{-1}S_{-1}S_{-1}S_{-1}], \tag{3.3}$$

respectively, where G, D, and S_{-1} are the good values.

Next, in order to inject faults at the inputs G and D, the following switch level parallel fault simulation *mask* operations are performed as

$$G' = [MIN(G,GM)]*GF \tag{3.4}$$

and

$$D' = [MIN(D,DM)]*DF, \tag{3.5}$$

where * is the CONNECTOR function, and MIN() is the MINIMUM function. Remember that GF, DF, GM, and DM are all vectors and to calculate G' and D' vector operations must be performed. The output of the switch is determined by

$$S = \text{Switch}[G,D,S_{-1}], \qquad (3.6)$$

where Switch is the switch function for the N-type or P-type transistor, and S-1 is the previous output value of the switch. In order to inject faults on the output, the *mask* operation is performed as

$$S' = [\text{MIN}(S,SM)]^{*}SF, \qquad (3.7)$$

where S is defined in Equation 3.6. The resulting output vector contains positions for the fault-free output as well as positions for all outputs in the presence of each specific fault. Each fault is considered detected if that fault position output is noticeably different from the fault-free position output. Noticeably different is defined as being a result of a 1 or an H instead of a 0 or an L. Output values of W, and X and not considered noticeably different.

An example of switch level parallel fault simulation is given below. The results for this example, shown as the SOURCE value after the output MASK operation, indicate that the three faults gate stuck-at-0, drain stuck-at-1, and source stuck-at-1 are detected for the test vector, while the three faults gate stuck-at-1, drain stuck-at-0, and source stuck-at-1 are not detected. The input test vector for this example is <D,G> = <0,1>, with the previous source value S_{-1} equal to 1.

First, use Equations 3.1 - 3.3 in order to expand gate, drain and previous source values as

INPUT

SOURCE_P (value = 1) = (1111111) (3.8)

DRAIN (value = 0) = (0000000) (3.9)

GATE (value = 1) = (1111111). (3.10)

Second, perform *mask* operation on the expanded drain and gate using Equations 3.4 and 3.5 as

MASK OPERATION

DRAIN'(value = "0001000") (3.11)

GATE' (value = "1101111") (3.12)

Next, perform the switch operation using Equations 3.6 as

SWITCH OPERATION

SOURCE' (value = "00HH000") (3.13)

Finally, perform the output *mask* operation using Equation 3.7 as

MASK OPERATION

SOURCE (value = "00HH010") (3.14)

Equation 3.14 gives the resulting vector that includes the fault free output and outputs in the presence of the stuck-at fault set. Since the fault free output is 0, and since there is an H in positions G0 and D1, the faults gate stuck-at-0 and drain stuck-at-1

are detected for the vector applied. Similarly, since there is a 1 in positions S1, the fault source stuck-at-1 is detected. Faults not detected are the faults of the positions whose values match that of the fault free output and have value 0.

The results for a second example show that the faults gate stuck-at-1, and source stuck-at-1 are detected for the test vector, while faults gate stuck-at-0, drain stuck-at-0, drain stuck-at-1, and source stuck-at-0 are not detected. Notice that the switch is sequential in nature since its output (state) is dependent not only on the inputs but on its previous output (state). The test vector for this example is <D,G> = <1,0>, with previous source value S_{-1} equal to 0.

INPUT

PREV_SOURCE (value = 0) (3.15)

DRAIN (value = 1) (3.16)

GATE (value = 0) (3.17)

MASK OPERATION

GATE (value = "0100000") (3.18)

DRAIN (value = "1111011") (3.19)

SWITCH OPERATION

SOURCE (value = "LHLLLLL") (3.20)

MASK OPERATION

SOURCE (value = "LHLLL10") (3.21)

3.3 Parallel switch level fault simulation summary

Using nine-valued logic, the switch state tables, and a minimum operation, parallel fault simulation has been extended to the switch level. The advantage of using this technique for switch level fault simulation is that all switch level faults can be simulated in parallel. The disadvantage is that general purpose computers do not have built in switch operations. With regard to this disadvantage and since the parallel switch level fault simulation technique is the core computation in switch level fault simulation, it is proposed that the parallel switch level fault simulation operation be performed in hardware and defined as one processing element (PE).

The switch level parallel fault simulation technique was verified for the 9^3 combinations of inputs and previous states using a VHDL simulator [Synop90]. Appendix A gives complete VHDL switch level parallel fault simulation results.

Chapter 4.

Circuit Partitioning

4.1 Introduction

Circuit partitioning is used for two reasons. The first reason is to take advantage of circuit latency. Latency is a property associated with concurrent fault simulation. Basically, when a fault is injected into a circuit only a small portion of the circuit's values will change states. The majority of the circuit is latent or inactive. The usefulness of circuit partitioning in this regard is to group or partition circuit elements that are tightly coupled, i.e., elements that depend on each others values. In concurrent fault simulation, only partitions that are active need to be simulated. Furthermore, simulation is halted when signal values stop changing. One such examples of partitioning to take advantage of circuit latency is switch simulation using partial orders [Agraw90.2].

The second reason for circuit partitioning is to take advantage of parallel processing. Since fault simulation is CPU intensive, computers simulating in parallel are used in order to reduce simulation time. However, most fault simulation implementations that use parallel processing perform partitioning of the fault list or of the test set and not

of the circuit elements [Mark90], and [Duba88]. With fault list partitioning or test set partitioning, multiple copies of the fault simulation program are distributed to any number of processors and the fault list or test set is just divided between them. Circuit partitioning for parallel processing has not been performed in the distributed environment because circuit simulation is very fine grained and the slow communication time of a distributed network adds too much overhead.

The work presented in this chapter addresses circuit partitioning for parallel processing and concurrent fault simulation. The goal of this work is to simulate a circuit partition in parallel with special purpose hardware.

4.2 Circuit Partition for Parallel Switch Level Fault Simulation

The original idea for the circuit partitioning used here came from the basic concept that fault simulation of an arbitrary fault in a CMOS circuit was difficult. However, if the CMOS circuit had no internal nodes, and hence only primary inputs and primary outputs, then fault simulation of an arbitrary fault would be easy. If simulated in the correct sequence as shown in Chapter 5, one reverse level of switches can in fact appear similar to a CMOS circuit with no internal nodes.

Reverse level ordering, somewhat similar to critical path tracing [Abram84], is of particular interest because it enables parallel fault simulation of individual levels of switches. This is because two properties, when taken together, enable the reverse level to be thought of as a single CMOS circuit that has no internal nodes. The first property that helps to enable the reverse level to be thought of as a single CMOS circuit is that all signals that fan-in at a node must all be present during processing of

that node. This property is important because fault propagation through a node can only occur with all fan-in values at the node present. The second property that helps to enable the reverse level to be thought of as a single CMOS circuit is that all input values on the switches of that level must be known. This property is important because switch simulation of the level can only occur with known input values.

With reverse level ordering, all of the switches that are inputs to a node are in the same level. Thus, if that level is processes in parallel, then all fan-in values at the nodes are known. In addition, if the reverse level ordered circuit is simulated from the primary inputs one level at a time to any reverse level, then all of the input switch values for that reverse level are known. Thus reverse level ordering is effectively used to isolate levels of switches for parallel processing.

The reverse level ordering of the example XNOR gate of Figure 4.1 is shown in Figure 4.2. The ordering process is started by labeling the primary outputs as Level 1. All transistors directly connected to the primary outputs are then traversed backwards and their inputs are labeled Level 2. Level 1 transistor inputs are connected to level 2 outputs or to primary inputs. This step is repeated until all transistors have been traversed and the primary inputs are reached. A reverse level of switches is important because it can be simulated in parallel since all switch input values have been determined from simulation of the previous level. For this reason, a reverse level of switches can be considered disjoint from other levels as long as order is preserved.

The reverse level process involves two algorithm. First, Algorithm 4.1 performs a forward pass on the circuit and is used to determine transistor inputs and outputs.

Next, Algorithm 4.2 performs a backward pass on the circuit and is used to actually reverse level the circuit. Some preliminary definitions needed are as follows:

Definition 4.1 The *transistor list* is the set of transistors in the circuit remaining to be processed.

Definition 4.2 The *input list* is the set of nodes of transistors from the transistor list that are known to be only input, i.e., signals that are not connected to any transistor outputs.

Definition 4.3 The *output list* is the set of nodes of transistors from the transistor list that are known to be only output, i.e., signals that are not connected to any transistor inputs.

Definition 4.4 The *search list* is the set of nodes of transistors from the transistor list that could be both input and outputs.

Starting at the primary inputs, Vdd and Ground, Algorithm 4.1 searches the circuit for transistors that have known inputs and outputs. The idea is to identify two nodes of the transistor as inputs and then by default the third node is an output. Once the transistor inputs and outputs are identified, the transistor can be removed from the search list. Algorithm 4.1 is as follows:

Algorithm 4.1

1. Let i be a signal in the input list. Scan the search list for signal i. Let T be the set of transistors connected to signal i and let Ts be the set node signals of the transistor set T.
2. Let j be an element of Ts. Scan the input list for signal j.
3. If the input list contains j then let the transistor connect to j have output labeled k or T(i,j,k). Now remove transistor T(i,j,k) from set T and from the transistor list. Also, remove the nodes of T(i,j,k) from the search list.
4. Scan the search list for node k. If k appears once and only once, then add k to the input list.
5. Next j. Loop to Step 2.
6. If set T is empty, then remove signal i from the input list.
7. Next i. Loop to Step 1.

Starting at the primary outputs, Algorithm 4.2 searches the circuit in reverse order one level of transistors at a time. The idea is to order the transistors from output to input and also to keep the transistors that fan - in to common nodes in the same level. Algorithm 4.2 is as follows:

Algorithm 4.2

1. Let k be a signal in the output list. Scan the search list for signal k. Let T be the set of transistors connected to signal k and let Ts be the node signal set of the transistor set T.
2. Let j be an element of Ts. Scan the search list for signal j.

3. If the search list contains j then let the transistor connect to j be called T(i,j,k). Now remove transistor T(i,j,k) from set T and from the transistor list. Also, remove the nodes of T(i,j,k) from the search list.

4. Scan the search list for node . If i and or j are not connect to any inputs then add i and or j to the output list.

5. Next j. Loop to Step 2.

6. If set T is empty, then remove signal k from the output list.

7. Next k. Loop to Step 1.

During reverse level ordering, inputs and outputs of the switch are now accounted for by assigning the inputs to be the gate of the switch and that node of the switch which is farthest from the primary output. Conversely, the output will be the node that is closest to the primary output.

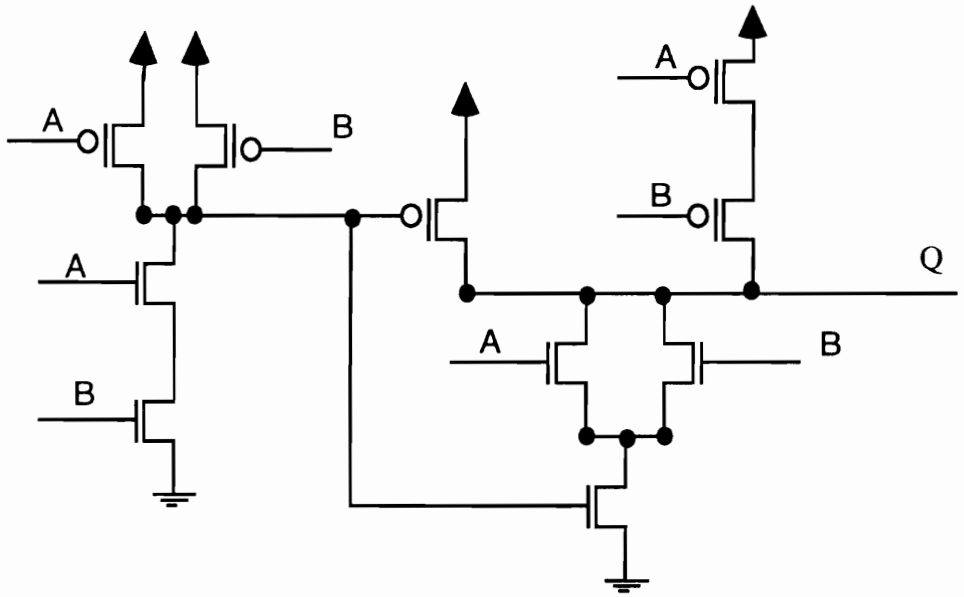


FIGURE 4.1 CMOS XNOR gate.

Since a reverse level of switches could be arbitrary large, processing an entire reverse level in parallel would require an arbitrary large number of PEs. Thus reverse levels are further divided into groups and subgroups of transistors. A *group* of transistors in a level is defined as those transistors that are electrically connected via their inputs or outputs. This excludes connection by virtue of sharing ground or Vdd nodes. Groups are by definition disjoint since if an input or output were in more than any one group, these groups would be connected and they would form a single larger group. Since groups in a reverse level are disjoint they can be simulated independent of one another. Two groups in a single reverse level are shown in Figure 4.3.

Subgroups of a group in a reverse level are defined as those transistors in a group that have a common output. Subgroups are disjoint in outputs but not necessarily in inputs. This division is important because output signals have fan-in and they have to be resolved with the CONNECTOR (*) function in hardware, while input signals have been resolved at the previous level. Figure 4.3 also contains an example of subgroups within a group in a reverse level.

In order to map subgroups to hardware, partitions of subgroups are now introduced. A *partition* is defined as the maximum number of transistors that can be processed in parallel on the PHAFS architecture. The maximum partition size is limited only by the number of processing elements (PEs) on the actual PHAFS board. The partition size will be discussed in Chapter 6.

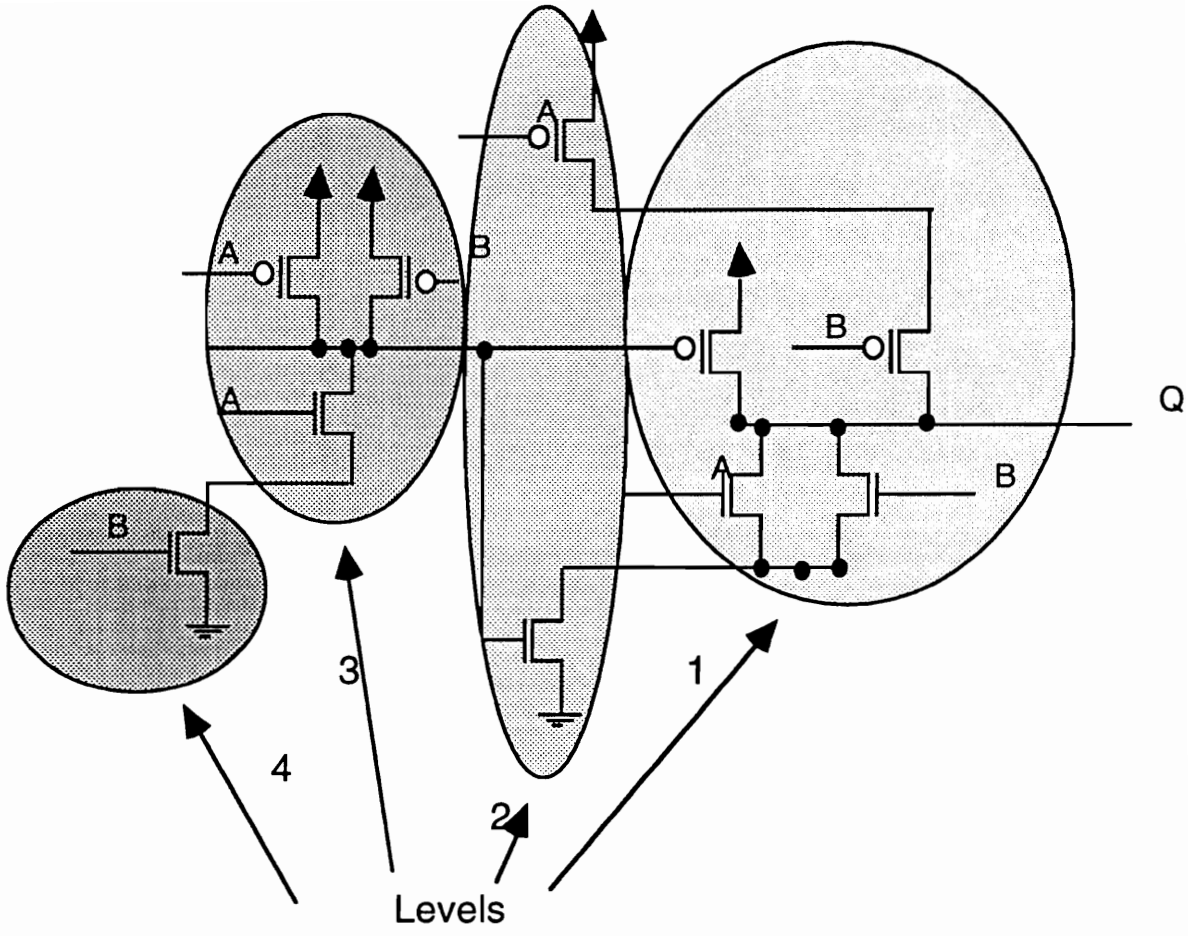


FIGURE 4.2 Reverse Level Ordering of CMOS XNOR gate.

As an example of circuit partitioning, the smallest ISCAS85 circuit is shown in Figure 4.4. For this example, the maximum partition size is 4. Since order must be preserved, partitions can only include switches from one reverse level. In addition, depending on subgroup sizes and the number of transistor in the reverse level, partition will not necessarily use the maximum number of processing elements available. The usage of processing elements or load will be discussed in detail in Chapter 6.

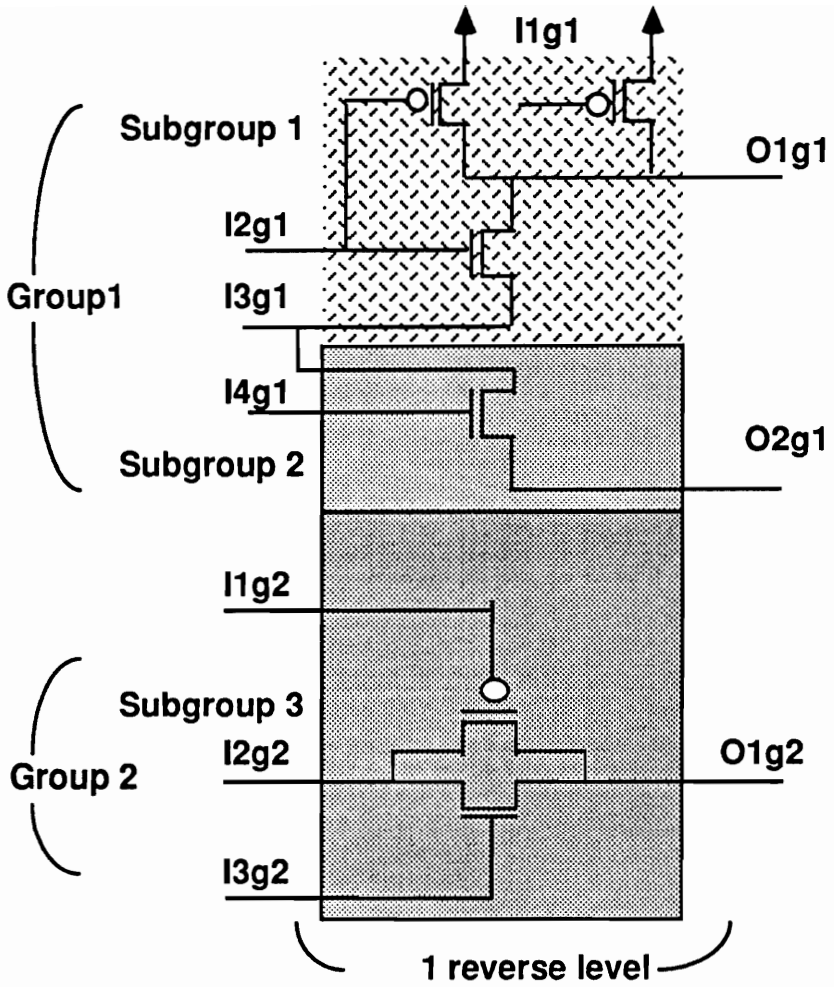


FIGURE 4.3 Groups and Subgroups in a Reverse Level.

Because the circuit partitioning used here partitions disjoint groups of transistors, the reverse level ordering circuit partitioning is of primary importance to the

parallel switch level fault simulation algorithm and hardware design issues as discussed in Chapters 5, 6, and 7.

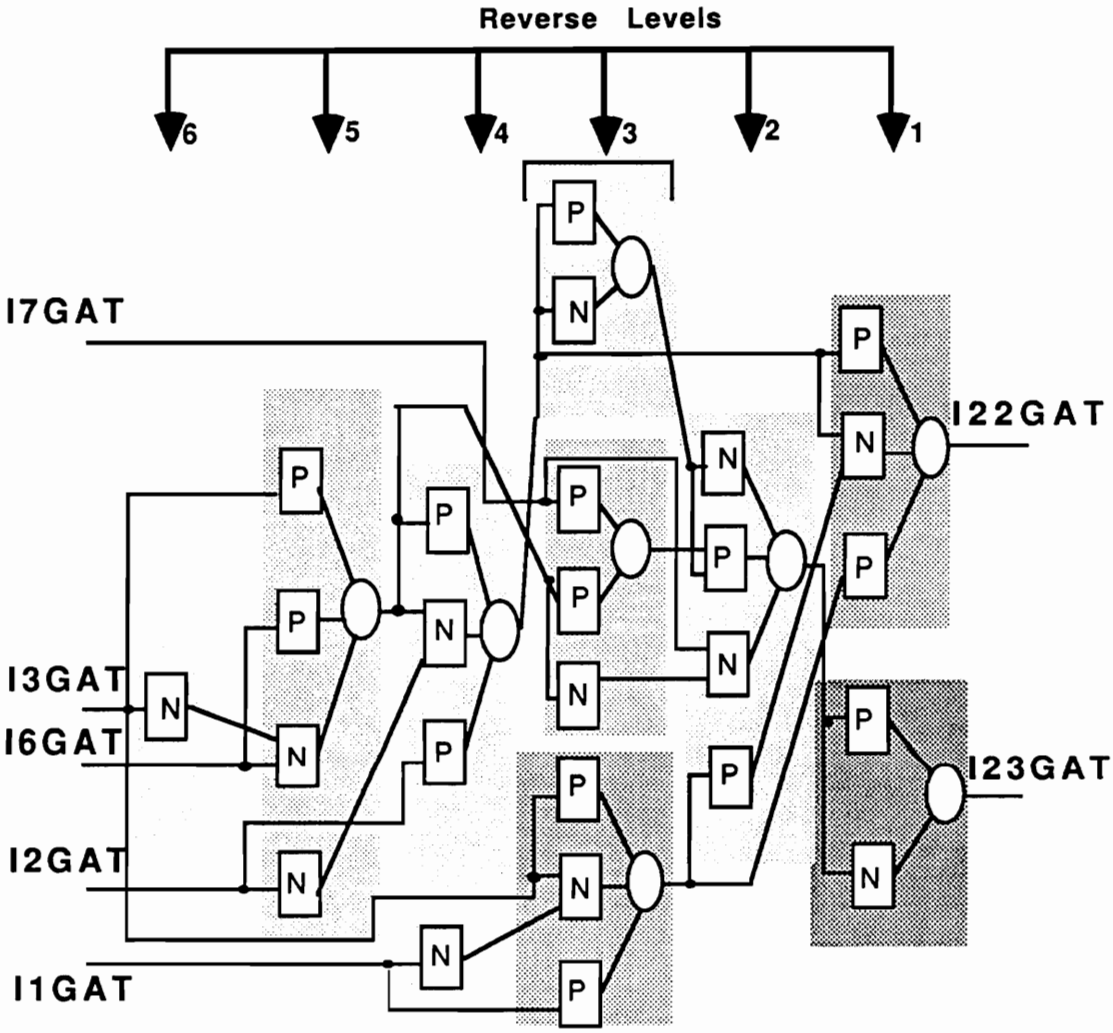


FIGURE 4.4 C17 reverse level ordered and partitioned.

Chapter 5.

Parallel Switch Level Fault Simulation Algorithm and Complexity

5.1 Introduction

This chapter uses the parallel switch level fault simulation technique given in Chapter 3 along with the reverse level order circuit partitioning method given in Chapter 4 as building blocks for the parallel fault simulation algorithm. The algorithm is given in Section 5.4. Presented in Section 5.5 is the upper bound of the algorithm's computational complexity, which is order L^2, where L is the number of reverse levels. Required for both the algorithm and the algorithm's complexity, Section 5.2 presents two-Dimensional parallel fault simulation for one reverse level, while 5.3 presents symbols, definitions, and theorems.

5.2 Two-Dimensional Parallel Switch Level Fault Simulation

The two-dimensional fault simulation performed here simulates devices (N - type and P - type switches) and faults in parallel for a single test vector. As a detailed example of the process of two-dimensional parallel fault simulation, consider the AND

gate of Figure 5.1. Using the reverse level ordering of the AND gate, one reverse level or one partition, whichever is smaller, of switches and faults would be simulated in parallel. The faults for one switch are simulated in parallel on a single PE using the switch level parallel fault simulation technique discussed in Chapter 3. The faults for all switches in a particular level or partition are simulated in parallel using the multiple processing elements of the hardware, while the outputs that fan-in at a node are resolved using the nine-valued connector operation which will be defined as an Interconnect Module (IM). The fault simulation algorithm for one reverse level of switches is as follows:

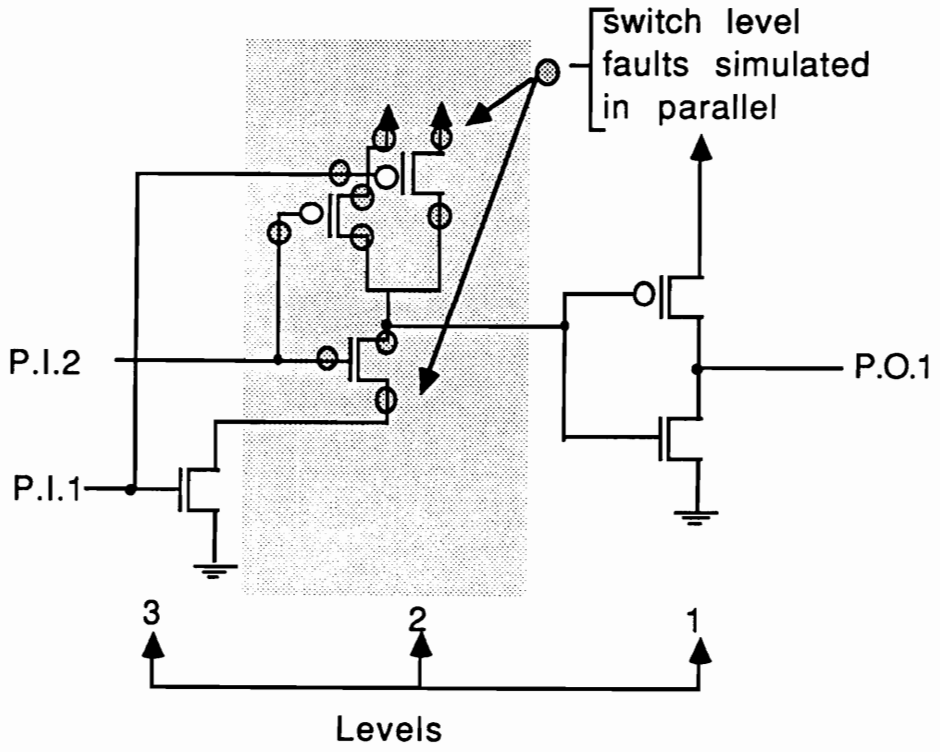


FIGURE 5.1 Two - Dimensional faults on CMOS AND Gate.

Algorithm 5.1

1. Label all switches for the simulation level as 1 to W and apply the test vector to all inputs.
2. Label all connected output lines from 1 to W/2 (W/2 is maximum)
3. Using the parallel processing elements from PHAFS, fault simulate all switches in parallel.
4. Resolve all output line values by performing the CONNECTOR(*) function on all connected output lines in parallel using the Interconnect Modules (IMs) from PHAFS in parallel during one simulation cycle.
5. Compare the resolved output lines with the parallel fault results to determine potentially detected faults for this level.

5.3 Definitions and Theorems

To show how to propagate the fault effects in one level for the fault simulation algorithm, which is given in section 5.4, some preliminary definitions and theorems are necessary. These definitions and theorems involve implication of faults for one test vector.

The *implication* or justification relation is well understood as applied at the gate level [Prad90]. At the switch level this *implication* has also been studied [Light82]. For the switch, implication or justification is defined as the set of inputs {D,G} required in order that the output S achieve a specific value. For example, to imply a logic low on the output S of an N - type switch input values of logic zero and one would be required on the drain and gate inputs, respectively. For a node or one reverse level of MOS switches, the implication relation is used to state some preliminary definitions concerning the propagation of the fault set simulated.

Definition of Symbols

For the definitions, theorems, algorithms, and complexity measure described in this chapter, the following symbols notations are used:

C - any feedback-free circuit

V - any test vector set

t - one test vector of **V**

F - stuck-at-1/0 fault set

f(D, G, S) - arbitrary fault in drain, gate, or source

OUT - primary output response of fault - free circuit

I - any line in circuit **C**

P - the set of all paths from **I** to **OUT**

p - shortest path in **P**

i - one level of transistors from reverse level ordering of **C**

W - the maximum number of transistors in any level **i**

j - one level of transistors from reverse level order

L - the number of levels from reverse level ordering of **C**

F' - test equivalent fault set

a - undetected fault dropping activity factor

b - detected fault dropping activity factor

g - subset of **F'** faults that propagate to primary output for test **t**

n - the number of transistors in **C**

Definition 5.1 A fan-in line is defined as any line that is connected to a node which has more than one input source.

Definition 5.2 A fan-out line is defined as any line that is connected to a node which has one and only one input source.

Property 5.1 A single line stuck-at fault f(D,G) on the switch input propagates to the switch output (S) if and only if f(D,G) implies NOT(S).

Property 5.2 A single line stuck-at fault f(S) on one switch output propagates to a fan-in line L if and only if f(S) implies NOT(L).

Property 5.3 A single line stuck-at fault $f_n(L)$ propagates through fan-out lines to the next level set of outputs $f_{n+1}(S)$ if and only if $f_n(L)$ implies NOT($f_{n+1}(S)$).

Figures 5.2 A, B, and C show examples of properties 5.1 through 5.3. For the single line stuck-at fault of the switch, properties 5.1 and 5.2 involve being able to propagate the fault forward toward the primary output through switches and fan-in nodes. Propagation of faults through the switch is shown in Figure 5.2A, while propagation of faults through the fan-in node is shown in Figure 5.2B. Unlike the gate level, the switch level can have fan-in points. The resulting value at these points is determined with the CONNECTOR function. The fault propagates through the fan-out lines in a similar manner as any good value, i.e., the value of the fault is duplicated for each fan-out line as given in property 5.3 and shown in Figure 5.2C.

Theorem 5.1 Let C be any feedback-free circuit, V be any test vector set, and F be the fault set consisting of switch stuck-at-1/0 and line stuck-at-1/0 faults. An arbitrary fault f which is an element of the fault set F is detected if and only if f propagates to the primary output for at least one test t which is an element of V.

Proof

=> Assume f is detected and the fault does not propagate to the primary output on any test t, then the circuit with the fault is indistinguishable from the circuit without the fault. This is a contradiction, thus, the fault must propagate to primary output on at least one test t.

<= Assume f propagates to a primary output for at least one test t and the fault is undetected. Then, by the definition of propagation for the fault set F given earlier, the primary output must be NOT(OUT). This is a contradiction, and f must be detected.

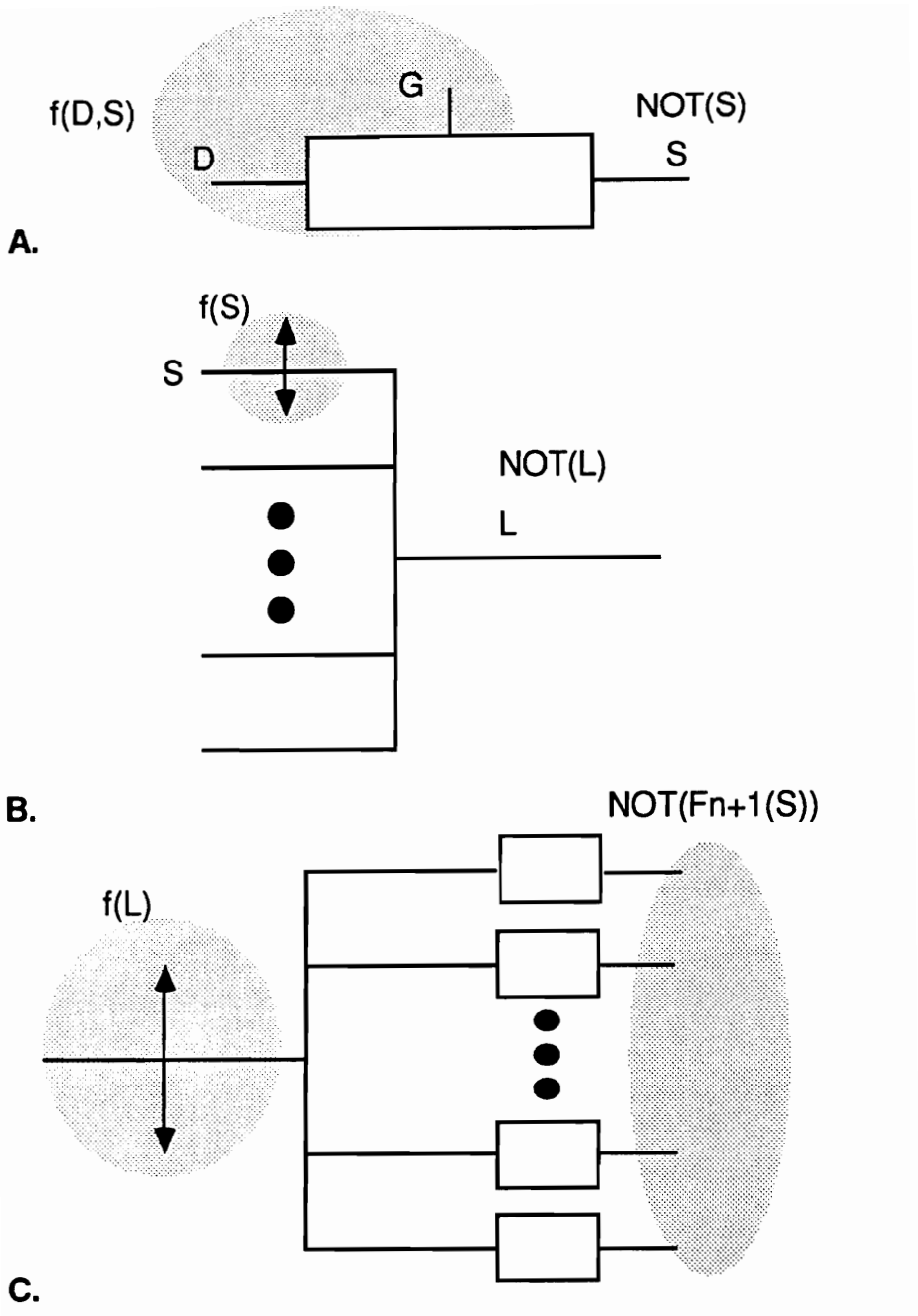


FIGURE 5.2. Fault Propagation. A. Switch. B. Fan-in node. C. Fan-out node.

Definition 5.3 Let C be an arbitrary circuit. Then the level of line l is defined as the minimum number of transistors traversed from l to any primary output.

Definition 5.4 If an arbitrary fault f propagates to the next level line(s) l-1 as value 'a' for test t, then the fault(s) stuck-at 'a' on level l-1 is (are) test equivalent to f for the test t. Furthermore, the fault f is said to be potentially detected.

Theorem 5.2 A stuck-at fault f on a line in level l in circuit C is detected if

(1) f propagates to the next level l-1 for at least one test t which is an element of V ,

and

(2) the fan-out test equivalent fault to f on level l-1 is detected for that same test t, if such a fan-out is present. If no such fan-out is present, then the single test equivalent fault to f on level l-1 is detected for that same test t.

Proof - By Theorem 5.1, if f does not propagate to the next level or primary output, then f is undetected. Since f does in fact propagate for that test t, there exists at least one test equivalent fault to f at level l-1 and possibly more if fan-out occurs.

Case 1 - There is only one test equivalent fault at the next level l-1, call it 'a' . Assume 'a' to be detected for test t, then the primary output is NOT(OUT) in the presents of fault 'a' at test t. Now, assume that the fault f is undetected at test t, then

the primary output is unaffected as OUT. This is a contradiction, and f must therefore be detected.

Case 2 - There is more than one test equivalent fault at the next level l-1. This implies that the line must have fan-out. If the test equivalent fault before fan-out is detected, then the same argument as in Case 1 applies.

Definition 5.5 One-dimensional parallel switch level simulation is defined as switch simulation of one reverse level of switches in parallel.

5.4 Parallel Fault Simulation Algorithm

The algorithm presented here is for all faults in the fault set to be determined as detected or undetected for an arbitrary feedback-free switch level circuit and test vector set. To be described below, Theorem 5.3 states the upper-bound on computational complexity for the algorithm. Two types of fault simulation techniques are used in the algorithm. First, two dimensional parallel fault simulation is used to fault simulate the switch line stuck-at faults. Here, both switches and stuck-at faults are simulated in parallel with PHAFS as discussed in Algorithm 5.1. The second fault simulation type, concurrent fault simulation, together with parallel device (switch) simulation is used to propagate faults to the primary output for detection or to drop from further processing any undetected faults occurring before the output.

Theorem 5.3 All single line stuck-at faults at fan-out points f in circuit C are known to be detected or undetected for a given test set V in computational complexity

$O(|V|W(L^2))$, where V is the number of test vectors, L is the number of levels from reverse level ordering, and W is the maximum number of switches in any level.

Proof - Using the definitions, theorems, and switch level fault simulation technique given previously, the proof of the complexity in Theorem 5.3 is shown by the parallel fault simulation algorithm for an arbitrary circuit which is given now.

Assumption 5.1: Special purpose hardware will perform one-dimensional parallel switch level simulation. The parallelism will be in device (switch) simulation. That is, the hardware will simulate one reverse level of switches (transistors) in parallel.

Assumption 5.2: Hardware will perform two-dimensional switch level fault simulation. The hardware will simulate all line stuck-at faults and all switches for one level in parallel.

Algorithm 5.2.

1. Let i be the level for parallel simulation and initialize as i = L, i.e., start at level L.
2. Parallel device simulate one test vector from primary input (level L) through level i+1 using PHAFS. Use these outputs as inputs for level i.
3. Two-Dimensional parallel fault simulate level i using PHAFS and find all potentially detected faults using Algorithm 5.1.
4. Determine the output test equivalent fault set W for the potentially detected faults in step 3. Let j = i-1 (the next level closest to output).

5. Inject W faults and parallel device simulate using PHAFS the test vector and injected faults through level j.
6. Compare the outputs at level j for the test vector and injected faulty vectors. Injected faults are undetected if no difference and thus drop fault from this simulation test vector pass.
7. Repeat steps 4, 5, and 6 until j = 0 (the primary output is reached) then proceed to step 8.
8. Let $w \leq W$ be the remaining injected faults, then these w faults along with the potentially detected test equivalent faults from step 3 are detected. Drop these faults from further test vector simulation.
9. Let $i = i-1$. If $i \neq 0$ then go to step 2.
10. Stop if no more test vectors, otherwise use next test vector and go to step 1.

5.5 Parallel Fault Simulation Complexity

Using the symbols given in Section 5.3, the complexity for the steps in the algorithm is as follows:

The complexity in step 2 is determined by simulation of each reverse level from level L to level l + 1 as

$$O(L-i-1). \qquad (5.1)$$

The complexity for the parallel fault simulation in step 3 is one simulation cycle or

$$O(1). \qquad (5.2)$$

Step 4, potentially detected faults are determined during the parallel fault simulation step 3 in hardware by the PHAFS PE and thus the complexity is considered part of Equation 5.2. The complexity in step 5. for simulation of one good test vector and W potentially detected faults through the level i - 1 is

$$O(1+ W). \qquad (5.3)$$

The complexity adjustment in step 6. for concurrent fault simulation or fault dropping of faults that are undetected for this test vector is

$$W = W/a. \qquad (5.4)$$

The complexity required in step 7. for propagation of one good test vector and all remaining not dropped faults to the output is determined by substituting equation 5.4 into equation 5.3 and then multiplying by i or

$$O((1+W/a)i). \qquad (5.5)$$

The complexity adjustment in step 8. for fault dropping of faults that are detected is

$$W = W/b. \qquad (5.6)$$

The complexity for fault simulation of all reverse level of faults is the loop of step 9. as

$$\sum_{i=0}^{i=L} (\text{steps } 2, 3, 7, 8), \qquad (5.7)$$

The complexity for fault simulation of all test vectors is the loop of step 10 as

$$\sum_{v=0}^{v=|V|} (\text{step } 9) \qquad (5.8)$$

Substituting complexities and fault dropping of Equations 5.1 through 5.7 into the loop of Equation 5.8 yields

$$O() = \sum_{v=0}^{v=|V|} \sum_{i=0}^{i=L} [(L-i-1) + 1 + (1+ W/ab)i]. \qquad (5.9)$$

Simplifying Equation 5.9 gives

$$O() = |V| \sum_{i=0}^{i=L} (L+ Wi/ab). \qquad (5.10)$$

Expanding summation of Equation 5.10 gives

$$O() = |V| (L^2+ WL(L+1)/2ab). \qquad (5.11)$$

Further simplifying of Equation 5.11 yields

$$O() = |V|(L^2)W/2ab + |V|LW/2ab + |V|(L^2), \qquad (5.12)$$

where |V| is the number of test vectors, L is the number of reverse levels of transistors, W is the maximum number of transistors in a level, "a" is the activity factor, and "b" is the fault dropping factor. Overhead for the algorithm includes reverse level ordering of the circuit, and programming of the PHAFS processors. Reverse level ordering has computational complexity on the order of O(n) where n is the number of transistors in the circuit, while the complexity of programming the special purpose hardware is of O(L). Both of these overheads have complexities that

are much lower than the earlier simulation computational complexities since they are independent of the size of the test vector set |V|.

5.6 Summary

While Chapter 3 introduced a novel switch level extension to parallel fault simulation and Chapter 4 presented the circuit partitioning necessary to fault simulate single levels of switches in parallel, this chapter has given the algorithm for all faults to be determined detected of undetected for an arbitrary test set. Also shown in this chapter was the algorithm's upper bound on computational complexity which is on order of L^2, where L is the number of reverse levels in the circuit. This bound on complexity is much less than the traditional fault simulation techniques.

Chapter 6

Results on Complexity, Load, and Partition Size

6.1 Introduction

The purpose of the work described in this chapter is to determine how much hardware, as measured in processing elements, is cost effective for performing hardware accelerated fault simulation. Measures of cost effectiveness could include usage or load, and speed or computational complexity. Two factors effecting these measures are circuit characteristics and the maximum number of processor elements in a partition. Section 6.2 contains a study of circuit characteristics of benchmark circuits. Section 6.3 derives a complexity increase formula describing the situation when a fixed maximum number of processor elements is available to simulate a partition. Finally, Section 6.4 compares the complexity and load versus partition size for the benchmark circuits.

6.2 Reverse Level, Group, and Subgroup Average and Maximum Sizes

A switch level implementation of the ISCAS85 combinational benchmark circuits [Kozm91], which range in size from 26 to 8854 transistors, have been partitioned

and studied. Detailed reverse level order sizes versus level for each of the benchmark circuits is given in Figures 6.1 through 6.11. Load, plotted on the y-axis, is defined as the width of the level in transistors divided by the maximum width of any level in the circuit. Maximum width of the largest reverse level is also shown in the box. The maximum width of the largest reverse level in a circuit is important because if an entire reverse level is to be simulated in parallel, then the number of PEs required for simulation would be equal to this maximum width. Further, if the number of PEs available is equal to the maximum width, then simulation of reverse levels that have fewer transistors would mean that PEs would sit idle. The load or PE usage, then, is an important measure because idle PE would mean wasted hardware.

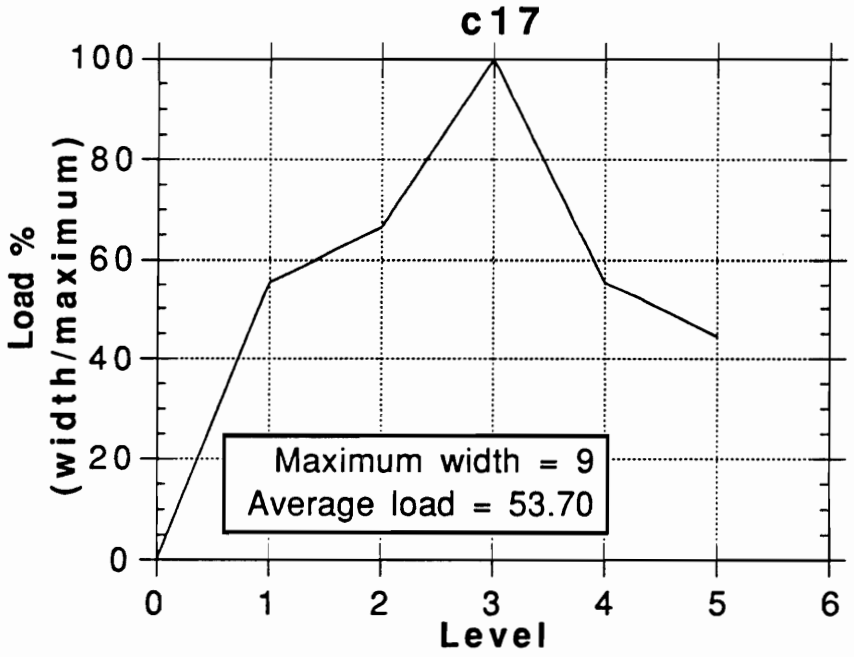


FIGURE 6.1 Circuit C17 reverse level order sizes.

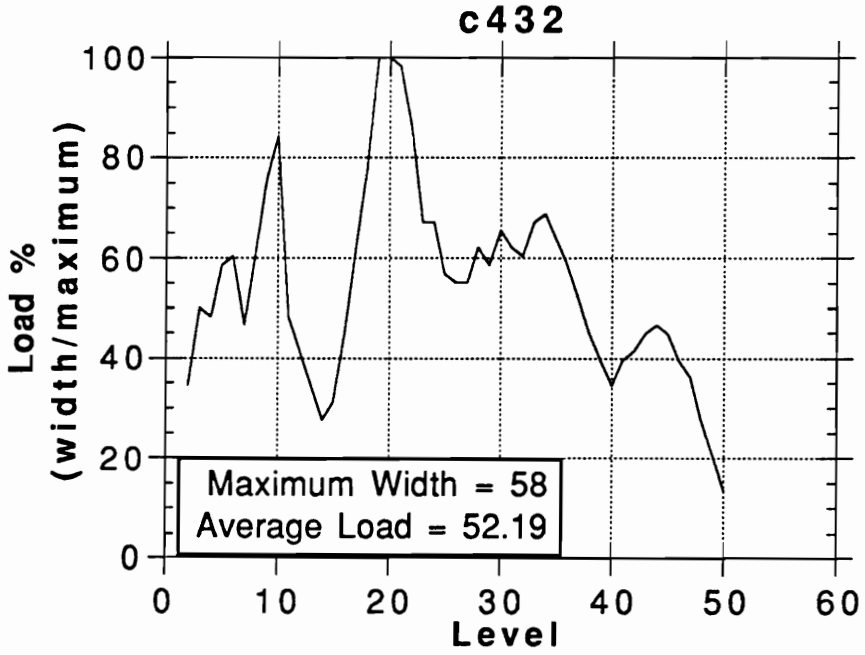


FIGURE 6.2 Circuit C432 reverse level order sizes.

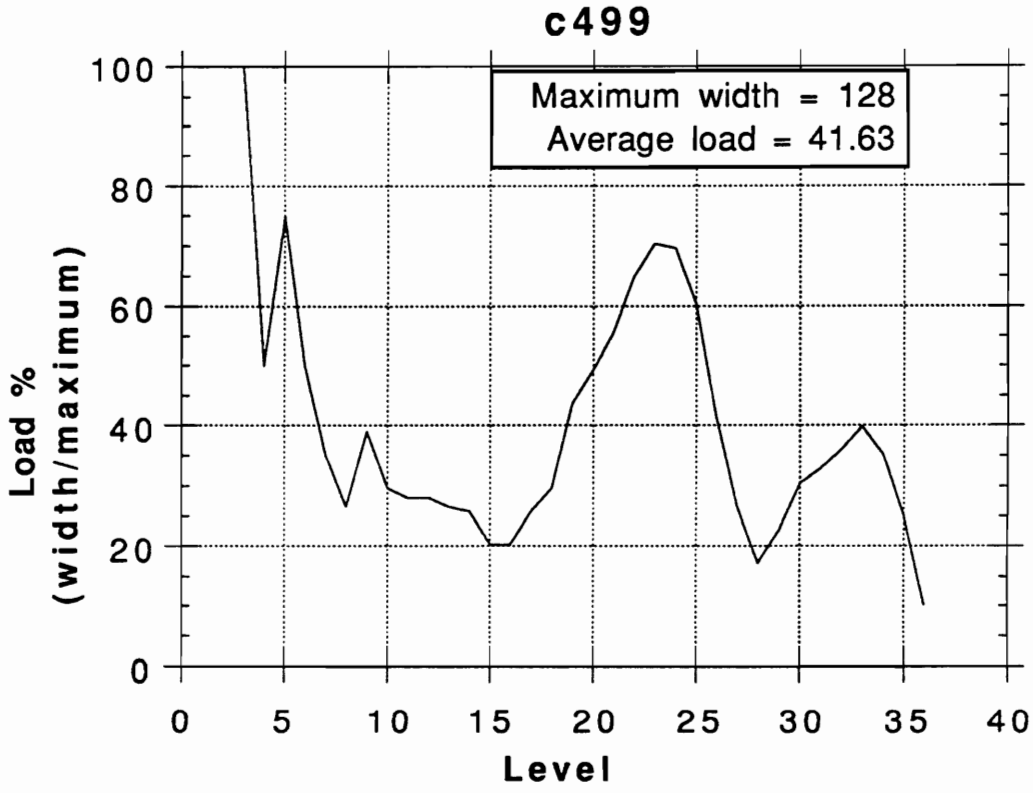


FIGURE 6.3 Circuit C499 reverse level order sizes.

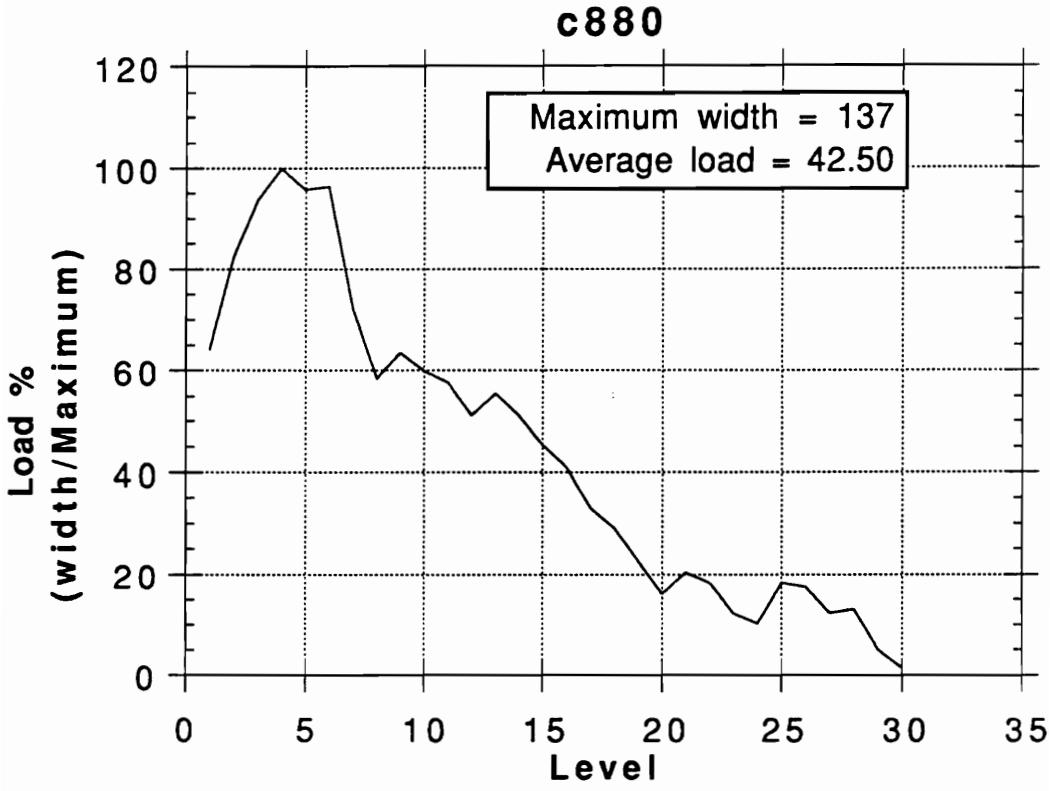


FIGURE 6.4 Circuit C880 reverse level order sizes.

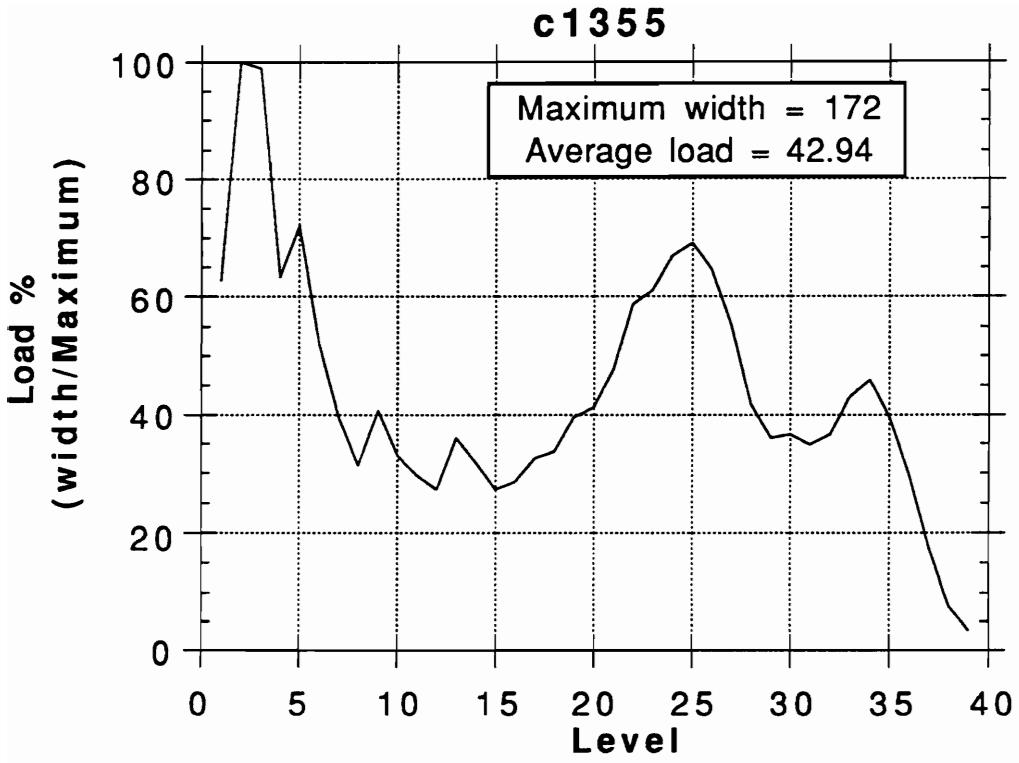


FIGURE 6.5 Circuit C1355 reverse level order sizes.

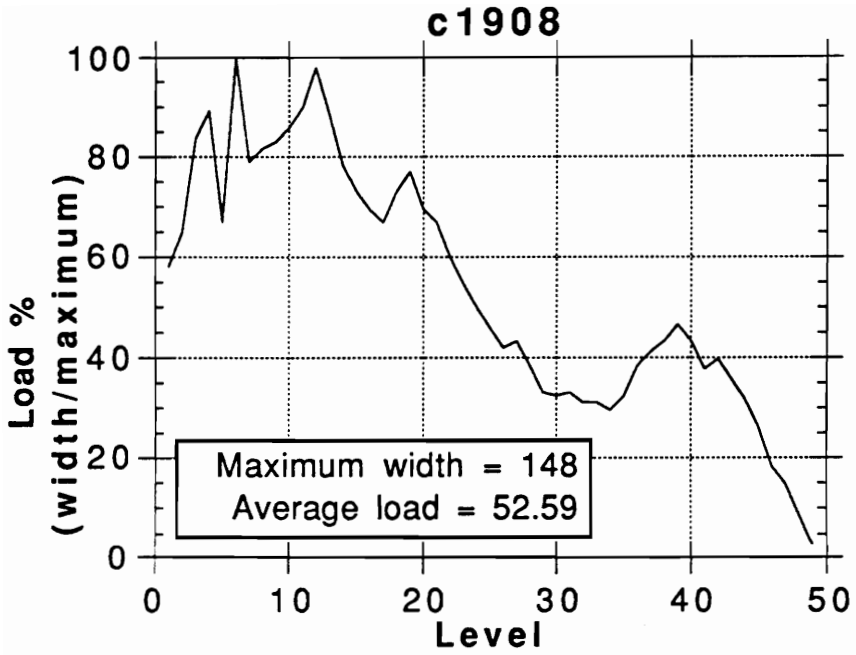


FIGURE 6.6 Circuit C1908 reverse level order sizes.

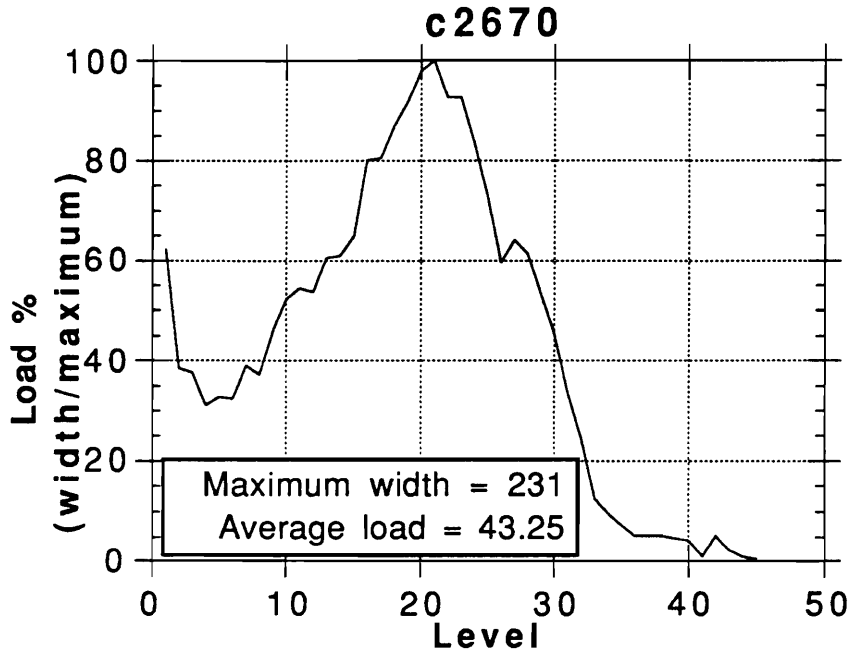


FIGURE 6.7 Circuit C2670 reverse level order sizes.

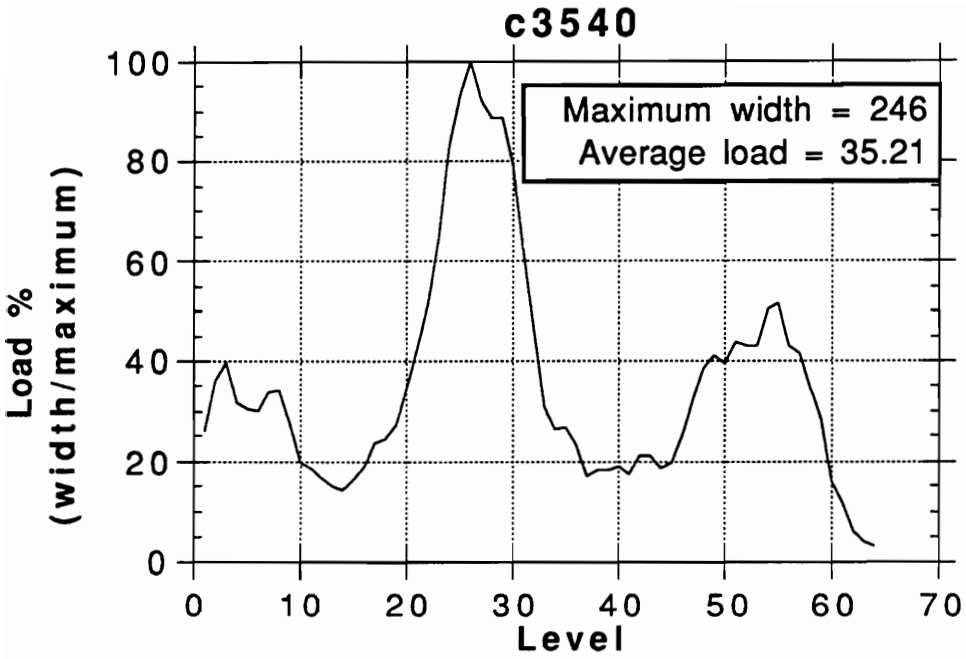


FIGURE 6.8 Circuit C3540 reverse level order sizes.

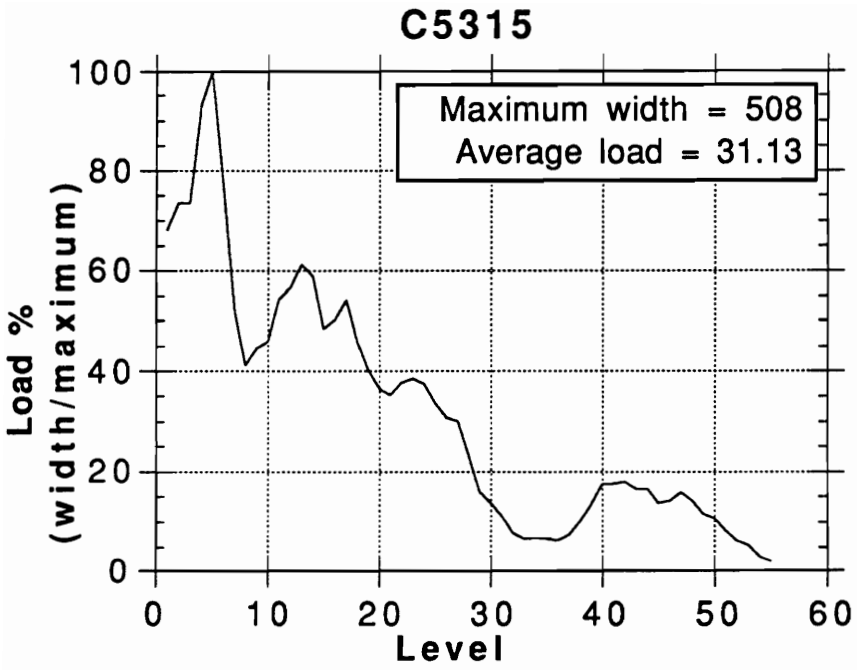


FIGURE 6.9 Circuit C5315 reverse level order sizes.

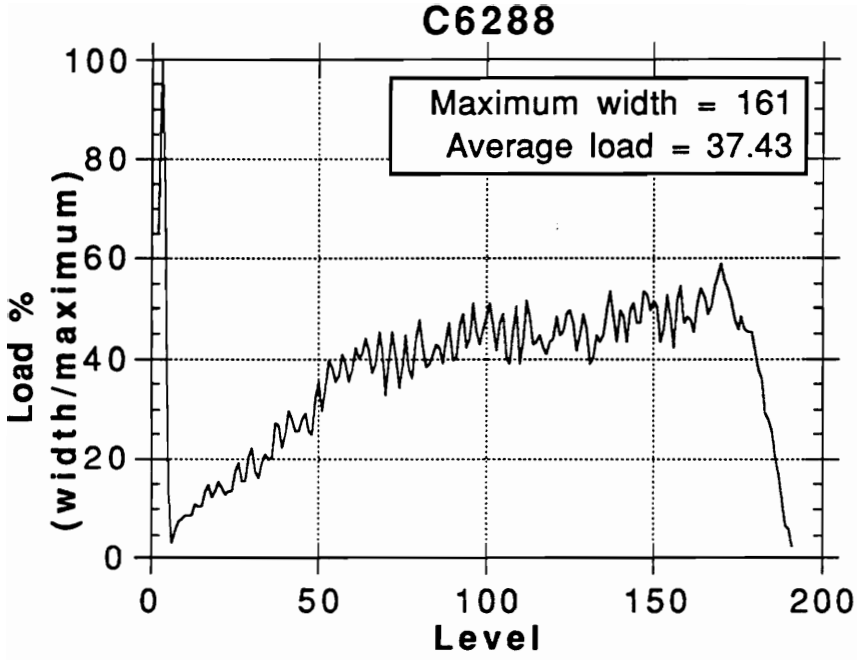


FIGURE 6.10 Circuit C6288 reverse level order sizes.

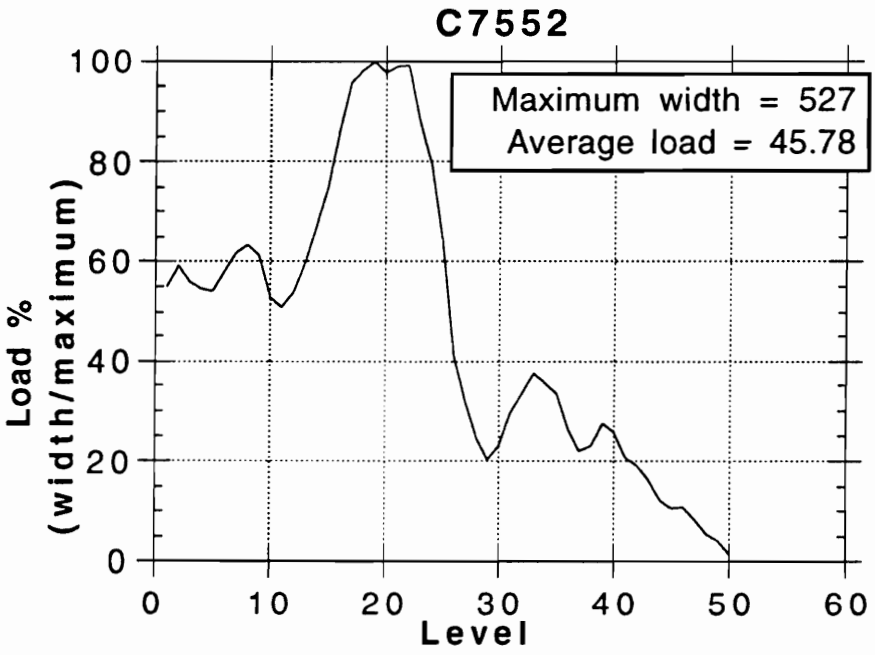


FIGURE 6.11 Circuit C7552 reverse level order sizes.

The average reverse level size has been calculated and is plotted versus circuit size as shown in Figure 6.12. The average group and subgroup size for the ISCAS85 circuits is

much smaller than the average reverse level size as shown in Figure 6.13. Each data point in Figure 6.13 represents a single circuit. This result indicates that most transistors in a reverse level are disjoint from the majority of transistors of that same level. Disjoint transistors in a reverse level do not depend on each others values. Thus, communications between a transistor a reverse level and the majority of the transistors in the same reverse level is unnecessary.

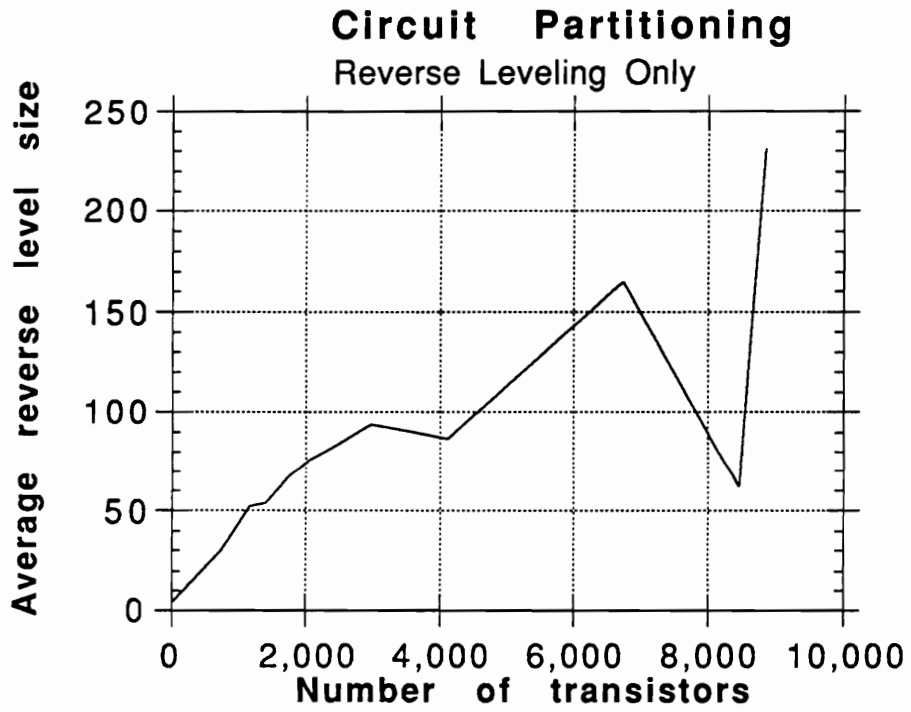


FIGURE 6.12 Average reverse level order sizes.

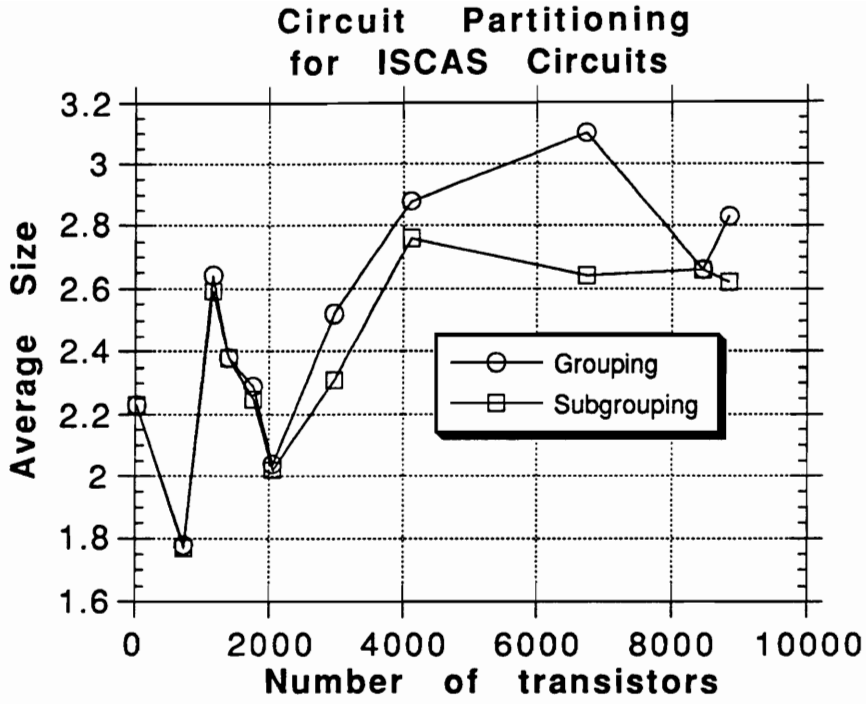


FIGURE 6.13 Average group and subgroup sizes.

The maximum reverse level size versus circuit size is shown in Figure 6.14. The largest reverse level size of all benchmark circuits is 527 (the maximum width of circuit C7552). This result on reverse level maximum size indicates that, in order to simulate one level of transistors in parallel for all of these circuits, 527 processing elements would be required, i.e. a maximum partition size of 527 is necessary. However, if 527 PEs were available, most PEs would sit idle while simulating other levels and circuits. The maximum group and subgroup sizes are also shown in Figure 6.14. The notation + grouping and + subgrouping in the legend of Figure 6.14 refer to grouping and subgrouping in addition to reverse level ordering partitioning. The largest group size of all the circuits is 140 (the maximum group size of circuit

C7552). The largest subgroup size of all the circuits is 30 (the maximum subgroup size of a number of circuits). This result on the subgroup maximum size indicates that, if the maximum partition size is defined to be larger than 30 (the maximum subgroup size) then communications between partition within a reverse level would be unnecessary. This is true because each subgroup would fit into at most one partitions and since subgroup are defined to be disjoint in output no fan-in (communications) would occur between partitions.

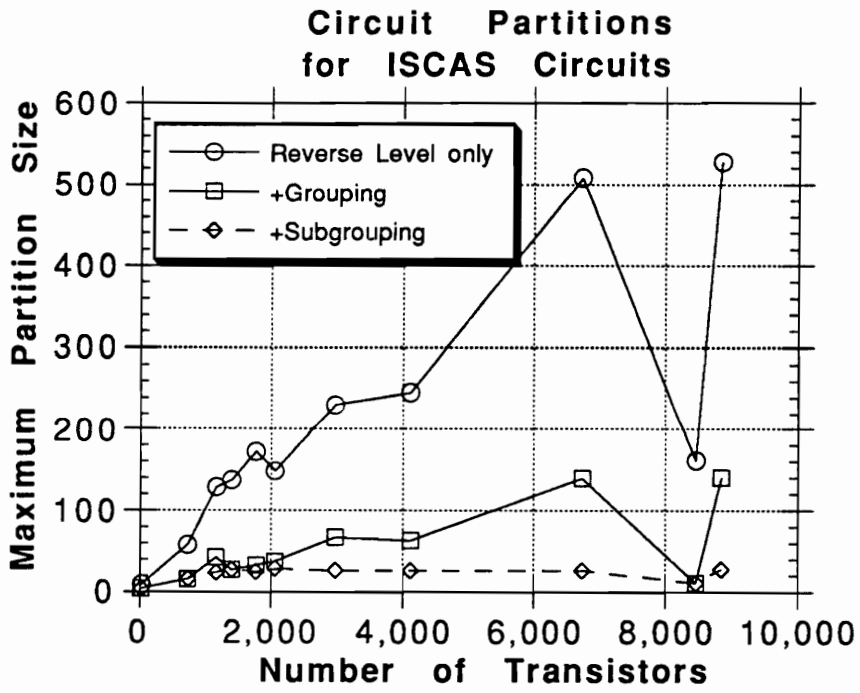


FIGURE 6.14 Maximum reverse level, group, and subgroup Sizes.

6.3 Complexity with Fixed Partition Size Included

From an examination of the experimental results generated for reverse level, group, and subgroup sizes for the ISCAS85 benchmark circuits, it is clear that most transistors in a reverse level are disjoint from the majority of transistors in that same level. Disjoint transistors in a reverse level do not depend on each others values. Thus, communications between a transistor a reverse level and the majority of the transistors in the same reverse level is unnecessary. If the maximum partition size is defined to be larger than the maximum subgroup size, then communications between partition within a reverse level would be unnecessary. This is true because each subgroup would fit into at most one partitions and, since subgroup are defined to be disjoint in output, no fan-in (communications) would occur between partitions.

The complexity measure given in Chapter 5 was calculated while assuming that the maximum partition size to be equal to the maximum number of transistors (width) of any reverse level in the circuit. However, the partition size is limited to the number of PEs on the PHAFS board which is limited by cost. For this purpose, the partition will be limited and its maximum size will be defined as p.

As an example of partition sizes equal to the maximum reverse level size, consider the ISCAS85 circuit C17 as shown in Figure 6.15. The N and P boxes represent N- and P-type switches, while the circle represents fan-in at a node. Also, for simplicity, Vdd and ground connections are not shown. For the complexity calculation given in Section 5.4, it is assumed that any reverse level can be fault simulated in parallel. Then, for this the example, the maximum number of the PEs required would be equal to the

maximum number of transistors in the largest level. In C17, level 3 has the maximum number of transistors equal to 8. This means that a maximum hardware partition size of 8 would need to be available for the complexity calculation of Equation 5.12 to hold true.

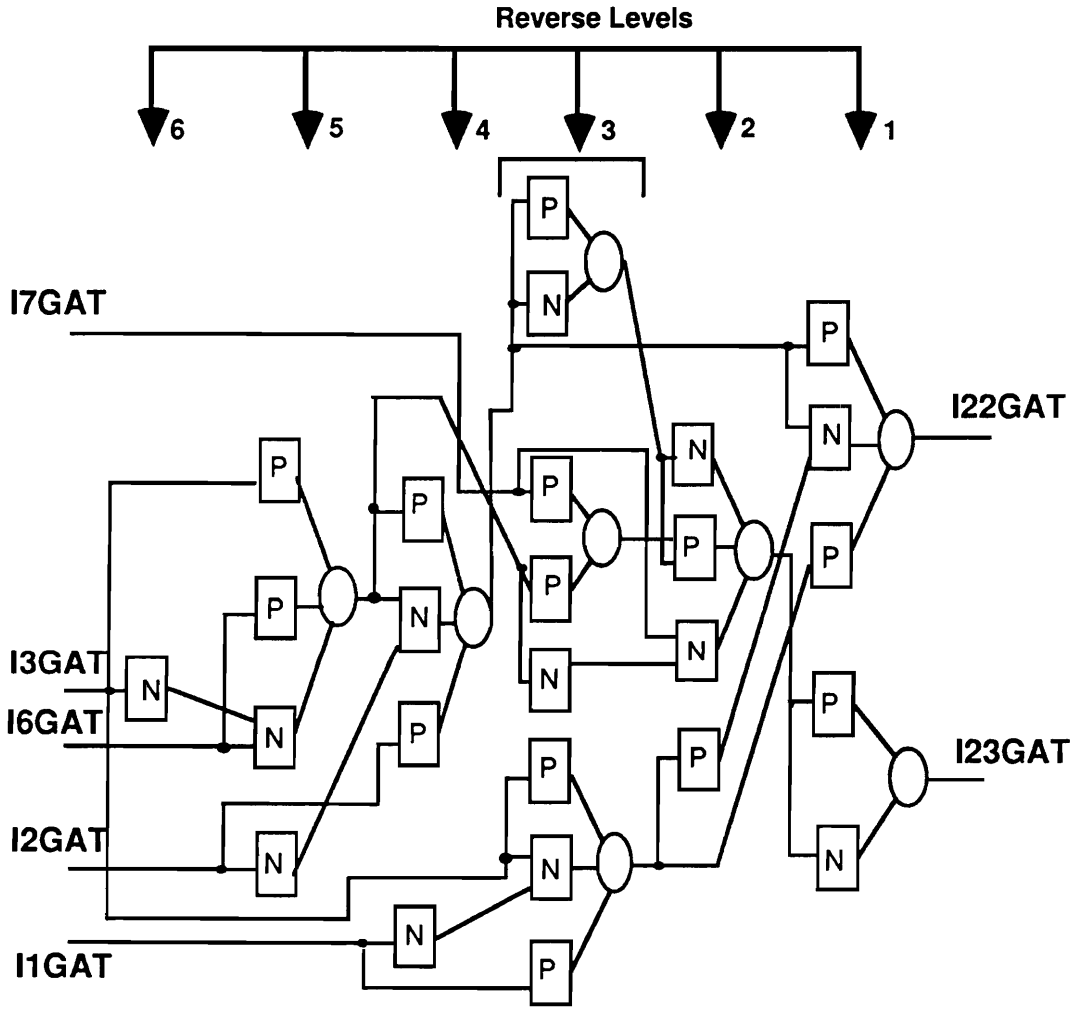


FIGURE 6.15 C17 reverse level ordered with no fixed partition size.

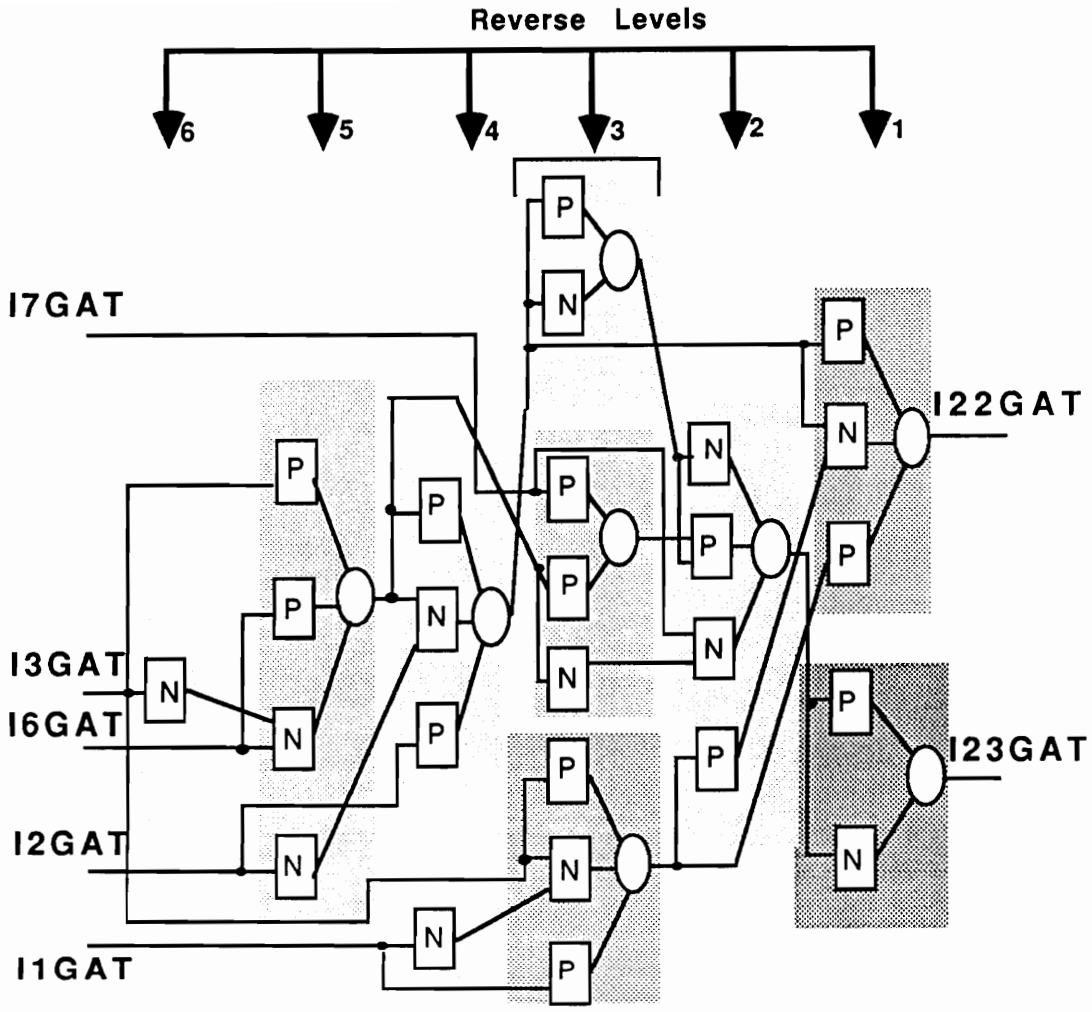


FIGURE 6.16 C17 reverse level ordered with fixed partition size equal 4.

Now, consider that a limited amount of hardware is available and that the fixed hardware partition size is less than the maximum reverse level size. This means that one or more reverse levels must be divided as shown in Figure 6.16. The separate

partitions are shown in the gray boxes. For this example, the maximum partition size is 4 PEs. This means that a maximum of 4 transistors can be mapped to a partition and the maximum partition size must be less than or equal to 4. Notice that, partitions can use fewer PEs than the maximum value of 4. This is dependent on the circuit topology. Furthermore, with respect to a reverse level, each partition can be simulated separately because the partitions are disjoint and their simulated values do not depend on one another. Therefore, the average number of partitions per level, defined as K, can be used for complexity calculations, i.e., simulation of one reverse level will require on average K simulation cycles.

The complexity for the steps in Algorithm 5.2, when the average number of partitions per level K is considered, is then:

The complexity in step 2 is determined by simulation of each reverse level, which on average takes K simulation cycles, from level L to level I + 1 as

$$O(K(L-i-1)). \qquad (6.1)$$

The complexity for the parallel fault simulation in step 3 takes on average K simulation cycles or

$$O(K). \qquad (6.2)$$

The complexity in step 5. for simulation of one good test vector and W potentially detected faults through the level i - 1 is

$$O(K(1+ W)). \qquad (6.3)$$

The complexity adjustment in step 6. for concurrent fault simulation or fault dropping of faults that are undetected for this test vector is

$$W = W/a. \qquad (6.4)$$

The complexity required in step 7. for propagation of one good test vector and all remaining not dropped faults to the output is determined by substituting equation 5.4 into equation 5.3 and then multiplying by i or

$$O(K(1+W_{/a})i). \qquad (6.5)$$

The complexity adjustment in step 8. for fault dropping of faults that are detected is

$$W = W_{/b}. \qquad (6.6)$$

The complexity for fault simulation of all reverse level of faults is the loop of step 9. as

$$\sum_{i=0}^{i=L} (\text{steps } 2, 3, 7, 8), \qquad (6.7)$$

The complexity for fault simulation of all test vectors is the loop of step 10 as

$$\sum_{v=0}^{v=|V|} (\text{step } 9) \qquad (6.8)$$

Substituting complexities and fault dropping of Equations 6.1 through 6.7 into the loop of Equation 6.8 and simplifying yields

$$O() = K(VL^2W_{/2ab} + VLW_{/2ab} + VL^2), \qquad (6.9)$$

or just

$$O()_{NEW} = K^*O()_{OLD}, \qquad (6.10)$$

where $O()_{OLD}$ is the complexity as given in Equation 5.12 while assuming a partition size equal to the reverse level size, and $O()_{NEW}$ is the complexity while assuming a fixed partition size equal to or larger than the maximum subgroup size. As shown in Equation 6.10, the complexity increase due to the fixed partition size is linearly

proportional to the average number of partitions per level K. A comparison of these worst case computational complexity with other fault simulation techniques is shown in Table 6.1. When accounting for fixed partition size, if K values are small, the complexity KL^2 is still much smaller than the complexity of the other traditional techniques. K values for the benchmark circuits will be discussed next.

TABLE 6.1 Fault simulation complexity.

Name	Type	Level	Technique	Complexity
parallel	software	gate	parallel	$O(n^3)$
concurrent	software	gate	concurrent	$O(n^2)$
deductive	software	gate	deductive	$O(n^2)$
FMOSSIM	software	switch	concurrent	$O(n^2)$
MOZART	software	hierarchical	concurrent	$O(n^2/k)$
CHIEFS	distributed	hierarchical	concurrent	O(nlog(n))
PHAFS	accelerator	switch	parallel	$O(KL^2)$

6.4 Processor Load and Complexity Increase versus Partition Size

Experiments examining processor load and increased complexity versus different partition sizes for switch level implementations of the ISCAS85 circuits were performed. Processor load versus partition size for benchmark circuits is important because extra PEs that are seldom used are not cost effective. Complexity increase versus partition size is important when comparing this proposed parallel fault

simulation technique with other fault simulation implementations. Again, for the purposes of this discussion, a processor is a simple single processor element on the PHAFS board.

For processor load experiments, the ISCAS85 circuits were studied for partition sizes of 32, 64, 128, 256, and 512 PEs. The idea was to partition the reverse levels of a circuit with partition sizes as close to the maximum partition size as possible. The load is then defined as the average number of PEs used per partition divided by the maximum partition size, or

$$\text{Load} = (1/l * \text{maximum partition size}) \sum_{i=1}^{i=l} (\#\text{ of PEs in partition } i), \quad (6.11)$$

where i is the current partition and l is the total number of partitions.

As an example, consider C17 with partition size of 4 as in Figure 6.16. There are a total of 9 partitions that have sizes 2, 3, 4, 2, 3, 3, 4, 4, 1. The sum of these values is 26, while the average partition size is 26/9 or 2.89. Then the average load is just 2.89/4.00 or .72. The value .72 is then just the load or average usage of the partition.

The processor element load for different partition sizes on the reversed level ordered, grouped, subgrouped ISCAS85 circuits was studied. The results, as shown in Figure 6.17 and Table 6.2, indicate that the load seems to reach a limit as circuit size grows.

TABLE 6.2 Processor element load versus partition size for ISCAS85 circuits.

Circuit Size (switches)	Partition Size (PE's) 32	Partition Size (PE's) 64	Partition Size (PE's) 128	Partition Size (PE's) 256	Partition Size (PE's) 512
26	0.18125	0.090600	0.045300	0.022700	0.011300
728	0.63640	0.48220	0.24110	0.12060	0.060300
1396	0.73280	0.67110	0.42800	0.21400	0.10700
1164	0.74440	0.57850	0.42350	0.23500	0.11750
1769	0.76980	0.72380	0.56190	0.29590	0.14790
2058	0.78712	0.74100	0.56130	0.31030	0.15510
2974	0.78260	0.68620	0.56640	0.39900	0.19950
4122	0.77200	0.72400	0.59670	0.34360	0.17180
6734	0.84510	0.81070	0.69780	0.51330	0.31450
8464	0.73940	0.58120	0.47010	0.23660	0.11830
8854	0.87040	0.87300	0.76110	0.60320	0.43650

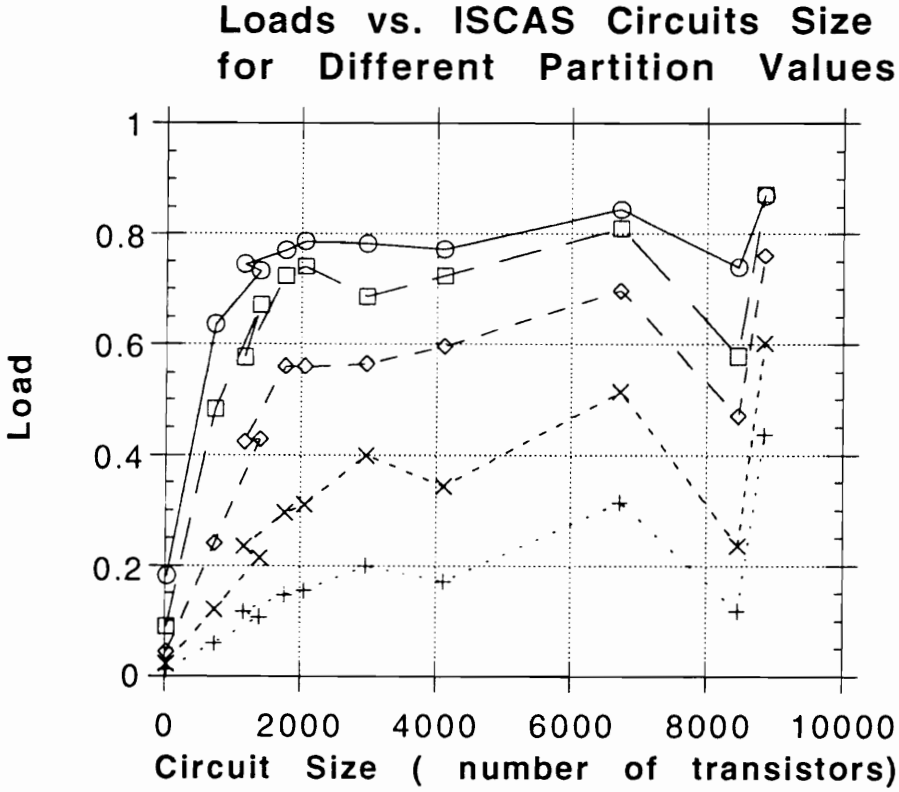


FIGURE 6.17 Processor element load versus partition size for ISCAS85 circuits.

Considering a maximum partition size, the complexity increase K from Equation 6.10 is the average number of partitions per level in the circuit. The complexity increase because of partitioning was studied using the ISCAS85 circuits for maximum partition sizes of 32, 64, 128, 256, and 512 . The average number of partitions per level was calculated by dividing the total number of partitions required for the circuit by the total number of levels in the circuit. For the example circuit C17 and maximum partition size of 4 that is shown in Figure 6.16, the number of partitions required would be 9, while the number of levels is 6. Thus the complexity increase is 9/6 or 1.5. This result means that it would take 1.5 times the complexity as calculated in Equation 5.12 to simulated this circuit.

As shown in Figure 6.18 and Table 6.3, complexity increase K, versus partition size for the reverse level ordered ISCAS85 circuits was also studied. Results show that for the smallest partition size of 32, the complexity increases less than a multiple of 8 for all circuits. Over the size range for the benchmark circuits (26 to 8854 transistors), a complexity increase of at most 8 or $8*(L^2)$ is still much less than the complexity for fault simulation using traditional techniques.

TABLE 6.3 Complexity increase versus partition size for ISCAS85 circuits.

Circuit Size	Partition Size 32	Partition Size 64	Partition Size 128	Partition Size 256	Partition Size 512
26	1.0000	1.0000	1.0000	1.0000	1.0000
728	1.4510	1.0000	1.0000	1.0000	1.0000
1396	2.1667	1.2500	1.0000	1.0000	1.0000
1164	2.2333	1.5330	1.1000	1.0000	1.0000
1769	2.7179	1.5897	1.0512	1.0000	1.0000
2058	2.8571	1.6326	1.1020	1.0000	1.0000
2974	3.6000	2.2000	1.3778	1.0000	1.0000
4122	3.1094	1.8281	1.1406	1.0000	1.0000
6734	5.2000	2.9273	1.7636	1.2181	1.0000
8464	2.3560	1.5759	1.0052	1.0000	1.0000
8854	7.7000	4.1600	2.4600	1.5800	1.1000

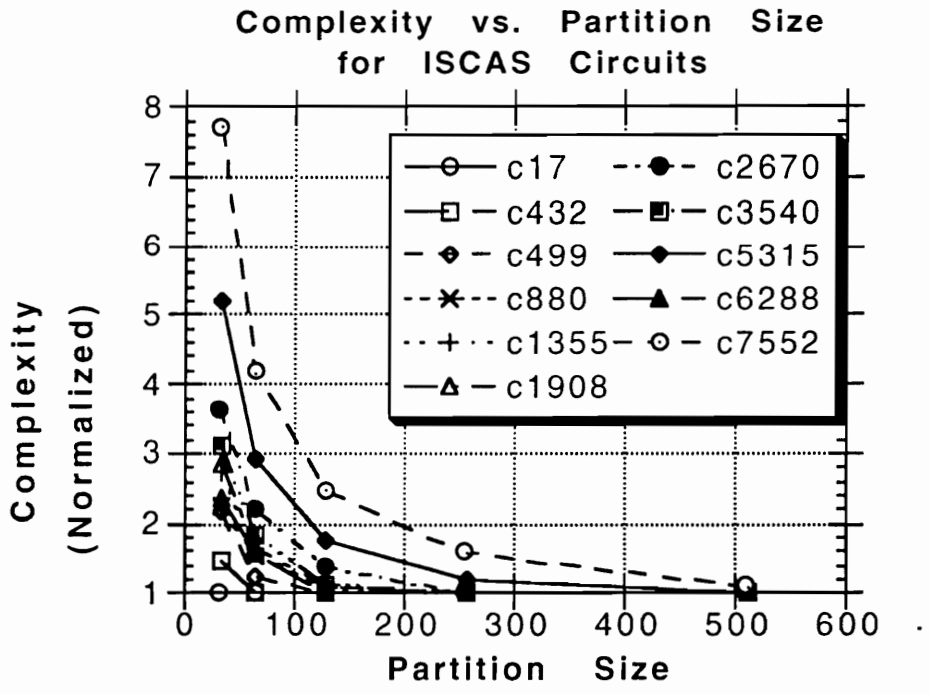


FIGURE 6.18 Complexity increase versus partition size for ISCAS85 circuits.

Chapter 7

Parallel Fault Simulation Algorithm/Complexity Verification

7.1 Introduction

The switch level parallel fault simulation technique, reverse level order circuit partitioning, and the fault simulation algorithm and complexity, given in Chapters 3, 4, and 5, respectively, were verified using C and VHDL implementations. This chapter describes the verification and gives results on measured complexity and parallel speed up for switch level fault simulation. A block diagram of the verification technique is shown in Figure 7.1.

The simulator process is accomplished by combining the circuit reverse level order partitioning and the compiled code simulator. First, a circuit is reverse level ordered. Next, the reverse level ordered switch Netlist is compiled to VHDL code as PEs. Finally, fault simulation is performed using the controller or VHDL Testbench and the user input test vectors. This process is shown in Figure 7.1.

7.2 Circuit Netlist Partitioning

As described in Figure 7.1, the switch level netlist is extracted from the circuit layout. One advantage of the switch level over the gate level is that since it is extracted from the layout it can more accurately model the circuit and any possible faults.

After extraction, the switch netlist is then reverse level ordered. For switch ordering, unlike the gate level, switches have two additional problems. The first problem, determining inputs and outputs, is handled by traversing the circuit with a forward pass from primary inputs (VDD and GND, also) to primary outputs. Inputs are determined to be the gates of transistors and the nodes that are connected to outputs of the transistors only. Outputs are determined to be the remaining node of the transistor.

The second problem encountered with switch level simulation is fan-in at nodes. Fan-in at nodes is a problem because conflicting signal values at the node must be resolved at the same level. Once the forward pass is complete, this problem is handled with a backward pass. The backward pass is just the reverse level ordering of transistors as discussed in Chapter 4. With the reverse level ordered pass, all transistors that fan-in to the same node can be put into the same level. All C programs that are used for circuit partitioning and VHDL code compilation are found in Appendix D.

7.3 Compiled Code VHDL Fault Simulator

This section describes the components that make up the simulator. The components include the PE, the IM, and the controller. The PEs and IMs structure is compiled from the reverse level ordered netlist. One transistor in the reverse level ordered netlist is

mapped to one PE in the VHDL netlist. The nodes in the reverse level ordered netlist are mapped to IMs in the VHDL netlist. Also discussed is the simulator process.

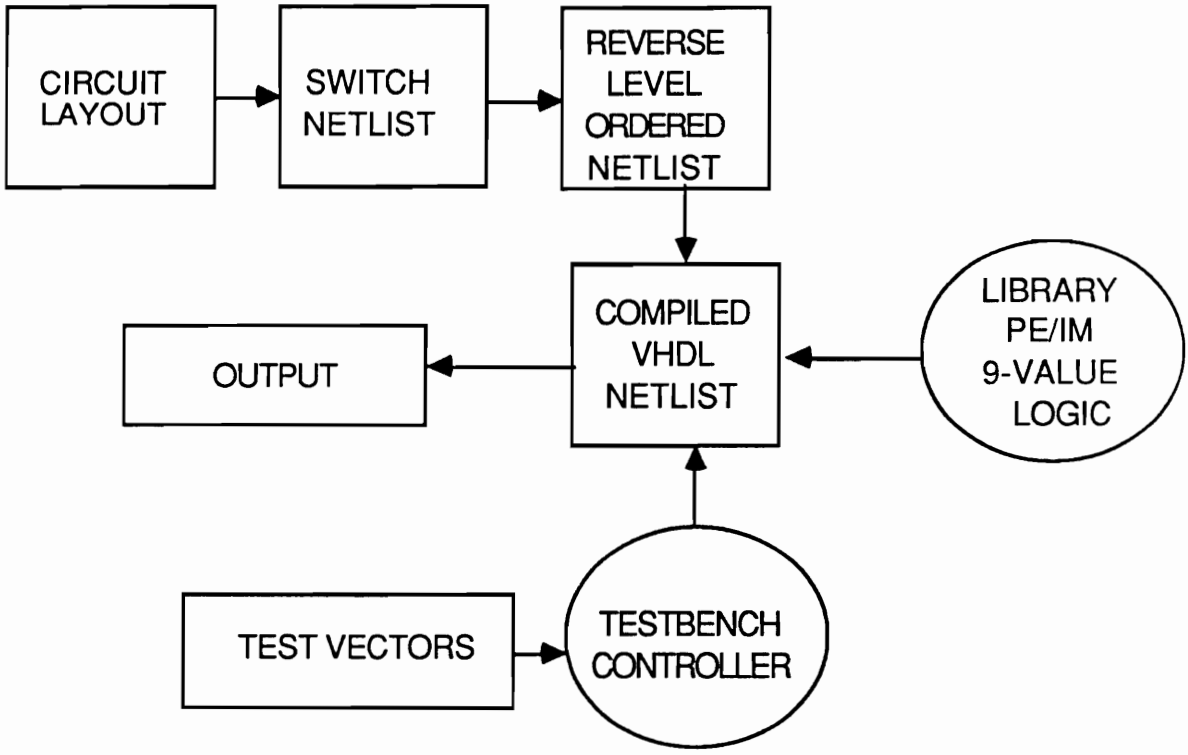


FIGURE 7.1 Fault simulation verification block diagram.

Each PE is used to model one switch of the circuit and performs parallel fault simulation using the technique described in Chapter 3. The four modes of operation of the PE, as used in fault simulation, are shown in Figure 7.2. The good simulation and the inject simulation modes model one non-faulty N - type or P - type switch. The difference in these modes is that good simulation is used on the good test vector while inject simulation is used to propagate faulty values. Parallel fault simulation mode simulates all switch level faults for one N - type or P - type switch. Fault inject mode

is used to inject faults into the simulated circuit. Finally, each PE keeps a table of its own faults as detected or undetected.

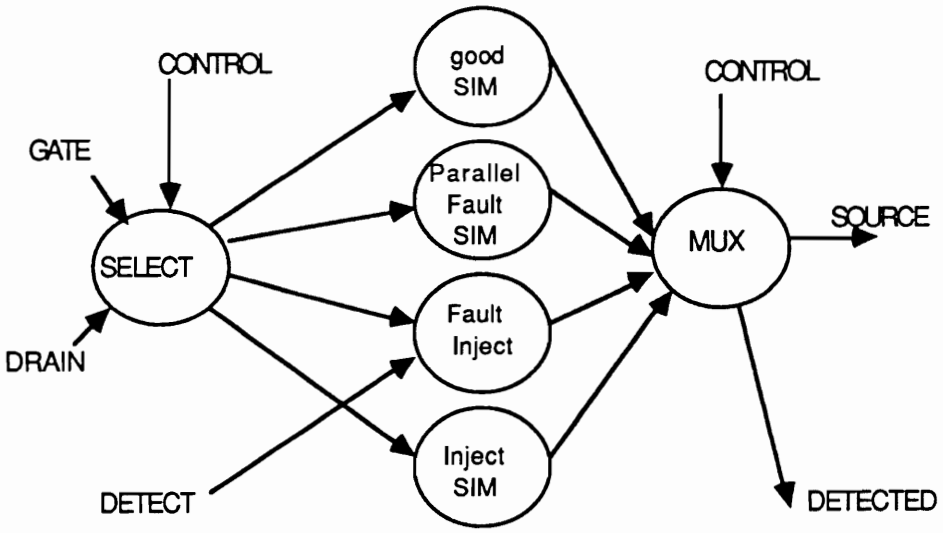


FIGURE 7.2 Processor element.

As shown symbolically in Figure 7.3, the interconnect module is used to resolve fan-in at a node in the circuit. Signal values input from lines C1, C2, ..., CN are resolved to output value C0 using the nine - value connector operation. The CONNECTOR operation is the nine-value resolve operator from the proposed standard IEEE nine-value logic [Bill91]. Since VHDL is designed for hardware modeling the CONNECTOR is easily TYPE defined. TYPE defined means that a variable or signal which includes an operation can be defined. Then at any time when this TYPE of variable or signal is used the operation is performed. In the case the CONNECTOR operation, gate, drain and source signal are defined with the CONNECTOR type. Thus the values at the fan - in nodes are resolved automatically.

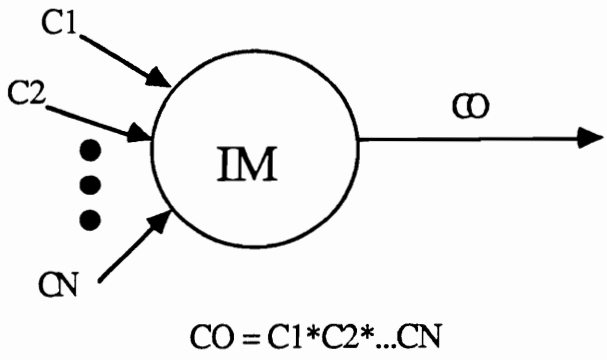


FIGURE 7.3 Interconnect module.

The controller shown in Figure 7.4 is used to model the parallel fault simulation algorithm. The controller is the VHDL stimulus file or Testbench and provides the control signals for the simulated PEs by sequencing through the four modes described earlier. This sequence follows the fault simulation algorithm as given in Chapter 5 and is the same for any compiled circuit. The Controller starts with good circuit simulation. This is followed by parallel fault simulation for one reverse level of switches. The potentially detected faults are, next, injected one at a time. These potentially detected faults are then simulated to the output or dropped in undetected.

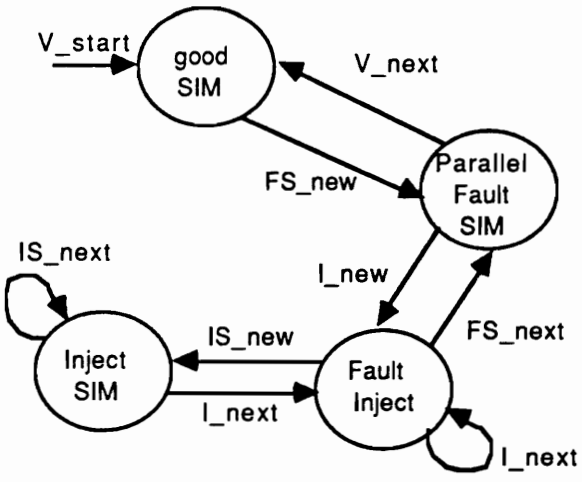


FIGURE 7.4 Testbench controller.

An example compiled benchmark circuit, C17, is shown in Figure 7.5. One processing element (PE) performs parallel switch level fault simulation for all faults on a single switch, while the interconnect modules (IMs) perform the fan-in operation at a node. Not shown in Figure 7.5 are the ground and Vdd inputs. The VHDL code for the PE, IM, example circuit, and Testbench controller is given in Appendix E. Also given in Appendix E are detailed simulation results for the ISCAS85 85 benchmark circuit C17.

7.4 Verification Results

Fault simulation verification was performed on the seven smaller ISCAS85 circuits. The input test vector sets used were obtained from the gate level test generation tool ATALANTA [Ha91]. Fault simulation complexity, the speed up due to the parallel partition size, and fault simulation switch level fault coverage results are all considered in the section.

Reducing fault simulation complexity was one of the motivating factors in this work. Here, simulation cycles are used to measure complexity. The simulation cycles were counted during fault simulation and compared to the upper bound complexity as given in Chapter 6.

Simulation cycles were measured while assuming a partition size equal to the reverse level size. Thus, when a reverse level was simulated the number of simulation cycles was incremented by one. In order to determine the simulation cycles when fixed partition sizes of 32, 64, 128, 256, and 512 are considered, the measured simulation cycles were multiplied by the complexity increase factors K from Table 6.3. The upper bound complexity is calculated using Equation 6.9.

A partition size of 1 was considered as traditional serial fault simulation. This is assuming that a general purpose computer can simulate one switch and all faults in for one switch in one simulation cycle. However, this is a very generous assumption because a general purpose computer does not have a build in switch operation.

The simulation cycles for a partition size of 1 were measured by counting every processing element simulation. The upper bound simulation cycles were calculated by multiplying the number of test vectors by the square of the number of transistors in the circuit or Vn^2. This upper bound serial fault simulation complexity is assuming traditional deductive fault simulation [Fuji85].

Shown in Figures 7.5 through 7.13 and Tables 7.1 through 7.7 are the predicted and measured complexity for the seven smaller ISCAS85 benchmark circuits (C17, C432, C499, C880, C1355, C1908, C2670). Notice that for all circuits studied measured fault simulation complexity was well below the theoretical upper bound by on average a factor of ten. The measured fault simulation complexity is the lower curves of Figures 7.5 through 7.13. This is due to fault dropping of detected faults and concurrent fault simulation of undetected faults.

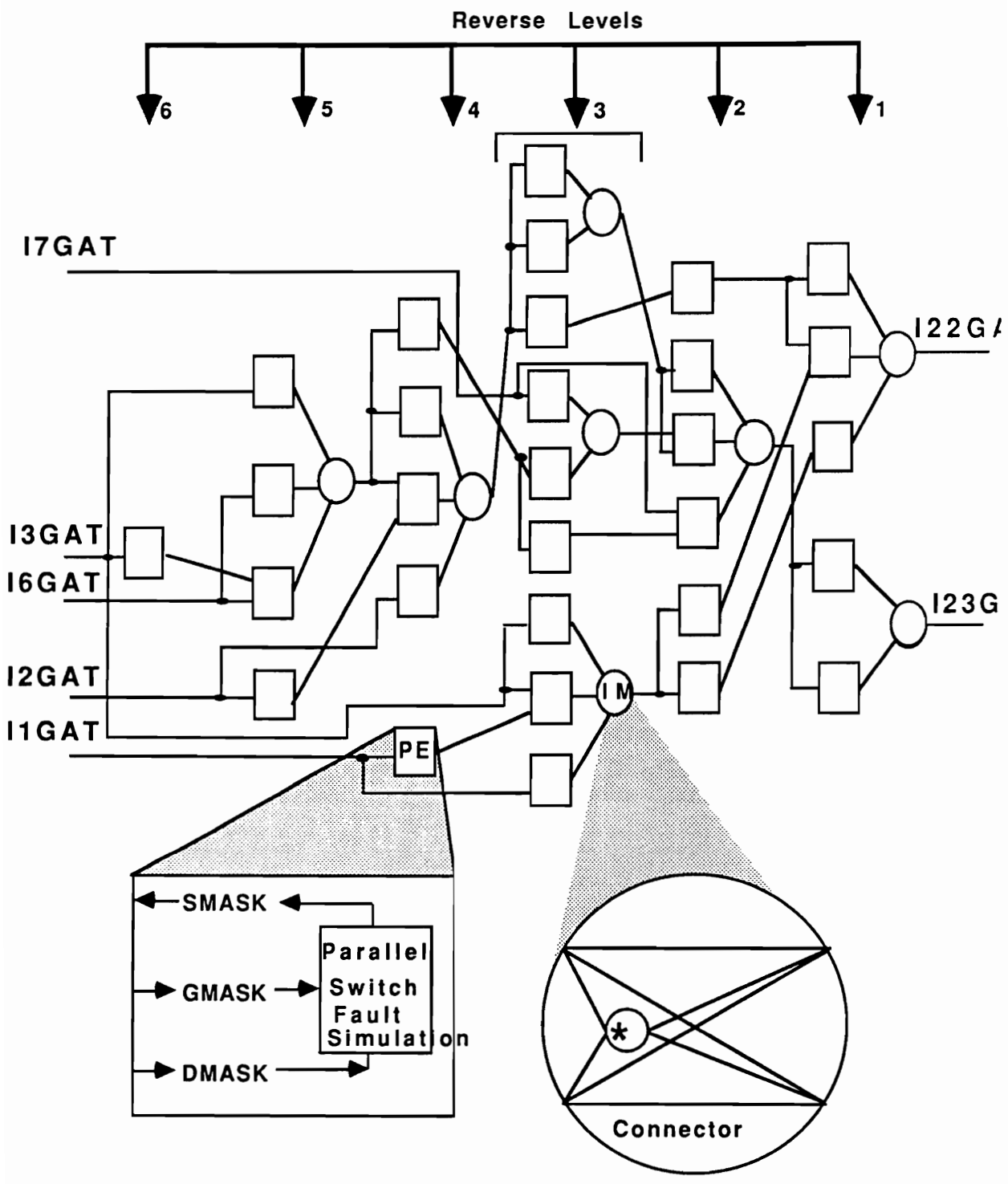


FIGURE 7.5 Compiled C17 reverse level ordering - PE's and IM's.

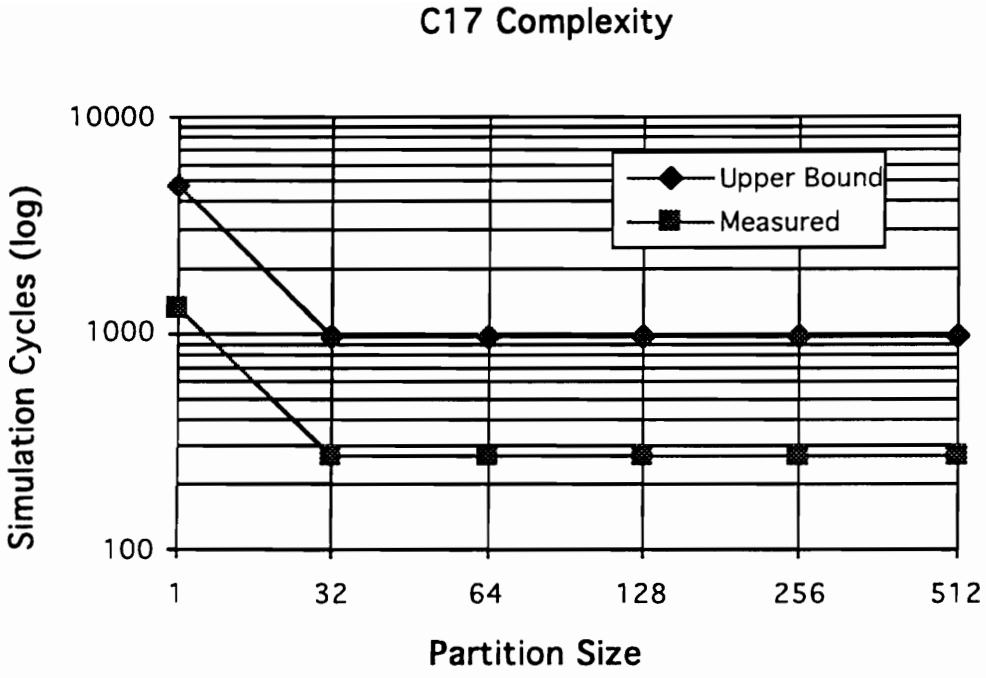


FIGURE 7.6 C17 fault simulation complexity.

TABLE 7.1 C17 fault simulation complexity.

Partition size	Simulation Cycles (Upper Bound)	Speed Up (Predicted)	Simulation Cycles (Measured)	Speed Up (Measured)
1	4732	1.000	1305	1.000
32	953.0	4.970	270.0	4.830
64	953.0	4.970	270.0	4.830
128	953.0	4.970	270.0	4.830
256	953.0	4.970	270.0	4.830
512	953.0	4.970	270.0	4.830

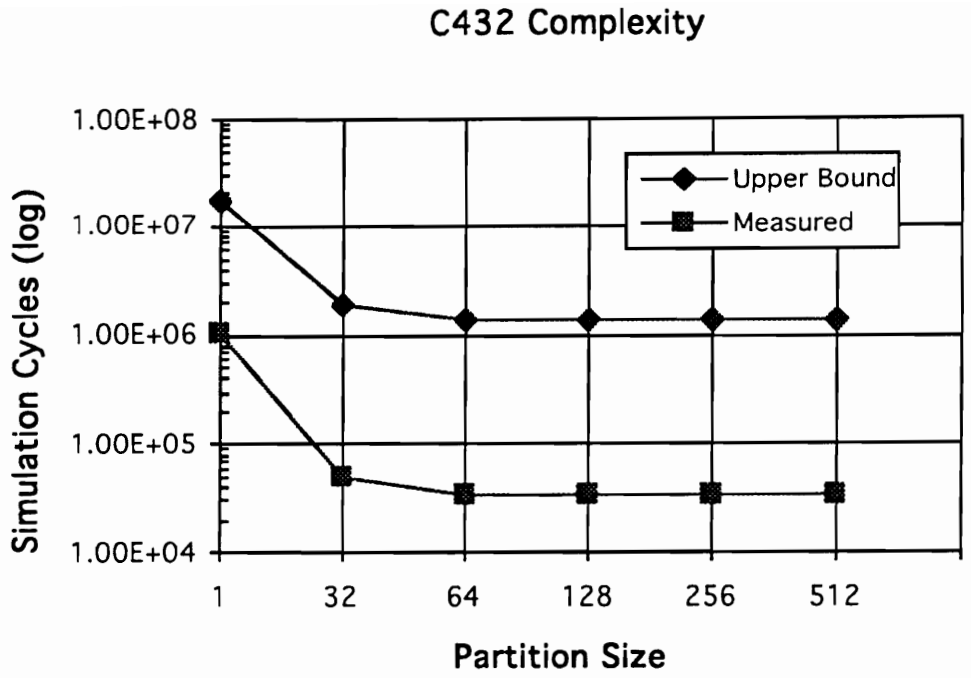


FIGURE 7.7 C432 fault simulation complexity.

TABLE 7.2 C432 fault simulation complexity.

Partition size	Simulation Cycles (Upper Bound)	Speed Up (Predicted)	Simulation Cycles (Measured)	Speed Up (Measured)
1	1.750e+07	1.000	1.020e+06	1.000
32	1.960e+06	8.930	4.926e+04	20.68
64	1.350e+06	12.96	3.395e+04	30.05
128	1.350e+06	12.96	3.395e+04	30.05
256	1.350e+06	12.96	3.395e+04	30.05
512	1.350e+06	12.96	3.395e+04	30.05

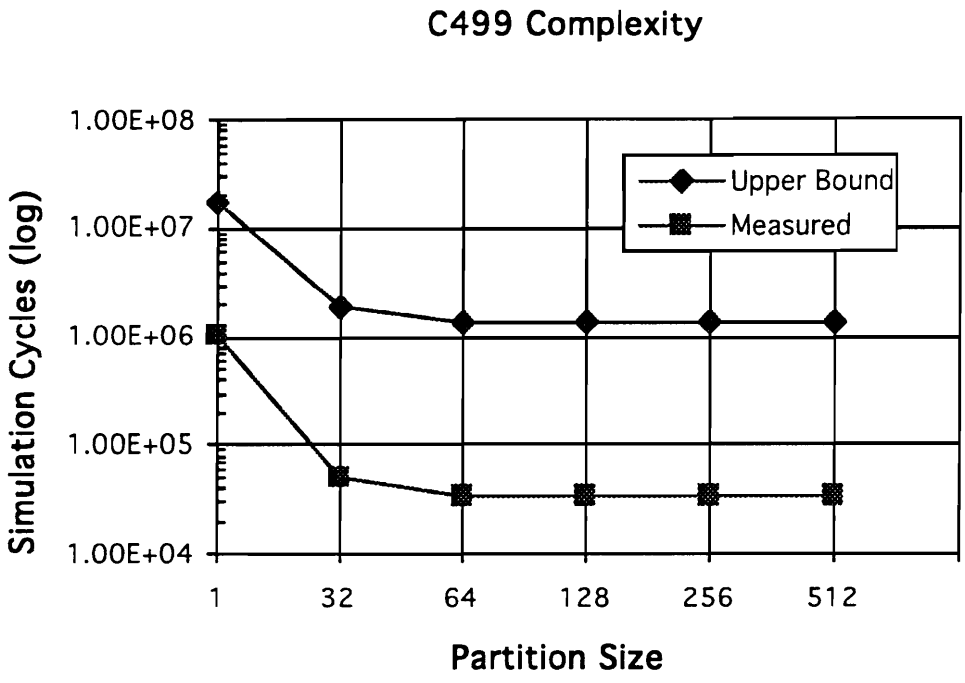


FIGURE 7.8 C499 fault simulation complexity

TABLE 7.3 C499 fault simulation complexity

Partition size	Simulation Cycles (Upper Bound)	Speed Up (Predicted)	Simulation Cycles (Measured)	Speed Up (Measured)
1	1.750e+07	1.000	1.020e+06	1.000
32	1.960e+06	8.930	4.926e+04	20.68
64	1.350e+06	12.96	3.395e+04	30.05
128	1.350e+06	12.96	3.395e+04	30.05
256	1.350e+06	12.96	3.395e+04	30.05
512	1.350e+06	12.96	3.395e+04	30.05

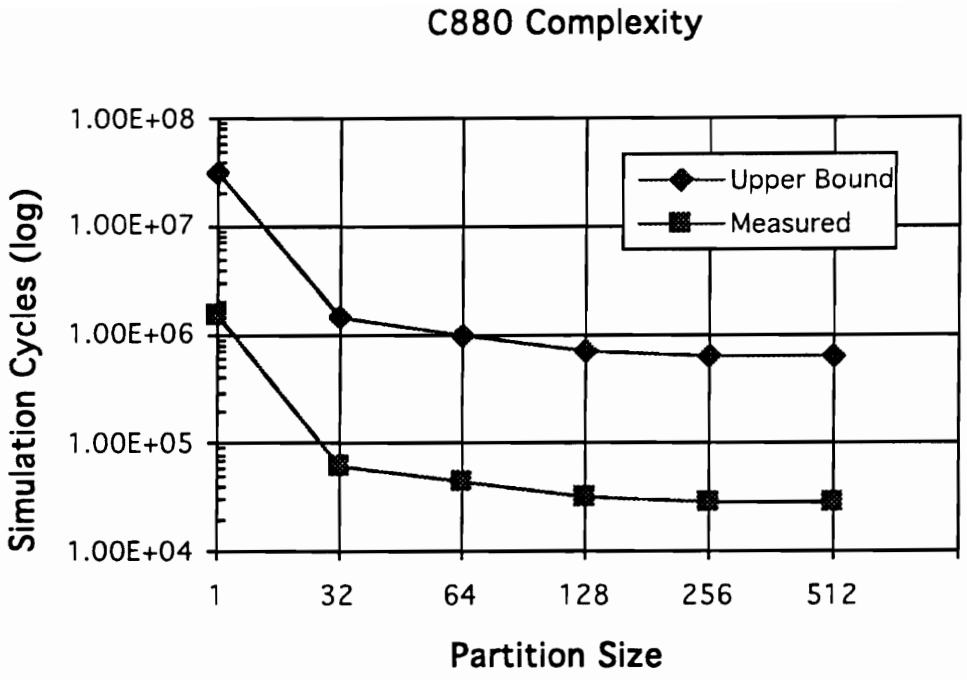


FIGURE 7.9 C880 fault simulation complexity.

TABLE 7.4 C880 fault simulation complexity.

Partition size	Simulation Cycles (Upper Bound)	Speed Up (Predicted)	Simulation Cycles (Measured)	Speed Up (Measured)
1	3.120e+07	1.000	1.550e+06	1.000
32	1.420e+06	21.97	6.406e+04	24.20
64	9.750e+05	32.00	4.398e+04	35.24
128	6.990e+05	44.64	3.156e+04	49.12
256	6.360e+05	49.10	2.869e+04	54.03
512	6.360e+05	49.10	2.869e+04	54.03

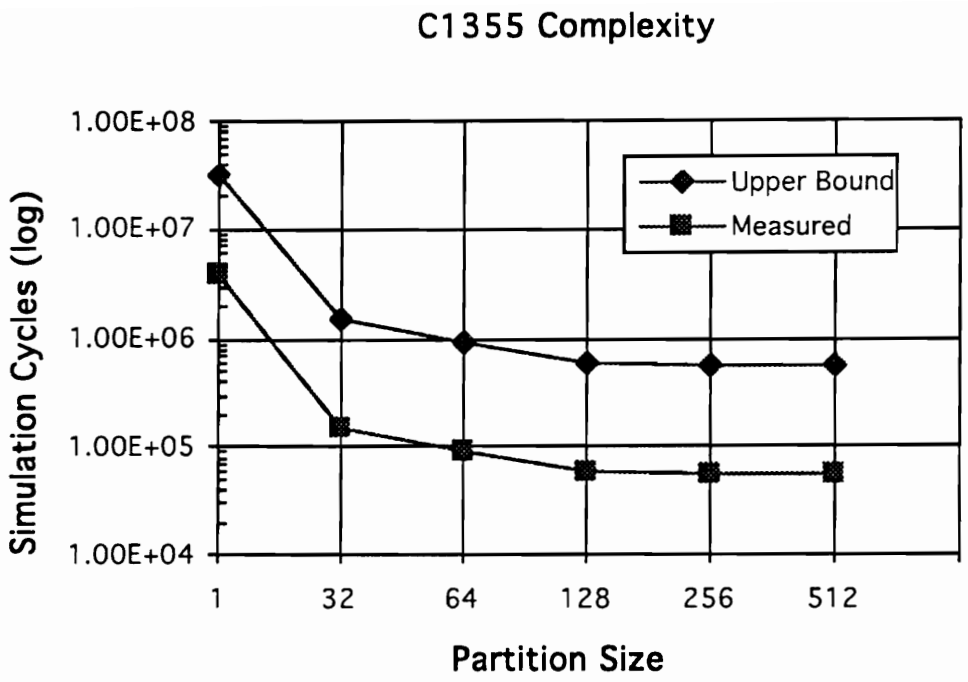


FIGURE 7.10 C1355 fault simulation complexity.

TABLE 7.5 C1355 fault simulation complexity.

Partition size	Simulation Cycles (Upper Bound)	Speed Up (Predicted)	Simulation Cycles (Measured)	Speed Up (Measured)
1	3.130e+07	1.000	3.820e+06	1.000
32	1.570e+06	19.94	1.528e+05	25.02
64	9.204e+05	34.01	8.938e+04	42.74
128	6.086e+05	51.43	5.911e+04	64.63
256	5.790e+05	54.05	5.623e+04	67.93
512	5.790e+05	54.05	5.623e+04	67.93

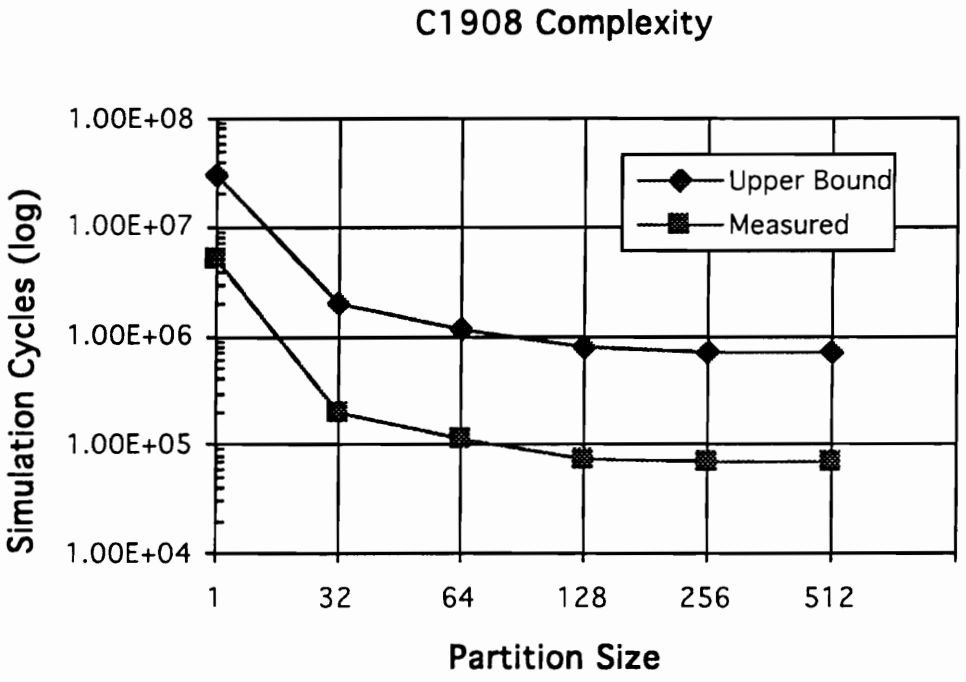


FIGURE 7.11 C1908 fault simulation complexity.

TABLE 7.6 C1908 fault simulation complexity.

Partition size	Simulation Cycles (Upper Bound)	Speed Up (Predicted)	Simulation Cycles (Measured)	Speed Up (Measured)
1	2.960e+07	1.000	5.200e+06	1.000
32	1.992e+06	14.86	1.956e+05	26.58
64	1.139e+06	25.98	1.118e+05	46.51
128	7.686e+05	38.51	7.546e+04	68.91
256	6.975e+05	42.44	6.847e+04	75.93
512	6.975e+05	42.44	6.847e+04	75.93

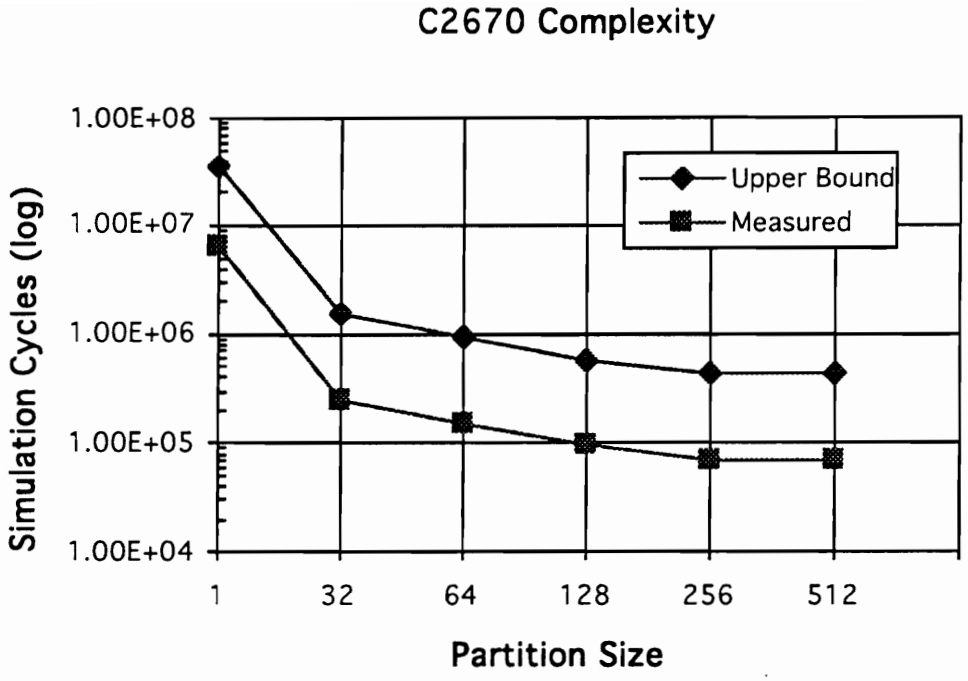


FIGURE 7.12 C2670 fault simulation complexity.

TABLE 7.7 C2670 fault simulation complexity.

Partition size	Simulation Cycles (Upper Bound)	Speed Up (Predicted)	Simulation Cycles (Measured)	Speed Up (Measured)
1	3.540e+07	1.000	6.580e+06	1.000
32	1.518e+06	23.32	2.529e+05	26.02
64	9.277e+05	38.16	1.546e+05	42.56
128	5.810e+05	60.93	9.681e+04	67.97
256	4.217e+05	83.95	7.026e+04	93.65
512	4.217e+05	83.95	7.026e+04	93.65

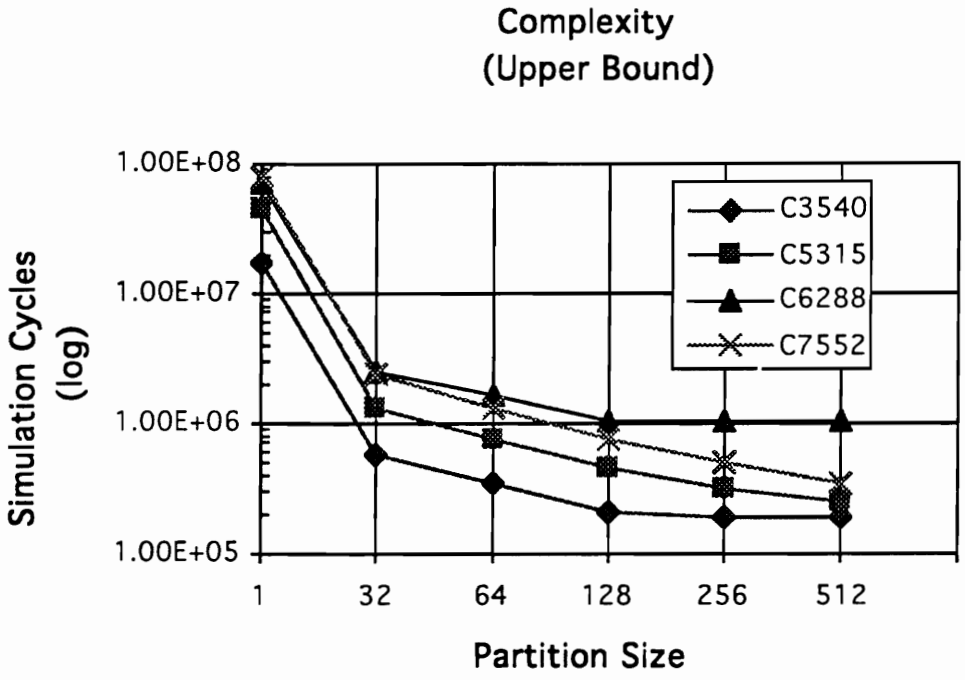


FIGURE 7.13 C3540, C5315, C6288, C7552 fault simulation complexity.

TABLE 7.8 C3540, C5315, C6288, C7552 fault simulation complexity.

Part size	C3540 Cycles	C3540 Speed Up	C5315 Cycles	C5315 Speed Up	C6288 Cycles	C6288 Speed Up	C7552 Cycles	C7552 Speed Up
1	1.699e+07	1.000	4.530e+07	1.000	7.160e+07	1.000	7.840e+07	1.000
32	5.729e+05	29.70	1.305e+06	34.70	2.428e+06	29.50	2.385e+06	32.90
64	3.368e+05	50.40	7.345e+05	61.70	1.624e+06	44.10	1.288e+06	60.80
128	2.101e+05	80.90	4.425e+05	102.4	1.036e+06	69.11	7.620e+05	102.9
256	1.842e+05	92.20	3.056e+05	148.2	1.031e+06	69.44	4.894e+05	160.2
512	1.842e+05	92.20	2.509e+05	180.6	1.031e+06	69.44	3.408e+05	230.0

To understand the reduced simulation time for parallel fault simulation, the speed up of parallel fault simulation over that of serial fault simulation was considered. The speed up is defined here as the ratio of the number of serial simulation cycles to the

number of parallel simulation cycles. Measured and predicted speed up versus partition size for the ISCAS85 circuits are given in Tables 7.1 through 7.8 and shown in Figures 7.14 through 7.18. The speed up for the circuits studied to plotted against the maximum partition size.

The speed up results for the benchmark circuits as shown in Figures 7.14 through 7.18 indicate that for all but the largest circuits the speed up is fairly constant and does not increase with partition sizes greater than 128. A plot of speed up versus circuit size is shown in Figures 7.19 and 7.20. Notice here that for partition of 32, 64, 128, the speed up has values that range from 5 to 80.

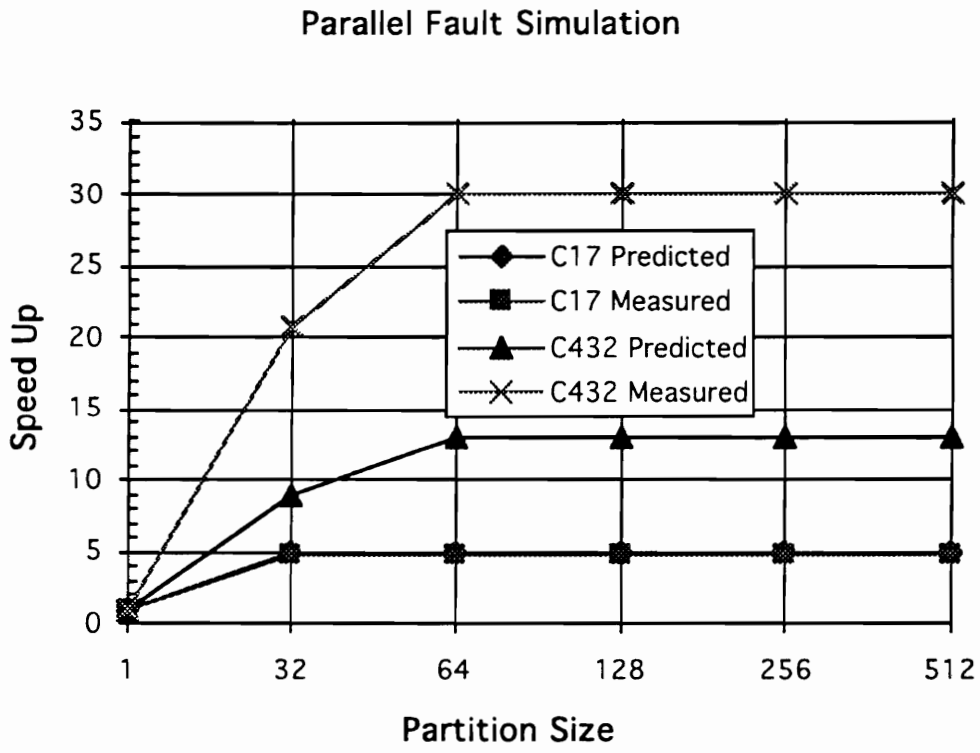


FIGURE 7.14 C17, C432 parallel fault simulation speed up.

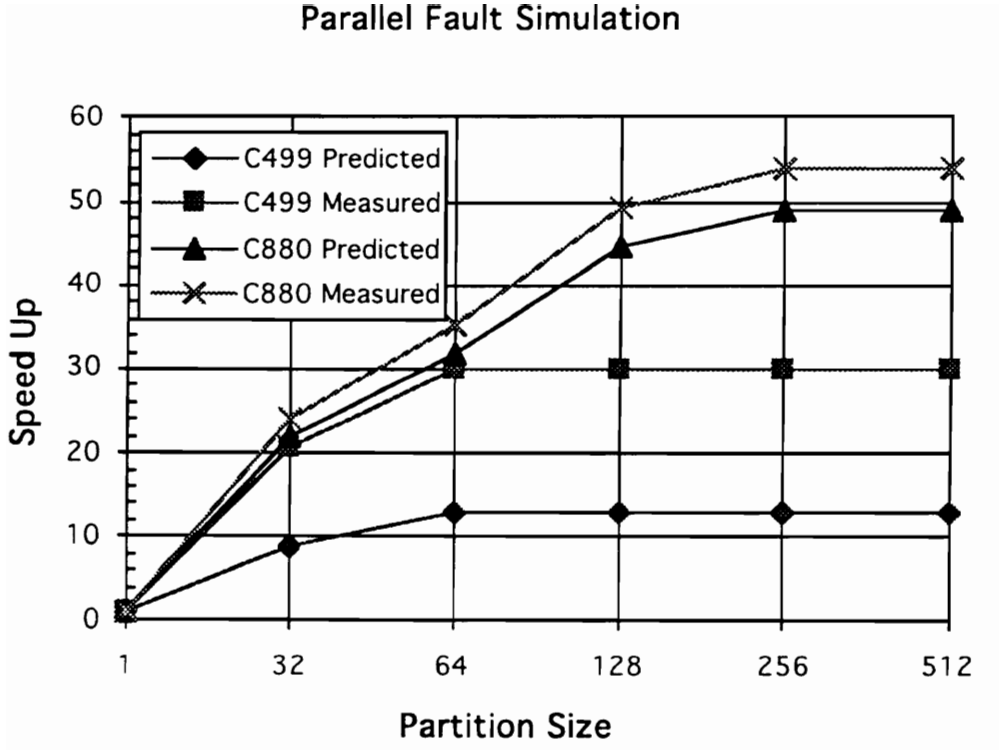


FIGURE 7.15 C499, C880 parallel fault simulation speed up.

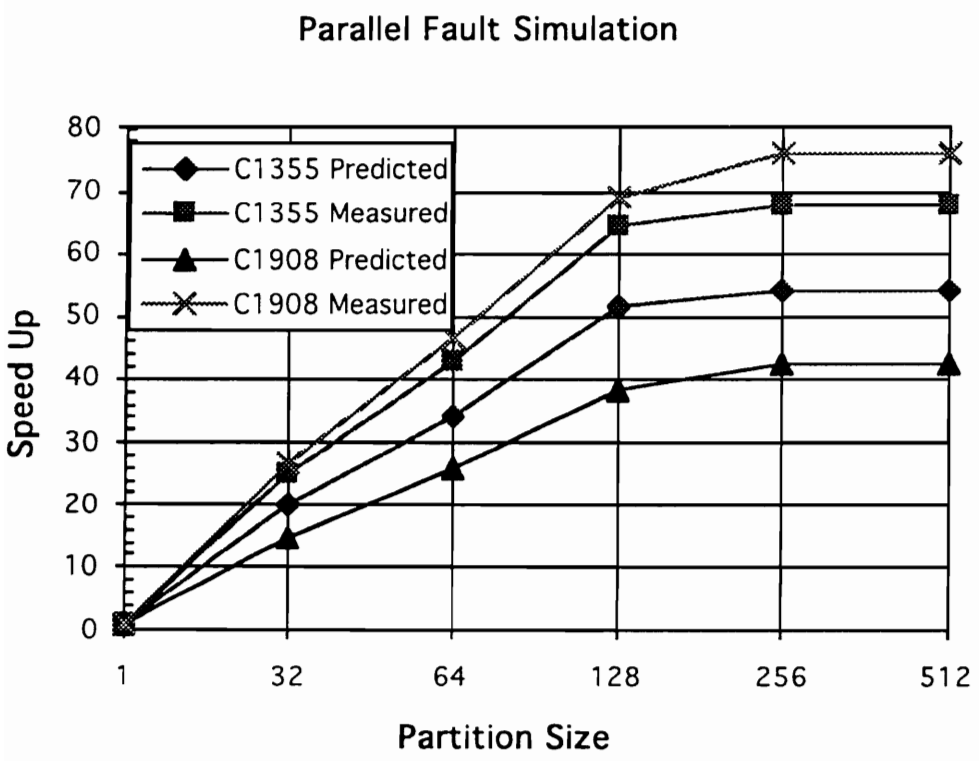


FIGURE 7.16 C1355, C1908 parallel fault simulation speed up.

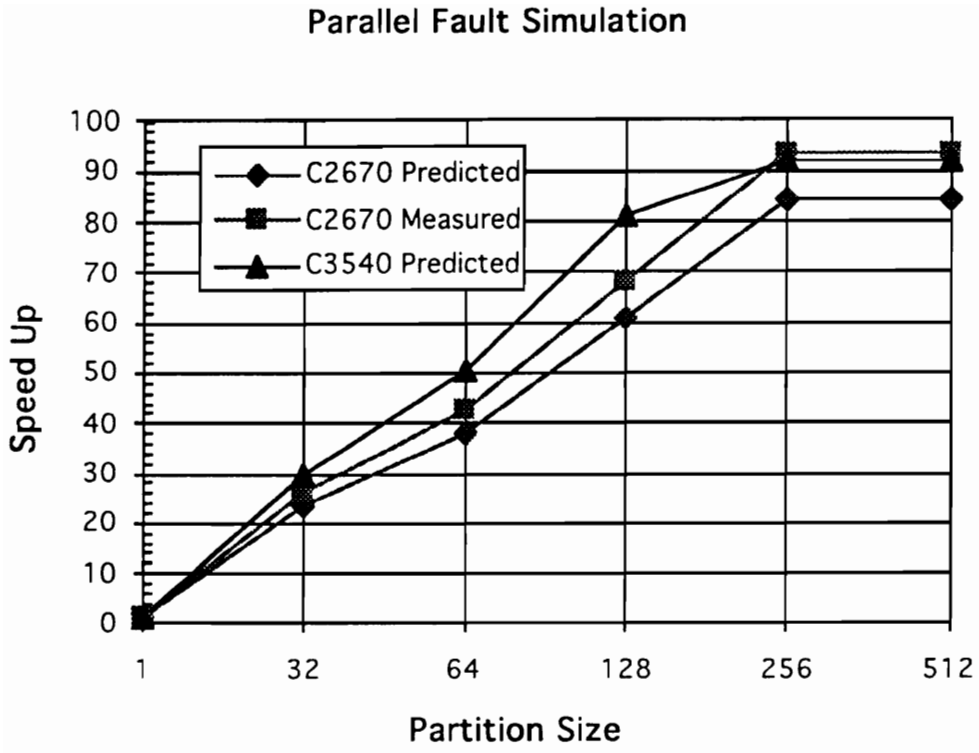


FIGURE 7.17 C2670, C3540 parallel fault simulation speed up.

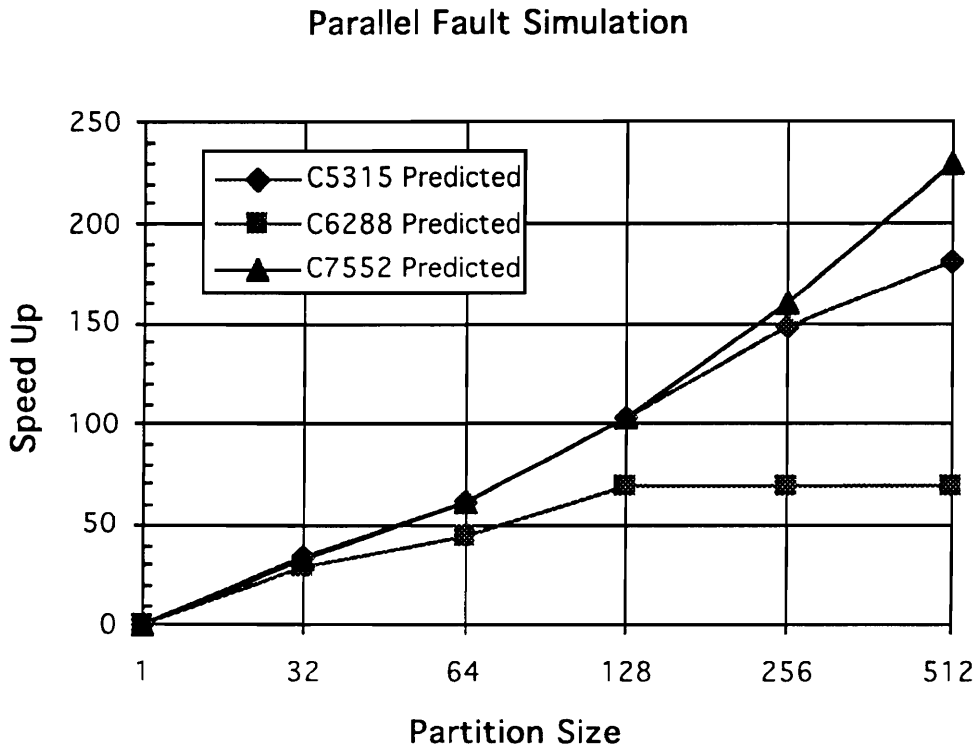


FIGURE 7.18 C5315, C6288, C7552 parallel fault simulation speed up.

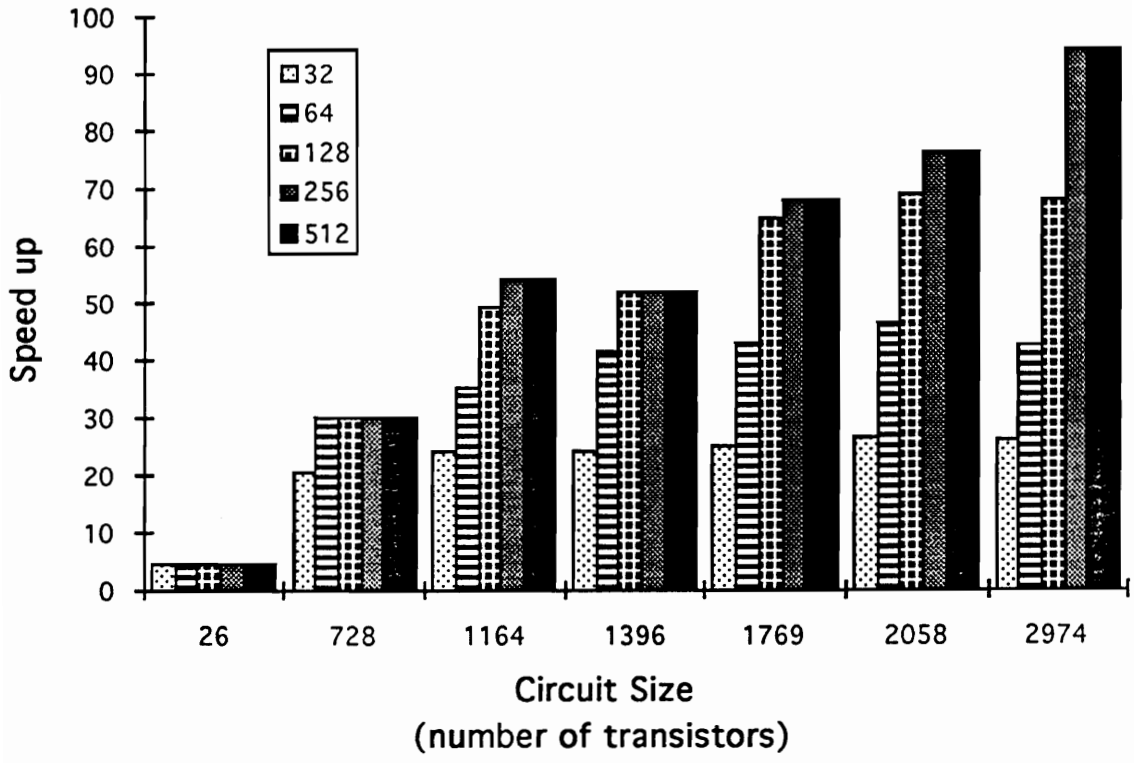


FIGURE 7.19 Measured parallel fault simulation speed up for different partition sizes (32, 64, 128, 256, 512).

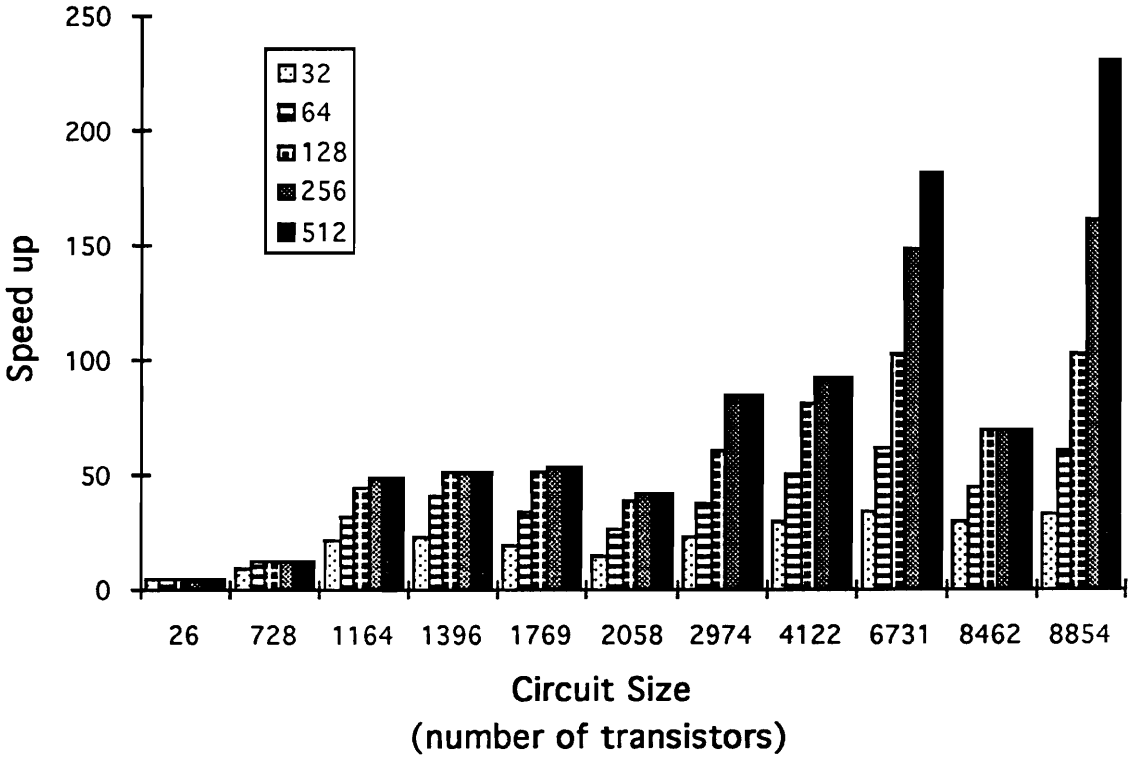


FIGURE 7.20 Predicted parallel fault simulation speed up for different partition sizes (32, 64, 128, 256, 512)..

Fault simulation results for the switch level and the gate level are shown in Table 7.9. Results show, as expected, that gate level test vector sets do not detect all switch level faults. This result, while not a surprise, does confirm the fact that switch fault simulation can be a better design verification tool. This is because a larger test set would be required to achieve a switch level fault coverage similar to the gate level fault coverage. Any possible switch level fault undetected for the gate level test set could be detected by the larger switch level test set.

Because of long simulation times, only partial test vector sets were used in Table 7.9. The host computer used for simulation was a SUN SPARC 2 with 32 Megabytes of RAM. The VHDL simulator used was SYNOPSYS. Shown in Table 7.10, the long simulation times were because parallel computation designed for special purpose hardware is being modeled and simulated in software on a general purpose computer.

TABLE 7.9 Fault simulation results.

CKT	# of test vectors	Gate detect Faults	Gate undet Faults	Gate Fault %Cover	Switch detect Faults	Switch undet Faults	Switch Xdet Faults	Switch Fault %Cover
c17	7	22	0	100	60	17	27	57.69
c432	33	460	64	87.79	1549	405	958	53.19
c499	5	547	211	72.16	2150	792	2642	38.50
c880	23	856	86	90.87	2651	634	1371	56.94
c1355	10	1035	1574	65.76	2574	1297	3201	36.40
c1908	7	1047	832	55.72	3273	1433	3526	39.76
c2670	4	1185	1562	43.14	4317	2066	5513	36.29

TABLE 7.10 Fault simulation run times.

Circuit	Number of transistors	Number of test vectors	CPU time (Seconds)
c17	26	7	29.04
c432	728	33	4,065
c499	1396	5	13,292
c880	1164	23	9,215
c1355	1769	10	20,106
c1908	2058	7	22,341
c2670	2974	4	32,395

Chapter 8.

Conclusion and Future Work

8.1 Conclusion

One problem with performing fault simulation at the gate level is that it has been shown that physical faults within a Metal Oxide Semiconductor (MOS) (IC) cannot be modeled by only gate level stuck-at faults. Additional fault models needed are: line stuck-at faults internal to the gate, transistor stuck-at faults, floating line faults, and bridging faults. Switch level simulation has greater accuracy and can model physical faults better.

One problem common to current switch level fault simulators is that of long simulation times. This problem occurs because of the upper bound on computational complexity of the traditional gate level fault simulation techniques used at the switch level. If n is the number of gates, the upper bounds on simulation time are $O(n^3)$, $O(n^2)$, and $O(n^2)$, for parallel, deductive, and concurrent fault simulation, respectively. When switch level simulation is considered, the problem is compounded since each CMOS gate is typically comprised of from two to ten transistors. This makes switch level fault simulation of even an average size integrated circuit unfeasible.

The work presented in this dissertation addresses the complexity and large simulation time for switch level fault simulation by examining the feasibility of two-dimensional parallel fault simulation using a Parallel Hardware Accelerated Fault Simulator (PHAFS).

Using nine-valued logic, switch state tables, a minimum operation, and the connector operation, a novel switch level extension to parallel fault simulation, was introduced in Chapter 3. This technique simulates all faults for the switch in parallel.

Chapter 4 introduced reverse level ordering as the circuit partitioning method for this work. Reverse level ordering is the choice for the circuit partitioning method because it isolates levels of transistors as disjoint when simulation order is preserved. In order to reduce the minimum required parallel partition size, reverse levels are further subdivided into groups and subgroups of transistors. It was shown that for the benchmark circuits, the average size of groups and subgroups are quite small. In addition, the maximum subgroup size has value of less than 32. These results indicate that transistors in a circuit are basically disjoint.

Chapter 5 presented the parallel fault simulation algorithm and showed upper bound complexity on order of $O(L^2)$, where L is the number of reverse levels in the circuits. This complexity is much less than the complexity for traditional techniques, which are on order of $O(n^2)$, where n is the number of transistors in the circuits.

Chapter 6 studied load and complexity increase versus parallel partition size. Load is important because any special purpose hardware that is unused is not cost effective.

Complexity increase is important when comparing the fixed partition size parallel fault simulation method with traditional fault simulation techniques. In this chapter, It was shown that even with a fixed partition size, parallel fault simulation is much faster than traditional techniques.

The techniques and algorithm introduced in Chapters 3, 4, 5, and 6 were verified using the C language and VHDL language as described in Chapter 7. The C language is used to model the software that performs the reverse level order circuit partitioning. VHDL is used to model the parallel switch level fault simulation technique, the fan-in connector operation, and the fault simulation algorithm. Results show that the complexity as measured in simulation cycles for the parallel fault simulation technique is less than the predicted upper bound complexity and much less than serial fault simulation. The measured complexity for the circuits studied approaches a constant value with a partition size greater than 128. For a partition size less than or equal to 128, results also show that the speed up for parallel fault simulation is between 5 and 80. This means that partition sizes of between 32 and 128 PEs would be cost effective in terms of the load or the usage of PEs, while worst-case complexity would be still be less than traditional techniques.

The feasibility of two-dimensional parallel fault simulation using a Parallel Hardware Accelerated Fault Simulator (PHAFS) has been studied. The worst - case computational complexity has been used as a measure of comparison between PHAFS and other techniques. The results given in this dissertation show that, even while taking into account the fixed partition size effects, the complexity for PHAFS is much less than that of traditional techniques. Results also show that the hardware or maximum

number of PEs required can be as little as 32 PEs. These two results makes PHAFS feasible in terms of speed benefits and overall cost.

8.2 Future Work

Future work on the PHAFS system might include a study of interconnection networks, a study of applicable single instruction multiple data (SIMD) architecture, and detailed ASIC design. The building block for the interconnection network might consist of the interconnect module (IM), while its overall functionality would be to resolve all fan - in signal values for a partition. The architecture would consist of PEs and IM along with a controller and memory. Issues of concern for the detailed ASIC design would include speed and size of the IC.

Future work on the PHAFS system would also include expanding the fault simulation to include bidirectional switches and synchronous circuits.

Bibliography

[Abram84] M. Abramovici, P.R. Menon, D.J. Miller, "Critical Path Tracing: An Alternative to Fault Simulation," *IEEE Design and Test of Computers*, Vol. 5, No. 4, pp. 83-93, Dec. 1984.

[Agraw87.1] P. Agrawal, W.J. Dally, A.K. Ezzat, W.C. Fischer, H.V. Jagadish, and A.S. Krishnakumar, "Architecture and Design of the MARS Hardware Accelerator," ACM IEEE 24th Design Automation Conference Proceedings, pp. 101 - 107, 1987.

[Agraw87.2] P. Agrawal, W.J. Dally, W.C. Fischer, H.V. Jagadish, A.S. Krishnakumar, and R. Tutundjian, "MARS: A Multiprocessor Based-Programmable Accelerator," *IEEE Design and Test of Computers*, Vol. 6, No. 5, pp. 28-36, Oct. 1987 .

[Agraw89] Prathima Agrawal, Vishwani D. Agrawal, Kwang-Ting Cheng, and Raffi Tutundjian, "Fault Simulation in a Pipelined Multiprocessor System," IEEE International Test Conference Proceedings, pp. 727 - 734, 1989.

[Agraw90] Ajit Agrawal, and Debashis Bhattacharya, "CMP3F: A High Speed Fault Simulator for the Connection Machine," IEEE International Test Conference Proceedings, pp. 410 - 416, 1990.

[Agraw90.2] P. Agrawal, S. Robinson, and T. Szymanski, "Automatic Modeling of Switch-Level Networks Using Partial Orders," *IEEE Trans. CAD*, Vol. 9, No. 7, pp. 696-707, July 1990.

[Barzi86] Z. Barzilai, D.K. Beece, L.M. Huisman, V.S. Iyengar, and G.M. Silberman, "SLS-A Fast Switch Level Simulator for Verification and Fault Coverage Analysis," IEEE 23rd Design Automation Conference, pp. 164 - 170, 1986.

[Beece88] Daniel K. Beece, George Deibert, Georgina Papp, and Frank Villante, "The IBM Engineering Verification Engine," ACM IEEE 25th Design Automation Conference Proceedings, pp. 218 - 223, 1988.

[Bill91] W.D. Billowitch, VHDL Tech Notes, The VHDL Consulting Group, Bethlehem, Pa., June 1990.

[Blank84] T. Blank,"A Survey of Hardware Accelerators Used in Computer-Aided Design, " *IEEE Design and Test of Computers*," Vol 5, No 4, pp 21-39, Aug. 1984.

[Bose82] A.K. Bose, P. Kozak, C-Y Lo, H.N. Nham, E. Pacas-Skewes, and K. Wu, "A Fault Simulator for MOS LSI Circuits," IEEE 19th Design Automation Conference Proceedings, pp. 400 - 409, 1982.

[Brglez85] F.Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran," Proc. IEEE Intl Symp. on Circuits and Systems, Kyoto, pp. 663-698, June 1985.

[Bryant85] Randal E. Bryant, and Michael D. Schuster, "Performance Evaluation of FMOSSIM, a Concurrent Switch-Level Fault Simulator," IEEE 22nd Design Automation Conference Proceedings, pp. 715 - 719, 1985.

[Bryant87] R.E. Bryant, D. Beatty, K. Brace, K. Cho, and T. Sheffler, "COSMOS: A Compiled Simulator for MOS circuits," ACM IEEE 24th Design Automation Conference Proceedings, pp. 9 - 16, 1987.

[Duba88] Patrick A. Duba, Rabindra K. Roy, Jacob A. Abraham, and William A. Rogers, "Fault Simulation in a Distributed Environment," ACM IEEE 24th Design Automation Conference Proceedings, pp. 686 - 691, 1988.

[Fuji85] H. Fujiwara, "Logic Testing and Design for Testability," MIT Press, pp. 84-102, 1985.

[Gai88a] S. Gai, P. L. Montessoro, and F. Somenzi, "MOZART: A Concurrent Multilevel Simulator," *IEEE Transactions on Computer Aided Design*, Vol. 7, No. 9, pp. 1005-1011, Sept. 1988.

[Gai88b] S. Gai, P. L. Montessoro and F. Somenzi, "The Performance of the Concurrent Fault Simulation Algorithms in MOZART," ACM IEEE 25th Design Automation Conference Proceedings, pp. 692 - 697, 1988.

[Goel80] P. Goel, "Test Generation Costs Analysis and Projections," 17th Design Automation Conference Proceedings, pp. 77 - 84, 1980.

[Ha92] D. Ha, Personal Communications, Virginia Polytechnic Institute and State University, Dec. 92.

[Hayes82] John P. Hayes, "A Fault Simulation Methodology for VLSI," IEEE 19th Design Automation Conference Proceedings, pp. 393 - 399, 1982.

[Hayes87] John P. Hayes, "An Introduction to Switch-Level Modeling," *IEEE Design and Test of Computers*, Vol. 6, No. 4, pp. 18-25, Aug. 1987.

[Hills85] W.D. Hills, The Connection Machine, MIT press, Cambridge, Mass., 1985.

[Ishi90] Nagisa Ishiura, Masaki Ito, and Shuzo Yajima, "Dynamic Two-Dimensional Parallel Simulation Technique for High-Speed Fault Simulation on a Vector Processor," *IEEE Transactions on Computer-Aided Design*, Vol. 9, No. 8., pp. 868-875, Aug. 1990.

[Kawai84] Masato Kawai, and John P.Hayes, "An Experimental MOS Fault Simulation Program CSASIM," IEEE 21st Design Automation Conference Proceedings, pp. 2 - 9, 1984.

[Koe89] W. Koe, "An Investigation of Sensitization Conditions and Test Effectiveness for CMOS Faults," Virginia Tech. Masters Thesis, 1989.

[Krav91] S.A. Kravitz, R.E. Bryant, and R.A. Rutenbar, "Massively Parallel Switch-level Simulation: A Feasibility Study," *IEEE Transactions On Computer-Aided Design*, Vol. 10, No. 7, pp. 871-894, 1991.

[Kush89] E.J. Kusher, "The iPSC/2: A Second Generation Concurrent Supercomputer from Intel Scientific Computers". Proceedings of the Fifth Conference on Multiprocessors and Array Processors, pp 95 - 103, 1989.

[Lee91] T. Lee, and I. Haji, "A Switch-Level Matrix Approach to Transistor-Level Fault Simulation," Proc. ICCD, pp. 554-557, Nov. 1991.

[Light82] M.R. Lightner and G.D. Hachtel, "Implication Algorithms for MOS Switch Level Functional Macromodeling Implication and Testing," IEEE 19th Design Automation Conference Proceedings, 1982, pp. 691 - 698.

[Lim89] B. Lim, "Fault Simulation for Supply Current Testing of Bridging Faults in CMOS Circuits," Virginia Tech. Masters Thesis, 1989.

[Maam90] F. Maamari, and J. Rajski, "A Method of Fault Simulation Based on Stem Regions," *IEEE Trans. CAD*, Vol. 8, pp. 212-220, February 1990.

[Mark90] T. Markas, M. Royals, and N. Kanopoulos Research Triangle Institute, "On Distributed Fault Simulation," *Computer*, Vol. 23, No. 1, pp. 40-52, Jan. 1990,

[Moto88] Akira Motohara, Motohide Murakami, Miki Urano, Yasuo Masuda, and Masahide Sugano, "An Approach to Fast Hierarchical Fault Simulation," ACM IEEE 25th Design Automation Conference Proceedings, pp 698 - 703, 1988.

[Nagel75] 7 L.W. Nagel,"SPICE2: A Computer Program to Simulate Semiconductor Circuits," Berkely, CA, Rep. No. ERL-M520, May1975.

[Prad90] D.K. Pradhan, "Fault-Tolerant Computing-Theory and Techniques Volume 1", pp. 31-35, 1989.

[Rajs87] R. Rajsuman, Y. K. Malaiya, and A. P. Jayasumana, "On Accuracy of Switch-Level Modeling of Bridging Faults in Complex Gates," 25th ACM/IEEE Design Automation Conference Proceedings, pp. 244 - 250, 1987.

[Saab90] 5 Daniel G. Saab, Robert B. Mueller-Thuns, David Blaauw, Joseph T. Rahmeh, and Jacob A. Abraham, "Hierarchical Multi-level Fault Simulation of Large Systems," *Journal of Electronic Testing: Theory and Applications*, Vol. 1, No. 2, pp. 139-149, May 1990.

[Shen85] John P. Shen, W. Maly, and F. Joel Ferguson, "Inductive Fault Analysis of MOS Integrated Circuits," *IEEE Design and Test of Computers*, Vol. 5, No. 4, pp. 13-26, Dec. 1985.

[Shih86] Hsi-Ching Shih, Joseph T. Rahmeh, and Jacob A. Abraham, "FAUST: An MOS Fault Simulator with Timing Information," *IEEE Transactions on Computer-Aided Design*, Vol. 5, No. 4, pp. 557-563, Oct. 1986.

[Smith87] Mark T. Smith, "A Hardware Switch Level Simulator for Large MOS Circuits," ACM IEEE 24th Design Automation Conference Proceedings, pp. 95 - 100, 1987.

[Synop90] Synopsys-The Synthesis Company, VHDL System Simulator Reference Manuel, 1990.

[Taka89] Shigeru Takasaki, Fumiyasu Hirose, and Akihiko Yamada, "Logic Simulation Engines in Japan," *IEEE Design and Test of Computers*, Vol. 6, No. 5, pp. 40-49, Dec. 1989.

[Waic85] J. A. Waicukauski, E. B. Eichelberger, D. O. Forlenza, E. Lindbloom and T. McCarthy, "Fault Simulation for Structured VLSI," VLSI Systems Design, Vol. 6, No. 12, pp. 22-32, Dec. 1985.

[Wu90] 3 Wu Tung Cheng, and Meng-Lin Yu, "Differential Fault Simulation for Sequential Circuits," *Journal of Electronic Testing: Theory and Applications*, Vol.1, No.1, pp. 7-13, 1990.

Appendix

Appendix A

VHDL N-type Switch Parallel Fault Simulation

A.1 Parallel Switch Fault Simulation for One N-type Switch

The VHDL code used to verify the parallel switch level fault simulation technique follows.

```
-------------------------------------------------------
--            Parallel Fault simulation          --
--           name - N_PARALL                --
--            chris ryan   prelim work          --
--           12-10-91                          --
-------------------------------------------------------
use WORK.logic_system.all, WORK.pnew1.all, WORK.pnew2.all, WORK.all ;
library  WORK;
package pnew is
type RESTYPEBBB is array (INTEGER range <>) of BIT;

function MASK_NEW (IN_T, MASK_IN, FVALUE_IN : std_LOGIC_vector(0 to 6) ) return
std_LOGIC_vector;
function EXPAND1_7 (A : std_LOGIC) return std_LOGIC_vector;
--function INVW ( INDATA: DATA8 ) return DATA8;
end pnew;

package body pnew is

function MASK_NEW (IN_T, MASK_IN, fvalue_IN: std_LOGIC_vector(0 to 6))return
  std_LOGIC_vector is
variable RES_VALUE: std_LOGIC_vector(0 to 6) := "ZZZZZZZ" ;
begin

RES_VALUE := CON(MIN_F(IN_T,fvalue_in),MASK_IN);

 return RES_VALUE;
 end MASK_NEW;

function EXPAND1_7 (A : std_LOGIC ) return std_LOGIC_vector is
variable RESOLVED_VALUE: std_LOGIC_vector(0 to 6) := "ZZZZZZZ" ;

 begin
   for I in 0 to 6 loop
           RESOLVED_VALUE(I) := A;
  end loop;
 return RESOLVED_VALUE;
 end EXPAND1_7;

end pnew;

use  work.logic_system.all,  work.pnew1.all,work.pnew2.all,  work.pnew.all,  work.all  ;

entity N_PARALL is
port(DRAIN     : in   STD_LOGIC          := 'Z'  ;
   GATE      : in   STD_LOGIC          := 'Z'  ;
```

```
    PREV_SOURCE : in   STD_LOGIC              := 'Z'  ;
    SOURCE_M    : out STD_LOGIC_vector(0 to 6) := "ZZZZZZZ";
    STOP        : in   BIT         );
end N_PARALL;


architecture ONLY of N_PARALL is

signal  SOURCE_S7, DRAIN_M, GATE_M, PREV_SOURCE_S7, DRAIN_S7, GATE_S7:
          STD_LOGIC_VECTOR(0 to 6)  := "ZZZZZZZ";
begin
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
IN_VECTOR:process(DRAIN,GATE,PREV_SOURCE)
begin
        DRAIN_S7       <= EXPAND1_7(DRAIN);
        GATE_S7       <= EXPAND1_7(GATE);
        PREV_SOURCE_S7 <= EXPAND1_7(PREV_SOURCE);
end process;


- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
IN_MASK: process(DRAIN_S7,GATE_S7)

constant MASK_D      : STD_LOGIC_VECTOR(0 to 6) := "ZZZ10ZZ";
constant FVALUE_D    : STD_LOGIC_VECTOR(0 to 6) := "UUUZZUU";
constant MASK_G      : STD_LOGIC_VECTOR(0 to 6) := "Z10ZZZZ";
constant FVALUE_G    : STD_LOGIC_VECTOR(0 to 6) := "UZZUUUU";
constant MASK_S      : STD_LOGIC_VECTOR(0 to 6) := "ZZZZZ10";
constant FVALUE_S    : STD_LOGIC_VECTOR(0 to 6) := "UUUUUZZ";


begin


        DRAIN_M <= MASK_NEW(DRAIN_S7,MASK_D,FVALUE_D);
        GATE_M  <= MASK_NEW(GATE_S7 ,MASK_G ,FVALUE_G);

end process;
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- -

SWITCHN: process(DRAIN_M,GATE_M,PREV_SOURCE_S7)
begin

   SOURCE_S7 <= SWITCH7_N(DRAIN_M, GATE_M, PREV_SOURCE_S7);

end process;

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - -
```

```
OUT_MASK: process(SOURCE_S7)
constant MASK_S    : STD_LOGIC_VECTOR(0 to 6) := "ZZZZZ10";
constant FVALUE_S  : STD_LOGIC_VECTOR(0 to 6) := "UUUUUZZ";
begin

        SOURCE_M <= MASK_NEW(SOURCE_S7,MASK_S,FVALUE_S);

end process;
-------------------------------------------------------------------
-------
end only;
```

A.2 Parallel Switch Fault Simulation Test_bench

The stimulus file use to test 9^3 combinations of input combination to the parallel switch fault simulation technique in appendix A..1.

```
use WORK.logic_system.all, WORK.pnew1.all,work.pnew2.all, work.pnew.all,
WORK.all;
entity TEST_BENCH
is end TEST_BENCH;

architecture N_PARALL_1 of TEST_BENCH is
  signal  DRAIN      : STD_LOGIC              := 'Z' ;
  signal  GATE       : STD_LOGIC              := 'Z' ;
  signal  PREV_SOURCE : STD_LOGIC             := 'Z' ;
  signal  SOURCE_M   : STD_LOGIC_VECTOR(0 to 6)  := "ZZZZZZZ";
  signal  STOP       : BIT                    := '0' ;
  signal  INIT       : BIT                    := '0' ;

  component N_PARALLA
    port (
        DRAIN          : in   STD_LOGIC          := 'Z';
        GATE           : in   STD_LOGIC          := 'Z';
        PREV_SOURCE    : in   STD_LOGIC          := 'Z';
     SOURCE_M   : out STD_LOGIC_VECTOR(0 to 6)  := "ZZZZZZZ";
        STOP           : in   BIT                := '1');
  end component;
   for L1: N_PARALLA  use entity N_PARALL(only);
begin
  L1: N_PARALLA
     port map(DRAIN,GATE,PREV_SOURCE,SOURCE_M,STOP);
INIT <= '1';
process(INIT)
variable GDEL : time := 0 ns;
  begin
  for I in 0 to 8 loop
      for J in 0 to 8 loop
           for K in 0 to 8 loop
             GDEL := GDEL + 1 ns;
              GATE       <= transport STD_LOGIC'val(I) after GDEL;
            DRAIN      <= transport STD_LOGIC'val(J) after GDEL;
           PREV_SOURCE <= transport STD_LOGIC'val(K) after GDEL;
        end loop;
     end loop;
  end loop;
  if (GDEL > 730 ns) then stop <= '1';
  end if;
end process;
end N_PARALL_1;
```

A.3 Parallel Switch Level Fault Simulation Results

Output results that verify the parallel switch level fault simulation technique. The output printed here has been condensed to the more interesting respounces (0,1,L,H).

n-type switch

```
N_PARALLA     COMPONENT                (no value)
M3          CE ACTIVE /TEST_BENCH/SOURCE_M
M2          CE ACTIVE /TEST_BENCH/PREV_SOURCE
M1          CE ACTIVE /TEST_BENCH/GATE
M           CE ACTIVE /TEST_BENCH/DRAIN
0 NS
    M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "ZZZZZ10")
    M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "WWWWW10")
    M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "WWZWX10")
165 NS
    M2:   ACTIVE /TEST_BENCH/PREV_SOURCE (value = '0')
     M:   ACTIVE /TEST_BENCH/DRAIN (value = 'U')
    M1:   ACTIVE /TEST_BENCH/GATE (value = '0')
    M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "LULLL10")
166 NS
    M2:   ACTIVE /TEST_BENCH/PREV_SOURCE (value = '1')
     M:   ACTIVE /TEST_BENCH/DRAIN (value = 'U')
    M1:   ACTIVE /TEST_BENCH/GATE (value = '0')
    M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "HUHHH10")
262 NS
    M2:   ACTIVE /TEST_BENCH/PREV_SOURCE (value = 'U')
     M:   ACTIVE /TEST_BENCH/DRAIN (value = '0')
    M1:   ACTIVE /TEST_BENCH/GATE (value = '1')
    M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "XXUH010")
    M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "00UH010")
263 NS
    M2:   ACTIVE /TEST_BENCH/PREV_SOURCE (value = 'X')
     M:   ACTIVE /TEST_BENCH/DRAIN (value = '0')
    M1:   ACTIVE /TEST_BENCH/GATE (value = '1')
    M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "00WH010")
264 NS
    M2:   ACTIVE /TEST_BENCH/PREV_SOURCE (value = '0')
     M:   ACTIVE /TEST_BENCH/DRAIN (value = '0')
    M1:   ACTIVE /TEST_BENCH/GATE (value = '1')
    M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "00LH010")
265 NS
    M2:   ACTIVE /TEST_BENCH/PREV_SOURCE (value = '1')
     M:   ACTIVE /TEST_BENCH/DRAIN (value = '0')
    M1:   ACTIVE /TEST_BENCH/GATE (value = '1')
    M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "00HH010")
271 NS
```

```
   M2:  ACTIVE /TEST_BENCH/PREV_SOURCE (value = 'U')
    M:  ACTIVE /TEST_BENCH/DRAIN (value = '1')
   M1:  ACTIVE /TEST_BENCH/GATE (value = '1')
   M3:  ACTIVE /TEST_BENCH/SOURCE_M (value = "00UH010")
   M3:  ACTIVE /TEST_BENCH/SOURCE_M (value = "HHUH010")
273 NS
   M2:  ACTIVE /TEST_BENCH/PREV_SOURCE (value = '0')
    M:  ACTIVE /TEST_BENCH/DRAIN (value = '1')
   M1:  ACTIVE /TEST_BENCH/GATE (value = '1')
   M3:  ACTIVE /TEST_BENCH/SOURCE_M (value = "HHLH010")
300 NS
   M2:  ACTIVE /TEST_BENCH/PREV_SOURCE (value = '0')
    M:  ACTIVE /TEST_BENCH/DRAIN (value = 'L')
   M1:  ACTIVE /TEST_BENCH/GATE (value = '1')
   M3:  ACTIVE /TEST_BENCH/SOURCE_M (value = "LLLH010")
301 NS
   M2:  ACTIVE /TEST_BENCH/PREV_SOURCE (value = '1')
    M:  ACTIVE /TEST_BENCH/DRAIN (value = 'L')
   M1:  ACTIVE /TEST_BENCH/GATE (value = '1')
   M3:  ACTIVE /TEST_BENCH/SOURCE_M (value = "LLHH010")
544 NS
   M2:  ACTIVE /TEST_BENCH/PREV_SOURCE (value = '1')
    M:  ACTIVE /TEST_BENCH/DRAIN (value = 'L')
   M1:  ACTIVE /TEST_BENCH/GATE (value = 'L')
   M3:  ACTIVE /TEST_BENCH/SOURCE_M (value = "HLHHH10")
586 NS
   M2:  ACTIVE /TEST_BENCH/PREV_SOURCE (value = 'U')
    M:  ACTIVE /TEST_BENCH/DRAIN (value = '0')
   M1:  ACTIVE /TEST_BENCH/GATE (value = 'H')
   M3:  ACTIVE /TEST_BENCH/SOURCE_M (value = "XXUH010")
   M3:  ACTIVE /TEST_BENCH/SOURCE_M (value = "00UH010")
587 NS
   M2:  ACTIVE /TEST_BENCH/PREV_SOURCE (value = 'X')
    M:  ACTIVE /TEST_BENCH/DRAIN (value = '0')
   M1:  ACTIVE /TEST_BENCH/GATE (value = 'H')
   M3:  ACTIVE /TEST_BENCH/SOURCE_M (value = "00WH010")
894 NS
   M2:  ACTIVE /TEST_BENCH/PREV_SOURCE (value = '0')
    M:  ACTIVE /TEST_BENCH/DRAIN (value = 'U')
   M1:  ACTIVE /TEST_BENCH/GATE (value = '0')
   M3:  ACTIVE /TEST_BENCH/SOURCE_M (value = "LULLL10")
895 NS
   M2:  ACTIVE /TEST_BENCH/PREV_SOURCE (value = '1')
    M:  ACTIVE /TEST_BENCH/DRAIN (value = 'U')
   M1:  ACTIVE /TEST_BENCH/GATE (value = '0')
   M3:  ACTIVE /TEST_BENCH/SOURCE_M (value = "HUHHH10")
903 NS
   M2:  ACTIVE /TEST_BENCH/PREV_SOURCE (value = '0')
    M:  ACTIVE /TEST_BENCH/DRAIN (value = 'X')
   M1:  ACTIVE /TEST_BENCH/GATE (value = '0')
   M3:  ACTIVE /TEST_BENCH/SOURCE_M (value = "LXLLL10")
```

```
904 NS
    M2:   ACTIVE /TEST_BENCH/PREV_SOURCE (value = '1')
     M:   ACTIVE /TEST_BENCH/DRAIN (value = 'X')
    M1:   ACTIVE /TEST_BENCH/GATE (value = '0')
    M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "HXHHH10")
905 NS
    M2:   ACTIVE /TEST_BENCH/PREV_SOURCE (value = 'Z')
     M:   ACTIVE /TEST_BENCH/DRAIN (value = 'X')
    M1:   ACTIVE /TEST_BENCH/GATE (value = '0')
908 NS
    M2:   ACTIVE /TEST_BENCH/PREV_SOURCE (value = '0')
     M:   ACTIVE /TEST_BENCH/DRAIN (value = '0')
    M1:   ACTIVE /TEST_BENCH/GATE (value = '0')
    M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "L0LLL10")
913 NS
    M2:   ACTIVE /TEST_BENCH/PREV_SOURCE (value = '1')
     M:   ACTIVE /TEST_BENCH/DRAIN (value = '0')
    M1:   ACTIVE /TEST_BENCH/GATE (value = '0')
    M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "H0HHH10")
921 NS
    M2:   ACTIVE /TEST_BENCH/PREV_SOURCE (value = '0')
     M:   ACTIVE /TEST_BENCH/DRAIN (value = '1')
    M1:   ACTIVE /TEST_BENCH/GATE (value = '0')
    M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "LHLLL10")
922 NS
    M2:   ACTIVE /TEST_BENCH/PREV_SOURCE (value = '1')
     M:   ACTIVE /TEST_BENCH/DRAIN (value = '1')
    M1:   ACTIVE /TEST_BENCH/GATE (value = '0')
    M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "HHHHH10")
993 NS
    M2:   ACTIVE /TEST_BENCH/PREV_SOURCE (value = '0')
     M:   ACTIVE /TEST_BENCH/DRAIN (value = '0')
    M1:   ACTIVE /TEST_BENCH/GATE (value = '1')
    M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "00LH010")
994 NS
    M2:   ACTIVE /TEST_BENCH/PREV_SOURCE (value = '1')
     M:   ACTIVE /TEST_BENCH/DRAIN (value = '0')
    M1:   ACTIVE /TEST_BENCH/GATE (value = '1')
    M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "00HH010")
995 NS
    M2:   ACTIVE /TEST_BENCH/PREV_SOURCE (value = 'Z')
     M:   ACTIVE /TEST_BENCH/DRAIN (value = '0')
    M1:   ACTIVE /TEST_BENCH/GATE (value = '1')
    M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "00ZH010")
1002 NS
    M2:   ACTIVE /TEST_BENCH/PREV_SOURCE (value = '0')
     M:   ACTIVE /TEST_BENCH/DRAIN (value = '1')
    M1:   ACTIVE /TEST_BENCH/GATE (value = '1')
    M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "HHLH010")
1003 NS
    M2:   ACTIVE /TEST_BENCH/PREV_SOURCE (value = '1')
```

```
    M:   ACTIVE /TEST_BENCH/DRAIN (value = '1')
   M1:   ACTIVE /TEST_BENCH/GATE (value = '1')
   M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "HHHH010")
1004 NS
   M2:  ACTIVE /TEST_BENCH/PREV_SOURCE (value = 'Z')
    M:   ACTIVE /TEST_BENCH/DRAIN (value = '1')
   M1:   ACTIVE /TEST_BENCH/GATE (value = '1')
   M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "HHZH010")
1027 NS
   M2:  ACTIVE /TEST_BENCH/PREV_SOURCE (value = 'U')
    M:   ACTIVE /TEST_BENCH/DRAIN (value = 'L')
   M1:   ACTIVE /TEST_BENCH/GATE (value = '1')
   M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "WWUH010")
   M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "LLUH010")
1028 NS
   M2:  ACTIVE /TEST_BENCH/PREV_SOURCE (value = 'X')
    M:   ACTIVE /TEST_BENCH/DRAIN (value = 'L')
   M1:   ACTIVE /TEST_BENCH/GATE (value = '1')
   M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "LLWH010")
1029 NS
   M2:  ACTIVE /TEST_BENCH/PREV_SOURCE (value = '0')
    M:   ACTIVE /TEST_BENCH/DRAIN (value = 'L')
   M1:   ACTIVE /TEST_BENCH/GATE (value = '1')
   M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "LLLH010")
1030 NS
   M2:  ACTIVE /TEST_BENCH/PREV_SOURCE (value = '1')
    M:   ACTIVE /TEST_BENCH/DRAIN (value = 'L')
   M1:   ACTIVE /TEST_BENCH/GATE (value = '1')
   M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "LLHH010")
1316 NS
   M2:  ACTIVE /TEST_BENCH/PREV_SOURCE (value = 'X')
    M:   ACTIVE /TEST_BENCH/DRAIN (value = '0')
   M1:   ACTIVE /TEST_BENCH/GATE (value = 'H')
   M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "00WH010")
1317 NS
   M2:  ACTIVE /TEST_BENCH/PREV_SOURCE (value = '0')
    M:   ACTIVE /TEST_BENCH/DRAIN (value = '0')
   M1:   ACTIVE /TEST_BENCH/GATE (value = 'H')
   M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "00LH010")
1318 NS
   M2:  ACTIVE /TEST_BENCH/PREV_SOURCE (value = '1')
    M:   ACTIVE /TEST_BENCH/DRAIN (value = '0')
   M1:   ACTIVE /TEST_BENCH/GATE (value = 'H')
   M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "00HH010")
1319 NS
   M2:  ACTIVE /TEST_BENCH/PREV_SOURCE (value = 'Z')
    M:   ACTIVE /TEST_BENCH/DRAIN (value = '0')
   M1:   ACTIVE /TEST_BENCH/GATE (value = 'H')
   M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "00ZH010")
1324 NS
   M2:  ACTIVE /TEST_BENCH/PREV_SOURCE (value = 'U')
```

```
     M:   ACTIVE /TEST_BENCH/DRAIN (value = '1')
    M1:   ACTIVE /TEST_BENCH/GATE (value = 'H')
    M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "00UH010")
    M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "HHUH010")
1325 NS
    M2:  ACTIVE /TEST_BENCH/PREV_SOURCE (value = 'X')
     M:   ACTIVE /TEST_BENCH/DRAIN (value = '1')
    M1:   ACTIVE /TEST_BENCH/GATE (value = 'H')
    M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "HHWH010")
1326 NS
    M2:   ACTIVE /TEST_BENCH/PREV_SOURCE (value = '0')
     M:   ACTIVE /TEST_BENCH/DRAIN (value = '1')
    M1:   ACTIVE /TEST_BENCH/GATE (value = 'H')
    M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "HHLH010")
1327 NS
    M2:   ACTIVE /TEST_BENCH/PREV_SOURCE (value = '1')
     M:   ACTIVE /TEST_BENCH/DRAIN (value = '1')
    M1:   ACTIVE /TEST_BENCH/GATE (value = 'H')
    M3:   ACTIVE /TEST_BENCH/SOURCE_M (value = "HHHH010")
```

Appendix B

Circuit Partitioning Example Netlist

B.1 C17

The netlist that follows is the reverse level ordered, grouped, example netlist of the ISCAS85 benchmark circuit C17. Level 1 is primary output.

```
/**************  Level  6  group  1  ****************/
n c17_ch_3_0/I3gat GND c17_row2_0/ai2s_1/6_8_22#
/**************  Level  5  group  1  ****************/
p c17_ch_3_0/I3gat Vdd c17_ch_2_0/I10
p c17_ch_3_0/I6gat Vdd c17_ch_2_0/I10
n c17_ch_3_0/I6gat c17_row2_0/ai2s_1/6_8_22# c17_ch_2_0/I10
/**************  Level  5  group  2  ****************/
n c17_ch_1_0/I2gat GND c17_row1_0/ai2s_1/6_8_22#
/**************  Level  4  group  2  ****************/
n c17_ch_1_0/I1gat GND c17_row1_0/ai2s_0/6_8_22#
/**************  Level  4  group  1  ****************/
p c17_ch_2_0/I10 Vdd c17_ch_2_0/I12
n c17_ch_2_0/I10 c17_row1_0/ai2s_1/6_8_22# c17_ch_2_0/I12
p c17_ch_1_0/I2gat Vdd c17_ch_2_0/I12
/**************  Level  3  group  2  ****************/
p c17_ch_1_0/I1gat Vdd c17_ch_2_0/I8
p c17_ch_3_0/I3gat Vdd c17_ch_2_0/I8
n c17_ch_3_0/I3gat c17_row1_0/ai2s_0/6_8_22# c17_ch_2_0/I8
/**************  Level  3  group  3  ****************/
p c17_ch_3_0/I7gat Vdd c17_row2_0/aoi21s_0/6_4_86#
p c17_ch_2_0/I10 Vdd c17_row2_0/aoi21s_0/6_4_86#
n c17_ch_2_0/I10 GND c17_row2_0/aoi21s_0/6_12_18#
/**************  Level  3  group  1  ****************/
n c17_ch_2_0/I12 GND c17_ch_2_0/I13
p c17_ch_2_0/I12 Vdd c17_ch_2_0/I13
/**************  Level  2  group  1  ****************/
n c17_ch_2_0/I8 GND c17_row2_0/ai2s_0/6_8_22#
/**************  Level  2  group  2  ****************/
n c17_ch_2_0/I13 GND c17_ch_2_0/I7
p c17_ch_2_0/I13 c17_row2_0/aoi21s_0/6_4_86# c17_ch_2_0/I7
n c17_ch_3_0/I7gat c17_row2_0/aoi21s_0/6_12_18# c17_ch_2_0/I7
/**************  Level  1  group  1  ****************/
p c17_ch_2_0/I8 Vdd c17_ch_2_0/I22gat
p c17_ch_2_0/I12 Vdd c17_ch_2_0/I22gat
n c17_ch_2_0/I12 c17_row2_0/ai2s_0/6_8_22# c17_ch_2_0/I22gat
/**************  Level  1  group  2  ****************/
n c17_ch_2_0/I7 GND c17_ch_1_0/I23gat
p c17_ch_2_0/I7 Vdd c17_ch_1_0/I23gat
```

Appendix C

C Code for Circuit Partitioning

Appendix C.1 Forward Preprocessing

The program that follows preprocesses a switch level netlist which has been extracted from layout. The preprocessing is a forward pass from primary input to primary output. This pass determines switch inputs and outputs.

```
/*****************************************************//*
* /
/ *     Program to forward preprocess circuit      * /
/*      Christopher A. Ryan                  * /
/ *     3/19/92                                    * /
/*      VPI&SU                                     * /
/*                                 */
/*                                          * /
/*****************************************************/


#include <stdio.h>
#include <string.h>
FILE *fm;FILE *fmn;FILE *fmtt;FILE *pd;FILE *pda;FILE *pd1a;FILE *pdn;FILE
*pdd;FILE *sd;FILE *mt;FILE *mtab;FILE *mttoo;FILE *lu;FILE *pi1;FILE *pi2;FILE
*pi3;FILE *pi4;FILE *pi5;FILE *pi6;FILE *pi7;FILE *pi8;FILE *pi9;

/******     Node
Parameters*************************************************************
*****/

char   primary_input[100]       ;       /* Primary output to search for*/
char   transistor[100]     ;       /* Transistor type N or P*/
char   gate[100]           ;       /* Gate of Transistor*/
char   drain[100]          ;       /* Drain of Transistor*/
char   source[100]         ;       /* Source of Transistor*/
char   transistor_ex[100] ;     /* Transistor type N or P                    */
char   gate_ex[100]          ;     /* Gate of Transistor                  */
char   drain_ex[100]          ;     /* Drain of Transistor                  */
char   source_ex[100]         ;     /* Source of Transistor                 */
char   gate_temp[100]      ;       /* Gate Temporary Storage*/
char   gate_new[100]       ;       /* Gate New Name*/
char   drain_new[100]      ;       /* Drain New Name*/
char   source_new[100]     ;       /* Source New Name*/
char   previous_in_1[100]          ;       /* Previous label input for current
transistor     * /
char   previous_in_2[100]          ;       /* Previous label input for current
transistor    */
char   dummy_1[100]       ;       /* Dummy variable*/
char   dummy_2[100]       ;       /* Dummy variable*/
char   dummy_3[100]       ;       /* Dummy variable*/
char   dummy_4[100]       ;       /* Dummy variable*/
char   dummy_5[100]    ;     /* Dummy variable                              */
```

```
char    check_name_1[100] ;
char     check_name_2[100] ;
char    check_name_3[100];
char    new_po_name[100];
char     new_po_name_2[100];
char     new_po_name_3[100];
int     L               ;       /* Level */
int     code            ;
int    code2        ;
int     check           ;
/******  Node
Parameters*************************************************************
*************/
main ()
{
        L = 1 ;
        pi1 = popen("csh ","w");
        fprintf(pi1,"new1.scr\n");
        pclose(pi1);
        pda = fopen("netlist.not","r");
        fm = fopen("netlist.not1","w");
        strcpy(dummy_5," new");
        while(
fscanf(pda,"%s%s%s%s%s%s%s%s\n",transistor,gate,drain,source,dummy_1,dummy
_2,dummy_3,dummy_4) != EOF)
        {
                fprintf(fm,"%s %s %s %s %s %s %s %s
%s\n",transistor,gate,drain,source,dummy_1,dummy_2,dummy_3,dummy_4,dummy_
5);

        }
        fclose(pda);
        fclose(fm);
       pi3 = popen("csh ","w");
          fprintf(pi3,"new2.scr\n");
       pclose(pi3);
        Strip_columbs();
        Primary_input();


repeat:         pdn = fopen("netlist_next.sim1","r");
        if (fscanf(pdn,"%s%s%s%s\n",transistor,gate,drain,source) == EOF)
        {       fclose(pdn);
                exit(0);
        }
        else
        {
                fclose(pdn);
                pi5 = popen("csh","w");
                fprintf(pi5,"scr.file\n");
                pclose(pi5);
```

```
        Primary_input();
        }
        goto repeat;
}
/****** Strip last 4 columbs from netlist  ******/
Strip_columbs()
{
        pd = fopen("netlist.sim1","w");
        pd1a = fopen("netlist.sim1a","w");
        fm = fopen("netlist.sim_new","r");
        while(fscanf(fm,"%s%s%s%s%s%s%s%s%s\n",transistor,gate,drain,source,d
ummy_1,dummy_2,dummy_3,dummy_4,dummy_5) != EOF )
        {
        if ( (strcmp(source,"Vdd") && strcmp(source,"GND")) == 0)
        {
                        strcpy(drain,source);
                        strcpy(source,dummy_1);
        }
                else if ( (strcmp(dummy_1,"Vdd") && strcmp(dummy_1,"GND")) ==
0 )
                {
                        strcpy(source,dummy_1);
                }
                fprintf(pd,"%s %s %s %s\n",transistor,gate,drain,source);
                fprintf(pd1a,"%s %s %s %s\n",transistor,gate,drain,source);
        }
        fclose(pd);
        fclose(pd1a);
        fclose(fm);
        return;
        }
/***** Locating transistors connected to primary input for this level *******/

Primary_input()
{
         fm = fopen("primary.input","r");
         while(fscanf(fm,"%s\n",primary_input) != EOF )
        {
                pd = fopen("netlist.sim1","r");
                pdn = fopen("netlist_next.sim1","w");
                sd = fopen("netlist.sim2","a");
                check = 0;
                while ( fscanf(pd,"%s%s%s%s\n",transistor,gate,drain,source) != EOF)
                {
                        if ( strcmp(source,"Vdd") && strcmp(source,"GND") == 0 )
                    {
                            strcpy(new_po_name,drain);
                            strcpy(new_po_name_2,gate);
                            strcpy(new_po_name_3,gate);
add_to_list(new_po_name,new_po_name_2,new_po_name_3,transistor,gate,drain,sou
rce);
```

```
                        if(code == 0)
                        {
                                fprintf(sd,"%s  %s %s
%s\n",transistor,gate,source,drain);
                                check = 1;
                        }
                        else
                        {
                                fprintf(pdn,"%s %s %s
%s\n",transistor,gate,source,drain);
                                check = 1;
                        }
                goto next2;
                }
                        if ( strcmp(drain,"Vdd") && strcmp(drain,"GND") == 0  )
                {
                        strcpy(new_po_name,source);
                        strcpy(new_po_name_2,gate);
                        strcpy(new_po_name_3,gate);
add_to_list(new_po_name,new_po_name_2,new_po_name_3,transistor,gate,drain,sou
rce);
                if(code == 0)
                {
                        fprintf(sd,"%s  %s %s %s\n",transistor,gate,drain,source);
                check = 1;
                }
                else
                {
                        fprintf(pdn,"%s %s %s %s\n",transistor,gate,drain,source);
                check = 1;
                        }
                goto next2;
                }

                if  (strcmp(primary_input,gate)  ==  0)
                {
                        strcpy(new_po_name,source);
                        strcpy(new_po_name_2,drain);
                        strcpy(new_po_name_3,gate);

add_to_list(new_po_name,new_po_name_2,new_po_name_3,transistor,gate,drain,sou
rce);
                        if(code  ==  0)
                        {
                                fprintf(sd,"%s   %s  %s
%s\n",transistor,gate,drain,source);
                                check = 1;
                        }
                        else
                        {
                                        strcpy(new_po_name,drain);
```

```
                                        strcpy(new_po_name_2,source);
                                        strcpy(new_po_name_3,gate);

add_to_list(new_po_name,new_po_name_2,new_po_name_3,transistor,gate,drain,sou
rce);
                        if(code == 0)
                        {
                                fprintf(sd,"%s    %s
%s%s\n",transistor,gate,source,drain);
                                check = 1;
                        }
                        else
                        {
                                fprintf(pdn,"%s  %s  %s
%s\n",transistor,gate,drain,source);
                                check = 1;
                        }
                }
                        goto next2;
                }
                if  (strcmp(primary_input,drain)  ==  0)
                {
                        if ( strcmp(source,"Vdd") && strcmp(source,"GND") == 0   )
                                {
                        strcpy(new_po_name,drain);
                        strcpy(new_po_name_2,gate);
                        strcpy(new_po_name_3,gate);
add_to_list(new_po_name,new_po_name_2,new_po_name_3,transistor,gate,drain,sou
rce);
                          if(code == 0)
                         {
                                fprintf(sd,"%s  %s %s
%s\n",transistor,gate,source,drain);
                                check = 1;
                         }
                         else
                         {
                                fprintf(pdn,"%s %s %s
%s\n",transistor,gate,source,drain);
                                check = 1;
                         }
                              }
                              else
                              {
                                        strcpy(new_po_name,source);
                                        strcpy(new_po_name_2,gate);
                                        strcpy(new_po_name_3,gate);

        add_to_list(new_po_name,new_po_name_2,new_po_name_3,transistor,gate,dr
ain,source);
                        if(code == 0)
```

```
                        {
                                fprintf(sd,"%s   %s  %s
%s\n",transistor,gate,drain,source);
                                        check = 1;
                        }
                        else
                        {
                                fprintf(pdn,"%s  %s  %s
%s\n",transistor,gate,drain,source);
                                check = 1;
                        }
                                }
                goto next2;
                }
                if  (strcmp(primary_input,source)  ==  0)
                {
                        if  ( strcmp(drain,"Vdd")  &&  strcmp(drain."GND")  ==  0  )
                        {
                        strcpy(new_po_name,source);
                        strcpy(new_po_name_2,gate);
                        strcpy(new_po_name_3,gate);
add_to_list(new_po_name,new_po_name_2,new_po_name_3,transistor,gate,drain,sou
rce);
                             if(code == 0)
                            {
                                fprintf(sd,"%s %s %s
%s\n",transistor,gate,drain,source);
                                check = 1;
                            }
                            else
                            {
                                fprintf(pdn,"%s %s %s
%s\n",transistor,gate,drain,source);
                                check = 1;
                            }
                        }
                        else
                        {
                              strcpy(new_po_name,drain);
                              strcpy(new_po_name_2,gate);
                              strcpy(new_po_name_3,gate);
add_to_list(new_po_name,new_po_name_2,new_po_name_3,transistor,gate,drain,sou
rce);
                             if(code == 0)
                        {
                                fprintf(sd,"%s  %s  %s
%s\n",transistor,gate,source,drain);
                                check = 1;
                        }
                        else
                        {
```

```
                        fprintf(pdn,"%s %s %s
%s\n",transistor,gate,source,drain);
                        check = 1;
                    }
                        }
            goto next2;
                    }
            if  (strcmp(primary_input,source)&&strcmp(primary_input,drain)  !=
0 )
            {
                fprintf(pdn,"%s %s %s %s\n",transistor,gate,source,drain);
                    }
next2:          ;           }
            if (check == 0)
            {
                    pi6 = popen("csh","w");
                    fprintf(pi6,"another.scr  %s\n",primary_input);
                    pclose(pi6);
                    fclose(pd);
            }
            fclose(pdn);
            fclose(sd);
            fclose(pd);
            pi7 = popen("csh","w");
            fprintf(pi7,"too.scr\n");
            pclose(pi7);

      }
     fclose(fm);
      L = L + 1;
      return;
}
/***************************** check new po list for name and add name
*****************/
add_to_list(new_po_name,new_po_name_2,new_po_name_3,transistor,gate,drain,sou
rce)
{
      pi4 = popen("csh","w");
        fprintf(pi4,"remove.scr %s %s %s %s\n",transistor,gate,drain,source);
     pclose(pi4);
       mtab = fopen("primary.output","r");
      fscanf(mtab,"%s\n",check_name_3);
      while(strcmp(check_name_3,new_po_name)  != 0)
      {
            if(fscanf(mtab,"%s\n",check_name_3)  == EOF)
                {
                    fclose(mtab);
                    goto tooo;
                }
      }
      fclose(mtab);
```

```
        code = 0;
        return;
tooo:    mt = fopen("primary.in2","r");
        while(fscanf(mt,"%s\n",check_name_3) != EOF)
     {
/***************  match  other  input
**************************************/
             if (strcmp(check_name_3,new_po_name_2) == 0)
              {
                    fclose(mt);
              pdd = fopen("primary.output","r");
              while(fscanf(pdd,"%s\n",check_name_1) != EOF)
              {
                          if (strcmp(check_name_1,new_po_name) == 0)
                          {      fclose(pdd);
                                 code = 0;
                                 code2 = 0;
                                 return ;
                          }
              }
              fclose(pdd);
                  pd1a = fopen("netlist.sim1a","r");

                        /***************  check  for  fan-in
*****************************/
        while(
fscanf(pd1a,"%s%s%s%s\n",transistor_ex,gate_ex,drain_ex,source_ex) != EOF )
              {
                    if(strcmp(gate_ex,new_po_name) == 0)
                          {
                                fclose(pd1a);
                          pd1a = fopen("netlist.sim1a","r");

       while(fscanf(pd1a,"%s%s%s%s\n",transistor_ex,gate_ex,drain_ex,source_ex
) != EOF)
                                {
       /*****************************  check  drain  or  source
*****************/
                                 if (strcmp(drain_ex,new_po_name) == 0)
                          {      fclose(pd1a);
                                 code = 0;
                                 code2 = 0;
                                 return ;
                          }
                              if (strcmp(source_ex,new_po_name) == 0)
                            {     fclose(pd1a);
                               code = 0;
                               code2 = 0;
                                return ;
                          }
                                 }
```

```
                                fclose(pd1a);
                                goto new;
                            }
                    }
                        fclose(pd1a);
new:                    fmn  =  fopen("primary.in2","r");
                    fscanf(fmn,"%s\n",check_name_2);
                    while ( strcmp(check_name_2,new_po_name) != 0)
                    {
                        if (fscanf(fmn,"%s\n",check_name_2) == EOF)
                        {
                                fclose(fmn);
                               fmn = fopen("primary.in2","a");
                            fprintf(fmn,"%s\n",new_po_name);
                           fclose(fmn);
                            fmtt  =  fopen("primary_next.input","a");
                                fprintf(fmtt,"%s\n",new_po_name);
                                fclose(fmtt);
                                code = 0;
                                code2 = 0;
                                return  ;
                            }
                        }
                        fclose(fmn);
                        code = 0;
                        code2 = 0;
                        return  ;
                    }
            }
            code = 1;
            code2 = 1;
            fclose(mt);
            return  ;
}
```

Appendix C.2 Reverse Level Ordering

The program that follows is used to reverse level order the switch level netlist after preprocessing.

```
/*****************************************************/
/*                                                   */
/*    Program to reverse levelize circuit            */
/*    Christopher A. Ryan                            */
/*    3/19/92                                        */
/*    VPI&SU                                         */
/*                                                   */
/*****************************************************/


#include <stdio.h>
#include <string.h>
FILE *fm;FILE *amn;FILE *pd;FILE *pda;FILE *bdn;FILE *cdn;FILE *ddd;FILE *sd;FILE
*mt;FILE *lu;FILE *pi1;FILE *pi2;

/******    Node
Parameters*************************************************************
*****/

char   primary_output[100]      ;       /* Primary output to search for*/
char   transistor[100]     ;       /* Transistor type N or P*/
char   gate[100]           ;       /* Gate of Transistor                    */
char   drain[100]          ;       /* Drain of Transistor
       */
char   source[100]         ;       /* Source of Transistor*/
char   gate_temp[100]      ;       /* Gate Temporary Storage*/
char   gate_new[100]       ;       /* Gate New Name*/
char   drain_new[100]      ;       /* Drain New Name*/
char   source_new[100]     ;       /* Source New Name*/
char   previous_in_1[100]          ;       /* Previous label input for current
transistor     */
char   previous_in_2[100]          ;       /* Previous label input for current
transistor   */
char   dummy_1[100]        ;       /* Dummy variable*/
char   dummy_2[100]        ;       /* Dummy variable*/
char   dummy_3[100]        ;       /* Dummy variable*/
char   dummy_4[100]        ;       /* Dummy variable*/
char    dummy_5[100]    ;      /* Dummy variable */
char   check_name_1[100] ;
char    check_name_2[100] ;
char   check_name_3[100];
char   new_po_name[100];
int    L               ;       /* Level */
```

```
/******  Node
Parameters**********************************************************
*************/
main ()
{
        L = 1 ;
        Primary_out();
repeat:         bdn  =  fopen("netlist_next.sim1","r");
        if (fscanf(bdn,"%s%s%s%s\n",transistor,gate,drain,source) == EOF)
        {       fclose(bdn);
                exit(0);
        }
        else
        {
                fclose(bdn);
                pi1 = popen("csh","w");
                fprintf(pi1,"scr.file\n");
                pclose(pi1);
                wait(200);
        Primary_out();
        }
        goto repeat;
}
Primary_out()
{
         fm = fopen("primary.out","r");
        sd = fopen("netlist.sim2","a");
        fprintf(sd,"/**************   Level   %d   ****************/\n",L);
         while(fscanf(fm,"%s\n",primary_output) != EOF )
        {
                pd = fopen("netlist.sim1","r");
                cdn = fopen("netlist_next.sim1","w");
                while ( fscanf(pd,"%s%s%s%s\n",transistor,gate,drain,source) != EOF)
                {
                        if ( strcmp(primary_output,source) == 0)
                        {
                                if( (strcmp(drain,"Vdd") && strcmp(drain,"GND") != 0)
)
                                {
                                        strcpy(new_po_name,drain);
                                        add_to_list(new_po_name);
                                }
                                if( (strcmp(gate,"Vdd") && strcmp(gate,"GND") != 0) )
                                {
                                        strcpy(new_po_name,gate);
                                        add_to_list(new_po_name);
                                }
                                fprintf(sd,"%s              %s          %s
        %s\n",transistor,gate,drain,source);

                        }
```

```
                    else
                    fprintf(cdn,"%s %s %s %s\n",transistor,gate,drain,source);
                }
        }
        fclose(pd);
        fclose(cdn);
        fclose(sd);
        pi2 = popen("csh","w");
        fprintf(pi2,"too.scr\n");
        pclose(pi2);
       fclose(fm);
        wait(200);
        L = L + 1;
       return;
}
/****************************** check new po list for name and add name
*****************/
add_to_list(new_po_name)
{
        ddd = fopen("primary.input","r");
        amn = fopen("primary_next.out","r");
       while(fscanf(amn,"%s\n",check_name_2) != EOF)
     {
              if (strcmp(check_name_2,new_po_name) == 0)
               {      fclose(amn);
                     fclose(ddd);
                     return;
              }
     }
       while(fscanf(ddd,"%s\n",check_name_1) != EOF)
     {
            if (strcmp(check_name_1,new_po_name) == 0)
         {      fclose(ddd);
                    fclose(amn);
               return;
          }
       }
      fclose(ddd);
       fclose(amn);
       mt = fopen("primary_next.out","a");
       fprintf(mt,"%s\n",new_po_name);
       fclose(mt);
       return;
}
```

Appendix C.3 Grouping C-code

The program that follows is used to group transistors in a reverse level

```
/*****************************************************/
/*                                                   */
/*    Program to group transistors in a reverse level circuit    */
/*    Christopher A. Ryan                            */
/*    7/23/92                                        */
/*    VPI&SU                                         */
/*                                                   */
/*****************************************************/
#include <stdio.h>
#include <string.h>
FILE *am;
FILE *fm;
FILE *amn;
FILE *pd;FILE *pda;FILE *bdn;FILE *cdn;FILE *ddd;FILE *sd;FILE *ad;FILE *ns2;FILE
*mt;
FILE *lu;FILE *pi1;FILE *pi2;FILE *pi3;FILE *pi4;FILE *pi6;FILE *pi9;
/******    Node
Parameters*************************************************************
*****/
char   primary_output[100]      ;      /* Primary output to search for*/
char   transistor[100]     ;      /* Transistor type N or P*/
char   gate[100]           ;      /* Gate of Transistor */
char   drain[100]          ;      /* Drain of Transistor*/
char   source[100]         ;      /* Source of Transistor*/
char   gate_temp[100]      ;      /* Gate Temporary Storage*/
char   gate_new[100]       ;      /* Gate New Name*/
char   drain_new[100]      ;      /* Drain New Name*/
char   source_new[100]     ;      /* Source New Name*/
char   previous_in_1[100]        ;      /* Previous label input for current
transistor    */
char   previous_in_2[100]        ;      /* Previous label input for current
transistor   */
char   dummy_1[100]      ;      /* Dummy variable*/
char   dummy_2[100]      ;      /* Dummy variable*/
char   dummy_3[100]      ;      /* Dummy variable*/
char   dummy_4[100]      ;      /* Dummy variable*/
char   dummy_5[100]   ;     /* Dummy variable                              */
char   check_name_1[100] ;
char    check_name_2[100] ;char  check_name_3[100];char   new_po_name[100];
int    L             ;      /* Level */
int    check         ;
int    g             ;
int    Lold          ;
```

```
/******  Node
Parameters**********************************************************
**************/
main ()
{
        L = 0 ;
        g = 1;
        bdn  =  fopen("netlist.sim1","r");
         ns2 = fopen("netlist_next.sim1","a");
         while ( fscanf(bdn,"%s%s%s%s\n",transistor,gate,drain,source) != EOF)
      {
              if ( strcmp(gate,"Level") == 0)
           {
                 fclose(ns2);
               pi1  =  popen("csh","w");
               fprintf(pi1,"rm   primary.out\n");
               pclose(pi1);
               ns2  =  fopen("netlist_next.sim1","r");
                       while
(fscanf(ns2,"%s%s%s%s\n",transistor,gate,drain,source)!=  EOF)
                       {
                              fclose(ns2);
                       fm  =  fopen("primary.out","w");
                       fprintf(fm,"%s\n",source);
                       fclose(fm);
                       Primary_out();
                              ns2  =  fopen("netlist_next.sim1","r");
                              g = g + 1;
                       }
                 fclose(ns2);
                L = L + 1;
                       g = 1;
               ns2  =  fopen("netlist_next.sim1","a");
           }
               else
               {
                              fprintf(ns2,"%s %s %s
%s\n",transistor,gate,drain,source);
               }
        }
       fclose(bdn);
       fclose(ns2);
        pi1 = popen("csh","w");
           fprintf(pi1,"rm  primary.out\n");
       pclose(pi1);
         ns2 = fopen("netlist_next.sim1","r");
         while (fscanf(ns2,"%s%s%s%s\n",transistor,gate,drain,source)!= EOF)
      {
            fclose(ns2);
              fm = fopen("primary.out","w");
               fprintf(fm,"%s\n",source);
```

```
            fclose(fm);
              Primary_out();
                ns2 = fopen("netlist_next.sim1","r");
            g = g + 1;
      }
      L = L + 1;
      g = 1;
         ns2 = fopen("netlist_next.sim1","a");
        fclose(ns2);
       exit(0);
}
Primary_out()
{
        check = 0;
         fm = fopen("primary.out","r");
        sd = fopen("netlist.sim2","a");
        fprintf(sd,"/*************** Level %d group %d
*****************/\n",L,g);
        while(fscanf(fm,"%s\n",primary_output) != EOF )
        {
              ns2 = fopen("netlist_next.sim1","r");
             ddd = fopen("netlist_nextL.sim1","a");
              while ( fscanf(ns2,"%s%s%s%s\n",transistor,gate,drain,source) != EOF)
          {
                      if (strcmp(primary_output,gate) == 0)
                      {
                              fprintf(sd,"%s %s %s
%s\n",transistor,gate,drain,source);
                              add_to_list(drain);
                              add_to_list(source);
                              goto next;
                      }
                      if (strcmp(primary_output,drain) == 0)
                      {
                              fprintf(sd,"%s %s %s
%s\n",transistor,gate,drain,source);
                              add_to_list(gate);
                              add_to_list(source);
                              goto next;
                      }
                      if (strcmp(primary_output,source) == 0)
                      {
                              fprintf(sd,"%s %s %s
%s\n",transistor,gate,drain,source);
                              add_to_list(drain);
                              add_to_list(gate);
                              goto next;
                      }
                              else
                      {
```

```
                                fprintf(ddd,"%s %s %s
%s\n",transistor,gate,drain,source);
                        }
                        next: ;
                }
                fclose(ns2);
                fclose(ddd);
        pi1 = popen("csh","w");
          fprintf(pi1,"cp netlist_nextL.sim1 netlist_next.sim1\n rm
netlist_nextL.sim1\n");
       pclose(pi1);
        }
        fclose(sd);
       fclose(fm);
        return;
}
/****************************** check new po list for name and add name
*******************/
add_to_list(new_po_name)
{
        pda = fopen("primary.out","r");
        if (strcmp("Vdd",new_po_name) == 0)
                        {
                                fclose(pda);
                        return;
                }
        if (strcmp("GND",new_po_name) == 0)
                        {
                        fclose(pda);
                        return;
                }
        while(fscanf(pda,"%s\n",check_name_2) != EOF)
    {
                if (strcmp(check_name_2,new_po_name) == 0)
                        {
                        fclose(pda);
                        return;
                }

    }
        fclose(pda);
        mt = fopen("primary.out","a");
        fprintf(mt,"%s\n",new_po_name);
        fclose(mt);
        return;
}
```

Appendix C.4 Subgrouping C code

The program that follows is used to subgroup groups of transistors in a reverse level.

```
/*****************************************************************
*********************/
/*                                                                 */
/*     Program to Subgroup a group of transistors in a reverse level circuit     */
/*     Christopher A. Ryan                                         */
/*     3/19/92                                                     */
/*     VPI&SU                                                      */
/*                                                                 */
/*****************************************************************
*********************/
#include <stdio.h>
#include <string.h>
FILE *am;FILE *fm;FILE *amn;FILE *pd;FILE *pda;FILE *bcd;FILE *bdn;FILE
*cdn;FILE *ddd;
FILE *sd;FILE *ad;FILE *mt;FILE *lu;FILE *ns2;FILE *pi1;FILE *pi2;FILE *pi3;FILE
*pi4;FILE  *pi6;
FILE *pi9;
/*******    Node
Parameters*************************************************************
*****/
char   primary_output[100]      ;      /* Primary output to search for*/
char   transistor[100]     ;      /* Transistor type N or P*/
char   gate[100]           ;      /* Gate of Transistor*/
char   drain[100]          ;      /* Drain of Transistor*/
char   source[100]         ;      /* Source of Transistor*/
char    transistor_temp[100] ;     /* Transistor type N or P   */
char   gate_temp[100]           ;     /* Gate of Transistor   */
char   drain_temp[100]           ;     /* Drain of Transistor   */
char   source_temp[100]          ;     /* Source of Transistor */
char     extra1[100]  ;char    extra2[100] ;
char   gate_temp[100]     ;      /* Gate Temporary Storage*/
char   gate_new[100]      ;      /* Gate New Name*/
char   drain_new[100]     ;      /* Drain New Name*/
char   source_new[100]    ;      /* Source New Name*/
char   previous_in_1[100]         ;      /* Previous label input for current
transistor    * /
char   previous_in_2[100]         ;      /* Previous label input for current
transistor   */
char   dummy1[100]        ;      /* Dummy variable*/
char   dummy2[100]        ;      /* Dummy variable*/
char   group[100]    ;
char   dummy_3[100]       ;      /* Dummy variable*/
char   dummy_4[100]       ;      /* Dummy variable*/
char   dummy_5[100]   ;     /* Dummy variable                          */
char   check_name_1[100] ;
```

```
char    check_name_2[100] ;char   check_name_3[100];char   Level[100];
int     L                 ;       /* Level */
int     check             ;int    count           ;
int     load[6000]        ;int    g               ;int    sg              ;
float   max               ;float  new             ;float  perload                 ;
float   cumul             ;float  average                 ;float  levels                  ;
/******  Node
Parameters*************************************************************
**************/
main ()
{
        L = 1 ;
        Count_load();
        Subgroup();
}
Count_load()
{
        check = 1;
        strcpy(Level,"Level");
        count = 0;
        fm = fopen("count.back","r");
        sd = fopen("count.stat","a");
         while(fscanf(fm,"%s%s%s%s\n",transistor,gate,drain,source) != EOF )
        {
                if ( strcmp(gate,Level) == 0)
                {
                        fscanf(fm,"%s%s\n",extra1,extra2);
                        check = 0;
                        load[L] = count;
                        count = 0;
                   L = L + 1;
                }
                else
                {
                        count = count + 1;
                }
        }
                levels = L;
                L = 1;
                max = 0;
                perload = 0;
                cumul = 0;
                while (L < levels)
                {
                    new = load[L];
                        if ( new > max )
                        {
                                max = new;
                        }
                        L = L + 1;
                }
```

```
                L = 1;
              while (L < levels)
             {
                    new = load[L];
                        perload  =  (new/max)*100;
                        cumul = perload + cumul;
                   L = L + 1;
             }
                  average  =  (cumul/levels)*max/100;
        fclose(fm);
         fclose(sd);
}
Subgroup()
{
        L = 1 ;
        sg = 1;
        g = 1;
        bdn  =  fopen("netlist.sim1","r");
         ns2 = fopen("netlist_next.sim1","a");
         while ( fscanf(bdn,"%s%s%s%s\n",transistor,gate,drain,source) != EOF)
      {
                if (load[L] >= 32)
                {
                        if (strcmp(gate,"Level") == 0)
                        {
                                strcpy(transistor_temp,transistor);
                                strcpy(gate_temp,gate);
                                strcpy(drain_temp,drain);
                                strcpy(source_temp,source);
                                fscanf(bdn,"%s%s\n",dummy1,dummy2);
                                        fclose(ns2);
                                pi1  =  popen("csh","w");
                                fprintf(pi1,"rm   primary.out\n");
                                pclose(pi1);
                                ns2  =  fopen("netlist_next.sim1","r");

        while(fscanf(ns2,"%s%s%s%s\n",transistor,gate,drain,source)!=  EOF)
                        {
                                fclose(ns2);
                                fm  =  fopen("primary.out","w");
                                fprintf(fm,"%s\n",source);
                                fclose(fm);
                                        Primary_out();
                                ns2  =  fopen("netlist_next.sim1","r");
                                sg = sg + 1;
                        }
                                bcd  =  fopen("netlist.sim2","a");
                        fprintf(bcd,"%s %s %s %s
%s%s\n",transistor_temp,gate_temp,
                        drain_temp,source_temp,dummy1,dummy2);
                        fclose(bcd);
```

```
                        fclose(ns2);
                        L = L + 1;
                g = 1;
                sg = 1;
                ns2 = fopen("netlist_next.sim1","a");
        }
                else
                {
                        fprintf(ns2,"%s %s %s
%s\n",transistor,gate,drain,source);
                }
        }
        else
        {
                bcd = fopen("netlist.sim2","a");
                if (strcmp(gate,"Level") == 0)
                        {
                        fscanf(bdn,"%s%s\n",dummy1,dummy2);
                        strcpy(group,dummy1);
        fprintf(bcd,"%s %s %s %s %s
%s\n",transistor,gate,drain,source,dummy1,dummy2);
                        L = L + 1;
                        check = 0;

                }
                else
                {
                        fprintf(bcd,"%s %s %s
%s\n",transistor,gate,drain,source);
                }
                        fclose(bcd);
        }
            strcpy(group,dummy1);
      }
     fclose(bdn);
     fclose(ns2);
      exit(0);
}
Primary_out()
{
      check = 0;
       fm = fopen("primary.out","r");
      sd = fopen("netlist.sim2","a");
      fprintf(sd,"/*************** group %s subgroup %d
*****************/\n",group,sg);
      while(fscanf(fm,"%s\n",primary_output) != EOF )
      {
            ns2 = fopen("netlist_next.sim1","r");
           ddd = fopen("netlist_nextL.sim1","a");
            while ( fscanf(ns2,"%s%s%s%s%s\n",transistor,gate,drain,source) != EOF)
        {
```

```
                        if (strcmp(primary_output,source) == 0)
                        {
                                fprintf(sd,"%s %s %s
%s\n",transistor,gate,drain,source);
                                goto next;
                        }
                                else
                        {
                                fprintf(ddd,"%s %s %s
%s\n",transistor,gate,drain,source);
                        }
                        next: ;
                }
                fclose(ns2);
                fclose(ddd);
        pi1 = popen("csh","w");
        fprintf(pi1,"cp netlist_nextL.sim1 netlist_next.sim1\n rm
netlist_nextL.sim1\n");
        pclose(pi1);
        }
        fclose(sd);
        fclose(fm);
        return;
}
/****************************** check new po list for name and add name
*****************/
add_to_list(new_po_name)
{
        pda = fopen("primary.out","r");
        if (strcmp("Vdd",new_po_name) == 0)
                        {
                                fclose(pda);
                        return;
                }
        if (strcmp("GND",new_po_name) == 0)
                        {
                        fclose(pda);
                        return;
                }
        while(fscanf(pda,"%s\n",check_name_2) != EOF)
    {
                if (strcmp(check_name_2,new_po_name) == 0)
                        {
                        fclose(pda);
                        return;
                }
    }
        fclose(pda);
        mt = fopen("primary.out","a");
        fprintf(mt,"%s\n",new_po_name);
        fclose(mt);
```

```
        return;
}
```

Appendix C.5 Reverse Level order Statistics C code

The program that follows is used to extract reverse level sizes.

```
/*****************************************************************
****/
/*                                                              */
/*    Program for reverse level order statistics circuit        */
/*    Christopher A. Ryan                                       */
/*    6/30/92                                                   */
/*    VPI&SU                                                    */
/*                                                              */
/*****************************************************************
****/
#include <stdio.h>
#include <string.h>
FILE *am;FILE *fm;FILE *amn;FILE *pd;FILE *pda;FILE *bdn;FILE *cdn;FILE *ddd;FILE
*sd;
FILE *ad;FILE *mt;FILE *lu;FILE *pi1;FILE *pi2;FILE *pi3;FILE *pi4;FILE *pi6;FILE
*pi9;
/******    Node
Parameters*************************************************************
*****/
char   primary_output[100]      ;      /* Primary output to search for*/
char   transistor[100]     ;      /* Transistor type N or P*/
char   gate[100]           ;      /* Gate of Transistor*/
char   drain[100]          ;      /* Drain of Transistor*/
char   source[100]         ;      /* Source of Transistor*/
char   gate_temp[100]      ;      /* Gate Temporary Storage*/
char   gate_new[100]       ;      /* Gate New Name*/
char   drain_new[100]      ;      /* Drain New Name*/
char   source_new[100]     ;      /* Source New Name*/
char   previous_in_1[100]          ;      /* Previous label input for current
transistor    * /
char   previous_in_2[100]          ;      /* Previous label input for current
transistor   */
char   dummy_1[100]        ;      /* Dummy variable*/
char   dummy_2[100]        ;      /* Dummy variable*/
char   dummy_3[100]        ;      /* Dummy variable*/
char   dummy_4[100]        ;      /* Dummy variable*/
char   dummy_5[100]   ;     /* Dummy variable                              */
char   check_name_1[100] ;char     check_name_2[100] ;char  check_name_3[100];
char   Level[100];int      L              ;      /* Level */
int    check          ;int    count          ;int    load[120]     ;
float  max            ;
float  new            ;
float  perload              ;
float  cumul          ;
float  average              ;
```

```
float    levels                  ;
/******   Node
Parameters**********************************************************
*************/
main ()
{
        L = 0 ;
        Primary_out();
}
Primary_out()
{
        check = 1;
        strcpy(Level,"Level");
        count = 0;
        fm = fopen("fan.back","r");
        sd = fopen("fan.new","a");
        fprintf(sd,"level       width\n");
         while(fscanf(fm,"%s%s%s%s\n",transistor,gate,drain,source) != EOF )
        {
                if ( strcmp(gate,Level) == 0)
                {
                        if ( check == 0)
                        {
                                check = 1;
                        }
                        else
                        {
                                check = 0;
                                fclose(fm);
                                fprintf(sd,"%d              %d\n", L,count);
                                load[L] = count;
                                count = 0;
                         L = L + 1;
                        }
                }
                else
                {
                        count = count + 1;
                }
      }
       if (fscanf(fm,"%s%s%s%s\n",transistor,gate,drain,source) == EOF)
    {
                levels = L;
              fprintf(sd,"%d            %d\n", L,count);
                L = 1;
                max = 0;
                perload = 0;
                cumul = 0;
                while (L < levels)
                {
                    new = load[L];
```

```
                    if ( new > max )
                    {
                            max = new;
                    }
                    L = L + 1;
            }
                fprintf(sd,"maximum width = %f\n", max );
            L = 1;
          fprintf(sd,"Level      load\n");
         while (L < levels)
        {
                new = load[L];
                    perload  =  (new/max)*100;
                    cumul = perload + cumul;
                fprintf(sd,"%d          %f\n",L,perload );
             L = L + 1;
        }
            average = cumul/levels;
           fprintf(sd,"average = %f\n",average );
     }
    fclose(fm);
     fclose(sd);
     exit(0);

}
```

Appendix C.6 Group/Subgroup Reverse Level Order Statistics C code

The program that follows is used to extract group and subgroup sizes.

```
/*****************************************************************
***********/
/*                                                               */
/*    Program to get Statistics on Grouping and Subgrouping in a */
/*          reverse level ordered circuit                        */
/*    Christopher A. Ryan                                        */
/*    8/8/92                                                     */
/*    VPI&SU                                                     */
/*                                                               */
/*****************************************************************
***********/
#include <stdio.h>
#include <string.h>
FILE *am;FILE *fm;FILE *amn;FILE *pd;FILE *pda;FILE *bdn;FILE *cdn;FILE *ddd;
FILE *sd;FILE *ad;FILE *mt;FILE *lu;FILE *pi1;FILE *pi2;FILE *pi3;FILE *pi4;FILE
*pi6;FILE  *pi9;
/******     Node
Parameters*****************************************************
*****/
char   primary_output[100]      ;      /* Primary output to search for*/
char   transistor[100]     ;      /* Transistor type N or P*/
char   gate[100]           ;      /* Gate of Transistor*/
char   drain[100]          ;      /* Drain of Transistor*/
char   source[100]         ;      /* Source of Transistor*/
char    extra1[100]  ;char    extra2[100] ;
char   gate_temp[100]      ;      /* Gate Temporary Storage*/
char   gate_new[100]       ;      /* Gate New Name*/
char   drain_new[100]      ;      /* Drain New Name*/
char   source_new[100]     ;      /* Source New Name*/
char   previous_in_1[100]          ;      /* Previous label input for current
transistor    * /
char   previous_in_2[100]          ;      /* Previous label input for current
transistor   */
char   dummy_1[100]      ;      /* Dummy variable*/
char   dummy_2[100]      ;      /* Dummy variable*/
char   dummy_3[100]      ;      /* Dummy variable*/
char   dummy_4[100]      ;      /* Dummy variable*/
char    dummy_5[100]    ;     /* Dummy variable  */
char   check_name_1[100] ;char    check_name_2[100] ;
char   check_name_3[100];char   Level[100];
int    L               ;      /* Level  */
int    check           ;int    count          ;int    load[5000]      ;
float  max             ;float  new            ;float  perload              ;
float  cumul           ;float  average                ;float  levels                ;
```

```
/******  Node
Parameters**********************************************************
*************/
main ()
{
        L = 0 ;
        Primary_out();
}
Primary_out()
{
        check = 1;
        strcpy(Level,"Level");
        count = 0;
        fm  =  fopen("count.back","r");
        sd  =  fopen("count.stat","a");
        fprintf(sd,"level          width\n");
        while(fscanf(fm,"%s%s%s%s\n",transistor,gate,drain,source)  !=  EOF  )
        {
                if  ((  strcmp(gate,Level)  ==  0)||(strcmp(gate,"group")  ==  0)==1)
                {
                        fscanf(fm,"%s%s\n",extra1,extra2);
                        check = 0;
                        fprintf(sd,"%d              %d\n", L,count);
                        load[L] = count;
                        count = 0;
                    L = L + 1;
                }
                else
                {
                        count = count + 1;
                }
        }
                levels = L;
                L = 1;
                max = 0;
                perload = 0;
                cumul = 0;
                while (L < levels)
                {
                    new = load[L];
                        if ( new > max )
                        {
                                max = new;
                        }
                        L = L + 1;
                }
                     fprintf(sd,"maximum width = %f\n", max );
                L = 1;
              while (L < levels)
             {
                        new = load[L];
```

```
                    perload = (new/max)*100;
                    cumul = perload + cumul;
                L = L + 1;
            }
                average = (cumul/levels)*max/100;
              fprintf(sd,"average = %f\n",average );
        fclose(fm);
         fclose(sd);
         exit(0);
}
```

Appendix C.7 Compiled Code VHDL C code

The program that follows is used to convert the switch level reverse level ordered netlist to a VHDL structural netlist.

```
/*****************************************************/
/*                                                   */
/*    Program to translate RLO netlist to            */
/*                    VHDL netlist                   */
/*    Christopher A. Ryan                            */
/*    1/11/93                                        */
/*    VPI&SU                                         */
/*                                                   */
/*****************************************************/
#include <stdio.h>
#include <string.h>
FILE *fm;FILE *fm2;FILE *sd;FILE *sd_2;FILE *sd_4;FILE *sd_3;FILE *ss;FILE
*ss_2;
FILE *s_in;FILE *s_out;FILE *s_IO;FILE *s_IO_test;FILE *s_IO_test_port;FILE
*mid2;
/******    Node
Parameters*************************************************************
******/
char   transistor[100]     ;      /* Transistor type N or P*/
char     transistor2[100] ;
char   gate[100]           ;      /* Gate of Transistor*/
char   gate2[100]          ;
char   drain[100]          ;      /* Drain of Transistor*/
char   drain2[100]         ;
char   source[100]         ;      /* Source of Transistor*/
char   source2[100]          ;char   level[100]   ;
char     extra1[100]  ;char    extra12[100]    ;
char     extra2[100]  ;char    extra22[100]    ;
char   Level1[100];char      Level2[100];
int    level_number        ;int    level_num[10000]  ;
char   PD[2]         ;char  signal[10000][100];
int    pd_count[10000] ;int       number_l_pd[10000];
char     source_pd[10000][100];char      gate_pd[10000][100];
int    check          ;int    count          ;
int    number               ;int    number_f      ;
int   number_l      ;int    number_sig     ;
int    number_in_out   ;int  numbers_max_temp[300];
int    numbers_max[300];int     i;
int    in_i;int      j;int  k;int    l;int    m;int    n;int    o;int       width;
/******  Node
Parameters*************************************************************
*************/
main ()
{
```

```
        Primary_out();
}
Primary_out()
{
        check = 1;
        number = 1;
        number_f = 1;
        number_l = 1;
        number_sig = 0;
        strcpy(Level1,"********");
        strcpy(Level2,"/***************");
        strcpy(PD,"PD");
        strcpy(level,"1000");
        count = 0;
        i = 0;
        m = 0;
        width = 0;
        fm  =  fopen("trans.rrnet","r");
        fm2  =  fopen("trans.rrnet2","r");
        s_IO   = fopen("trans.IO","w");
        ss   = fopen("trans.signal","w");
        mid2   = fopen("trans.mid2","w");
        s_IO_test   = fopen("trans.IO_test","w");
        s_IO_test_port   = fopen("trans.IO_test_port","w");
/*****************  Primary      Inputs
*************************************************************/
        fprintf(s_IO,"-------------    inputs    ---------------\n");
        s_in    = fopen("trans.in","r");
        while(fscanf(s_in,"%s\n",signal[i])  !=  EOF)
        {
                fprintf(ss,"signal   %s   : STD_LOGIC   := 'Z';\n",signal[i]);
                fprintf(mid2,"%s  <=  transport  l_gat_in(%i)   ;  \n",signal[i],i);
                i = i + 1;
        }
        fprintf(s_IO,"l_gat_in : inout STD_LOGIC_VECTOR(0 to %d) ;\n",i-1);
        fclose(s_in);
/*****************  Primary      Outputs
*************************************************************/
        fprintf(s_IO,"-------------    outputs    ---------------\n");
        s_out  = fopen("trans.out","r");
        in_i  =  i-1;
        while(fscanf(s_out,"%s\n",signal[i])  !=  EOF)
        {
                fprintf(ss,"signal   %s   : STD_LOGIC   := 'Z';\n",signal[i]);
                fprintf(mid2,"l_gat_out(%d)  <=  transport  %s  ;  \n",i-1-
in_i,signal[i]);
                i = i + 1;
        }
        fprintf(s_IO,"l_gat_out : inout STD_LOGIC_VECTOR(0 to %d) ;\n",i-1-in_i);
        fclose(s_out);
        fclose(s_IO_test_port);
```

```
        fclose(s_IO_test);
        fclose(ss);
        fclose(mid2);
        fclose(s_IO);
        number_sig = i ;
        number_in_out = i ;
        sd  =  fopen("trans.rrrnet","w");
        ss  =  fopen("trans.signal","a");
/*****************   Read   Netlist
**************************************************************/
        while(fscanf(fm,"%s%s%s%s%s\n",transistor,gate,drain,source)  !=  EOF  )
        {
                if  (  strcmp(transistor,Level1)  ==  0)
                {
        fscanf(fm2,"%s%s%s%s%s%s%s\n",transistor2,gate2,drain2,source2,extra12,e
xtra22);
                        fscanf(fm,"%s%s\n",extra1,extra2);
                }
                else
                {
                        if  (  strcmp(transistor,Level2)  ==  0)
                        {
fscanf(fm2,"%s%s%d%s%s%s\n",transistor2,gate2,level_number,source2,extra12,e
xtra22);

                        fscanf(fm,"%s%s\n",extra1,extra2);
                                if  (  strcmp(drain,level)  !=  0)
                                {
                                        strcpy(level,drain);
                                        numbers_max_temp[m] = number - 1;
                                        if( width < number -1)
                                        {
                                                width = number -1;
                                        }
                                        else
                                        {
                                        }
                                        number = 1;
                                        m = m + 1;
                                }
                                else
                                {
                                }
                        }
                        else
                        {

      fscanf(fm2,"%s%s%s%s%s\n",transistor2,gate2,drain2,source2);
                                if  (  strcmp(transistor,PD)  ==  0)
                                {
                                        strcat(source,PD);
```

```
/***************  Record  Signals
**************************************************/
                    j = number_sig;
                    while( (strcmp(source,signal[j]) != 0) && (j >= 0))
                    {
                            j = j - 1;
                            if ( j == -1)
                            {
                                    number_sig = number_sig + 1;
                                    strcpy(signal[number_sig],source);
                                            pd_count[number_sig] = 1;
                                            number_l_pd[number_sig] = number_l;
                                            strcpy(source_pd[number_sig],source);
                                            strcpy(gate_pd[number_sig],gate);
                                            level_num[number_sig] = level_number;
                                    number_l = number_l + 1;
                            }
                            else
                            {
                            }
                    }
                                    if (j != -1)
                                    {
                                            pd_count[j] = pd_count[j] + 1;
                                    }
                                    else
                                    {
                                    }
                            }
                            else
                            {
/***************  Print  VHDL  Netlist
**************************************************/
                                    fprintf(sd,"T%d:%s_faulta\n       port
map(%s,%s,%s,FAULTS(%d),DETECT,DETECTED,SIM_GOOD,FAULT_SIM,INJECT,INJT_
SIM,NUMBER_CNTL,LEVELS_P1(%s),LEVELS(%s),NUMBER_%d,STOP);\n",number_f
,transistor,gate,drain,source,number_f,level,level,number);
                            number = number + 1;
                            number_f = number_f + 1;
                            }
                            j = number_sig;
                            while( (strcmp(gate,signal[j]) != 0) && (j >= 0))
                            {
                                    j = j - 1;
                                    if ( j == -1)
                                    {
                                            number_sig = number_sig + 1;
                                            strcpy(signal[number_sig],gate);
                                            pd_count[number_sig] = 0;
                                    }
                                    else
```

```
                                {
                                }
                        }
                        j = number_sig;
                        while( (strcmp(drain,signal[j]) != 0) && (j >= 0))
                        {
                        j = j - 1;
                        if ( j == -1)
                        {
                                number_sig = number_sig + 1;
                                strcpy(signal[number_sig],drain);
                                        pd_count[number_sig] = 0;
                        }
                        else
                        {
                        }
                        }
                        j = number_sig;
                        while( (strcmp(source,signal[j]) != 0) && (j >= 0))
                        {
                        j = j - 1;
                        if ( j == -1)
                        {
                                number_sig = number_sig + 1;
                                strcpy(signal[number_sig],source);
                                        pd_count[number_sig] = 0;
                        }
                        else
                        {
                        }
                        }
                }
        }

        }
/**************** Print VHDL   I / O
*****************************************************/
        s_IO = fopen("trans.IO","a");
        s_IO_test = fopen("trans.IO_test","a");
        fprintf(s_IO,"-------------------------------------\n");
        fprintf(s_IO,"FAULTS        : inout X01Z_DIMEN( 1 to %d);\n",number_f - 1);
        fprintf(s_IO_test,"signal FAULTS        : X01Z_DIMEN( 1 to %d);\n",number_f -
1);
        fprintf(s_IO,"MAX_NUMBERS  : out integer_array(0 to %d)    ;\n",m);
        fprintf(s_IO,"LEVELS_P1     : in BIT_VECTOR(1 to %d)     ;\n",m+1);
    f  printf(s_IO,"LEVEL         : in BIT_VECTOR(0 to %d)     ;\n",m);
/**************** Print VHDL I / O test_bench
*****************************************/
        fprintf(s_IO_test,"signal MAX_NUMBERS  : integer_array(0 to %d)    ;\n",m);
        fclose(s_IO);
        fclose(s_IO_test);
```

```
        mid2 = fopen("trans.mid2","a");
        numbers_max_temp[m] = number - 1;
        if( width < number -1)
        {
                width = number -1;
        }
        else
        {
        }
/***************** Assign Level Numbers
*************************************************/
        o = 1;
        n = m ;
        while ( n > 0)
      {
                n = n - 1;
                o = o + 1;
      }
/***************** Assign Transistor Numbers in level
*****************************************/
        n = 1 ;
        while ( n <= width )
        {
                fprintf(ss,"signal   NUMBER_%d  : integer := %d;\n",n,n);
                n = n + 1;
        }
        o = 1;
        n = m ;
/***************** Vdd      GND Maximum Level
*************************************************/
        fprintf(mid2,"Vdd <= transport '1' ;\n");
        fprintf(mid2,"GND <= transport '0' ;\n");
        fprintf(mid2,"MAX_LEVELS <= %d ; \n", m);
        fprintf(mid2,"MAX_NUMBERS(0 to %d) <= (\n          (0),",n);
        while ( n > 0)
        {
                fprintf(mid2,"(%d),",numbers_max_temp[n]);
                n = n - 1;
                o = o + 1;
        }
        fprintf(mid2,"\n    );\n");
        fclose(mid2);
        ss_2 = fopen("PD.number","w");
/***************** Declaring All Signals in VHDL
*****************************************/
        sd_3 = fopen("trans.rrrnet_pdF","w");
        l = number_in_out + 1;
        while(l <= number_sig  )
        {
                if ( pd_count[l] == 0)
                {
```

```
                fprintf(ss,"signal    %s    : STD_LOGIC := 'Z';\n",signal[l]);
                l = l + 1;
        }
        else
        {
                fprintf(ss,"signal    %s    : STD_LOGIC := 'Z';\n",signal[l]);
                fprintf(ss_2,"  %s  %d  \n",signal[l],pd_count[l]);
                l = l + 1;
        }
    }
    fclose(fm);
    fclose(fm2);
    fclose(sd);
    fclose(sd_3);
    fclose(ss);
    fclose(ss_2);
    exit(0);
}
```

Appendix D

VHDL Code Fault Simulation Verification

D.1 N-type Processor Element (PE) VHDL code

The program that follows models switch level parallel fault simulation for one N-type switch. This program differs from that in appendix A..1 in that this model is the building block that is used in the verification of the fault simulation algorithm. The program in A..1 is used only to verify the novel switch level parallel fault simulation technique for a stand alone switch.

```
-------------------------------------------------------
--            Parallel Fault simulation              --
--           name - N_FAULT_OPT                      --
--           chris ryan                              --
--           11-4-92                                 --
--                                                   --
-------------------------------------------------------
use WORK.logic_system.all, WORK.all ;
library WORK;

---------------------------------  PACKAGE  PNEW1  ------------
-----------------------------------------
package pnew1 is
type RESTYPEBBB is array (INTEGER range <>) of BIT;
function MASK_NEW (IN_T, MASK_IN, FVALUE_IN : std_LOGIC_vector(0 to 6) ) return
std_logic_vector;
function EXPAND1_7 (A : std_LOGIC) return std_LOGIC_vector;
function EXPAND_X01_1_7 (A : X01) return X01_vector;
function EXPAND_X01Z_1_7 (A : X01) return X01Z_vector;
end pnew1;
package body pnew1 is
function MASK_NEW (IN_T, MASK_IN, fvalue_IN: std_LOGIC_vector(0 to 6))return
  std_LOGIC_vector is
variable RES_VALUE: std_LOGIC_vector(0 to 6) := "ZZZZZZZ" ;
begin
RES_VALUE := CON(MIN_F(IN_T,fvalue_in),MASK_IN);
 return RES_VALUE;
 end MASK_NEW;
function EXPAND1_7 (A : std_LOGIC ) return std_LOGIC_vector is
variable RESOLVED_VALUE: std_LOGIC_vector(0 to 6) := "ZZZZZZZ" ;
  begin
    for I in 0 to 6 loop
            RESOLVED_VALUE(I) := A;
   end loop;
 return RESOLVED_VALUE;
 end EXPAND1_7;
function EXPAND_X01_1_7 (A : X01 ) return X01_vector is
variable RESOLVED_VALUE: X01_vector(0 to 6) := "XXXXXXX" ;
  begin
    for I in 0 to 6 loop
            RESOLVED_VALUE(I) := A;
```

```
   end loop;
 return RESOLVED_VALUE;
 end EXPAND_X01_1_7;
function EXPAND_X01Z_1_7 (A : X01 ) return X01Z_vector is
variable RESOLVED_VALUE: X01Z_vector(0 to 6) := "XXXXXXX" ;
 begin
   for I in 0 to 6 loop
            RESOLVED_VALUE(I) := A;
   end loop;
 return RESOLVED_VALUE;
 end EXPAND_X01Z_1_7;
end pnew1;

use WORK.logic_system.all, WORK.all ;
library  WORK;
---------------------------------     PACKAGE     PNEW8     ------------
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
package pnew8 is
type X01Z_DIMEN is array (natural range <>) of X01Z_VECTOR(1 to 6);
function SWITCH_N (IN_D, IN_G, PREV_OUT_S :STD_LOGIC) return STD_LOGIC;
function SWITCH7_N (IN_D, IN_G, PREV_OUT_S : STD_LOGIC_VECTOR(0 to 6)) return
STD_LOGIC_VECTOR;
end pnew8;
package body pnew8 is
type STDLOGIC_1D is array (STD_LOGIC) of STD_LOGIC;
- - - - - - - - - - -
constant n_switch_on_array : STDLOGIC_1D := (
- -
-- IN_G        = 1,H
-- PREV_OUT_S  = DON'T CARE
- -
--                                        ------------------------------------------------
-- IN_D       = | U   X   0   1   Z   W   L   H   -   |
--                                        ------------------------------------------------
-- OUT_S      =
                ( 'U', 'X', '0', 'H', 'Z', 'W', 'L', 'W', '-' ))
- - - - - - - - - - -
constant n_switch_off_array : STDLOGIC_1D := (
- -
-- IN_G        = 0,L
-- IN_D       = DON'T CARE
--                                        ------------------------------------------------
-- PREV_OUT_S = | U   X   0   1   Z   W   L   H   -   |
--                                        ------------------------------------------------
-- OUT_S      =
                ( 'U', 'W', 'L', 'H', 'Z', 'Z', 'Z', 'Z', 'Z' ));

- - - - - - - - - - -
constant n_switch_other_array : STDLOGIC_1D := (
- -
-- IN_G        = U,X,Z,W,-
```

```
-- PREV_OUT_S  = DON'T CARE
- -
--                                          ------------------------------------------------
-- IN_D       = I  U   X   0   1   Z   W   L   H   -   I
--                                          ------------------------------------------------
-- OUT_S      =
                ( 'X', 'X', 'X', 'W', 'W', 'W', 'W', 'W', 'W' ));
- - - - - - - - - - -
function SWITCH_N (IN_D, IN_G, PREV_OUT_S : STD_LOGIC) return STD_LOGIC is
variable RESULT : STD_LOGIC := 'Z';
begin
  case IN_G is
     when '1' =>
         RESULT := n_switch_on_array(IN_D);
     when 'H' =>
         RESULT := n_switch_on_array(IN_D);
     when '0' =>
        RESULT := n_switch_off_array(PREV_OUT_S);
     when 'L' =>
        RESULT := n_switch_off_array(PREV_OUT_S);
     when 'U' =>
         RESULT := n_switch_other_array(IN_D);
     when 'X' =>
         RESULT := n_switch_other_array(IN_D);
     when 'W' =>
         RESULT := n_switch_other_array(IN_D);
      when '-' =>
         RESULT := n_switch_other_array(IN_D);
     when 'Z' =>
         RESULT := n_switch_other_array(IN_D);
  end case;
   return RESULT;
  end SWITCH_N;
- - - - - - - - - - -
function SWITCH7_N (IN_D, IN_G, PREV_OUT_S : STD_LOGIC_VECTOR(0 to 6)) return
STD_LOGIC_VECTOR is
variable RESULT : STD_LOGIC_VECTOR(0 to 6) := "ZZZZZZZ";
begin
for I in 0 to 6 loop
   case IN_G(I) is
     when '1' =>
          RESULT(I)  :=  n_switch_on_array(IN_D(I));
     when 'H' =>
          RESULT(I) := n_switch_on_array(IN_D(I));
     when '0' =>
          RESULT(I)  :=  n_switch_off_array(PREV_OUT_S(I));
     when 'L' =>
          RESULT(I)  :=  n_switch_off_array(PREV_OUT_S(I));
     when 'U' =>
          RESULT(I)  :=  n_switch_other_array(IN_D(I));
     when 'X' =>
```

```
        RESULT(I) := n_switch_other_array(IN_D(I));
    when 'W' =>
        RESULT(I) := n_switch_other_array(IN_D(I));
    when '-' =>
        RESULT(I) := n_switch_other_array(IN_D(I));
    when 'Z' =>
        RESULT(I) := n_switch_other_array(IN_D(I));
  end case;
end loop;
  return RESULT;
  end SWITCH7_N;
end pnew8;

use work.logic_system.all, work.pnew8.all, work.pnew1.all, work.all ;

--------------------------------  N-type  parallel  fault  simulation  ---
-----------------------------------------
entity N_FAULT is
port(
   GATE     : in  STD_LOGIC          := 'Z'  ;
   DRAIN    : in  STD_LOGIC          := 'Z'  ;
   SOURCE   : inout STD_LOGIC        := 'Z'  ;
   FAULTS   : inout X01Z_vector( 1 to 6)    := "111Z1Z";
   DETECT   : in  X01                       := '1'  ;
   DETECTED : inout X01_A_RES               := '1'  ;
   LEVEL_ALL        : in LEVEL_R                    ;
   NUMBER   : in integer                    := 0  ;
   STOP     : in  BIT               := '0' );
end N_FAULT;

architecture ONLY of N_FAULT is

signal GOOD_SOURCE_S7 : STD_LOGIC_VECTOR(0 to 6) := "ZZZZZZZ";
signal GOOD_SOURCE_1 : STD_LOGIC  := 'Z';
signal SOURCE_S7, DRAIN_M, GATE_M, PREV_SOURCE_S7, DRAIN_S7, GATE_S7:
       STD_LOGIC_VECTOR(0 to 6) := "ZZZZZZZ";
signal PREV_SOURCE,GOOD_SOURCE : STD_LOGIC          := 'Z'  ;
signal SWITCH_DETECT_9v : STD_LOGIC_VECTOR(1 to 6) := "000000";
signal SOURCE_G_SIG : STD_LOGIC           := '0'  ;
signal SWITCH_DETECT : X01Z_VECTOR(1 to 6) := "000000";
signal GOOD_SOURCE_X01 : X01              := 'X';
signal SOURCE_X01 : X01           := 'X';
signal DETECT_S7 : X01_VECTOR(0 to 6)          := "XXXXXXX";
signal INJECT_OK : STD_LOGIC                   := 'Z';
signal INJT_SIM_2P : BIT                  := '0';
signal INJECT_2P : BIT                 := '0';
signal FAULTS_NEW : X01_VECTOR(1 to 6)         := "XXXXXX";
signal GOOD_SIM_N : BIT                   := '0';
signal FAULT_C_SIG : BIT                  := '0';
signal OUT_RES : STD_LOGIC                := 'Z';
signal SOURCE_GS: STD_LOGIC               := 'Z';
```

```
signal SOURCE_IS: STD_LOGIC                := 'Z';
signal SOURCE_I: STD_LOGIC                 := 'Z';
signal SOURCE_FS: STD_LOGIC                := 'Z';
signal SOURCE_M: STD_LOGIC_VECTOR(0 to 6)  := "ZZZZZZZ";
signal DETECTED_IS: X01_A_RES              := '1';
signal DETECTED_IS_2P: X01_A_RES           := '1';
signal DETECTED_I_2P: X01_A_RES            := '1';
signal DETECTED_I: X01_A_RES               := '1';

begin

----------FAULT SIMULATION ----------
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

IN_VECTOR:process(LEVEL_ALL.T_CNTL(1))
begin
        IF (LEVEL_ALL.T_CNTL(1)= '1')
        THEN
                FAULT_C_SIG  <= NOT(FAULT_C_SIG);
                SOURCE_FS <= GOOD_SOURCE;
                OUT_RES <= '1' after 100 pS;
        ELSE
                SOURCE_FS <= GOOD_SOURCE;
                OUT_RES <= '0' after 100 pS;
                SWITCH_DETECT_9v <= "xor"(SOURCE_M(1 to 6),GOOD_SOURCE_S7(1
to 6));
            SWITCH_DETECT <= CONVERTZ("xor"(SOURCE_M(1
to6),GOOD_SOURCE_S7(1 to6)));
        END IF;
end process;

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
IN_MASK: process(FAULT_C_SIG)
constant MASK_D     : STD_LOGIC_VECTOR(0 to 6) := "ZZZ10ZZ";
constant FVALUE_D   : STD_LOGIC_VECTOR(0 to 6) := "UUUZZUU";
constant MASK_G     : STD_LOGIC_VECTOR(0 to 6) := "Z10ZZZZ";
constant FVALUE_G   : STD_LOGIC_VECTOR(0 to 6) := "UZZUUUU";
constant MASK_S     : STD_LOGIC_VECTOR(0 to 6) := "ZZZZZ10";
constant FVALUE_S   : STD_LOGIC_VECTOR(0 to 6) := "UUUUUZZ";

begin
        DRAIN_M <= MASK_NEW(DRAIN_S7,MASK_D,FVALUE_D);
        GATE_M  <= MASK_NEW(GATE_S7 ,MASK_G ,FVALUE_G);
end process;
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- -
SWITCHN: process(DRAIN_M,GATE_M)
begin
   SOURCE_S7 <= SWITCH7_N(DRAIN_M, GATE_M, PREV_SOURCE_S7);
end process;
```

```
-----------------------------------------------------------
---
OUT_MASK: process(SOURCE_S7)
constant MASK_S    : STD_LOGIC_VECTOR(0 to 6) := "ZZZZZ10";
constant FVALUE_S  : STD_LOGIC_VECTOR(0 to 6) := "UUUUUZZ";
begin
        IF(LEVEL_ALL.T_CNTL(1) = '1')
         THEN
         SOURCE_M <= MASK_NEW(SOURCE_S7,MASK_S,FVALUE_S);
         GOOD_SOURCE_S7 <= EXPAND1_7(GOOD_SOURCE_1);
         ELSE
         END IF;
end process;

-----------------------------------GOOD      SIMULATION------------
-------------------------

Previous:  process(LEVEL_ALL.T_CNTL(0))
begin
        IF (LEVEL_ALL.T_CNTL(0)= '1')
      THEN
         PREV_SOURCE <= GOOD_SOURCE_1;
         DRAIN_S7       <= EXPAND1_7(DRAIN);
         GATE_S7       <= EXPAND1_7(GATE);
         GOOD_SIM_N <=  '1' after 1 ns;
      ELSE
         GOOD_SIM_N <=  '0';
       END IF;
end process;
-----------------------------------------------------------
------
REG_SWITCH_1: process(GOOD_SIM_N) --,LEVEL)
begin
       IF (GOOD_SIM_N = '1')
      THEN
         PREV_SOURCE_S7 <= EXPAND1_7(PREV_SOURCE);
         SOURCE_GS <= SWITCH_N(DRAIN,GATE,PREV_SOURCE);
         SOURCE_G_SIG <= '1' after 2 ns;
         OUT_RES <=  '1' after 1 nS;
      ELSE
         SOURCE_G_SIG <= '0' after 2 ns;
         OUT_RES <= '0' after 1 NS;
       END IF;
end process;
-----------------------------------------------------------
-------
REG_SWITCH_3: process(SOURCE_G_SIG)
begin
        IF (LEVEL_ALL.T_CNTL(0) = '1')
      THEN
         GOOD_SOURCE <= SOURCE_GS;
```

```
        GOOD_SCURCE_1 <= SOURCE;
      ELSE
      END IF;
end process;
-------------------------------------------------------------------
-------
REG_SWITCH_4: process(GOOD_SOURCE_1)
begin
       IF (LEVEL_ALL.T_CNTL(0) = '1')
      THEN
        GOOD_SCURCE_X01 <= CONVERT(GOOD_SOURCE_1);
      ELSE
      END IF;
end process;

---------------------------------- INJECTED FAULT SIMULATION ----
---------------------------------------

REG_SWITCH_2: process(LEVEL_ALL.T_CNTL(4), LEVEL_ALL.T_CNTL(5))
begin
        IF (LEVEL_ALL.T_CNTL(4) = '1' )
        THEN
                SOURCE_IS <= SWITCH_N(DRAIN,GATE,PREV_SOURCE);
                OUT_RES <= '1' after 100 pS;
        ELSIF(LEVEL_ALL.T_CNTL(5) = '1' )
        THEN
                OUT_RES <= '1' after 100 pS;
        ELSE
                OUT_RES <= '0' after 100 pS;
        END IF;
end process;

--------------------------------- OUTPUT MUX FOR SOURCE AND
DETECTED-------------------------

MUX_RES: process(OUT_RES)
begin
        IF ( GOOD_SIM_N = '1')
        THEN SOURCE <= SOURCE_GS;
                DETECTED <= '1';
        ELSIF ( LEVEL_ALL.T_CNTL(4) = '1')
        THEN
                SOURCE <= SOURCE_IS;
        ELSIF(LEVEL_ALL.T_CNTL(5) = '1' )
        THEN
                IF (SOURCE = 'X')
                THEN
                        DETECTED <= 'X';
                ELSIF ( CONVERT(SOURCE) = 'X')
                THEN
                        DETECTED <= '1';
```

```
            ELSIF ( (SOURCE /= 'X') and (GOOD_SOURCE_X01 /= 'X') and
(CONVERT(SOURCE)                    = not(GOOD_SOURCE_X01)))
            THEN
                  DETECTED <= '0';
            ELSE
                  DETECTED <= '1';
            END IF;
      ELSIF ((LEVEL_ALL.T_CNTL(2) = '1') or (LEVEL_ALL.T_CNTL(3) = '1'))
      THEN
            SOURCE <= SOURCE_I;
            DETECTED <= DETECTED_I;
      ELSIF (LEVEL_ALL.T_CNTL(1) = '1')
            THEN SOURCE <= SOURCE_FS;
      ELSE
      END IF;
end process;
-------------------------------------    FAULT    INJECTION    ----------
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
INJT_VCTR: process(LEVEL_ALL.T_CNTL(2),LEVEL_ALL.T_CNTL(3))
variable DETECT_S7_V : X01Z_VECTOR(0 to 6) := "XXXXXXX";
variable FAULTS_F : X01Z_VECTOR(1 to 6) := "111111";
variable INJECT_OK_V : X01Z := 'Z' ;
variable NDEL : time := 0 ps;
begin
      INJECT_OK_V := 'Z' ;
      ndel := 100 ps;
      INJECT_OK_V := '0';
      IF((LEVEL_ALL.T_CNTL(2) = '1') or (LEVEL_ALL.T_CNTL(3) = '1'))
      THEN
            DETECT_S7_V := EXPAND_X01Z_1_7(DETECT);
            IF (NUMBER = LEVEL_ALL.T_NUMBER_CNTL)
            THEN
                  IF (INJECT_OK = '0')
                  THEN
                        SOURCE_I <= GOOD_SOURCE;
                        DETECTED_I <= '1';
                        OUT_RES <= '1' after 100 pS;
                  ELSIF (CONVERT(GOOD_SOURCE_1)/='X')
                  THEN
                        IF (LEVEL_ALL.T_CNTL(2) = '1')
                        THEN
                              SOURCE_I <= not(GOOD_SOURCE_1);
                              DETECTED_I <= '0';
                              OUT_RES <= '1' after 100 pS;
                        ELSIF (LEVEL_ALL.T_CNTL(3) = '1')
                        THEN
                              IF ((GATE = '1') or (GATE = 'H'))
                              THEN
                                    SOURCE_I <= 'Z';
                                    DETECTED_I <= '0';
                                    OUT_RES <= '1' after 100 pS;
```

```
                                ELSE
                                        SOURCE_I <= GOOD_SOURCE;
                                        DETECTED_I <= '1';
                                        OUT_RES <= '1' after 100 pS;
                                END IF;
                        END IF;
                END IF;
                ELSIF ( NUMBER = LEVEL_ALL.T_NUMBER_CNTL - 1)
                THEN
                        DETECTED_I <= '1';
                                SOURCE_I <= GOOD_SOURCE;
                                OUT_RES <= '1' after 100 pS;
                        IF (INJECT_OK = '0')
                        THEN
                                INJECT_OK <= '0';
                        ELSE
                                IF(LEVEL_ALL.T_CNTL(2) = '1')
                                THEN
FAULTS_F := ANDDZ(NOOTZ(ANDDZ(NOOTZ(DETECT_S7_V(1 to
6)),SWITCH_DETECT)),FAULTS);
                                        IF ( FAULTS_F(1) = '0')
                                        THEN
                                                FAULTS_F(1) := 'X';
                                        END IF;
                                        FAULTS_F(4) := FAULTS(4);
                                        FAULTS_F(6) := FAULTS(6);
                                        FAULTS <= FAULTS_F;
                                        for I in 2 to 6 loop
                        INJECT_OK_V := CON(INJECT_OK_V,FAULTS(I));
                                                NDEL := NDEL+10 ps;
                                        END LOOP;
                                        INJECT_OK <= INJECT_OK_V after NDEL;
                                ELSE
                                        IF (DETECT_S7_V(2) = '0')
                                        THEN
                                                FAULTS(2) <= '0';
                                        END IF;
                                        for I in 3 to 6 loop
                                                INJECT_OK_V :=
CON(INJECT_OK_V,FAULTS(I));
                                                NDEL := NDEL+10 ps;
                                        END LOOP;
                                        INJECT_OK <= INJECT_OK_V after NDEL;
                                END IF;
                        END IF;
                ELSIF ( NUMBER < LEVEL_ALL.T_NUMBER_CNTL - 1)
                THEN
                                SOURCE_I <= GOOD_SOURCE;
                                DETECTED_I <= '1';
                                OUT_RES <= '1' after 100 pS;
                ELSIF ( NUMBER > LEVEL_ALL.T_NUMBER_CNTL )
```

```
                    THEN
                                SOURCE_I <= GOOD_SOURCE;
                                DETECTED_I <= '1';
                                OUT_RES <= '1' after 100 pS;
                    END IF;
        ELSE
                OUT_RES <= '0' after 100 pS;
        END IF;
end process;
-------------------------------------------------------------------
--------
end only;
```

D.2 p-type Processor Element (PE) VHDL code

The program that follows models switch level parallel fault simulation for one P-type switch.

```
-------------------------------------------------------
--          Parallel Fault simulation          --
--          name - P_FAULT                     --
--          chris ryan   prelim work           --
--          11-04-92                           --
--                                             --
-------------------------------------------------------
use WORK.logic_system.all, WORK.all ;
library WORK;

--------------------------------  PACKAGE  PNEW  ------------
-------------------------------

package pnew is
type RESTYPEBBB is array (INTEGER range <>) of BIT;
function MASK_NEW (IN_T, MASK_IN, FVALUE_IN : std_LOGIC_vector(0 to 6) ) return
std_logic_vector;
function EXPAND1_7 (A : std_LOGIC) return std_LOGIC_vector;
function EXPAND_X01_1_7 (A : X01) return X01_vector;
function EXPAND_X01Z_1_7 (A : X01) return X01Z_vector;
end pnew;

package body pnew is
function MASK_NEW (IN_T, MASK_IN, fvalue_IN: std_LOGIC_vector(0 to 6))return
  std_LOGIC_vector is
variable RES_VALUE: std_LOGIC_vector(0 to 6) := "ZZZZZZZ" ;
begin
        RES_VALUE := CON(MIN_F(IN_T,fvalue_in),MASK_IN);
        return RES_VALUE;
end MASK_NEW;

function EXPAND1_7 (A : std_LOGIC ) return std_LOGIC_vector is
variable RESOLVED_VALUE: std_LOGIC_vector(0 to 6) := "ZZZZZZZ" ;
begin
        for I in 0 to 6 loop
                RESOLVED_VALUE(I) := A;
        end loop;
        return RESOLVED_VALUE;
end EXPAND1_7;

function EXPAND_X01_1_7 (A : X01 ) return X01_vector is
variable RESOLVED_VALUE: X01_vector(0 to 6) := "XXXXXXX" ;
begin
        for I in 0 to 6 loop
```

```
                RESOLVED_VALUE(I) := A;
        end loop;
 return RESOLVED_VALUE;
 end EXPAND_X01_1_7;

function EXPAND_X01Z_1_7 (A : X01 ) return X01Z_vector is
variable RESOLVED_VALUE: X01Z_vector(0 to 6) := "XXXXXXX" ;
 begin
        for I in 0 to 6 loop
                RESOLVED_VALUE(I) := A;
        end loop;
        return RESOLVED_VALUE;
  end EXPAND_X01Z_1_7;

end pnew;

use WORK.logic_system.all, WORK.all ;
library  WORK;

---------------------------------     PACKAGE     PNEW9     ------------
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

package pnew9 is
function SWITCH_P (IN_D, IN_G, PREV_OUT_S :STD_LOGIC) return STD_LOGIC;
function SWITCH7_P (IN_D, IN_G, PREV_OUT_S : STD_LOGIC_VECTOR(0 to 6)) return
STD_LOGIC_VECTOR;
end pnew9;

package body pnew9 is
type STDLOGIC_1D is array (STD_LOGIC) of STD_LOGIC;
constant p_switch_on_array : STDLOGIC_1D := (
- -
-- IN_G         = 0,L
-- PREV_OUT_S  = DON'T CARE
--                                    ------------------------------------------------
-- IN_D       = I  U    X    0    1    Z    W    L    H    -   I
--                                    ------------------------------------------------
-- OUT_S       =
                 ( 'U', 'X', 'L', '1', 'Z', 'W', 'W', 'H', '-' ));
- - - - - - - - - - -
constant p_switch_off_array : STDLOGIC_1D := (
- -
-- IN_G         = 1,H
-- IN_D       = DON'T CARE
- -
--                                    ------------------------------------------------
-- PREV_OUT_S  = I  U    X    0    1    Z    W    L    H    -   I
--                                    ------------------------------------------------
-- OUT_S       =
                 ( 'U', 'W', 'L', 'H', 'Z', 'Z', 'Z', 'Z', 'Z' ));
- - - - - - - - - - -
```

```
constant p_switch_other_array : STDLOGIC_1D := (
- -
-- IN_G         = U,X,Z,W,-
-- PREV_OUT_S  = DON'T CARE
- -
--                                          ------------------------------------------------
-- IN_D       = I  U   X   0   1   Z   W   L   H   -   I
--                                          ------------------------------------------------
-- OUT_S      =
                ( 'X', 'X', 'X', 'W', 'W', 'W', 'W', 'W', 'W' ));
- - - - - - - - - - -
function SWITCH_P (IN_D, IN_G, PREV_OUT_S : STD_LOGIC) return STD_LOGIC is
variable RESULT : STD_LOGIC := 'Z';
begin
  case IN_G is
     when '0' =>
         RESULT := p_switch_on_array(IN_D);
     when 'L' =>
         RESULT := p_switch_on_array(IN_D);
     when '1' =>
        RESULT := p_switch_off_array(PREV_OUT_S);
     when 'H' =>
        RESULT := p_switch_off_array(PREV_OUT_S);
     when 'U' =>
         RESULT := p_switch_other_array(IN_D);
     when 'X' =>
         RESULT := p_switch_other_array(IN_D);
     when 'W' =>
         RESULT := p_switch_other_array(IN_D);
     when '-' =>
         RESULT := p_switch_other_array(IN_D);
     when 'Z' =>
         RESULT := p_switch_other_array(IN_D);
  end case;
   return RESULT;
  end SWITCH_P;
- - - - - - - - - - -
function SWITCH7_P (IN_D, IN_G, PREV_OUT_S : STD_LOGIC_VECTOR(0 to 6)) return
STD_LOGIC_VECTOR is
variable RESULT : STD_LOGIC_VECTOR(0 to 6) := "ZZZZZZZ";
begin

for I in 0 to 6 loop
   case IN_G(I) is
     when '0' =>
          RESULT(I)  :=  p_switch_on_array(IN_D(I));
     when 'L' =>
          RESULT(I) := p_switch_on_array(IN_D(I));
     when '1' =>
          RESULT(I)  :=  p_switch_off_array(PREV_OUT_S(I));
     when 'H' =>
```

```
        RESULT(I) := p_switch_off_array(PREV_OUT_S(I));
    when 'U' =>
        RESULT(I) := p_switch_other_array(IN_D(I));
    when 'X' =>
        RESULT(I) := p_switch_other_array(IN_D(I));
    when 'W' =>
        RESULT(I) := p_switch_other_array(IN_D(I));
    when '-' =>
        RESULT(I) := p_switch_other_array(IN_D(I));
    when 'Z' =>
        RESULT(I) := p_switch_other_array(IN_D(I));
  end case;
end loop;
  return RESULT;
  end SWITCH7_P;
end pnew9;

use work.logic_system.all, work.pnew9.all, work.pnew.all, work.all ;

---------------------------------- P-type parallel fault simulation ---
-----------------------------------------

entity P_FAULT is
port(
   GATE     : in  STD_LOGIC          := 'Z'  ;
   DRAIN    : in  STD_LOGIC          := 'Z'  ;
   SOURCE   : inout STD_LOGIC        := 'Z'  ;
   FAULTS   : inout X01Z_vector( 1 to 6)    := "11Z1Z1";
   DETECT   : in  X01                       := '1'  ;
   DETECTED : inout X01_A_RES               := '1'  ;
   LEVEL_ALL : in LEVEL_R                   ;
   NUMBER   : in integer                    := 0  ;
   STOP     : in  BIT               := '0' );
end P_FAULT;

architecture ONLY of P_FAULT is

signal GOOD_SOURCE_S7 : STD_LOGIC_VECTOR(0 to 6) := "ZZZZZZZ";
signal GOOD_SOURCE_1 : STD_LOGIC  := 'Z';
signal SOURCE_S7, DRAIN_M, GATE_M, PREV_SOURCE_S7, DRAIN_S7, GATE_S7:
        STD_LOGIC_VECTOR(0 to 6) := "ZZZZZZZ";
signal PREV_SOURCE,GOOD_SOURCE : STD_LOGIC       := 'Z'  ;
signal SWITCH_DETECT_9v : STD_LOGIC_VECTOR(1 to 6) := "000000";
signal SOURCE_G_SIG : STD_LOGIC          := '0'  ;
signal SWITCH_DETECT : X01Z_VECTOR(1 to 6) := "000000";
signal GOOD_SOURCE_X01 : X01             := 'X';
signal SOURCE_X01 : X01          := 'X';
signal DETECT_S7 : X01_VECTOR(0 to 6)        := "XXXXXXX";
signal INJECT_OK : STD_LOGIC                 := 'Z';
signal INJT_SIM_2P : BIT                 := '0';
signal INJECT_2P : BIT                   := '0';
```

```
signal FAULTS_NEW : X01_VECTOR(1 to 6)          := "XXXXXX";
signal GOOD_SIM_N : BIT                          :=  '0';
signal FAULT_C_SIG : BIT                        := '0';
signal OUT_RES : STD_LOGIC                      := 'Z';
signal SOURCE_GS: STD_LOGIC                      := 'Z';
signal SOURCE_IS: STD_LOGIC                      := 'Z';
signal SOURCE_I: STD_LOGIC                      := 'Z';
signal SOURCE_FS: STD_LOGIC                      := 'Z';
signal SOURCE_M: STD_LOGIC_VECTOR(0 to 6)     := "ZZZZZZZ";
signal DETECTED_IS: X01_A_RES                  := '1';
signal DETECTED_I: X01_A_RES                     := '1';
signal DETECTED_I_2P: X01_A_RES                  := '1';
signal DETECTED_IS_2P: X01_A_RES                   := '1';

begin
--------------------------------  FAULT  SIMULATION  -----------
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
IN_VECTOR:process(LEVEL_ALL.T_CNTL(1))
begin
        IF  (LEVEL_ALL.T_CNTL(1)=  '1')
        THEN
                FAULT_C_SIG   <= NOT(FAULT_C_SIG);
                SOURCE_FS <= GOOD_SOURCE;
                OUT_RES <= '1' after 100 pS;
        ELSE
                SOURCE_FS <= GOOD_SOURCE;
                OUT_RES <= '0' after 100 pS;
                SWITCH_DETECT_9v <= "xor"(SOURCE_M(1 to 6),GOOD_SOURCE_S7(1
to  6));
        SWITCH_DETECT <= CONVERTZ("xor"(SOURCE_M(1 to 6),GOOD_SOURCE_S7(1
to  6)));
        END IF;
end process;
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
IN_MASK: process(FAULT_C_SIG)
constant MASK_D     : STD_LOGIC_VECTOR(0 to 6) := "ZZZ10ZZ";
constant FVALUE_D   : STD_LOGIC_VECTOR(0 to 6) := "UUUZZUU";
constant MASK_G     : STD_LOGIC_VECTOR(0 to 6) := "Z10ZZZZ";
constant FVALUE_G   : STD_LOGIC_VECTOR(0 to 6) := "UZZUUUU";
constant MASK_S     : STD_LOGIC_VECTOR(0 to 6) := "ZZZZZ10";
constant FVALUE_S   : STD_LOGIC_VECTOR(0 to 6) := "UUUUUZZ";
begin
        DRAIN_M <= MASK_NEW(DRAIN_S7,MASK_D,FVALUE_D);
        GATE_M  <= MASK_NEW(GATE_S7 ,MASK_G ,FVALUE_G);
end process;
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- -
SWITCHP: process(DRAIN_M,GATE_M)
begin
        SOURCE_S7 <= SWITCH7_P(DRAIN_M, GATE_M, PREV_SOURCE_S7);
end process;
```

```
--------------------------------------------------------------------
---
OUT_MASK: process(SOURCE_S7)
constant MASK_S    : STD_LOGIC_VECTOR(0 to 6) := "ZZZZZ10";
constant FVALUE_S  : STD_LOGIC_VECTOR(0 to 6) := "UUUUUZZ";
begin
        IF(LEVEL_ALL.T_CNTL(1)  =  '1')
        THEN
                SOURCE_M <= MASK_NEW(SOURCE_S7,MASK_S,FVALUE_S);
                GOOD_SOURCE_S7 <= EXPAND1_7(GOOD_SOURCE_1); --_1
        ELSE
        END IF;
end process;
----------------------------------   GOOD    SIMULATION    ------------
---------------------------------
Previous:  process(LEVEL_ALL.T_CNTL(0))
begin
        IF  (LEVEL_ALL.T_CNTL(0)=  '1')
        THEN
                PREV_SOURCE <= GOOD_SOURCE_1;
                DRAIN_S7       <= EXPAND1_7(DRAIN);
                GATE_S7       <= EXPAND1_7(GATE);
                GOOD_SIM_N <=  '1' after 1 ns;
        ELSE
                GOOD_SIM_N <=  '0';
        END IF;
end process;
--------------------------------------------------------------------
-------
REG_SWITCH_1: process(GOOD_SIM_N)
begin
        IF (GOOD_SIM_N = '1')
        THEN
                PREV_SOURCE_S7 <= EXPAND1_7(PREV_SOURCE);
                SOURCE_GS <= SWITCH_P(DRAIN,GATE,PREV_SOURCE);
                SOURCE_G_SIG <= '1' after 2 ns ;
                OUT_RES <= '1' after 1 nS;
        ELSE
                SOURCE_G_SIG <= '0' after 2 ns ;
                OUT_RES <= '0' after 1 NS;
     END IF;
end process;
--------------------------------------------------------------------
-------
REG_SWITCH_3: process(SOURCE_G_SIG)
begin
       IF (LEVEL_ALL.T_CNTL(0) = '1')
     THEN
        GOOD_SOURCE <= SOURCE_GS;
        GOOD_SOURCE_1 <= SOURCE;
     ELSE
```

```
    END IF;
end process;
-------------------------------------------------------------------
-------
REG_SWITCH_4: process(GOOD_SOURCE_1)
begin
      IF (LEVEL_ALL.T_CNTL(0) = '1')
    THEN
       GOOD_SOURCE_X01 <= CONVERT(GOOD_SOURCE_1);
    ELSE
    END IF;
end process;
-------------------------------- INJECTED FAULT SIMULATION -------
-----------------------------
REG_SWITCH_2: process(LEVEL_ALL.T_CNTL(4), LEVEL_ALL.T_CNTL(5))
begin
       IF (LEVEL_ALL.T_CNTL(4) = '1' )
       THEN
              SOURCE_IS <= SWITCH_P(DRAIN,GATE,PREV_SOURCE);
              OUT_RES <= '1' after 100 pS;
       ELSIF(LEVEL_ALL.T_CNTL(5) = '1' )
       THEN
              OUT_RES <= '1' after 100 pS;
       ELSE
              OUT_RES <= '0' after 100 pS;
       END IF;
end process;
------------------------------- OUTPUT MUX FOR SOURCE AND
DETECTED ----------------------------
MUX_RES: process(OUT_RES)
begin
       IF ( GOOD_SIM_N = '1')
       THEN
              SOURCE <= SOURCE_GS;
              DETECTED <= '1';
       ELSIF ( LEVEL_ALL.T_CNTL(4) = '1')
       THEN
              SOURCE <= SOURCE_IS;
       ELSIF(LEVEL_ALL.T_CNTL(5) = '1' )
       THEN
              IF (SOURCE = 'X')
              THEN
                     DETECTED <= 'X';
              ELSIF ( CONVERT(SOURCE) = 'X')
              THEN
                     DETECTED <= '1';
              ELSIF ( (SOURCE /= 'X') and (GOOD_SOURCE_X01 /= 'X')
and(CONVERT(SOURCE)                                                    =
not(GOOD_SOURCE_X01)))
              THEN
                     DETECTED <= '0';
```

```
            ELSE
                    DETECTED <= '1';
            END IF;
        ELSIF ((LEVEL_ALL.T_CNTL(2) = '1') or (LEVEL_ALL.T_CNTL(3) = '1'))
        THEN
            SOURCE <= SOURCE_I ;
            DETECTED <= DETECTED_I;
        ELSIF (LEVEL_ALL.T_CNTL(1) = '1')
        THEN SOURCE <= SOURCE_FS;
        ELSE
        END IF;
end process;
-------------------------------        FAULT    INJECTION        -------------
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
INJT_VCTR: process(LEVEL_ALL.T_CNTL(2),LEVEL_ALL.T_CNTL(3))
variable DETECT_S7_V : X01Z_VECTOR(0 to 6) := "XXXXXXX";
variable FAULTS_F : X01Z_VECTOR(1 to 6) := "111111";
variable INJECT_OK_V : X01Z := 'Z';
variable NDEL : time := 0 ps;
begin
        INJECT_OK_V := 'Z';
        IF((LEVEL_ALL.T_CNTL(2) = '1') or (LEVEL_ALL.T_CNTL(3) = '1'))
        THEN
            DETECT_S7_V := EXPAND_X01Z_1_7(DETECT);
            IF (NUMBER = LEVEL_ALL.T_NUMBER_CNTL)
            THEN
            IF ((INJECT_OK /= '0') and (CONVERT(GOOD_SOURCE_1)/='X'))
            THEN
                IF (LEVEL_ALL.T_CNTL(2) = '1')
                THEN
                    SOURCE_I <= not(GOOD_SOURCE_1);
                    DETECTED_I <= '0';
                    OUT_RES <= '1' after 100 pS;
                ELSIF (LEVEL_ALL.T_CNTL(3) = '1')
                THEN
                    IF ((GATE = '0') or (GATE = 'L'))
                    THEN
                        SOURCE_I <= 'Z';
                        DETECTED_I <= '0';
                        OUT_RES <= '1' after 100 pS;
                    ELSE
                        SOURCE_I <= GOOD_SOURCE;
                        DETECTED_I <= '1';
                        OUT_RES <= '1' after 100 pS;
                    END IF;
                END IF;
            END IF;
        ELSIF ( NUMBER = LEVEL_ALL.T_NUMBER_CNTL - 1)
        THEN
            DETECTED_I <= '1';
            SOURCE_I <= GOOD_SOURCE;
```

```
                OUT_RES <=  '1' after 100 pS;
                IF (INJECT_OK = '0')
                THEN
                        INJECT_OK <= '0';
                ELSE
                        INJECT_OK <= 'Z';
                        IF(LEVEL_ALL.T_CNTL(2) = '1')
                        THEN
                                FAULTS_F :=
ANDDZ(NOOTZ(ANDDZ(nootZ(DETECT_S7_V(1 to
                        6)),SWITCH_DETECT)),FAULTS);
                                IF (FAULTS_F(2) = '0')
                                THEN
                                        FAULTS_F(2) := 'X';
                                END IF;
                                FAULTS_F(3) := FAULTS(3);
                                FAULTS_F(5) := FAULTS(5);
                                FAULTS <= FAULTS_F;
                                for I in 3 to 6 loop
                                        INJECT_OK_V := CON(INJECT_OK_V,FAULTS(I));
                                END LOOP;
                                INJECT_OK_V := CON(INJECT_OK_V,FAULTS(1));
                                INJECT_OK <= INJECT_OK_V after NDEL;
                                ELSE
                                        IF (DETECT_S7_V(1) = '0')
                                        THEN
                                                FAULTS(1) <= '0';
                                        END IF;
                                        for I in 3 to 6 loop
                                                INJECT_OK_V :=
CON(INJECT_OK_V,FAULTS(I));
                                        END LOOP;
                                        INJECT_OK <= INJECT_OK_V after NDEL;
                                END IF;
                        END IF;
                ELSIF ( NUMBER < LEVEL_ALL.T_NUMBER_CNTL - 1)
                THEN
                        SOURCE_I <= GOOD_SOURCE;
                        DETECTED_I <= '1';
                        OUT_RES <= '1' after 100 pS;
                ELSIF ( NUMBER > LEVEL_ALL.T_NUMBER_CNTL )
                THEN
                        SOURCE_I <= GOOD_SOURCE;
                        DETECTED_I <= '1';
                        OUT_RES <= '1' after 100 pS;
                END IF;
        ELSE
                OUT_RES <= '0' after 100 pS;
        END IF;
end process;
```

```
--------------------------------------------------------------------------------
--------
end only;
```

D.3 VHDL Nine-Valued Logic

The program that follow is the nine-valued logic used for parallel fault simulation. Additional modifications are the addition of a MINIMUM operation, four-valued logic, nive-value to binary, binary to nine-value, level record declaration, and four-value operators.

logic_system.pkg.vhdl

```
--        ------------------------------------------------------------
---------
-- (c) Copyright 1990          IEEE VHDL Model Standards Group
-- (c) Copyright 1989 - 1990    THE VHDL CONSULTING GROUP
- -
--      This source file may be used and distributed without restriction
--     provided that the copyright is not removed and that the code is
--      not used or distributed for profit.
--        ------------------------------------------------------------
---------
- -
--   File name :  logic_system.pkg.vhdl
--  Title     :  LOGIC_SYSTEM package ( multivalue logic system )
--   Library   :  STD
--   Author(s) :  W. Billowitch ( The VHDL Consulting Group )
--  Purpose   :  This packages defines a standard for digital designers
--           :  to use in describing the interconnection data types used in
--          :  modeling common ttl, cmos, GaAs, nmos, pmos, and ecl
--           :  digital devices.
--           :
--  Notes     :  The logic system defined in this package may
--            :  be insufficient for modeling switched transistors,
--           :  since that requirement is out of the scope of this
--           :  effort.
--          :
--           :  No other declarations or definitions shall be included
--          :  in this package. Any additional declarations shall be
--           :  placed in other orthogonal packages ( ie. timing, etc )
--          :
--        ------------------------------------------------------------
---------
--   Modification History :
--        ------------------------------------------------------------
---------
--  Version No:I Author:I  Mod. Date:I Changes Made:
--    v2.000   I  wdb   I   6/19/90  I DRAFT STANDARD
--    v2.100   I  wdb   I   7/16/90  I Addition of 'U' and '-' states
--        ------------------------------------------------------------
---------
--      Last Additional Modifications 2/18/93
```

```
- -     By: Chris Ryan
- -     For: Parallel Fault Simulation
- -     Where: Virginia Tech
-------------------------------------------------------------------
------------
Library STD; -- library location of this package
PACKAGE logic_system is
        attribute REFLEXIVE: boolean;
        attribute RESULT_INITIAL_POSITION : integer;
        attribute TABLE_NAME: string;
   attribute FUNCTION_IS_A_TABLE : boolean;
                    ------------------------------------------------------
--------
    -- Logic State System  (unresolved)
                    ------------------------------------------------------
--------
    TYPE std_ulogic is ( 'U',  -- Unitialized
                  'X',  -- Forcing  0 or 1
                  '0',  -- Forcing  0
                  '1',  -- Forcing  1
                 'Z',  -- High Impedance
                 'W',  -- Weak     0 or 1
                 'L',  -- Weak     0       ( for ECL open emitter )
                 'H',  -- Weak     1       ( for open Drain or Collector )
                 '-'   -- don't care
               );
                    ------------------------------------------------------
--------
    -- Unconstrained array of std_ulogic for use with the resolution function
                    ------------------------------------------------------
--------
   TYPE std_ulogic_vector IS ARRAY ( NATURAL RANGE <> ) of std_ulogic;
   TYPE INTEGER_ARRAY is ARRAY (NATURAL RANGE <>) of INTEGER;
                    ------------------------------------------------------
--------
    -- Resolution function
                    ------------------------------------------------------
--------
   FUNCTION resolved ( s : std_ulogic_vector ) RETURN std_ulogic;
    -- Hints to the simulator for optimization
        attribute REFLEXIVE of resolved : function is TRUE;
        attribute RESULT_INITIAL_POSITION of resolved: function is
std_ulogic'POS('Z');
        attribute TABLE_NAME of resolved : function is "resolution_table";
                    ------------------------------------------------------
--------
    -- *** Industry Standard Logic Type ***
                    ------------------------------------------------------
--------
   SUBTYPE std_logic IS resolved std_ulogic;
```

```
                    ------------------------------------------------------
- - - - - - -
    -- Unconstrained array of std_logic for use in declaring signal arrays
                    ------------------------------------------------------
- - - - - - -
   TYPE std_logic_vector IS ARRAY ( NATURAL RANGE <>) of std_logic;
                    ------------------------------------------------------
- - - - - - -
    -- Three basic states
                    ------------------------------------------------------
- - - - - - -
    SUBTYPE X01 is std_ulogic RANGE 'X' to '1'; -- ('X','0','1')
    SUBTYPE X01Z is std_ulogic RANGE 'X' to 'Z'; -- ('X','0','1','Z')
                    ------------------------------------------------------
- - - - - - -
    -- Unconstrained array of state for use in declaring registers
                    ------------------------------------------------------
- - - - - - -
   TYPE X01_vector IS ARRAY ( NATURAL RANGE <> ) of X01;
   TYPE X01Z_vector IS ARRAY ( NATURAL RANGE <> ) of X01Z;
                    ------------------------------------------------------
- - - - - - -
   TYPE time_vector IS ARRAY ( NATURAL RANGE <> ) of time;
                    ------------------------------------------------------
- - - - - - -
    -- Unconstrained array of X01_A_RES for use in declaring signal arrays
                    ------------------------------------------------------
- - - - - - -
    -- Resolution function AND_X01
                    ------------------------------------------------------
- - - - - - -
   FUNCTION resolved_and ( s : X01_vector ) RETURN X01;
    -- Hints to the simulator for optimization
      attribute REFLEXIVE of resolved_and : function is TRUE;
      attribute TABLE_NAME of resolved_and : function is "and";
   FUNCTION resolved_andz ( s : X01Z_vector ) RETURN X01Z;
    -- Hints to the simulator for optimization
      attribute REFLEXIVE of resolved_andz : function is TRUE;
      attribute TABLE_NAME of resolved_andz : function is "and";
                    ------------------------------------------------------
- - - - - - -
    -- Resolution function max_time
                    ------------------------------------------------------
- - - - - - -
   FUNCTION resolved_max_time ( s : time_vector ) RETURN time;
    -- Hints to the simulator for optimization
      attribute REFLEXIVE of resolved_max_time : function is TRUE;
                    ------------------------------------------------------
- - - - - - -
   SUBTYPE X01_A_RES IS resolved_and X01;
   SUBTYPE X01Z_A_RES IS resolved_andz X01Z;
```

```
-------------------------------------------------------------
- - - - - - -
   SUBTYPE TIME_RES IS resolved_max_time time;
-------------------------------------------------------------
- - - - - - -
   TYPE X01_A_RES_vector IS ARRAY ( NATURAL RANGE <>) of X01_A_RES;
   TYPE X01Z_A_RES_vector IS ARRAY ( NATURAL RANGE <>) of X01Z_A_RES;
   TYPE LEVEL_R is record
               T_CNTL : BIT_VECTOR( 0 to 5);
               T_NUMBER_CNTL : INTEGER ;
   END record;
   TYPE LEVEL_R_VECTOR is array ( NATURAL RANGE <>) of LEVEL_R;
-------------------------------------------------------------
- - - - - - -
   -- Overloaded Logical Operators
-------------------------------------------------------------
- - - - - - -
   FUNCTION "and"  ( l : std_ulogic; r : std_ulogic ) RETURN std_ulogic;
   FUNCTION ANDD  ( l : X01 ; r : X01 ) RETURN X01 ;
   FUNCTION ANDDZ  ( l : X01Z ; r : X01Z ) RETURN X01Z ;
   FUNCTION CONVERT  ( l : std_logic ) RETURN X01 ;
   FUNCTION CONVERTZ  ( l : std_logic ) RETURN X01Z ;
   FUNCTION CONVERT_B_9  ( l : bit) RETURN std_logic;
   FUNCTION CONVERT_9_B  ( l : STD_LOGIC) RETURN BIT;
   FUNCTION "nand" ( l : std_ulogic; r : std_ulogic ) RETURN std_ulogic;
   FUNCTION "or"   ( l : std_ulogic; r : std_ulogic ) RETURN std_ulogic;
   FUNCTION "nor"  ( l : std_ulogic; r : std_ulogic ) RETURN std_ulogic;
   FUNCTION "xor"  ( l : std_ulogic; r : std_ulogic ) RETURN std_ulogic;
   FUNCTION XOOR  ( l : X01 ; r : X01 ) RETURN X01 ;
   FUNCTION XOORZ  ( l : X01Z ; r : X01Z ) RETURN X01Z ;
   FUNCTION noot  ( l : X01  ) RETURN X01 ;
   FUNCTION nootZ  ( l : X01Z  ) RETURN X01Z ;
   FUNCTION "not"  ( l : std_ulogic              ) RETURN std_ulogic;
-------------------------------------------------------------
- - - - - - - - -
   -- My Additional functions CAR
-------------------------------------------------------------
- - - - - - - - -
    function MIN_F (l, r : std_logic) return std_logic;
    function MIN_F (l, r : std_logic_vector) return std_logic_vector;
    function MIN_F_X01Z (l, r : X01Z) return X01Z;
    function MIN_F_X01Z (l, r : X01Z_vector) return X01Z_vector;
    function CON (l, r : std_logic) return std_logic;
    function CON (l, r : std_logic_vector) return std_logic_vector;
-------------------------------------------------------------
- - - - - - -
   -- Vectorized Overloaded Logical Operators
-------------------------------------------------------------
- - - - - - -
   FUNCTION "and"  ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;
   FUNCTION ANDD  ( L,R  : X01_vector ) RETURN X01_vector ;
```

```
    FUNCTION ANDDZ  ( L,R  : X01Z_vector ) RETURN X01Z_vector ;
    FUNCTION CONVERT  ( L : std_logic_vector ) RETURN X01_vector ;
    FUNCTION CONVERTZ  ( L : std_logic_vector ) RETURN X01Z_vector ;
    FUNCTION CONVERT_B_9  ( L : bit_vector ) RETURN std_logic_vector ;
    FUNCTION CONVERT_9_B  ( L : std_logic_vector) RETURN bit_vector ;
     FUNCTION "nand" ( l, r : std_logic_vector ) RETURN std_logic_vector;
     FUNCTION "or"   ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;
     FUNCTION "nor"  ( l, r : std_logic_vector ) RETURN std_logic_vector;
     FUNCTION "xor"  ( l, r : std_logic_vector ) RETURN std_logic_vector;
    FUNCTION XOOR  ( L,R  : X01_vector ) RETURN X01_vector ;
    FUNCTION XOORZ  ( L,R  : X01Z_vector ) RETURN X01Z_vector ;
    FUNCTION noot  ( L  : X01_vector ) RETURN X01_vector ;
    FUNCTION nootZ  ( L  : X01Z_vector ) RETURN X01Z_vector ;
     FUNCTION "not"  ( l    : std_ulogic_vector ) RETURN std_ulogic_vector;
                        ------------------------------------------------------------
- - - - - - -
    -- Types
                        ------------------------------------------------------------
- - - - - - -
     TYPE logic_X01_table is array (std_ulogic'low to std_ulogic'high) of X01;
     TYPE logic_X01Z_table is array (std_ulogic'low to std_ulogic'high) of X01Z;
     TYPE logic_bit_table is array (bit'low to bit'high) of bit;
                        ------------------------------------------------------------
- - - - - - -
    -- Tables
                        ------------------------------------------------------------
- - - - - - -
    -- Table name : convert_to_X01
     --
    -- Parameters :
    --        in  :: std_ulogic  -- some logic value
    -- Returns    :  X01          -- state value of logic value
    -- Purpose    :  to convert state-strength to state only
     --
     -- Example     : if (convert_to_X01 (input_signal) = '1' ) then ...
    CONSTANT convert_to_X01 : logic_X01_table := (
                    'X',  -- 'U'
                    X',  -- 'X'
                     '0',  -- '0'
                     '1',  -- '1'
                    'X',  -- 'Z'
                    'X',  -- 'W'
                    '0',  -- 'L'
                    '1',  -- 'H'
                    'X'   -- '-'
                  );
    CONSTANT convert_to_9v : logic_bit_table := (
                     '0',  -- '0'
                     '1'  -- '1'
                  );
    CONSTANT convert_to_X01Z : logic_X01Z_table := (
```

```
                    'X',  -- 'U'
                    'X',  -- 'X'
                     '0',  -- '0'
                     '1',  -- '1'
                    'Z',  -- 'Z'
                    'Z',  -- 'W'
                     '0',  -- 'L'
                     '1',  -- 'H'
                     'Z'   -- '-'
                  );
                      ------------------------------------------------------------
- - - - - - - -
    -- Edge Detection
                      ------------------------------------------------------------
- - - - - - - -
   FUNCTION rising_edge  (SIGNAL s : std_ulogic) RETURN boolean;
   FUNCTION falling_edge (SIGNAL s : std_ulogic) RETURN boolean;
END logic_system;
```

D.4

logic_system.pkg_body.vhdl

```
--        ------------------------------------------------------------
---------
-- (c) Copyright 1990          IEEE VHDL Model Standards Group
-- (c) Copyright 1989 - 1990     THE VHDL CONSULTING GROUP
- -
--       This source file may be used and distributed without restriction
--      provided that the copyright is not removed and that the code is
--       not used or distributed for profit.
--         ------------------------------------------------------------
---------
- -
--    File name :  logic_system.pkg_body.vhdl
--    Title     :  LOGIC_SYSTEM package ( multivalue logic system )
--    Library   :  STD
--    Author(s) :  W. Billowitch ( The VHDL Consulting Group )
--    Purpose   :  To define the declarations made in the package
--              :
--    Notes     :
--         ------------------------------------------------------------
---------
--    Modification History :
--         ------------------------------------------------------------
---------
--   Version No:| Author:|  Mod. Date:| Changes Made:
--     v2.000   |  wdb   |   6/19/90  | DRAFT STANDARD
--     v2.100   |  wdb   |   7/16/90  | Addition of 'U' and '-' states
--         ------------------------------------------------------------
---------
--       Last Additional Modifications 2/18/93
- -      By: Chris Ryan
- -      For: Parallel Fault Simulation
- -      Where: Virginia Tech
------------------------------------------------------------------
-----------
Library STD; -- library location of this package
PACKAGE BODY logic_system is
                    ------------------------------------------------------
-------
    -- Local Types
                    ------------------------------------------------------
-------
    TYPE stdlogic_1D is array (std_ulogic) of std_ulogic;
    TYPE stdlogic_table is array(std_ulogic, std_ulogic) of std_ulogic;
                    ------------------------------------------------------
-------
    -- Resolution function
```

```
                    ------------------------------------------------------
-------
   CONSTANT resolution_table : stdlogic_table := (
      --                       ---------------------------------------------
-----
      --    | U   X   0   1   Z   W   L   H   -     | |
      --                       ---------------------------------------------
-----
         ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
         ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
          ( 'U', 'X', '0', 'X', '0', '0', '0', '0', '0' ), -- | 0 |
          ( 'U', 'X', 'X', '1', '1', '1', '1', '1', '1' ), -- | 1 |
          ( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-' ), -- | Z |
          ( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'W' ), -- | W |
          ( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'L' ), -- | L |
          ( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'H' ), -- | H |
           ( 'U', 'X', '0', '1', '-', 'W', 'L', 'H', '-' )  -- | - |
     );
   FUNCTION resolved ( s : std_ulogic_vector ) RETURN std_ulogic IS
      VARIABLE result : std_ulogic := 'Z';  -- weakest state default
  BEGIN
     IF   (s'LENGTH = 1) THEN    RETURN s(s'LOW);
    ELSE
      FOR i IN s'RANGE LOOP
              result := resolution_table(result, s(i));
      END LOOP;
       RETURN result;
    END IF;
  END resolved;
                    ------------------------------------------------------
-------
   -- Tables for Logical Operations
                    ------------------------------------------------------
-------
      -- truth table for "and" function
  CONSTANT and_table : stdlogic_table := (
      --                       ---------------------------------------------
      --    | U   X   0   1   Z   W   L   H   -     | |
      --                       ---------------------------------------------
            ( 'X', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ),  -- | U |
            ( 'X', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ),  -- | X |
      ( '0', '0', '0', '0', '0', '0', '0', '0', '0' ),  -- | 0 |
      ( 'X', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ),  -- | 1 |
      ( 'X', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ),  -- | Z |
      ( 'X', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ),  -- | W |
      ( '0', '0', '0', '0', '0', '0', '0', '0', '0' ),  -- | L |
      ( 'X', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ),  -- | H |
      ( 'X', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' )   -- | - |
  );
  FUNCTION resolved_and ( s : X01_vector ) RETURN X01 IS
     VARIABLE result : X01 := '1';  -- weakest state default
```

```
BEGIN
    IF    (s'LENGTH = 1) THEN    RETURN s(s'LOW);
   ELSE
     FOR i IN s'RANGE LOOP
            result := and_table (result, s(i));
     END LOOP;
      RETURN result;
   END IF;
END resolved_and;

FUNCTION resolved_andZ ( s : X01Z_vector ) RETURN X01Z IS
    VARIABLE result : X01Z := '1';  -- weakest state default
BEGIN
    IF    (s'LENGTH = 1) THEN    RETURN s(s'LOW);
   ELSE
     FOR i IN s'RANGE LOOP
            result := and_table (result, s(i));
     END LOOP;
      RETURN result;
   END IF;
END resolved_andZ;

FUNCTION resolved_max_time ( s : time_vector ) RETURN time IS
    VARIABLE result : time := 1 NS;  -- weakest state default
BEGIN
    IF    (s'LENGTH = 1) THEN    RETURN s(s'LOW);
   ELSE
     FOR i IN s'RANGE LOOP
            if   (s(i)>result)
            then
            result  :=  s(i);
            else
            END IF;
     END LOOP;
      RETURN result;
   END IF;
END resolved_max_time;
    -- truth table for "or" function
    CONSTANT or_table : stdlogic_table := (
    --                      ---------------------------------------------------
    --    | U   X   0   1   Z   W   L   H   -      | |
    --                      ---------------------------------------------------
           ( 'X', 'X', 'X', '1', 'X', 'X', 'X', '1', 'X' ),  -- | U |
           ( 'X', 'X', 'X', '1', 'X', 'X', 'X', '1', 'X' ),  -- | X |
           ( 'X', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ),  -- | 0 |
            ( '1', '1', '1', '1', '1', '1', '1', '1', '1' ),  -- | 1 |
           ( 'X', 'X', 'X', '1', 'X', 'X', 'X', '1', 'X' ),  -- | Z |
           ( 'X', 'X', 'X', '1', 'X', 'X', 'X', '1', 'X' ),  -- | W |
           ( 'X', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ),  -- | L |
            ( '1', '1', '1', '1', '1', '1', '1', '1', '1' ),  -- | H |
           ( 'X', 'X', 'X', '1', 'X', 'X', 'X', '1', 'X' )   -- | - |
```

```
    );
        -- truth table for "xor" function
    -- NOTE SPECIAL CASE FOR 'U' XOR 'U' NOT!!!!!
        CONSTANT xor_table : stdlogic_table := (
        --                      ---------------------------------------------------
        --    | U   X   0   1   Z   W   L   H   -      | |
        --                      ---------------------------------------------------
              ( 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ),  -- | U |
              ( 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ),  -- | X |
              ( 'X', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ),  -- | 0 |
              ( 'X', 'X', '1', '0', 'X', 'X', '1', '0', 'X' ),  -- | 1 |
              ( 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ),  -- | Z |
              ( 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ),  -- | W |
              ( 'X', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ),  -- | L |
              ( 'X', 'X', '1', '0', 'X', 'X', '1', '0', 'X' ),  -- | H |
              ( 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' )   -- | - |
    );
        -- truth table for "not" function
        CONSTANT not_table: stdlogic_1D :=
        --              ---------------------------------------------------
        -- | U   X   0   1   Z   W   L   H   - |
        --              ---------------------------------------------------
            ( 'X', 'X', '1', '0', 'X', 'X', '1', '0', 'X' );
        -- truth table for MIN function
        CONSTANT MIN_table : stdlogic_table := (
        --                      ---------------------------------------------------
        --    | U   X   0   1   Z   W   L   H   -      | |
        --                      ---------------------------------------------------
              ( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-' ),  -- | U |
              ( 'X', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-' ),  -- | X |
              ( '0', '0', '0', '0', 'Z', 'W', 'L', 'H', '-' ),  -- | 0 |
              ( '1', '1', '0', '1', 'Z', 'W', 'L', 'H', '-' ),  -- | 1 |
              ( 'Z', 'Z', 'Z', 'Z', 'Z', 'Z', 'Z', 'Z', 'Z' ),  -- | Z |
              ( 'W', 'W', 'W', 'W', 'Z', 'W', 'L', 'H', '-' ),  -- | W |
              ( 'L', 'L', 'L', 'L', 'Z', 'L', 'L', 'L', '-' ),  -- | L |
              ( 'H', 'H', 'H', 'H', 'Z', 'H', 'L', 'H', '-' ),  -- | H |
                ( '-', '-', '-', '-', 'Z', '-', '-', '-', '-' )   -- | - |
    );
                    ---------------------------------------------------------
- - - - - - - -
    -- Overloaded Logical Operators ( with optimizing hints )
                    ---------------------------------------------------------
- - - - - - - -
    FUNCTION "and"  ( l : std_ulogic; r : std_ulogic ) RETURN std_ulogic IS
    BEGIN
        RETURN (and_table(L, R));
    END "and";
    FUNCTION ANDD  ( l : X01; r :  X01) RETURN  X01 IS
    BEGIN
        RETURN (and_table(L, R));
    END ANDD;
```

```
 FUNCTION ANDDZ  ( l : X01Z; r :  X01Z) RETURN  X01Z IS
BEGIN
     RETURN (and_table(L, R));
END ANDDZ;
 FUNCTION "nand" ( l : std_ulogic; r : std_ulogic ) RETURN std_ulogic IS
BEGIN
      RETURN  (not_table ( and_table(L, R)));
END "nand";
 FUNCTION "or"   ( l : std_ulogic; r : std_ulogic ) RETURN std_ulogic IS
BEGIN
      RETURN (or_table(L, R));
 END "or";
 FUNCTION "nor"  ( l : std_ulogic; r : std_ulogic ) RETURN std_ulogic IS
BEGIN
      RETURN  (not ( or_table( L, R )));
 END "nor";
 FUNCTION "xor"  ( l : std_ulogic; r : std_ulogic ) RETURN std_ulogic IS
BEGIN
      RETURN (xor_table(L, R));
 END "xor";
FUNCTION XOOR  ( l : X01; r :  X01) RETURN  X01 IS
BEGIN
      RETURN (xor_table(L, R));
END XOOR;
FUNCTION XOORZ  ( l : X01Z; r :  X01Z) RETURN  X01Z IS
BEGIN
      RETURN (xor_table(L, R));
END XOORZ;
 FUNCTION "not"  ( l : std_ulogic ) RETURN std_ulogic IS
BEGIN
      RETURN (not_table(L));
 END "not";
FUNCTION noot  ( l : X01 ) RETURN X01 IS
BEGIN
      RETURN (not_table(L));
END noot;
FUNCTION nootZ  ( l : X01Z ) RETURN X01Z IS
BEGIN
      RETURN (not_table(L));
END nootZ;
 FUNCTION MIN_F  ( l, r : std_logic) RETURN std_logic IS
BEGIN
      RETURN (MIN_table(L, R));
END MIN_F;
 FUNCTION MIN_F_X01Z  ( l, r : X01Z) RETURN X01Z IS
BEGIN
      RETURN (MIN_table(L, R));
 END MIN_F_X01Z;
FUNCTION CON  ( l, r : std_logic) RETURN std_logic IS
BEGIN
      RETURN (resolution_table(L, R));
```

```
END CON;
FUNCTION CONVERT ( I : std_logic) RETURN X01 IS
BEGIN
      RETURN (convert_to_X01(L));
END CONVERT;
FUNCTION CONVERT_B_9 ( I : bit) RETURN STD_LOGIC IS
BEGIN
      IF ( I = '0')
      then
            return ( '0');
      else
            return ( '1');
      end if;
END CONVERT_B_9;
FUNCTION CONVERT_9_B ( I : STD_LOGIC ) RETURN bit IS
BEGIN
     IF ( I = '0')
    then
          return ( '0');
     elsif ( I = 'L')
      then
          return ( '0');
      else
            return ( '1');
    end if;
END CONVERT_9_B;
FUNCTION CONVERTZ ( I : std_logic) RETURN X01Z IS
BEGIN
      RETURN (convert_to_X01Z(L));
END CONVERTZ;
                  ------------------------------------------------------------
- - - - - - -
   -- Vectorized Overloaded Logical Operators
                  ------------------------------------------------------------
- - - - - - -
   FUNCTION MIN_F ( L,R : std_logic_vector ) RETURN std_logic_vector IS
      -- Note : Implementations may use aliases instead of the variables
      VARIABLE LV : std_logic_vector ( 1 to L'length ) := L;
      VARIABLE RV : std_logic_vector ( 1 to R'length ) := R;
       VARIABLE result : std_logic_vector ( 1 to L'length ) := (Others => 'X');
  begin
      if ( L'length /= R'length ) then
        assert false
         report "Arguments of overloaded 'and' operator are not of the same length"
        severity FAILURE;
    else
         for i in result'range loop
               result(i) := MIN_table (LV(i), RV(i));
       end loop;
     end if;
       return result;
```

```
    end MIN_F;
    FUNCTION MIN_F_X01Z  ( L,R : X01Z_vector ) RETURN X01Z_vector IS
        VARIABLE LV : X01Z_vector ( 1 to L'length ) := L;
        VARIABLE RV : X01Z_vector ( 1 to R'length ) := R;
        VARIABLE result : X01Z_vector ( 1 to L'length ) := (Others => 'X');
    begin
        if ( L'length /= R'length ) then
          assert false
report "Arguments of overloaded 'and' operator are not of the same length"
          severity FAILURE;
       else
          for i in result'range loop
                result(i) := MIN_table (LV(i), RV(i));
          end loop;
        end if;
        return result;
    end MIN_F_X01Z;
    FUNCTION CONVERT  ( L : std_logic_vector ) RETURN X01_vector IS
        VARIABLE LV : std_logic_vector ( 1 to L'length ) := L;
        VARIABLE result : X01_vector ( 1 to L'length ) := (Others => 'X');
    begin
          for i in result'range loop
                result(i) := convert_to_X01(LV(i));
          end loop;
        return result;
    end CONVERT;
    FUNCTION CONVERT_B_9  ( L : bit_vector ) RETURN std_logic_vector IS
        VARIABLE LV :  bit_vector ( 1 to L'length ) := L;
        VARIABLE result : std_logic_vector ( 1 to L'length ) := (Others => 'X');
    begin
          for i in result'range loop
                IF  ( LV(i)  =  '0')
                THEN
                      result(i)  :=  '0';
                else
                      result(i)  :=  '1';
                end if;
          end loop;
        return result;
    end CONVERT_B_9;
    FUNCTION CONVERT_9_B  ( L : std_logic_vector ) RETURN bit_vector IS
        VARIABLE LV :  std_logic_vector ( 1 to L'length ) := L;
        VARIABLE result : bit_vector ( 1 to L'length ) := (Others => '0');
    begin
          for i in result'range loop
             IF ( LV(i) = '0')
          THEN
                result(i) := '0';
             elsif ( LV(i) = 'L')
               THEN
                result(i) := '0';
```

```
            else
                    result(i)  :=  '1';
          end if;
       end loop;
       return result;
end CONVERT_9_B;
FUNCTION CONVERTZ  ( L :  std_logic_vector) RETURN X01Z_vector IS
     VARIABLE LV : std_logic_vector ( 1 to L'length ) := L;
     VARIABLE result : X01Z_vector ( 1 to L'length ) := (Others => 'X');
begin
        for i in result'range loop
              result(i) := convert_to_X01Z(LV(i));
       end loop;
       return result;
end CONVERTZ;
FUNCTION CON  ( L,R : std_logic_vector ) RETURN std_logic_vector IS
     VARIABLE LV : std_logic_vector ( 1 to L'length ) := L;
     VARIABLE RV : std_logic_vector ( 1 to R'length ) := R;
     VARIABLE result : std_logic_vector ( 1 to L'length ) := (Others => 'X');
begin
     if ( L'length /= R'length ) then
       assert false
        report "Arguments of overloaded 'and' operator are not of the same length"
       severity FAILURE;
   else
        for i in result'range loop
              result(i) := resolution_table (LV(i), RV(i));
       end loop;
    end if;
     return result;
end CON;
FUNCTION "and"  ( L,R : std_ulogic_vector ) RETURN std_ulogic_vector IS
     VARIABLE LV : std_ulogic_vector ( 1 to L'length ) := L;
     VARIABLE RV : std_ulogic_vector ( 1 to R'length ) := R;
     VARIABLE result : std_ulogic_vector ( 1 to L'length ) := (Others => 'X');
begin
     if ( L'length /= R'length ) then
       assert false
        report "Arguments of overloaded 'and' operator are not of the same length"
       severity FAILURE;
   else
        for i in result'range loop
              result(i) := and_table (LV(i), RV(i));
       end loop;
    end if;
     return result;
end "and";
FUNCTION "nand" ( L,R : std_logic_vector ) RETURN std_logic_vector IS
     -- Note : Implementations may use aliases instead of the variables
     VARIABLE LV : std_logic_vector ( 1 to L'length ) := L;
     VARIABLE RV : std_logic_vector ( 1 to R'length ) := R;
```

```
    VARIABLE result : std_logic_vector ( 1 to L'length ) := (Others => 'X');
begin
    if ( L'length /= R'length ) then
      assert false
       report "Arguments of overloaded 'nand' operator are not of the same length"
       severity FAILURE;
  else
       for i in result'range loop
            result(i) := not_table(and_table (LV(i), RV(i)));
     end loop;
   end if;
    return result;
end "nand";
 FUNCTION "or"  ( L,R : std_ulogic_vector ) RETURN std_ulogic_vector IS
    -- Note : Implementations may use aliases instead of the variables
    VARIABLE LV : std_ulogic_vector ( 1 to L'length ) := L;
    VARIABLE RV : std_ulogic_vector ( 1 to R'length ) := R;
    VARIABLE result : std_ulogic_vector ( 1 to L'length ) := (Others => 'X');
begin
    if ( L'length /= R'length ) then
      assert false
       report "Arguments of overloaded 'or' operator are not of the same length"
       severity FAILURE;
  else
       for i in result'range loop
            result(i) := or_table (LV(i), RV(i));
     end loop;
   end if;
    return result;
end "or";
 FUNCTION "nor"  ( L,R : std_logic_vector ) RETURN std_logic_vector IS
    -- Note : Implementations may use aliases instead of the variables
    VARIABLE LV : std_logic_vector ( 1 to L'length ) := L;
    VARIABLE RV : std_logic_vector ( 1 to R'length ) := R;
    VARIABLE result : std_logic_vector ( 1 to L'length ) := (Others => 'X');
begin
    if ( l'length /= r'length ) then
      assert false
       report "Arguments of overloaded 'nor' operator are not of the same length"
       severity FAILURE;
  else
       for i in result'range loop
            result(i) := not_table(or_table (LV(i), RV(i)));
     end loop;
   end if;
    return result;
end "nor";
FUNCTION "xor"  ( L,R : std_logic_vector ) RETURN std_logic_vector IS
    -- Note : Implementations may use aliases instead of the variables
    VARIABLE LV : std_logic_vector ( 1 to L'length ) := L;
    VARIABLE RV : std_logic_vector ( 1 to R'length ) := R;
```

```
    VARIABLE result : std_logic_vector ( 1 to L'length ) := (Others => 'X');
begin
     if ( l'length /= r'length ) then
      assert false
       report "Arguments of overloaded 'xor' operator are not of the same length"
      severity FAILURE;
  else
       for i in result'range loop
            result(i) := xor_table (LV(i), RV(i));
     end loop;
   end if;
    return result;
end "xor";
FUNCTION ANDD  ( L,R : X01_vector ) RETURN X01_vector IS
    -- Note : Implementations may use aliases instead of the variables
    VARIABLE LV : X01_vector ( 1 to L'length ) := L;
    VARIABLE RV : X01_vector ( 1 to R'length ) := R;
     VARIABLE result : X01_vector ( 1 to L'length ) := (Others => 'X');
begin
     if ( l'length /= r'length ) then
      assert false
      severity FAILURE;
  else
       for i in result'range loop
            result(i) := and_table (LV(i), RV(i));
     end loop;
   end if;
    return result;
end ANDD;
FUNCTION ANDDZ  ( L,R : X01Z_vector ) RETURN X01Z_vector IS
    -- Note : Implementations may use aliases instead of the variables
    VARIABLE LV : X01Z_vector ( 1 to L'length ) := L;
    VARIABLE RV : X01Z_vector ( 1 to R'length ) := R;
     VARIABLE result : X01Z_vector ( 1 to L'length ) := (Others => 'X');
begin
     if ( l'length /= r'length ) then
      assert false
      severity FAILURE;
  else
       for i in result'range loop
            result(i) := and_table (LV(i), RV(i));
     end loop;
   end if;
    return result;
end ANDDZ;
FUNCTION XOOR  ( L,R : X01_vector ) RETURN X01_vector IS
    -- Note : Implementations may use aliases instead of the variables
    VARIABLE LV : X01_vector ( 1 to L'length ) := L;
    VARIABLE RV : X01_vector ( 1 to R'length ) := R;
     VARIABLE result : X01_vector ( 1 to L'length ) := (Others => 'X');
begin
```

```
      if ( l'length /= r'length ) then
       assert false
       severity FAILURE;
    else
        for i in result'range loop
              result(i) := xor_table (LV(i), RV(i));
      end loop;
    end if;
     return result;
end XOOR;
 FUNCTION XOORZ  ( L,R : X01Z_vector ) RETURN X01Z_vector IS
    -- Note : Implementations may use aliases instead of the variables
    VARIABLE LV : X01Z_vector ( 1 to L'length ) := L;
    VARIABLE RV : X01Z_vector ( 1 to R'length ) := R;
     VARIABLE result : X01Z_vector ( 1 to L'length ) := (Others => 'X');
 begin
      if ( l'length /= r'length ) then
       assert false
       severity FAILURE;
   else
        for i in result'range loop
              result(i) := xor_table (LV(i), RV(i));
      end loop;
    end if;
     return result;
end XOORZ;
 FUNCTION noot  ( l : X01_vector ) RETURN X01_vector IS
    -- Note : Implementations may use aliases instead of the variables
    VARIABLE LV : X01_vector ( 1 to L'length ) := L;
     VARIABLE result : X01_vector ( 1 to L'length ) := (Others => 'X');
 begin
    for i in result'range loop
         result(i) := not_table( LV(i) );
   end loop;
     return result;
end;
 FUNCTION nootZ  ( l : X01Z_vector ) RETURN X01Z_vector IS
    -- Note : Implementations may use aliases instead of the variables
    VARIABLE LV : X01Z_vector ( 1 to L'length ) := L;
     VARIABLE result : X01Z_vector ( 1 to L'length ) := (Others => 'X');
begin
    for i in result'range loop
         result(i) := not_table( LV(i) );
   end loop;
     return result;
end;
 FUNCTION "not"  ( l : std_ulogic_vector ) RETURN std_ulogic_vector IS
    -- Note : Implementations may use aliases instead of the variables
    VARIABLE LV : std_ulogic_vector ( 1 to L'length ) := L;
     VARIABLE result : std_ulogic_vector ( 1 to L'length ) := (Others => 'X');
begin
```

```
      for i in result'range loop
          result(i) := not_table( LV(i) );
      end loop;
      return result;
  end;
                    ------------------------------------------------------------
- - - - - - -
   -- Edge Detection
                    ------------------------------------------------------------
- - - - - - -
  FUNCTION rising_edge  (SIGNAL s : std_ulogic) RETURN boolean is
  begin
      return (s'event and (convert_to_X01(s) = '1') and
                    (convert_to_X01(s'last_value) = '0'));
  end;
  FUNCTION falling_edge (SIGNAL s : std_ulogic) RETURN boolean is
  begin
      return (s'event and (convert_to_X01(s) = '0') and
                    (convert_to_X01(s'last_value) = '1'));
  end;
END logic_system;
```

D.5 Example Circuit C17 Netlist VHDL code

```
-------------------------------------------------------
--              Parallel Fault simulation             - -
--              name - S_C17                          - -
- -     chris  ryan                    - -
--              12-05-92                        - -
--                                      - -
-------------------------------------------------------
library  WORK;
library  STD;
use STD.TEXTIO.all, work.logic_system.all,   work.pnew1.all,
work.pnew.all,work.pnew9.all,  work.pnew8.all,    work.all  ;


entity S_C17 is
port(
------------      inputs      ---------------
I_gat_in : inout STD_LOGIC_VECTOR(0 to 4) := "ZZZZZ" ;
------------      outputs     ---------------
I_gat_out : inout STD_LOGIC_VECTOR(0 to 1) := "ZZ" ;
---------------------------------------
FAULTS      : inout X01Z_DIMEN( 1 to 26);
MAX_NUMBERS  : out integer_array(0 to 6)   ;
   DETECT      : in  X01                          := '1'   ;
   DETECTED   : inout X01_A_RES                  := '1'   ;
   SIM_GOOD   : in BIT                          := '0'     ;
    LEVELS_ALL  : LEVEL_R_VECTOR(0 to 6)                  ;
    MAX_LEVELS  : out integer                          := 6     ;
   COUNT       : in integer                      := 0;
    STOP       : in   BIT                 := '0'  ;
    FAULTS_DET_T : out REAL                       :=              0.0;
    FAULTS_UDET_T : out REAL                      :=          0.0;
    FAULTS_PDET_T : out REAL                      :=          0.0;
    FAULTS_ZDET_T : out REAL                      :=          0.0;
  COVERAGE     : out REAL              :=              0.0 ) ;
end S_c17;


architecture FIRST of S_C17 is

signal  c17_ch_1_0_I1gat  : STD_LOGIC  := 'Z';
signal  c17_ch_1_0_I2gat  : STD_LOGIC  := 'Z';
signal  c17_ch_3_0_I3gat  : STD_LOGIC  := 'Z';
signal  c17_ch_3_0_I6gat  : STD_LOGIC  := 'Z';
signal  c17_ch_3_0_I7gat  : STD_LOGIC  := 'Z';
signal  c17_ch_2_0_I22gat  : STD_LOGIC  := 'Z';
signal  c17_ch_1_0_I23gat  : STD_LOGIC  := 'Z';
signal  GND  : STD_LOGIC := '0';
```

```
signal   c17_row2_0_ai2s_1_6_8_22  : STD_LOGIC := 'Z';
signal  Vdd : STD_LOGIC := '1';
signal   c17_ch_2_0_I10  : STD_LOGIC := 'Z';
signal   c17_row1_0_ai2s_1_6_8_22  : STD_LOGIC := 'Z';
signal   c17_row1_0_ai2s_0_6_8_22  : STD_LOGIC := 'Z';
signal   c17_ch_2_0_I10PD  : STD_LOGIC := 'Z';
signal   c17_ch_2_0_I12  : STD_LOGIC := 'Z';
signal   c17_ch_2_0_I8  : STD_LOGIC := 'Z';
signal   c17_row2_0_aoi21s_0_6_4_86  : STD_LOGIC := 'Z';
signal   c17_row2_0_aoi21s_0_6_12_18  : STD_LOGIC := 'Z';
signal   c17_ch_2_0_I12PD  : STD_LOGIC := 'Z';
signal   c17_ch_2_0_I13  : STD_LOGIC := 'Z';
signal   c17_ch_2_0_I8PD  : STD_LOGIC := 'Z';
signal   c17_row2_0_ai2s_0_6_8_22  : STD_LOGIC := 'Z';
signal   c17_ch_2_0_I7  : STD_LOGIC := 'Z';
signal  NUMBER_1 : integer      := 1;
signal  NUMBER_2 : integer      := 2;
signal  NUMBER_3 : integer      := 3;
signal  NUMBER_4 : integer      := 4;
signal  NUMBER_5 : integer      := 5;
signal  NUMBER_6 : integer      := 6;
signal  NUMBER_7 : integer      := 7;
signal  NUMBER_8 : integer      := 8;

component N_FAULTA

port(
    GATE      : in   STD_LOGIC           := 'Z'  ;
    DRAIN     : in   STD_LOGIC           := 'Z'  ;
    SOURCE    : inout  STD_LOGIC         := 'Z'  ;
    FAULTS    : inout X01Z_vector( 1 to 6)      := "111111";
    DETECT    : in  X01                  := '1'  ;
    DETECTED  : inout X01_A_RES          := '1'  ;
    LEVEL_ALL : in LEVEL_R               ;
    NUMBER : in integer                  := 0;
    STOP      : in   BIT                 := '0' );
end component;

component P_FAULTA
port(
    GATE      : in   STD_LOGIC           := 'Z'  ;
    DRAIN     : in   STD_LOGIC           := 'Z'  ;
    SOURCE    : inout  STD_LOGIC         := 'Z'  ;
    FAULTS    : inout X01Z_vector( 1 to 6)      := "111111";
    DETECT    : in  X01                  := '1'  ;
    DETECTED  : inout X01_A_RES          := '1'  ;
    LEVEL_ALL : in LEVEL_R               ;
    NUMBER : in integer                  := 0;
    STOP      : in   BIT                 := '0' );
end component;
```

```
component PD_FAULTA
port(
   DRAIN     : in   STD_LOGIC            := 'Z'  ;
   SOURCE    : inout  STD_LOGIC          := 'Z'  ;
   DETECTED  : inout X01_A_RES           := '1'  ;
   LEVEL_ALL : in LEVEL_R_VECTOR               ;
   STOP      : in   BIT                  := '0' );
end component;

for all: N_FAULTA use entity N_FAULT(ONLY);

for all: P_FAULTA use entity P_FAULT(ONLY);

for all: PD_FAULTA use entity PD_FAULT(ONLY);


begin
--------------------------------      READ   INPUT   VECTORS      -----------
- - - - - - - - - - - - - - - - - - - - - -
new_vect: process
variable VLINE: LINE;
variable V: bit_vector(0 to 4);
file INVECT: TEXT is "c17.test_in";
begin
while not(ENDFILE(INVECT)) loop
        READLINE(INVECT,VLINE);
        READ(VLINE,V);
        I_gat_in <= convert_B_9(V);
     wait on STOP;
end loop;
end process;
--------------------------------    WRITE   GOOD  OUTPUT  RESPOUNSES   --
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
outwrite: process(SIM_GOOD)
variable FAULTS_DET,FAULTS_UDET,FAULTS_PDET,FAULTS_ZDET : REAL := 0.0;
variable FAULTS_TEMP : X01Z_vector( 1 to 6);
variable OLINE: LINE;
variable O: bit_vector(0 to 1);
file OUTVECT: TEXT is out "c17.test_out";
begin
IF (SIM_GOOD = '0')
then
-------------------     FAULT  COVERAGE     ------
     FOR i in 1 to 26 loop
          FAULTS_TEMP := FAULTS(i);
          FOR J in 1 to 6 loop
               IF (FAULTS_TEMP(J) = '0')
            THEN
                 FAULTS_DET := FAULTS_DET + 1.0;
              ELSIF (FAULTS_TEMP(J) = '1')
            THEN
```

```
                    FAULTS_UDET := FAULTS_UDET + 1.0;
                ELSIF (FAULTS_TEMP(J) = 'X')
                THEN
                    FAULTS_PDET := FAULTS_PDET + 1.0;
                ELSIF (FAULTS_TEMP(J) = 'Z')
                THEN
                    FAULTS_ZDET := FAULTS_ZDET + 1.0;
                END IF;
            END LOOP;
        END LOOP;
        FAULTS_DET_T <= FAULTS_DET;
        FAULTS_UDET_T <= FAULTS_UDET;
        FAULTS_PDET_T <= FAULTS_PDET;
        FAULTS_ZDET_T <= FAULTS_ZDET;
        COVERAGE <= FAULTS_DET/(FAULTS_DET + FAULTS_UDET + FAULTS_PDET);
        O := convert_9_b(I_GAT_OUT);
        WRITE(OLINE,O);
        WRITELINE(OUTVECT,OLINE);
        FAULTS_DET := 0.0;
        FAULTS_UDET := 0.0;
        FAULTS_PDET := 0.0;
        FAULTS_ZDET := 0.0;
end if;
end process;
--------------------------------        INPUTS          --------------------
------------
c17_ch_1_0_I1gat <= transport I_gat_in(0) ;
c17_ch_1_0_I2gat <= transport I_gat_in(1) ;
c17_ch_3_0_I3gat <= transport I_gat_in(2) ;
c17_ch_3_0_I6gat <= transport I_gat_in(3) ;
c17_ch_3_0_I7gat <= transport I_gat_in(4) ;
--------------------------------        OUTPUTS         --------------------
------------
I_gat_out(0) <= transport c17_ch_2_0_I22gat ;
I_gat_out(1) <= transport c17_ch_1_0_I23gat ;
--------------------------------      VDD   GND         --------------------
------------
Vdd <= transport '1' ;
GND <= transport '0' ;
--------------------------------    NUMBER OF REVERSE LEVELS      ------
--
MAX_LEVELS <= 6 ;
--------------------------------     REVERSE LEVEL WIDTHS       ---------
------
MAX_NUMBERS(0 to 6) <= (
            (0),(5),(4),(8),(4),(4),(1)
      ) ;
-----------------     VHDL CIRCUIT NETLIST       -----------------------
-------
T1:n_faulta
    port
```

```
      map(c17_ch_3_0_I3gat,GND,c17_row2_0_ai2s_1_6_8_22,FAULTS(1),DET
ECT,
                              DETECTED,LEVELS_ALL(6),NUMBER_1,STOP);
T2:p_faulta
      port  map(c17_ch_3_0_I3gat,Vdd,c17_ch_2_0_I10PD,FAULTS(2),DETECT,
                              DETECTED,LEVELS_ALL(5),NUMBER_1,STOP);
T3:p_faulta
      port  map(c17_ch_3_0_I6gat,Vdd,c17_ch_2_0_I10PD,FAULTS(3),DETECT,
                              DETECTED,LEVELS_ALL(5),NUMBER_2,STOP);
T4:n_faulta
      port
map(c17_ch_3_0_I6gat,c17_row2_0_ai2s_1_6_8_22,c17_ch_2_0_I10PD,

      FAULTS(4),DETECT,DETECTED,LEVELS_ALL(5),NUMBER_3,STOP);
T5:n_faulta
       port
map(c17_ch_1_0_I2gat,GND,c17_row1_0_ai2s_1_6_8_22,FAULTS(5),DETECT,
                              DETECTED,LEVELS_ALL(5),NUMBER_4,STOP);
T6:n_faulta
      port
map(c17_ch_1_0_I1gat,GND,c17_row1_0_ai2s_0_6_8_22,FAULTS(6),DETECT,
                              DETECTED,LEVELS_ALL(4),NUMBER_1,STOP);
PD1:PD_faulta
      port  map(c17_ch_2_0_I10PD,c17_ch_2_0_I10,DETECTED,LEVELS_ALL(4  to
4),STOP);
T7:p_faulta
      port  map(c17_ch_2_0_I10PD,Vdd,c17_ch_2_0_I12PD,FAULTS(7),DETECT,
                              DETECTED,LEVELS_ALL(4),NUMBER_2,STOP);
T8:n_faulta
      port
map(c17_ch_2_0_I10PD,c17_row1_0_ai2s_1_6_8_22,c17_ch_2_0_I12PD,

      FAULTS(8),DETECT,DETECTED,LEVELS_ALL(4),NUMBER_3,STOP);
T9:p_faulta
      port  map(c17_ch_1_0_I2gat,Vdd,c17_ch_2_0_I12PD,FAULTS(9),DETECT,
                              DETECTED,LEVELS_ALL(4),NUMBER_4,STOP);
T10:p_faulta
      port  map(c17_ch_1_0_I1gat,Vdd,c17_ch_2_0_I8PD,FAULTS(10),DETECT,
                              DETECTED,LEVELS_ALL(3),NUMBER_1,STOP);
T11:p_faulta
      port  map(c17_ch_3_0_I3gat,Vdd,c17_ch_2_0_I8PD,FAULTS(11),DETECT,
                        DETECTED,LEVELS_ALL(3),NUMBER_2,STOP);
T12:n_faulta
      port
map(c17_ch_3_0_I3gat,c17_row1_0_ai2s_0_6_8_22,c17_ch_2_0_I8PD,

      FAULTS(12),DETECT,DETECTED,LEVELS_ALL(3),NUMBER_3,STOP);
T13:p_faulta
   port
map(c17_ch_3_0_I7gat,Vdd,c17_row2_0_aoi21s_0_6_4_86,FAULTS(13),DETECT,
                              DETECTED,LEVELS_ALL(3),NUMBER_4,STOP);
```

```
T14:p_faulta
    port
map(c17_ch_2_0_I10,Vdd,c17_row2_0_aoi21s_0_6_4_86,FAULTS(14),DETECT,
                        DETECTED,LEVELS_ALL(3),NUMBER_5,STOP);
T15:n_faulta
    port
map(c17_ch_2_0_I10,GND,c17_row2_0_aoi21s_0_6_12_18,FAULTS(15),DETECT,
                        DETECTED,LEVELS_ALL(3),NUMBER_6,STOP);
PD2:PD_faulta
       port map(c17_ch_2_0_I12PD,c17_ch_2_0_I12,DETECTED,LEVELS_ALL(2 to
3),STOP);
T16:n_faulta
       port map(c17_ch_2_0_I12PD,GND,c17_ch_2_0_I13,FAULTS(16),DETECT,
                        DETECTED,LEVELS_ALL(3),NUMBER_7,STOP);
T17:p_faulta
       port map(c17_ch_2_0_I12PD,Vdd,c17_ch_2_0_I13,FAULTS(17),DETECT,
                        DETECTED,LEVELS_ALL(3),NUMBER_8,STOP);
PD3:PD_faulta
       port map(c17_ch_2_0_I8PD,c17_ch_2_0_I8,DETECTED,LEVELS_ALL(2 to
2),STOP);
T18:n_faulta
       port
map(c17_ch_2_0_I8PD,GND,c17_row2_0_ai2s_0_6_8_22,FAULTS(18),DETECT,
                        DETECTED,LEVELS_ALL(2),NUMBER_1,STOP);
T19:n_faulta
       port map(c17_ch_2_0_I13,GND,c17_ch_2_0_I7,FAULTS(19),DETECT,
                        DETECTED,LEVELS_ALL(2),NUMBER_2,STOP);
T20:p_faulta
       port
map(c17_ch_2_0_I13,c17_row2_0_aoi21s_0_6_4_86,c17_ch_2_0_I7,

       FAULTS(20),DETECT,DETECTED,LEVELS_ALL(2),NUMBER_3,STOP);
T21:n_faulta
       port
map(c17_ch_3_0_I7gat,c17_row2_0_aoi21s_0_6_12_18,c17_ch_2_0_I7,

       FAULTS(21),DETECT,DETECTED,LEVELS_ALL(2),NUMBER_4,STOP);
T22:p_faulta
       port map(c17_ch_2_0_I8,Vdd,c17_ch_2_0_I22gat,FAULTS(22),DETECT,
                        DETECTED,LEVELS_ALL(1),NUMBER_1,STOP);
T23:p_faulta
        port map(c17_ch_2_0_I12,Vdd,c17_ch_2_0_I22gat,FAULTS(23),DETECT,
                        DETECTED,LEVELS_ALL(1),NUMBER_2,STOP);
T24:n_faulta
       port
map(c17_ch_2_0_I12,c17_row2_0_ai2s_0_6_8_22,c17_ch_2_0_I22gat,

       FAULTS(24),DETECT,DETECTED,LEVELS_ALL(1),NUMBER_3,STOP);
T25:n_faulta
       port map(c17_ch_2_0_I7,GND,c17_ch_1_0_I23gat,FAULTS(25),DETECT,
                                DETECTED,LEVELS_ALL(1),NUMBER_4,STOP);
```

```
T26:p_faulta
        port  map(c17_ch_2_0_I7,Vdd,c17_ch_1_0_I23gat,FAULTS(26),DETECT,
                                        DETECTED,LEVELS_ALL(1),NUMBER_5,STOP);

end FIRST ;
```

D.6 Test_Bench Controller VHDL code

The VHDL program that follows provides the control signals that stimulate the compiled circuit. The control signals are used to model the fault simulation algorithm. Except for Inputs, Outputs, Fault numbers, level number, and level widths, this file is the same for all circuits.

```
-------------------------------------------------------
--      Parallel Fault simulation            - -
--      name - c17                    - -
- -     chris ryan                    - -
- -     12-09-92                      - -
-------------------------------------------------------
library WORK;
use work.logic_system.all,   work.pnew1.all, work.pnew.all,work.pnew9.all,
work.pnew8.all,work.all  ;
entity TEST_BENCH_c17 is
end TEST_BENCH_c17;
architecture C17_1 of TEST_BENCH_c17 is
signal          I_gat_in :STD_LOGIC_vector(0 to 4)   := "01111";
signal  I_gat_out :STD_LOGIC_vector(0 to 1)  := "ZZ";
signal     FAULTS      : X01Z_DIMEN( 1 to 26);
signal     DETECT     : X01_A_RES                  := '1'  ;
signal     DETECTED   : X01_A_RES                ;
signal     SIM_GOOD   : BIT                  := '0'    ;
signal     NUMBER_CNTL : integer          := 0   ;
signal     LEVELS_ALL  : LEVEL_R_VECTOR(0 to 6)      ;
signal    COUNT        :  integer                      := 0;
signal     MAX_LEVELS  :  integer                 := 6;
signal     MAX_NUMBERS :  integer_ARRAY(0 to 6)       ;
signal     STOP       : BIT                 := '0'  ;
signal     FAULTS_DET_T : REAL          :=          0.0;
signal     FAULTS_UDET_T : REAL         :=          0.0;
signal     FAULTS_PDET_T : REAL         :=          0.0;
signal     FAULTS_ZDET_T : REAL         :=          0.0;
signal     COVERAGE     : REAL        :=          0.0 ;
signal     INIT       : BIT                 := '0'  ;
signal     LEVEL_CNTL_V  : integer                 := 0   ;
signal     LEVEL_CNTL_FS  : integer                 := 0   ;
signal     LEVEL_CNTL_IS  : integer                 := 0   ;
signal     LEVEL_CNTL_I_1  : integer                 := 0   ;
signal     LEVEL_CNTL_I_X  : integer                 := 0   ;
signal     LEVEL_CNTL_2PH_V  : integer                 := 0   ;
signal     LEVEL_CNTL_2PH_FS  : integer                 := 0   ;
signal     LEVEL_CNTL_2PH_IS  : integer                 := 0   ;
signal     LEVEL_CNTL_2PH_I  : integer                 := 0   ;
signal     LEVEL_STRB_1  : BIT                  := '0'   ;
signal     LEVEL_STRB_2  : BIT                  := '0'   ;
signal     LEVEL_STRB_3  : BIT                  := '0'   ;
```

```
signal     L_VAL: integer    := 6   ;
signal     M_VAL: integer      := 1    ;
signal     N_VAL: integer    := 1    ;
signal     ACT_V_2P: BIT                    :=  '0';
signal     ACT_V_NEXT: BIT                  :=  '0';
signal     ACT_V_NEXT_2P: BIT                :=  '0';
signal     ACT_V_AGAIN: BIT                :=  '0';
signal     ACT_FS_NEW: BIT                  :=  '0';
signal     ACT_FS_NEXT: BIT                 :=  '0';
signal     ACT_FS_2P: BIT                :=  '0';
signal     ACT_I_NEW: BIT                   :=  '0';
signal     ACT_I_2P: BIT                 :=  '0';
signal     ACT_I_NEXT: BIT                  :=  '0';
signal     ACT_IS_NEW: BIT                  :=  '0';
signal     ACT_IS_NEXT: BIT                 :=  '0';
signal     ACT_IS_2P: BIT                :=  '0';
signal     INJECT_SWITCH: BIT                :=  '1';
signal     GDEL : TIME_RES                          :=  0 NS;
signal     COUNT_1  : integer                       :=  0;
signal     COUNT_2  : integer                       :=  0;
signal     COUNT_3  : integer                       :=  0;
signal     COUNT_4  : integer                       :=  0;
signal  c17_ch_1_0_I1gat_V  :integer:=  2;
signal   c17_ch_3_0_I3gat_V  :integer:=  3;
signal   c17_ch_3_0_I6gat_V  :integer  :=3  ;
signal   c17_ch_1_0_I2gat_V  :integer:=  3;
signal   c17_ch_3_0_I7gat_V   :integer:=3  ;
component C17_A
port(I_gat_in     :inout  STD_LOGIC_vector(0  to  4)   :=  "01111";
     I_gat_out   :inout STD_LOGIC_vector(0 to 1)   :=  "ZZ";
    FAULTS       : inout X01Z_D!MEN( 1 to 26) ;
     MAX_NUMBERS :  out integer_array(0 to 6)        ;
    DETECT       : inout      X01                        :=  '1'    ;
    DETECTED    : inout X01_A_RES                ;
    SIM_GOOD    : in BIT                  :=  '0'       ;
     LEVELS_ALL : IN LEVEL_R_VECTOR(0 to 6)                          ;
     MAX_LEVELS  :  out integer              :=  6      ;
     COUNT       : in integer                :=  0;
     STOP        : in    BIT                 :=  '0'  ;
     FAULTS_DET_T : out REAL           :=          0.0;
     FAULTS_UDET_T : out REAL          :=          0.0;
     FAULTS_PDET_T : out REAL          :=          0.0;
     FAULTS_ZDET_T : out REAL          :=          0.0;
    COVERAGE     : out REAL         :=          0.0 ) ;
end component;
for  L1:  C17_A  use  entity  S_C17(first);
begin
  L1: C17_A
      port
map(I_gat_in,I_gat_out,FAULTS,MAX_NUMBERS,DETECT,DETECTED,SIM_GOOD,
```

```
        LEVELS_ALL,MAX_LEVELS,COUNT,STOP,FAULTS_DET_T,FAULTS_UDET_T,
                FAULTS_PDET_T,FAULTS_ZDET_T,COVERAGE);
INIT <= '1';
--------------------------------  GOOD  CIRCUIT  SIMULATION  --------
- - - - - - - - - - - - - - - - - - - - - - - - - - - -
VECTORS:process(ACT_V_NEXT,ACT_V_2P,INIT)
variable EDEL : time := 0 NS;
variable Q : integer := 6;
begin
        EDEL :=  100 PS;
        IF ((ACT_V_NEXT'EVENT) or (INIT'EVENT))
        THEN
                STOP <= not(STOP);
                Q := MAX_LEVELS;
                SIM_GOOD <= '1' after EDEL;
                EDEL := EDEL + 100 PS;
                While ( Q > -1) loop
                        EDEL := EDEL + 5 NS;
                        LEVEL_CNTL_V <= transport Q after EDEL ;
                        EDEL := EDEL + 1 NS;
                        COUNT_1 <= COUNT_1 + 1;
                        EDEL := EDEL + 4 NS;
                        LEVEL_CNTL_2PH_V <= transport Q after EDEL ;
                        Q := Q - 1 ;
                end loop;
                Q := 1;
                EDEL := EDEL + 4 NS;
                ACT_V_2P <= not(ACT_V_2P) after EDEL;
        ELSIF (ACT_V_2P'EVENT)
        THEN
                LEVEL_CNTL_2PH_V <= transport Q after EDEL ;
                EDEL := EDEL + 1 NS;
                SIM_GOOD   <=  transport '0' after EDEL ;
                EDEL := EDEL + 4 NS;
                COUNT <= COUNT_1 + COUNT_2 + COUNT_3 + COUNT_4;
                ACT_FS_NEW <= not(ACT_FS_NEW) after EDEL;
        END IF;
end process;
--------------------------------FAULT    SIMULATION          ------------
- - - - - - - - - - - - - - - - - - - - - - -
FAULT_SIM_P   :process(ACT_FS_NEW,ACT_FS_NEXT,ACT_FS_2P)
variable FDEL : time := 0 NS;
begin
        FDEL := 0 NS;
        IF(ACT_FS_NEW'EVENT)
        THEN
                L_VAL <= MAX_LEVELS ;
                FDEL := FDEL + 100 PS;
                FDEL := FDEL + 100 PS;
                LEVEL_CNTL_FS  <= transport MAX_LEVELS after FDEL;
```

```
                COUNT_2 <= COUNT_2 + 1;
                FDEL := FDEL + 5 ns;
                ACT_FS_2P <= not(ACT_FS_2P) after FDEL;
        ELSIF(ACT_FS_NEXT'EVENT)
        THEN
                FDEL := FDEL + 100 ps;
                FDEL := FDEL + 100 ps;
                LEVEL_CNTL_FS  <= transport L_VAL after FDEL;
                FDEL := FDEL + 5 ns;
                ACT_FS_2P <= not(ACT_FS_2P) after FDEL;
        ELSIF(ACT_FS_2P'EVENT)
        THEN
                FDEL := FDEL + 5 ns;
                LEVEL_CNTL_2PH_FS  <= transport L_VAL after FDEL;
                FDEL := FDEL + 5 ns;
                IF (L_VAL = 0)
                THEN
                        ACT_V_NEXT <= not(ACT_V_NEXT) after FDEL;
                ELSE
                        L_VAL  <= L_VAL-1 ;
                        ACT_I_NEW <= not(ACT_I_NEW) after FDEL;
                END IF;
        END IF;
end process;
----------------------------------------FAULT     INJECTION     ---------
- - - - - - - - - - - - - - - - - - - - -
INJECT_P   :process(ACT_I_NEW,ACT_I_NEXT,ACT_I_2P)
variable HDEL : time := 0 NS;
begin
        HDEL := 100 PS;
        IF(ACT_I_NEW'EVENT)
        THEN
                M_VAL <= 1;
                NUMBER_CNTL  <= 1;
                HDEL := HDEL + 1 ns;
                IF (INJECT_SWITCH = '1')
                THEN
                        HDEL := HDEL + 1 ns;
                        LEVEL_CNTL_I_1 <= TRANSPORT L_VAL + 1 after HDEL;
                ELSE
                        HDEL := HDEL + 1 ns;
                        LEVEL_CNTL_I_X <= TRANSPORT L_VAL + 1 after HDEL;
                END IF;
                COUNT_3 <= COUNT_3 + 1;
                HDEL := HDEL + 5 ns;
                ACT_I_2P <= not(ACT_I_2P) after HDEL;
        ELSIF(ACT_I_NEXT'EVENT)
        THEN
                NUMBER_CNTL <= M_VAL ;
                HDEL := HDEL + 1 ns;
                IF (INJECT_SWITCH = '1')
```

```
            THEN
                HDEL := HDEL + 1 ns;
                LEVEL_CNTL_I_1 <= TRANSPORT L_VAL + 1 after HDEL;
            ELSE
                HDEL := HDEL + 1 ns;
                LEVEL_CNTL_I_X <= TRANSPORT L_VAL + 1 after HDEL;
            END IF;
            HDEL := HDEL + 5 ns;
            ACT_I_2P <= not(ACT_I_2P) after HDEL;
        ELSIF(ACT_I_2P'EVENT)
        THEN
            LEVEL_CNTL_2PH_I <= TRANSPORT L_VAL + 1 after HDEL;
            HDEL := HDEL + 1 ns;
            LEVEL_CNTL_I_1 <= TRANSPORT -1 after HDEL;
            LEVEL_CNTL_I_X <= TRANSPORT -1 after HDEL;
            HDEL := HDEL + 5 ns;
            LEVEL_CNTL_2PH_I <= TRANSPORT 0 after HDEL;
            IF (M_VAL > MAX_NUMBERS(L_VAL+1))
            THEN
                IF (INJECT_SWITCH = '1')
                THEN
                    M_VAL <= 1;
                    NUMBER_CNTL <= 1;
                    INJECT_SWITCH <= '0';
                    HDEL := HDEL + 1 ns;
                    ACT_IS_NEW <= not(ACT_IS_NEW) after HDEL;
                ELSE
                    INJECT_SWITCH <= '1';
                    HDEL := HDEL + 100 ps;
                    ACT_FS_NEXT <= not(ACT_FS_NEXT) after HDEL;
                END IF;
            ELSE
                M_VAL <= M_VAL + 1 after HDEL;
                ACT_IS_NEW <= not(ACT_IS_NEW) after HDEL;
            END IF;
        END IF;
end process;
---------------------- INJECTED FAULT SIMULATION ----------------
- - - - - - - - - - - - - - - - - - - - -
INJT_SIM_p:Process(ACT_IS_NEW,ACT_IS_NEXT,ACT_IS_2P)
variable IDEL : time := 0 NS;
begin
        IDEL := 100 ps;
        IF (ACT_IS_NEW'EVENT)
        THEN
            N_VAL <= L_VAL ;
            ACT_IS_NEXT <= not(ACT_IS_NEXT) after IDEL;
        ELSIF (ACT_IS_NEXT'EVENT)
        THEN
            IF (N_VAL > -1)
            THEN
```

```
                IF (DETECTED = '1')
                THEN
                                DETECT <= '1';
                        IDEL := IDEL + 100 ps;
                        LEVEL_CNTL_IS <= transport -2  after IDEL;
                        IDEL := IDEL + 13 ns;
                        ACT_IS_2P <= not(ACT_IS_2P) after IDEL;
                ELSIF (DETECTED = '0')
                THEN
                        LEVEL_CNTL_IS <= transport -2  after IDEL;
                        IDEL := IDEL + 5 ns;
                        IDEL := IDEL + 100 ps;
                        LEVEL_CNTL_IS  <= transport N_VAL after IDEL;
                        COUNT_4 <= COUNT_4 + 1;
                        IDEL := IDEL + 8 ns;
                        ACT_IS_2P <= not(ACT_IS_2P) after IDEL;
                ELSE
                                IDEL := IDEL + 100 ps;
                                LEVEL_CNTL_IS <= transport -1 after IDEL;
                                IDEL := IDEL + 5 ns;
                                ACT_IS_2P <= not(ACT_IS_2P) after IDEL;
                END IF;
        ELSE
                IF (DETECTED = '0')
                THEN
                        DETECT <= '0';
                ELSIF (DETECTED = '1')
                THEN
                        DETECT <= '1';
                ELSE
                        DETECT <= 'X';
                END IF;
                        IDEL := IDEL + 100 ps;
                LEVEL_CNTL_IS <= transport -2 after IDEL;
                IDEL := IDEL + 5 ns;
                ACT_IS_2P <= not(ACT_IS_2P) after IDEL;
        END IF;
ELSIF (ACT_IS_2P'EVENT)
THEN
        LEVEL_CNTL_2PH_IS <= transport N_VAL after IDEL;
        IDEL := IDEL + 5 ns;
        IF(( DETECTED = '1') or (N_VAL < 0) or ( DETECTED = 'X'))
        THEN
                IF( DETECTED = 'X')
                THEN
                        DETECT <= 'X';
                ELSIF (DETECTED = '1')
                        THEN
                        DETECT <= '1';
                        END IF;
                        IDEL := IDEL + 300 ps;
```

```
                ACT_I_NEXT <= not(ACT_I_NEXT) after IDEL;
        ELSE
                IDEL := IDEL + 300 ps;
                N_VAL <= N_VAL - 1 after IDEL;
                IDEL := IDEL + 100 ps;
                ACT_IS_NEXT <= not(ACT_IS_NEXT) after IDEL;
        END IF;
    END IF;
end process;
------------------------------ OUTPUT MUX FOR CONTROL SIGNALS -----
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
MUX_LEVEL:
process(LEVEL_CNTL_V,LEVEL_CNTL_FS,LEVEL_CNTL_IS,LEVEL_CNTL_I_1,LEVEL_CN
TL_I_X,LEVEL_CNTL_2PH_V,LEVEL_CNTL_2PH_FS,LEVEL_CNTL_2PH_IS)
begin
    IF (LEVEL_CNTL_V'EVENT)
    THEN
        LEVELS_ALL(LEVEL_CNTL_V).T_CNTL(0) <=  '1';
    ELSIF ((LEVEL_CNTL_FS'EVENT)and(LEVEL_CNTL_FS /= 0))
    THEN
        LEVELS_ALL(LEVEL_CNTL_FS).T_CNTL(1) <=  '1';
    ELSIF (LEVEL_CNTL_IS'EVENT)
    THEN
        IF(LEVEL_CNTL_IS >= 0)
        THEN
            LEVELS_ALL(LEVEL_CNTL_IS).T_CNTL(4) <=  TRANSPORT '1';
            LEVELS_ALL(LEVEL_CNTL_IS).T_CNTL(4) <=  TRANSPORT '0'
after 4 ns;
            LEVELS_ALL(LEVEL_CNTL_IS + 1).T_CNTL(5) <=transport '1'
after8ns;
            LEVELS_ALL(LEVEL_CNTL_IS + 1).T_CNTL(5)
<=transport'0'after12ns;
        ELSIF(LEVEL_CNTL_IS = -1 )
        THEN
            LEVELS_ALL(LEVEL_CNTL_IS + 1).T_CNTL(5)
<=transport'1'after  8  ns;
            LEVELS_ALL(LEVEL_CNTL_IS + 1).T_CNTL(5) <=
transport'0'after12ns;
        ELSE
        END IF;
    ELSIF ((LEVEL_CNTL_I_1'EVENT)and(LEVEL_CNTL_I_1 /= -1))
    THEN
        LEVELS_ALL(LEVEL_CNTL_I_1).T_NUMBER_CNTL <= NUMBER_CNTL ;
        LEVELS_ALL(LEVEL_CNTL_I_1).T_CNTL(2) <=  '1'after 1 ns;
    ELSIF ((LEVEL_CNTL_I_X'EVENT)and(LEVEL_CNTL_I_X /= -1))
    THEN
        LEVELS_ALL(LEVEL_CNTL_I_X).T_NUMBER_CNTL <= NUMBER_CNTL ;
        LEVELS_ALL(LEVEL_CNTL_I_X).T_CNTL(3) <=  '1' after 1 ns;
    ELSE
        LEVELS_ALL(LEVEL_CNTL_2PH_V).T_CNTL(0) <=  '0';
        LEVELS_ALL(LEVEL_CNTL_2PH_FS).T_CNTL(1) <=  '0';
```

```
        I    F((LEVEL_CNTL_2PH_IS <= MAX_LEVELS)and(LEVEL_CNTL_2PH_IS >=
0 ) )
             THEN
                  LEVELS_ALL(LEVEL_CNTL_2PH_IS).T_CNTL(4) <=  '0';
             ELSE
             END IF;
             IF(LEVEL_CNTL_2PH_IS < MAX_LEVELS)
             THEN
                  LEVELS_ALL(LEVEL_CNTL_2PH_IS + 1).T_CNTL(5) <= '0';
             ELSE
             END IF;
             LEVELS_ALL(LEVEL_CNTL_2PH_I).T_CNTL(2) <=  '0';
             LEVELS_ALL(LEVEL_CNTL_2PH_I).T_CNTL(3) <=  '0';
      END IF;
end process;
end C17_1;
```

D.7 Detailed VHDL Simulation Output Results For C17

Fault simulation results for the switch description of the ISCAS85 circuit C17 follows. The circuit has 26 transistors. The fault values are 1, 0, X, and Z. These fault values stand for undetected, detected, potentially detected, and undetectable, respectively.

```
C17_A        COMPONENT                (no value)
M9          CE ACTIVE /TEST_BENCH_C17/COUNT
M8          CE ACTIVE /TEST_BENCH_C17/COVERAGE
M7          CE ACTIVE /TEST_BENCH_C17/FAULTS_ZDET_T
M6          CE ACTIVE /TEST_BENCH_C17/FAULTS_PDET_T
M5          CE ACTIVE /TEST_BENCH_C17/FAULTS_UDET_T
M4          CE ACTIVE /TEST_BENCH_C17/FAULTS_DET_T
M3          CE ACTIVE /TEST_BENCH_C17/I_GAT_IN
M2          CE ACTIVE /TEST_BENCH_C17/I_GAT_OUT
M1          CE ACTIVE /TEST_BENCH_C17/FAULTS
M           CE ACTIVE /TEST_BENCH_C17/MAX_NUMBERS
0 PS
    M8:   ACTIVE /TEST_BENCH_C17/COVERAGE (value = 0.0)
     M7:   ACTIVE /TEST_BENCH_C17/FAULTS_ZDET_T (value = 52.0)
     M6:   ACTIVE /TEST_BENCH_C17/FAULTS_PDET_T (value = 0.0)
     M5:   ACTIVE /TEST_BENCH_C17/FAULTS_UDET_T (value = 104.0)
     M4:   ACTIVE /TEST_BENCH_C17/FAULTS_DET_T (value = 0.0)
      M:    ACTIVE /TEST_BENCH_C17/MAX_NUMBERS (value = (0, 5, 4, 8, 4, 4, 1))
     M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "ZZ")
      M3:    ACTIVE /TEST_BENCH_C17/I_GAT_IN (value = "11010")
      M3:    ACTIVE /TEST_BENCH_C17/I_GAT_IN (value = "11010")
57200 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "11")
74200 PS
    M9:   ACTIVE /TEST_BENCH_C17/COUNT (value = 1)
75300 PS
    M8:   ACTIVE /TEST_BENCH_C17/COVERAGE (value = 0.0)
     M7:   ACTIVE /TEST_BENCH_C17/FAULTS_ZDET_T (value = 52.0)
     M6:   ACTIVE /TEST_BENCH_C17/FAULTS_PDET_T (value = 0.0)
     M5:   ACTIVE /TEST_BENCH_C17/FAULTS_UDET_T (value = 104.0)
     M4:   ACTIVE /TEST_BENCH_C17/FAULTS_DET_T (value = 0.0)
129500 PS
     M1:    ACTIVE /TEST_BENCH_C17/FAULTS (value = ("111Z1Z", "11Z1Z1",
"11Z1Z1", "111Z1Z", "111Z1Z", "111Z1Z", "11Z1Z1", "111Z1Z", "11Z1Z1",
"11Z1Z1", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1", "111Z1Z", "111Z1Z",
"11Z1Z1", "111Z1Z", "111Z1Z", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1",
"111Z1Z", "111Z1Z", "11Z1Z1"))
294400 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "00")
340200 PS
     M1:    ACTIVE /TEST_BENCH_C17/FAULTS (value = ("111Z1Z", "X1Z0Z0",
"11Z1Z1", "111Z1Z", "111Z1Z", "111Z1Z", "11Z1Z1", "111Z1Z", "11Z1Z1",
```

```
"11Z1Z1", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1", "111Z1Z", "111Z1Z",
"11Z1Z1", "111Z1Z", "111Z1Z", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1",
"111Z1Z", "111Z1Z", "11Z1Z1"))
411800 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "XX")
457600 PS
     M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("111Z1Z", "X1Z0Z0",
"X1ZXZX", "111Z1Z", "111Z1Z", "111Z1Z", "11Z1Z1", "111Z1Z", "11Z1Z1",
"11Z1Z1", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1", "111Z1Z", "111Z1Z",
"11Z1Z1", "111Z1Z", "111Z1Z", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1",
"111Z1Z", "111Z1Z", "11Z1Z1"))
500200 PS
     M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("111Z1Z", "X1Z0Z0",
"X1ZXZX", "XX1Z1Z", "111Z1Z", "111Z1Z", "11Z1Z1", "111Z1Z", "11Z1Z1",
"11Z1Z1", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1", "111Z1Z", "111Z1Z",
"11Z1Z1", "111Z1Z", "111Z1Z", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1",
"111Z1Z", "111Z1Z", "11Z1Z1"))
571800 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "00")
617600 PS
     M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("111Z1Z", "X1Z0Z0",
"X1ZXZX", "XX1Z1Z", "1X0Z0Z", "111Z1Z", "11Z1Z1", "111Z1Z", "11Z1Z1",
"11Z1Z1", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1", "111Z1Z", "111Z1Z",
"11Z1Z1", "111Z1Z", "111Z1Z", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1",
"111Z1Z", "111Z1Z", "11Z1Z1"))
690200 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "11")
1046900 PS
     M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("111Z1Z", "X1Z0Z0",
"X1ZXZX", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "11Z1Z1", "111Z1Z", "11Z1Z1",
"11Z1Z1", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1", "111Z1Z", "111Z1Z",
"11Z1Z1", "111Z1Z", "111Z1Z", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1",
"111Z1Z", "111Z1Z", "11Z1Z1"))
1089500 PS
     M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("111Z1Z", "X1Z0Z0",
"X1ZXZX", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "111Z1Z", "11Z1Z1",
"11Z1Z1", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1", "111Z1Z", "111Z1Z",
"11Z1Z1", "111Z1Z", "111Z1Z", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1",
"111Z1Z", "111Z1Z", "11Z1Z1"))
1142400 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "00")
1188200 PS
     M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("111Z1Z", "X1Z0Z0",
"X1ZXZX", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "11Z1Z1",
"11Z1Z1", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1", "111Z1Z", "111Z1Z",
"11Z1Z1", "111Z1Z", "111Z1Z", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1",
"111Z1Z", "111Z1Z", "11Z1Z1"))
1241100 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "XX")
1286900 PS
```

```
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("111Z1Z", "X1Z0Z0",
"X1ZXZX", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"11Z1Z1", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1", "111Z1Z", "111Z1Z",
"11Z1Z1", "111Z1Z", "111Z1Z", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1",
"111Z1Z", "111Z1Z", "11Z1Z1"))
1515900 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("111Z1Z", "X1Z0Z0",
"X1ZXZX", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1", "111Z1Z", "111Z1Z",
"11Z1Z1", "111Z1Z", "111Z1Z", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1",
"111Z1Z", "111Z1Z", "11Z1Z1"))
1550100 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "11")
1595900 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("111Z1Z", "X1Z0Z0",
"X1ZXZX", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "111Z1Z", "11Z1Z1", "11Z1Z1", "111Z1Z", "111Z1Z",
"11Z1Z1", "111Z1Z", "111Z1Z", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1",
"111Z1Z", "111Z1Z", "11Z1Z1"))
1657200 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("111Z1Z", "X1Z0Z0",
"X1ZXZX", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "XXXZ1Z", "11Z1Z1", "11Z1Z1", "111Z1Z", "111Z1Z",
"11Z1Z1", "111Z1Z", "111Z1Z", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1",
"111Z1Z", "111Z1Z", "11Z1Z1"))
1737200 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("111Z1Z", "X1Z0Z0",
"X1ZXZX", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "XXXZ1Z", "X1Z0Z0", "11Z1Z1", "111Z1Z", "111Z1Z",
"11Z1Z1", "111Z1Z", "111Z1Z", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1",
"111Z1Z", "111Z1Z", "11Z1Z1"))
1779800 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("111Z1Z", "X1Z0Z0",
"X1ZXZX", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "XXXZ1Z", "X1Z0Z0", "X1ZXZX", "111Z1Z", "111Z1Z",
"11Z1Z1", "111Z1Z", "111Z1Z", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1",
"111Z1Z", "111Z1Z", "11Z1Z1"))
1859800 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("111Z1Z", "X1Z0Z0",
"X1ZXZX", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "XXXZ1Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "111Z1Z",
"11Z1Z1", "111Z1Z", "111Z1Z", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1",
"111Z1Z", "111Z1Z", "11Z1Z1"))
1902400 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("111Z1Z", "X1Z0Z0",
"X1ZXZX", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "XXXZ1Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"11Z1Z1", "111Z1Z", "111Z1Z", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1",
"111Z1Z", "111Z1Z", "11Z1Z1"))
1936600 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "10")
```

```
1982400 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("111Z1Z", "X1Z0Z0",
"X1ZXZX", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "XXXZ1Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "111Z1Z", "111Z1Z", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1",
"111Z1Z", "111Z1Z", "11Z1Z1"))
2017600 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "11")
2483200 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("111Z1Z", "X1Z0Z0",
"X1ZXZX", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "XXXZ1Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "111Z1Z", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1",
"111Z1Z", "111Z1Z", "11Z1Z1"))
2498700 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "10")
2544500 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("111Z1Z", "X1Z0Z0",
"X1ZXZX", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "XXXZ1Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "11Z1Z1", "111Z1Z", "11Z1Z1", "11Z1Z1",
"111Z1Z", "111Z1Z", "11Z1Z1"))
2560000 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "1X")
2605800 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("111Z1Z", "X1Z0Z0",
"X1ZXZX", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "XXXZ1Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "111Z1Z", "11Z1Z1", "11Z1Z1",
"111Z1Z", "111Z1Z", "11Z1Z1"))
2648400 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("111Z1Z", "X1Z0Z0",
"X1ZXZX", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "XXXZ1Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "11Z1Z1", "11Z1Z1",
"111Z1Z", "111Z1Z", "11Z1Z1"))
2688800 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "11")
2731400 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "1X")
2816800 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "11")
2834900 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "X1")
2877400 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("111Z1Z", "X1Z0Z0",
"X1ZXZX", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "XXXZ1Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "11Z1Z1",
"111Z1Z", "111Z1Z", "11Z1Z1"))
2877500 PS
```

```
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "01")
2920000 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("111Z1Z", "X1Z0Z0",
"X1ZXZX", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "XXXZ1Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"111Z1Z",  "111Z1Z",  "11Z1Z1"))
2920100 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "X1")
2962600 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("111Z1Z", "X1Z0Z0",
"X1ZXZX", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "XXXZ1Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z",  "111Z1Z",  "11Z1Z1"))
2962700 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "1X")
3005200 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("111Z1Z", "X1Z0Z0",
"X1ZXZX", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "XXXZ1Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "11Z1Z1"))
3005300 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "10")
3047800 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("111Z1Z", "X1Z0Z0",
"X1ZXZX", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "XXXZ1Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
3047900 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "11")
3112700 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "Z1")
3155300 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "11")
3219100 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "1Z")
3261700 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "11")
3287000 PS
    M3:   ACTIVE /TEST_BENCH_C17/I_GAT_IN (value = "11100")
3361200 PS
    M9:   ACTIVE /TEST_BENCH_C17/COUNT (value = 88)
3362300 PS
    M8:   ACTIVE /TEST_BENCH_C17/COVERAGE (value = 0.230769)
    M7:   ACTIVE /TEST_BENCH_C17/FAULTS_ZDET_T (value = 52.0)
    M6:   ACTIVE /TEST_BENCH_C17/FAULTS_PDET_T (value = 50.0)
    M5:   ACTIVE /TEST_BENCH_C17/FAULTS_UDET_T (value = 30.0)
    M4:   ACTIVE /TEST_BENCH_C17/FAULTS_DET_T (value = 24.0)
```

```
3520700  PS
     M1:    ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1ZXZX", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "XXXZ1Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
3736200  PS
     M2:    ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "10")
3782000  PS
     M1:    ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "XXXZ1Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
3853600  PS
     M2:    ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "1X")
3899400  PS
     M1:    ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "XXXZ1Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
4157500  PS
     M1:    ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "XXXZ1Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
4224000  PS
     M1:    ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "XXXZ1Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
4415600  PS
     M1:    ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "XXXZ1Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
4473700  PS
     M2:    ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "11")
4519500  PS
     M1:    ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
4631400  PS
     M1:    ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
```

```
"X1ZXZX", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
4705900 PS
    M1:  ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
4906500 PS
    M1:  ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
5211200 PS
    M2:  ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "1X")
5238300 PS
    M1:  ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
5280900 PS
    M1:  ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
5451900 PS
    M2:  ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "11")
5470000 PS
    M2:  ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "X1")
5512500 PS
    M1:  ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
5512600 PS
    M2:  ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "11")
5536500 PS
    M2:  ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "X1")
5579000 PS
    M1:  ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
5579100 PS
    M2:  ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "1X")
```

```
5621600 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
5621700 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "11")
5874000 PS
    M3:   ACTIVE /TEST_BENCH_C17/I_GAT_IN (value = "10011")
5931200 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "01")
5948200 PS
    M9:   ACTIVE /TEST_BENCH_C17/COUNT (value = 135)
5949300 PS
    M8:   ACTIVE /TEST_BENCH_C17/COVERAGE (value = 0.298077)
    M7:   ACTIVE /TEST_BENCH_C17/FAULTS_ZDET_T (value = 52.0)
    M6:   ACTIVE /TEST_BENCH_C17/FAULTS_PDET_T (value = 47.0)
    M5:   ACTIVE /TEST_BENCH_C17/FAULTS_UDET_T (value = 26.0)
    M4:   ACTIVE /TEST_BENCH_C17/FAULTS_DET_T (value = 31.0)
6203200 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
6517300 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZ1",
"X1ZXZX", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
6602100 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "11")
6647900 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "0XZ0Z0",
"X1ZXZX", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
6701800 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "01")
6896200 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "11")
6942000 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "0XZ0Z0",
"X1ZXZX", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
7004700 PS
```

```
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "X1")
7050500  PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "0XZ0Z0",
"X1ZXZX", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZX", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
7156400  PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "01")
7202200  PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "0XZ0Z0",
"X1ZXZX", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZ0", "1X0Z0Z", "XXXZ1Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
7314100  PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "0XZ0Z0",
"X1ZXZX", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZ0", "1X0Z0Z", "X00Z0Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
7669000  PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "0XZ0Z0",
"X1ZXZX", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZ0", "1X0Z0Z", "X00Z0Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
7827400  PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "0X")
7854500  PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "0XZ0Z0",
"X1ZXZX", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZ0", "1X0Z0Z", "X00Z0Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1XXZXZ", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
7870000  PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "00")
7915800  PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "0XZ0Z0",
"X1ZXZX", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZ0", "1X0Z0Z", "X00Z0Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "X1ZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
7932300  PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "01")
8108000  PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "0X")
8153600  PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "01")
8171700  PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "X1")
```

```
8214200 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "0XZ0Z0",
"X1ZXZX", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZ0", "1X0Z0Z", "X00Z0Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZX", "X1Z0Z0",
"XXXZ1Z", "XXXZ1Z", "X1Z0Z0"))
8214300 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "01")
8238200 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "11")
8280700 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "0XZ0Z0",
"X1ZXZX", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZ0", "1X0Z0Z", "X00Z0Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZX", "X1Z0Z0",
"X00Z0Z", "XXXZ1Z", "X1Z0Z0"))
8280800 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "0X")
8323300 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "0XZ0Z0",
"X1ZXZX", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZ0", "1X0Z0Z", "X00Z0Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZX", "X1Z0Z0",
"X00Z0Z", "XXXZ1Z", "X1Z0Z0"))
8323400 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "01")
8444000 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "H1")
8486500 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "XX1Z1Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "0XZ0Z0",
"X1ZXZX", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZ0", "1X0Z0Z", "X00Z0Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZX", "X1Z0Z0",
"X00Z0Z", "XXXZ1Z", "X1Z0Z0"))
8486600 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "01")
8575700 PS
    M3:   ACTIVE /TEST_BENCH_C17/I_GAT_IN (value = "00111")
8632900 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "00")
8649900 PS
    M9:   ACTIVE /TEST_BENCH_C17/COUNT (value = 183)
8651000 PS
    M8:   ACTIVE /TEST_BENCH_C17/COVERAGE (value = 0.413462)
    M7:   ACTIVE /TEST_BENCH_C17/FAULTS_ZDET_T (value = 52.0)
    M6:   ACTIVE /TEST_BENCH_C17/FAULTS_PDET_T (value = 39.0)
    M5:   ACTIVE /TEST_BENCH_C17/FAULTS_UDET_T (value = 22.0)
    M4:   ACTIVE /TEST_BENCH_C17/FAULTS_DET_T (value = 43.0)
8933900 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "01")
8979700 PS
```

```
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "X00Z0Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "0XZ0Z0",
"X1ZXZX", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZ0", "1X0Z0Z", "X00Z0Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZX", "X1Z0Z0",
"X00Z0Z", "XXXZ1Z", "X1Z0Z0"))
9051300 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "00")
9265400 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "01")
9311200 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "X00Z0Z", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "0XZ0Z0",
"X1ZXZX", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZ0", "1X0Z0Z", "X00Z0Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZX", "X1Z0Z0",
"X00Z0Z", "XXXZ1Z", "X1Z0Z0"))
9382800 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "00")
9531600 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "X00Z0Z", "1X0Z0Z", "1X0Z0Z", "XXZXZX", "1X0Z0Z", "0XZ0Z0",
"X1ZXZX", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZ0", "1X0Z0Z", "X00Z0Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZX", "X1Z0Z0",
"X00Z0Z", "XXXZ1Z", "X1Z0Z0"))
9837300 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "10")
9883100 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "X00Z0Z", "1X0Z0Z", "1X0Z0Z", "XXZXZX", "1X0Z0Z", "0XZ0Z0",
"X1Z0Z0", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1ZXZ0", "1X0Z0Z", "X00Z0Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZX", "X1Z0Z0",
"X00Z0Z", "XXXZ1Z", "X1Z0Z0"))
9917300 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "00")
10061100 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "01")
10106900 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "X00Z0Z", "1X0Z0Z", "1X0Z0Z", "XXZXZX", "1X0Z0Z", "0XZ0Z0",
"X1Z0Z0", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1Z0Z0", "1X0Z0Z", "X00Z0Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "XXZXZ1", "1X0Z0Z", "XXZXZX", "X1Z0Z0",
"X00Z0Z", "XXXZ1Z", "X1Z0Z0"))
10141100 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "00")
10684000 PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "01")
10729800 PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "X00Z0Z", "1X0Z0Z", "1X0Z0Z", "XXZXZX", "1X0Z0Z", "0XZ0Z0",
"X1Z0Z0", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1Z0Z0", "1X0Z0Z", "X00Z0Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "0XZ0Z0", "1X0Z0Z", "XXZXZX", "X1Z0Z0",
"X00Z0Z", "XXXZ1Z", "X1Z0Z0"))
```

```
10745300  PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "00")
10903300  PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "01")
10949100  PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "X00Z0Z", "1X0Z0Z", "1X0Z0Z", "XXZXZX", "1X0Z0Z", "0XZ0Z0",
"X1Z0Z0", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1Z0Z0", "1X0Z0Z", "X00Z0Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "0XZ0Z0", "1X0Z0Z", "XXZXZX", "X1Z0Z0",
"X00Z0Z", "XXXZ1Z", "X1Z0Z0"))
10964600  PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "00")
11039000  PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "X0")
11081500  PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "X00Z0Z", "1X0Z0Z", "1X0Z0Z", "XXZXZX", "1X0Z0Z", "0XZ0Z0",
"X1Z0Z0", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1Z0Z0", "1X0Z0Z", "X00Z0Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "0XZ0Z0", "1X0Z0Z", "XXZXZX", "X1Z0Z0",
"X00Z0Z", "XXXZ1Z", "X1Z0Z0"))
11081600  PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "00")
11137400  PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "01")
11179900  PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "X00Z0Z", "1X0Z0Z", "1X0Z0Z", "XXZXZX", "1X0Z0Z", "0XZ0Z0",
"X1Z0Z0", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1Z0Z0", "1X0Z0Z", "X00Z0Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "0XZ0Z0", "1X0Z0Z", "XXZXZX", "X1Z0Z0",
"X00Z0Z", "X00Z0Z", "X1Z0Z0"))
11180000  PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "00")
11351200  PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "0H")
11393700  PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "X00Z0Z", "1X0Z0Z", "1X0Z0Z", "XXZXZX", "1X0Z0Z", "0XZ0Z0",
"X1Z0Z0", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1Z0Z0", "1X0Z0Z", "X00Z0Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "0XZ0Z0", "1X0Z0Z", "XXZXZX", "X1Z0Z0",
"X00Z0Z", "X00Z0Z", "X1Z0Z0"))
11393800  PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "00")
11461700  PS
    M3:   ACTIVE /TEST_BENCH_C17/I_GAT_IN (value = "10101")
11518900  PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "11")
11535900  PS
    M9:   ACTIVE /TEST_BENCH_C17/COUNT (value = 240)
11537000  PS
    M8:   ACTIVE /TEST_BENCH_C17/COVERAGE (value = 0.528846)
    M7:   ACTIVE /TEST_BENCH_C17/FAULTS_ZDET_T (value = 52.0)
```

```
    M6:   ACTIVE /TEST_BENCH_C17/FAULTS_PDET_T (value = 32.0)
    M5:   ACTIVE /TEST_BENCH_C17/FAULTS_UDET_T (value = 17.0)
    M4:   ACTIVE /TEST_BENCH_C17/FAULTS_DET_T (value = 55.0)
12075600  PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "X00Z0Z", "1X0Z0Z", "1X0Z0Z", "XXZXZX", "1X0Z0Z", "0XZ0Z0",
"X1Z0Z0", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1Z0Z0", "1X0Z0Z", "X00Z0Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "0XZ0Z0", "1X0Z0Z", "XXZXZX", "X1Z0Z0",
"X00Z0Z", "X00Z0Z", "X1Z0Z0"))
13216900  PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "01")
13259400  PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "X00Z0Z", "1X0Z0Z", "1X0Z0Z", "XXZXZX", "1X0Z0Z", "0XZ0Z0",
"X1Z0Z0", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1Z0Z0", "1X0Z0Z", "X00Z0Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "0XZ0Z0", "1X0Z0Z", "0XZ0Z0", "X1Z0Z0",
"X00Z0Z", "X00Z0Z", "X1Z0Z0"))
13259500  PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "11")
13430700  PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "L1")
13473200  PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "X00Z0Z", "1X0Z0Z", "1X0Z0Z", "XXZXZX", "1X0Z0Z", "0XZ0Z0",
"X1Z0Z0", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1Z0Z0", "1X0Z0Z", "X00Z0Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "0XZ0Z0", "1X0Z0Z", "0XZ0Z0", "X1Z0Z0",
"X00Z0Z", "X00Z0Z", "X1Z0Z0"))
13473300  PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "11")
13636900  PS
    M3:   ACTIVE /TEST_BENCH_C17/I_GAT_IN (value = "11111")
13694100  PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "10")
13711100  PS
    M9:   ACTIVE /TEST_BENCH_C17/COUNT (value = 253)
13712200  PS
    M8:   ACTIVE /TEST_BENCH_C17/COVERAGE (value = 0.557692)
    M7:   ACTIVE /TEST_BENCH_C17/FAULTS_ZDET_T (value = 52.0)
    M6:   ACTIVE /TEST_BENCH_C17/FAULTS_PDET_T (value = 29.0)
    M5:   ACTIVE /TEST_BENCH_C17/FAULTS_UDET_T (value = 17.0)
    M4:   ACTIVE /TEST_BENCH_C17/FAULTS_DET_T (value = 58.0)
14261100  PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "11")
14306900  PS
    M1:   ACTIVE /TEST_BENCH_C17/FAULTS (value = ("1X0Z0Z", "X1Z0Z0",
"X1Z0Z0", "X00Z0Z", "1X0Z0Z", "1X0Z0Z", "XXZ0Z0", "1X0Z0Z", "0XZ0Z0",
"X1Z0Z0", "X1Z0Z0", "X00Z0Z", "X1Z0Z0", "X1Z0Z0", "1X0Z0Z", "X00Z0Z",
"X1Z0Z0", "1X0Z0Z", "1X0Z0Z", "0XZ0Z0", "1X0Z0Z", "0XZ0Z0", "X1Z0Z0",
"X00Z0Z", "X00Z0Z", "X1Z0Z0"))
14359800  PS
    M2:   ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "10")
```

```
15910900  PS
     M3:    ACTIVE /TEST_BENCH_C17/I_GAT_IN (value = "00000")
15968100  PS
    M2:    ACTIVE /TEST_BENCH_C17/I_GAT_OUT (value = "00")
15985100  PS
    M9:    ACTIVE /TEST_BENCH_C17/COUNT (value = 270)
15986200  PS
    M8:    ACTIVE /TEST_BENCH_C17/COVERAGE (value = 0.576923)
    M7:    ACTIVE /TEST_BENCH_C17/FAULTS_ZDET_T (value = 52.0)
    M6:    ACTIVE /TEST_BENCH_C17/FAULTS_PDET_T (value = 27.0)
    M5:    ACTIVE /TEST_BENCH_C17/FAULTS_UDET_T (value = 17.0)
    M4:    ACTIVE /TEST_BENCH_C17/FAULTS_DET_T (value = 60.0)
 c17_opt_gen3.vhd(151):              READ(VLINE,V);
```

D.8 VHDL Fault Simulation Output Results For C432

```
COVERAGE      SIGNAL                  0.0
M2          CE ACTIVE /TEST_BENCH_C432/COVERAGE
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.0)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.0)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.15728)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.15728)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.265797)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.302198)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.330357)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.34272)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.382555)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.407624)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.433379)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.461882)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.469093)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.472871)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.476305)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.48386)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.487294)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.5)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.50103)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.504808)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.507555)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.512706)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.512706)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.516827)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.523008)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.525069)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.526099)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.526099)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.528159)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.52919)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.529876)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.530907)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.531937)
    M2:   ACTIVE /TEST_BENCH_C432/COVERAGE (value = 0.531937)
COUNT       SIGNAL                  0
COUNT_1       SIGNAL                  0
COUNT_2       SIGNAL                  0
COUNT_3       SIGNAL                  0
COUNT_4       SIGNAL                  0
M7          CE ACTIVE /TEST_BENCH_C432/COUNT
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 1)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 6545)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 7118)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 13441)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 15639)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 17904)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 18764)
```

```
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 21583)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 23601)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 24964)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 27063)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 27743)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 28402)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 28800)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 29429)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 29818)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 30460)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 30653)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 30987)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 31229)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 31653)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 31821)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 32188)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 32430)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 32627)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 32779)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 32906)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 33170)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 33323)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 33528)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 33675)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 33825)
    M7:   ACTIVE /TEST_BENCH_C432/COUNT (value = 33946)
FAULTS_DET_T   SIGNAL                  0.0
FAULTS_UDET_T  SIGNAL                  0.0
FAULTS_PDET_T  SIGNAL                  0.0
FAULTS_ZDET_T  SIGNAL                  0.0
M6          CE ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T
M5          CE ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T
M4          CE ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T
M3          CE ACTIVE /TEST_BENCH_C432/FAULTS_DET_T
    M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
    M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 0.0)
    M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 2912.0)
    M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 0.0)
    M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
    M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 0.0)
    M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 2912.0)
    M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 0.0)
    M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
    M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 1467.0)
    M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 987.0)
    M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 458.0)
    M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
    M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 1467.0)
    M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 987.0)
    M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 458.0)
    M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
```

```
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 1428.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 710.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 774.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 1414.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 618.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 880.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 1362.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 588.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 962.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 1338.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 576.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 998.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 1259.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 539.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 1114.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 1208.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 517.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 1187.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 1158.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 492.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 1262.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 1106.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 461.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 1345.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 1091.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 455.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 1366.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 1084.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 451.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 1377.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 1078.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 447.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 1387.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 1059.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 444.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 1409.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 1049.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 444.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 1419.0)
```

```
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 1024.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 432.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 1456.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 1025.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 428.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 1459.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 1015.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 427.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 1470.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 1007.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 427.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 1478.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 992.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 427.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 1493.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 995.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 424.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 1493.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 989.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 418.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 1505.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 978.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 411.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 1523.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 973.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 410.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 1529.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 970.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 410.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 1532.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 970.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 410.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 1532.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 966.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 408.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 1538.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 963.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 408.0)
```

```
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 1541.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 961.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 408.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 1543.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 958.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 408.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 1546.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 958.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 405.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 1549.0)
M6:   ACTIVE /TEST_BENCH_C432/FAULTS_ZDET_T (value = 1456.0)
M5:   ACTIVE /TEST_BENCH_C432/FAULTS_PDET_T (value = 958.0)
M4:   ACTIVE /TEST_BENCH_C432/FAULTS_UDET_T (value = 405.0)
M3:   ACTIVE /TEST_BENCH_C432/FAULTS_DET_T (value = 1549.0)
```

D.9 VHDL Fault Simulation Output Results For C499

```
COVERAGE      SIGNAL                  0.0
M2          CE ACTIVE /TEST_BENCH_C499/COVERAGE
    M2:   ACTIVE /TEST_BENCH_C499/COVERAGE (value = 0.0)
    M2:   ACTIVE /TEST_BENCH_C499/COVERAGE (value = 0.0)
     M2:   ACTIVE /TEST_BENCH_C499/COVERAGE (value = 0.129298)
     M2:   ACTIVE /TEST_BENCH_C499/COVERAGE (value = 0.236569)
     M2:   ACTIVE /TEST_BENCH_C499/COVERAGE (value = 0.252865)
     M2:   ACTIVE /TEST_BENCH_C499/COVERAGE (value = 0.385029)
    M7          CE ACTIVE /TEST_BENCH_C499/COUNT
     M7:   ACTIVE /TEST_BENCH_C499/COUNT (value = 1)
     M7:   ACTIVE /TEST_BENCH_C499/COUNT (value = 9997)
     M7:   ACTIVE /TEST_BENCH_C499/COUNT (value = 34185)
     M7:   ACTIVE /TEST_BENCH_C499/COUNT (value = 36209)
     M7:   ACTIVE /TEST_BENCH_C499/COUNT (value = 73852)
     M6:   ACTIVE /TEST_BENCH_C499/FAULTS_ZDET_T (value = 2792.0)
     M5:   ACTIVE /TEST_BENCH_C499/FAULTS_PDET_T (value = 0.0)
     M4:   ACTIVE /TEST_BENCH_C499/FAULTS_UDET_T (value = 5584.0)
     M3:   ACTIVE /TEST_BENCH_C499/FAULTS_DET_T (value = 0.0)
     M6:   ACTIVE /TEST_BENCH_C499/FAULTS_ZDET_T (value = 2792.0)
     M5:   ACTIVE /TEST_BENCH_C499/FAULTS_PDET_T (value = 0.0)
     M4:   ACTIVE /TEST_BENCH_C499/FAULTS_UDET_T (value = 5584.0)
     M3:   ACTIVE /TEST_BENCH_C499/FAULTS_DET_T (value = 0.0)
     M6:   ACTIVE /TEST_BENCH_C499/FAULTS_ZDET_T (value = 2792.0)
     M5:   ACTIVE /TEST_BENCH_C499/FAULTS_PDET_T (value = 3422.0)
     M4:   ACTIVE /TEST_BENCH_C499/FAULTS_UDET_T (value = 1440.0)
     M3:   ACTIVE /TEST_BENCH_C499/FAULTS_DET_T (value = 722.0)
     M6:   ACTIVE /TEST_BENCH_C499/FAULTS_ZDET_T (value = 2792.0)
     M5:   ACTIVE /TEST_BENCH_C499/FAULTS_PDET_T (value = 3203.0)
     M4:   ACTIVE /TEST_BENCH_C499/FAULTS_UDET_T (value = 1060.0)
     M3:   ACTIVE /TEST_BENCH_C499/FAULTS_DET_T (value = 1321.0)
     M6:   ACTIVE /TEST_BENCH_C499/FAULTS_ZDET_T (value = 2792.0)
     M5:   ACTIVE /TEST_BENCH_C499/FAULTS_PDET_T (value = 3163.0)
     M4:   ACTIVE /TEST_BENCH_C499/FAULTS_UDET_T (value = 1009.0)
     M3:   ACTIVE /TEST_BENCH_C499/FAULTS_DET_T (value = 1412.0)
     M6:   ACTIVE /TEST_BENCH_C499/FAULTS_ZDET_T (value = 2792.0)
     M5:   ACTIVE /TEST_BENCH_C499/FAULTS_PDET_T (value = 2642.0)
     M4:   ACTIVE /TEST_BENCH_C499/FAULTS_UDET_T (value = 792.0)
     M3:   ACTIVE /TEST_BENCH_C499/FAULTS_DET_T (value = 2150.0)
```

D.10 Detailed VHDL Simulation Output Results For C880

```
COVERAGE      SIGNAL                  0.0
M2          CE ACTIVE /TEST_BENCH/COVERAGE
    M2:   ACTIVE /TEST_BENCH/COVERAGE (value = 0.0)
    M2:   ACTIVE /TEST_BENCH/COVERAGE (value = 0.0)
     M2:   ACTIVE /TEST_BENCH/COVERAGE (value = 0.172251)
     M2:   ACTIVE /TEST_BENCH/COVERAGE (value = 0.271692)
     M2:   ACTIVE /TEST_BENCH/COVERAGE (value = 0.332689)
     M2:   ACTIVE /TEST_BENCH/COVERAGE (value = 0.405284)
     M2:   ACTIVE /TEST_BENCH/COVERAGE (value = 0.440077)
     M2:   ACTIVE /TEST_BENCH/COVERAGE (value = 0.462844)
     M2:   ACTIVE /TEST_BENCH/COVERAGE (value = 0.477234)
     M2:   ACTIVE /TEST_BENCH/COVERAGE (value = 0.48604)
     M2:   ACTIVE /TEST_BENCH/COVERAGE (value = 0.508591)
     M2:   ACTIVE /TEST_BENCH/COVERAGE (value = 0.520619)
     M2:   ACTIVE /TEST_BENCH/COVERAGE (value = 0.531143)
     M2:   ACTIVE /TEST_BENCH/COVERAGE (value = 0.535438)
     M2:   ACTIVE /TEST_BENCH/COVERAGE (value = 0.540808)
     M2:   ACTIVE /TEST_BENCH/COVERAGE (value = 0.548754)
     M2:   ACTIVE /TEST_BENCH/COVERAGE (value = 0.554339)
     M2:   ACTIVE /TEST_BENCH/COVERAGE (value = 0.55799)
     M2:   ACTIVE /TEST_BENCH/COVERAGE (value = 0.562285)
     M2:   ACTIVE /TEST_BENCH/COVERAGE (value = 0.565292)
     M2:   ACTIVE /TEST_BENCH/COVERAGE (value = 0.566366)
     M2:   ACTIVE /TEST_BENCH/COVERAGE (value = 0.567655)
     M2:   ACTIVE /TEST_BENCH/COVERAGE (value = 0.568084)
     M2:   ACTIVE /TEST_BENCH/COVERAGE (value = 0.569373)
COUNT_1       SIGNAL                  0
COUNT_2       SIGNAL                  0
COUNT_3       SIGNAL                  0
COUNT_4       SIGNAL                  0
M7          CE ACTIVE /TEST_BENCH/COUNT
    M7:   ACTIVE /TEST_BENCH/COUNT (value = 1)
     M7:   ACTIVE /TEST_BENCH/COUNT (value = 7025)
     M7:   ACTIVE /TEST_BENCH/COUNT (value = 12336)
     M7:   ACTIVE /TEST_BENCH/COUNT (value = 15237)
     M7:   ACTIVE /TEST_BENCH/COUNT (value = 18843)
     M7:   ACTIVE /TEST_BENCH/COUNT (value = 20788)
     M7:   ACTIVE /TEST_BENCH/COUNT (value = 22244)
     M7:   ACTIVE /TEST_BENCH/COUNT (value = 23370)
     M7:   ACTIVE /TEST_BENCH/COUNT (value = 24129)
     M7:   ACTIVE /TEST_BENCH/COUNT (value = 25241)
     M7:   ACTIVE /TEST_BENCH/COUNT (value = 25912)
     M7:   ACTIVE /TEST_BENCH/COUNT (value = 26344)
     M7:   ACTIVE /TEST_BENCH/COUNT (value = 26733)
     M7:   ACTIVE /TEST_BENCH/COUNT (value = 27197)
     M7:   ACTIVE /TEST_BENCH/COUNT (value = 27445)
     M7:   ACTIVE /TEST_BENCH/COUNT (value = 27695)
```

```
    M7:   ACTIVE /TEST_BENCH/COUNT (value = 27905)
    M7:   ACTIVE /TEST_BENCH/COUNT (value = 28126)
    M7:   ACTIVE /TEST_BENCH/COUNT (value = 28303)
    M7:   ACTIVE /TEST_BENCH/COUNT (value = 28399)
    M7:   ACTIVE /TEST_BENCH/COUNT (value = 28495)
    M7:   ACTIVE /TEST_BENCH/COUNT (value = 28575)
    M7:   ACTIVE /TEST_BENCH/COUNT (value = 28688)

FAULTS_DET_T   SIGNAL                  0.0
FAULTS_UDET_T  SIGNAL                  0.0
FAULTS_PDET_T  SIGNAL                  0.0
FAULTS_ZDET_T  SIGNAL                  0.0
M6          CE ACTIVE /TEST_BENCH/FAULTS_ZDET_T
M5          CE ACTIVE /TEST_BENCH/FAULTS_PDET_T
M4          CE ACTIVE /TEST_BENCH/FAULTS_UDET_T
M3          CE ACTIVE /TEST_BENCH/FAULTS_DET_T
    M6:   ACTIVE /TEST_BENCH/FAULTS_ZDET_T (value = 2328.0)
    M5:   ACTIVE /TEST_BENCH/FAULTS_PDET_T (value = 0.0)
    M4:   ACTIVE /TEST_BENCH/FAULTS_UDET_T (value = 4656.0)
    M3:   ACTIVE /TEST_BENCH/FAULTS_DET_T (value = 0.0)
    M6:   ACTIVE /TEST_BENCH/FAULTS_ZDET_T (value = 2328.0)
    M5:   ACTIVE /TEST_BENCH/FAULTS_PDET_T (value = 0.0)
    M4:   ACTIVE /TEST_BENCH/FAULTS_UDET_T (value = 4656.0)
    M3:   ACTIVE /TEST_BENCH/FAULTS_DET_T (value = 0.0)
    M6:   ACTIVE /TEST_BENCH/FAULTS_ZDET_T (value = 2328.0)
    M5:   ACTIVE /TEST_BENCH/FAULTS_PDET_T (value = 2334.0)
    M4:   ACTIVE /TEST_BENCH/FAULTS_UDET_T (value = 1520.0)
    M3:   ACTIVE /TEST_BENCH/FAULTS_DET_T (value = 802.0)
    M6:   ACTIVE /TEST_BENCH/FAULTS_ZDET_T (value = 2328.0)
    M5:   ACTIVE /TEST_BENCH/FAULTS_PDET_T (value = 2241.0)
    M4:   ACTIVE /TEST_BENCH/FAULTS_UDET_T (value = 1150.0)
    M3:   ACTIVE /TEST_BENCH/FAULTS_DET_T (value = 1265.0)
    M6:   ACTIVE /TEST_BENCH/FAULTS_ZDET_T (value = 2328.0)
    M5:   ACTIVE /TEST_BENCH/FAULTS_PDET_T (value = 2159.0)
    M4:   ACTIVE /TEST_BENCH/FAULTS_UDET_T (value = 948.0)
    M3:   ACTIVE /TEST_BENCH/FAULTS_DET_T (value = 1549.0)
    M6:   ACTIVE /TEST_BENCH/FAULTS_ZDET_T (value = 2328.0)
    M5:   ACTIVE /TEST_BENCH/FAULTS_PDET_T (value = 1953.0)
    M4:   ACTIVE /TEST_BENCH/FAULTS_UDET_T (value = 816.0)
    M3:   ACTIVE /TEST_BENCH/FAULTS_DET_T (value = 1887.0)
    M6:   ACTIVE /TEST_BENCH/FAULTS_ZDET_T (value = 2328.0)
    M5:   ACTIVE /TEST_BENCH/FAULTS_PDET_T (value = 1852.0)
    M4:   ACTIVE /TEST_BENCH/FAULTS_UDET_T (value = 755.0)
    M3:   ACTIVE /TEST_BENCH/FAULTS_DET_T (value = 2049.0)
    M6:   ACTIVE /TEST_BENCH/FAULTS_ZDET_T (value = 2328.0)
    M5:   ACTIVE /TEST_BENCH/FAULTS_PDET_T (value = 1771.0)
    M4:   ACTIVE /TEST_BENCH/FAULTS_UDET_T (value = 730.0)
    M3:   ACTIVE /TEST_BENCH/FAULTS_DET_T (value = 2155.0)
    M6:   ACTIVE /TEST_BENCH/FAULTS_ZDET_T (value = 2328.0)
    M5:   ACTIVE /TEST_BENCH/FAULTS_PDET_T (value = 1717.0)
    M4:   ACTIVE /TEST_BENCH/FAULTS_UDET_T (value = 717.0)
```

```
M3:   ACTIVE /TEST_BENCH/FAULTS_DET_T (value = 2222.0)
M6:   ACTIVE /TEST_BENCH/FAULTS_ZDET_T (value = 2328.0)
M5:   ACTIVE /TEST_BENCH/FAULTS_PDET_T (value = 1693.0)
M4:   ACTIVE /TEST_BENCH/FAULTS_UDET_T (value = 700.0)
M3:   ACTIVE /TEST_BENCH/FAULTS_DET_T (value = 2263.0)
M6:   ACTIVE /TEST_BENCH/FAULTS_ZDET_T (value = 2328.0)
M5:   ACTIVE /TEST_BENCH/FAULTS_PDET_T (value = 1600.0)
M4:   ACTIVE /TEST_BENCH/FAULTS_UDET_T (value = 688.0)
M3:   ACTIVE /TEST_BENCH/FAULTS_DET_T (value = 2368.0)
M6:   ACTIVE /TEST_BENCH/FAULTS_ZDET_T (value = 2328.0)
M5:   ACTIVE /TEST_BENCH/FAULTS_PDET_T (value = 1560.0)
M4:   ACTIVE /TEST_BENCH/FAULTS_UDET_T (value = 672.0)
M3:   ACTIVE /TEST_BENCH/FAULTS_DET_T (value = 2424.0)
M6:   ACTIVE /TEST_BENCH/FAULTS_ZDET_T (value = 2328.0)
M5:   ACTIVE /TEST_BENCH/FAULTS_PDET_T (value = 1522.0)
M4:   ACTIVE /TEST_BENCH/FAULTS_UDET_T (value = 661.0)
M3:   ACTIVE /TEST_BENCH/FAULTS_DET_T (value = 2473.0)
M6:   ACTIVE /TEST_BENCH/FAULTS_ZDET_T (value = 2328.0)
M5:   ACTIVE /TEST_BENCH/FAULTS_PDET_T (value = 1504.0)
M4:   ACTIVE /TEST_BENCH/FAULTS_UDET_T (value = 659.0)
M3:   ACTIVE /TEST_BENCH/FAULTS_DET_T (value = 2493.0)
M6:   ACTIVE /TEST_BENCH/FAULTS_ZDET_T (value = 2328.0)
M5:   ACTIVE /TEST_BENCH/FAULTS_PDET_T (value = 1487.0)
M4:   ACTIVE /TEST_BENCH/FAULTS_UDET_T (value = 651.0)
M3:   ACTIVE /TEST_BENCH/FAULTS_DET_T (value = 2518.0)
M6:   ACTIVE /TEST_BENCH/FAULTS_ZDET_T (value = 2328.0)
M5:   ACTIVE /TEST_BENCH/FAULTS_PDET_T (value = 1460.0)
M4:   ACTIVE /TEST_BENCH/FAULTS_UDET_T (value = 641.0)
M3:   ACTIVE /TEST_BENCH/FAULTS_DET_T (value = 2555.0)
M6:   ACTIVE /TEST_BENCH/FAULTS_ZDET_T (value = 2328.0)
M5:   ACTIVE /TEST_BENCH/FAULTS_PDET_T (value = 1436.0)
M4:   ACTIVE /TEST_BENCH/FAULTS_UDET_T (value = 639.0)
M3:   ACTIVE /TEST_BENCH/FAULTS_DET_T (value = 2581.0)
M6:   ACTIVE /TEST_BENCH/FAULTS_ZDET_T (value = 2328.0)
M5:   ACTIVE /TEST_BENCH/FAULTS_PDET_T (value = 1420.0)
M4:   ACTIVE /TEST_BENCH/FAULTS_UDET_T (value = 638.0)
M3:   ACTIVE /TEST_BENCH/FAULTS_DET_T (value = 2598.0)
M6:   ACTIVE /TEST_BENCH/FAULTS_ZDET_T (value = 2328.0)
M5:   ACTIVE /TEST_BENCH/FAULTS_PDET_T (value = 1400.0)
M4:   ACTIVE /TEST_BENCH/FAULTS_UDET_T (value = 638.0)
M3:   ACTIVE /TEST_BENCH/FAULTS_DET_T (value = 2618.0)
M6:   ACTIVE /TEST_BENCH/FAULTS_ZDET_T (value = 2328.0)
M5:   ACTIVE /TEST_BENCH/FAULTS_PDET_T (value = 1390.0)
M4:   ACTIVE /TEST_BENCH/FAULTS_UDET_T (value = 634.0)
M3:   ACTIVE /TEST_BENCH/FAULTS_DET_T (value = 2632.0)
M6:   ACTIVE /TEST_BENCH/FAULTS_ZDET_T (value = 2328.0)
M5:   ACTIVE /TEST_BENCH/FAULTS_PDET_T (value = 1385.0)
M4:   ACTIVE /TEST_BENCH/FAULTS_UDET_T (value = 634.0)
M3:   ACTIVE /TEST_BENCH/FAULTS_DET_T (value = 2637.0)
M6:   ACTIVE /TEST_BENCH/FAULTS_ZDET_T (value = 2328.0)
M5:   ACTIVE /TEST_BENCH/FAULTS_PDET_T (value = 1379.0)
```

```
    M4:   ACTIVE /TEST_BENCH/FAULTS_UDET_T (value = 634.0)
    M3:   ACTIVE /TEST_BENCH/FAULTS_DET_T (value = 2643.0)
    M6:   ACTIVE /TEST_BENCH/FAULTS_ZDET_T (value = 2328.0)
    M5:   ACTIVE /TEST_BENCH/FAULTS_PDET_T (value = 1377.0)
    M4:   ACTIVE /TEST_BENCH/FAULTS_UDET_T (value = 634.0)
    M3:   ACTIVE /TEST_BENCH/FAULTS_DET_T (value = 2645.0)
    M6:   ACTIVE /TEST_BENCH/FAULTS_ZDET_T (value = 2328.0)
    M5:   ACTIVE /TEST_BENCH/FAULTS_PDET_T (value = 1371.0)
    M4:   ACTIVE /TEST_BENCH/FAULTS_UDET_T (value = 634.0)
    M3:   ACTIVE /TEST_BENCH/FAULTS_DET_T (value = 2651.0)
I_GAT_IN     SIGNAL                  COUNT       SIGNAL                  0
```

D.11 VHDL Simulation Output Results For C1355

```
COVERAGE      SIGNAL                  0.0
M2            CE ACTIVE /TEST_BENCH_C1355/COVERAGE
    M2:   ACTIVE /TEST_BENCH_C1355/COVERAGE (value = 0.0)
    M2:   ACTIVE /TEST_BENCH_C1355/COVERAGE (value = 0.0)
     M2:   ACTIVE /TEST_BENCH_C1355/COVERAGE (value = 0.133484)
     M2:   ACTIVE /TEST_BENCH_C1355/COVERAGE (value = 0.222144)
     M2:   ACTIVE /TEST_BENCH_C1355/COVERAGE (value = 0.26711)
     M2:   ACTIVE /TEST_BENCH_C1355/COVERAGE (value = 0.303874)
     M2:   ACTIVE /TEST_BENCH_C1355/COVERAGE (value = 0.322257)
     M2:   ACTIVE /TEST_BENCH_C1355/COVERAGE (value = 0.331165)
     M2:   ACTIVE /TEST_BENCH_C1355/COVERAGE (value = 0.335124)
     M2:   ACTIVE /TEST_BENCH_C1355/COVERAGE (value = 0.341912)
     M2:   ACTIVE /TEST_BENCH_C1355/COVERAGE (value = 0.363971)
COUNT         SIGNAL                  0
COUNT_1       SIGNAL                   0
COUNT_2       SIGNAL                   0
COUNT_3       SIGNAL                   0
COUNT_4       SIGNAL                   0
M7            CE ACTIVE /TEST_BENCH_C1355/COUNT
    M7:   ACTIVE /TEST_BENCH_C1355/COUNT (value = 1)
     M7:   ACTIVE /TEST_BENCH_C1355/COUNT (value = 16717)
     M7:   ACTIVE /TEST_BENCH_C1355/COUNT (value = 30332)
     M7:   ACTIVE /TEST_BENCH_C1355/COUNT (value = 37933)
     M7:   ACTIVE /TEST_BENCH_C1355/COUNT (value = 43062)
     M7:   ACTIVE /TEST_BENCH_C1355/COUNT (value = 47124)
     M7:   ACTIVE /TEST_BENCH_C1355/COUNT (value = 49130)
     M7:   ACTIVE /TEST_BENCH_C1355/COUNT (value = 50423)
     M7:   ACTIVE /TEST_BENCH_C1355/COUNT (value = 52587)
     M7:   ACTIVE /TEST_BENCH_C1355/COUNT (value = 56227)

FAULTS_DET_T   SIGNAL                  0.0
FAULTS_UDET_T  SIGNAL                   0.0
FAULTS_PDET_T  SIGNAL                   0.0
FAULTS_ZDET_T  SIGNAL                   0.0
M6            CE ACTIVE /TEST_BENCH_C1355/FAULTS_ZDET_T
M5            CE ACTIVE /TEST_BENCH_C1355/FAULTS_PDET_T
M4            CE ACTIVE /TEST_BENCH_C1355/FAULTS_UDET_T
M3            CE ACTIVE /TEST_BENCH_C1355/FAULTS_DET_T
    M6:   ACTIVE /TEST_BENCH_C1355/FAULTS_ZDET_T (value = 3536.0)
    M5:   ACTIVE /TEST_BENCH_C1355/FAULTS_PDET_T (value = 0.0)
    M4:   ACTIVE /TEST_BENCH_C1355/FAULTS_UDET_T (value = 7072.0)
    M3:   ACTIVE /TEST_BENCH_C1355/FAULTS_DET_T (value = 0.0)
    M6:   ACTIVE /TEST_BENCH_C1355/FAULTS_ZDET_T (value = 3536.0)
    M5:   ACTIVE /TEST_BENCH_C1355/FAULTS_PDET_T (value = 0.0)
    M4:   ACTIVE /TEST_BENCH_C1355/FAULTS_UDET_T (value = 7072.0)
    M3:   ACTIVE /TEST_BENCH_C1355/FAULTS_DET_T (value = 0.0)
    M6:   ACTIVE /TEST_BENCH_C1355/FAULTS_ZDET_T (value = 3536.0)
    M5:   ACTIVE /TEST_BENCH_C1355/FAULTS_PDET_T (value = 3579.0)
```

```
M4:   ACTIVE /TEST_BENCH_C1355/FAULTS_UDET_T (value = 2549.0)
M3:   ACTIVE /TEST_BENCH_C1355/FAULTS_DET_T (value = 944.0)
M6:   ACTIVE /TEST_BENCH_C1355/FAULTS_ZDET_T (value = 3536.0)
M5:   ACTIVE /TEST_BENCH_C1355/FAULTS_PDET_T (value = 3588.0)
M4:   ACTIVE /TEST_BENCH_C1355/FAULTS_UDET_T (value = 1913.0)
M3:   ACTIVE /TEST_BENCH_C1355/FAULTS_DET_T (value = 1571.0)
M6:   ACTIVE /TEST_BENCH_C1355/FAULTS_ZDET_T (value = 3536.0)
M5:   ACTIVE /TEST_BENCH_C1355/FAULTS_PDET_T (value = 3578.0)
M4:   ACTIVE /TEST_BENCH_C1355/FAULTS_UDET_T (value = 1605.0)
M3:   ACTIVE /TEST_BENCH_C1355/FAULTS_DET_T (value = 1889.0)
M6:   ACTIVE /TEST_BENCH_C1355/FAULTS_ZDET_T (value = 3536.0)
M5:   ACTIVE /TEST_BENCH_C1355/FAULTS_PDET_T (value = 3464.0)
M4:   ACTIVE /TEST_BENCH_C1355/FAULTS_UDET_T (value = 1459.0)
M3:   ACTIVE /TEST_BENCH_C1355/FAULTS_DET_T (value = 2149.0)
M6:   ACTIVE /TEST_BENCH_C1355/FAULTS_ZDET_T (value = 3536.0)
M5:   ACTIVE /TEST_BENCH_C1355/FAULTS_PDET_T (value = 3374.0)
M4:   ACTIVE /TEST_BENCH_C1355/FAULTS_UDET_T (value = 1419.0)
M3:   ACTIVE /TEST_BENCH_C1355/FAULTS_DET_T (value = 2279.0)
M6:   ACTIVE /TEST_BENCH_C1355/FAULTS_ZDET_T (value = 3536.0)
M5:   ACTIVE /TEST_BENCH_C1355/FAULTS_PDET_T (value = 3333.0)
M4:   ACTIVE /TEST_BENCH_C1355/FAULTS_UDET_T (value = 1397.0)
M3:   ACTIVE /TEST_BENCH_C1355/FAULTS_DET_T (value = 2342.0)
M6:   ACTIVE /TEST_BENCH_C1355/FAULTS_ZDET_T (value = 3536.0)
M5:   ACTIVE /TEST_BENCH_C1355/FAULTS_PDET_T (value = 3316.0)
M4:   ACTIVE /TEST_BENCH_C1355/FAULTS_UDET_T (value = 1386.0)
M3:   ACTIVE /TEST_BENCH_C1355/FAULTS_DET_T (value = 2370.0)
M6:   ACTIVE /TEST_BENCH_C1355/FAULTS_ZDET_T (value = 3536.0)
M5:   ACTIVE /TEST_BENCH_C1355/FAULTS_PDET_T (value = 3287.0)
M4:   ACTIVE /TEST_BENCH_C1355/FAULTS_UDET_T (value = 1367.0)
M3:   ACTIVE /TEST_BENCH_C1355/FAULTS_DET_T (value = 2418.0)
M6:   ACTIVE /TEST_BENCH_C1355/FAULTS_ZDET_T (value = 3536.0)
M5:   ACTIVE /TEST_BENCH_C1355/FAULTS_PDET_T (value = 3201.0)
M4:   ACTIVE /TEST_BENCH_C1355/FAULTS_UDET_T (value = 1297.0)
M3:   ACTIVE /TEST_BENCH_C1355/FAULTS_DET_T (value = 2574.0)
```

D.12 Detailed VHDL Simulation Output Results For C1908

```
COVERAGE      SIGNAL                  0.0
M2            CE ACTIVE /TEST_BENCH_C1908/COVERAGE
    M2:    ACTIVE /TEST_BENCH_C1908/COVERAGE (value = 0.0)
    M2:    ACTIVE /TEST_BENCH_C1908/COVERAGE (value = 0.0)
     M2:    ACTIVE /TEST_BENCH_C1908/COVERAGE (value = 0.144558)
     M2:    ACTIVE /TEST_BENCH_C1908/COVERAGE (value = 0.229956)
     M2:    ACTIVE /TEST_BENCH_C1908/COVERAGE (value = 0.283892)
     M2:    ACTIVE /TEST_BENCH_C1908/COVERAGE (value = 0.338678)
     M2:    ACTIVE /TEST_BENCH_C1908/COVERAGE (value = 0.355685)
     M2:    ACTIVE /TEST_BENCH_C1908/COVERAGE (value = 0.378401)
     M2:    ACTIVE /TEST_BENCH_C1908/COVERAGE (value = 0.397595)
COUNT         SIGNAL                  0
COUNT_1       SIGNAL                  0
COUNT_2       SIGNAL                  0
COUNT_3       SIGNAL                  0
COUNT_4       SIGNAL                  0
M7            CE ACTIVE /TEST_BENCH_C1908/COUNT
    M7:    ACTIVE /TEST_BENCH_C1908/COUNT (value = 1)
     M7:    ACTIVE /TEST_BENCH_C1908/COUNT (value = 20117)
     M7:    ACTIVE /TEST_BENCH_C1908/COUNT (value = 35859)
     M7:    ACTIVE /TEST_BENCH_C1908/COUNT (value = 43765)
     M7:    ACTIVE /TEST_BENCH_C1908/COUNT (value = 53380)
     M7:    ACTIVE /TEST_BENCH_C1908/COUNT (value = 57912)
     M7:    ACTIVE /TEST_BENCH_C1908/COUNT (value = 62775)
     M7:    ACTIVE /TEST_BENCH_C1908/COUNT (value = 68475)
FAULTS_DET_T   SIGNAL                 0.0
FAULTS_UDET_T  SIGNAL                 0.0
FAULTS_PDET_T  SIGNAL                 0.0
FAULTS_ZDET_T  SIGNAL                 0.0
M6            CE ACTIVE /TEST_BENCH_C1908/FAULTS_ZDET_T
M5            CE ACTIVE /TEST_BENCH_C1908/FAULTS_PDET_T
M4            CE ACTIVE /TEST_BENCH_C1908/FAULTS_UDET_T
M3            CE ACTIVE /TEST_BENCH_C1908/FAULTS_DET_T
     M6:    ACTIVE /TEST_BENCH_C1908/FAULTS_ZDET_T (value = 4116.0)
     M5:    ACTIVE /TEST_BENCH_C1908/FAULTS_PDET_T (value = 0.0)
     M4:    ACTIVE /TEST_BENCH_C1908/FAULTS_UDET_T (value = 8232.0)
     M3:    ACTIVE /TEST_BENCH_C1908/FAULTS_DET_T (value = 0.0)
     M6:    ACTIVE /TEST_BENCH_C1908/FAULTS_ZDET_T (value = 4116.0)
     M5:    ACTIVE /TEST_BENCH_C1908/FAULTS_PDET_T (value = 0.0)
     M4:    ACTIVE /TEST_BENCH_C1908/FAULTS_UDET_T (value = 8232.0)
     M3:    ACTIVE /TEST_BENCH_C1908/FAULTS_DET_T (value = 0.0)
     M6:    ACTIVE /TEST_BENCH_C1908/FAULTS_ZDET_T (value = 4116.0)
     M5:    ACTIVE /TEST_BENCH_C1908/FAULTS_PDET_T (value = 4279.0)
     M4:    ACTIVE /TEST_BENCH_C1908/FAULTS_UDET_T (value = 2763.0)
     M3:    ACTIVE /TEST_BENCH_C1908/FAULTS_DET_T (value = 1190.0)
     M6:    ACTIVE /TEST_BENCH_C1908/FAULTS_ZDET_T (value = 4116.0)
     M5:    ACTIVE /TEST_BENCH_C1908/FAULTS_PDET_T (value = 4133.0)
     M4:    ACTIVE /TEST_BENCH_C1908/FAULTS_UDET_T (value = 2206.0)
```

```
M3:   ACTIVE /TEST_BENCH_C1908/FAULTS_DET_T (value = 1893.0)
M6:   ACTIVE /TEST_BENCH_C1908/FAULTS_ZDET_T (value = 4116.0)
M5:   ACTIVE /TEST_BENCH_C1908/FAULTS_PDET_T (value = 4095.0)
M4:   ACTIVE /TEST_BENCH_C1908/FAULTS_UDET_T (value = 1800.0)
M3:   ACTIVE /TEST_BENCH_C1908/FAULTS_DET_T (value = 2337.0)
M6:   ACTIVE /TEST_BENCH_C1908/FAULTS_ZDET_T (value = 4116.0)
M5:   ACTIVE /TEST_BENCH_C1908/FAULTS_PDET_T (value = 3828.0)
M4:   ACTIVE /TEST_BENCH_C1908/FAULTS_UDET_T (value = 1616.0)
M3:   ACTIVE /TEST_BENCH_C1908/FAULTS_DET_T (value = 2788.0)
M6:   ACTIVE /TEST_BENCH_C1908/FAULTS_ZDET_T (value = 4116.0)
M5:   ACTIVE /TEST_BENCH_C1908/FAULTS_PDET_T (value = 3729.0)
M4:   ACTIVE /TEST_BENCH_C1908/FAULTS_UDET_T (value = 1575.0)
M3:   ACTIVE /TEST_BENCH_C1908/FAULTS_DET_T (value = 2928.0)
M6:   ACTIVE /TEST_BENCH_C1908/FAULTS_ZDET_T (value = 4116.0)
M5:   ACTIVE /TEST_BENCH_C1908/FAULTS_PDET_T (value = 3624.0)
M4:   ACTIVE /TEST_BENCH_C1908/FAULTS_UDET_T (value = 1493.0)
M3:   ACTIVE /TEST_BENCH_C1908/FAULTS_DET_T (value = 3115.0)
M6:   ACTIVE /TEST_BENCH_C1908/FAULTS_ZDET_T (value = 4116.0)
M5:   ACTIVE /TEST_BENCH_C1908/FAULTS_PDET_T (value = 3526.0)
M4:   ACTIVE /TEST_BENCH_C1908/FAULTS_UDET_T (value = 1433.0)
M3:   ACTIVE /TEST_BENCH_C1908/FAULTS_DET_T (value = 3273.0)
```

D.13 Detailed VHDL Simulation Output Results For C2670

```
COVERAGE      SIGNAL                  0.0
M2          CE ACTIVE /TEST_BENCH_C2670/COVERAGE
    M2:   ACTIVE /TEST_BENCH_C2670/COVERAGE (value = 0.0)
    M2:   ACTIVE /TEST_BENCH_C2670/COVERAGE (value = 0.0)
     M2:   ACTIVE /TEST_BENCH_C2670/COVERAGE (value = 0.160054)
     M2:   ACTIVE /TEST_BENCH_C2670/COVERAGE (value = 0.272697)
     M2:   ACTIVE /TEST_BENCH_C2670/COVERAGE (value = 0.334987)
     M2:   ACTIVE /TEST_BENCH_C2670/COVERAGE (value = 0.362895)
COUNT         SIGNAL                  0
COUNT_1       SIGNAL                   0
COUNT_2       SIGNAL                   0
COUNT_3       SIGNAL                   0
COUNT_4       SIGNAL                   0
M7          CE ACTIVE /TEST_BENCH_C2670/COUNT
    M7:   ACTIVE /TEST_BENCH_C2670/COUNT (value = 1)
     M7:   ACTIVE /TEST_BENCH_C2670/COUNT (value = 26395)
     M7:   ACTIVE /TEST_BENCH_C2670/COUNT (value = 49352)
     M7:   ACTIVE /TEST_BENCH_C2670/COUNT (value = 63219)
     M7:   ACTIVE /TEST_BENCH_C2670/COUNT (value = 70262)
FAULTS_DET_T   SIGNAL                  0.0
FAULTS_UDET_T  SIGNAL                   0.0
FAULTS_PDET_T  SIGNAL                   0.0
FAULTS_ZDET_T  SIGNAL                   0.0
M6          CE ACTIVE /TEST_BENCH_C2670/FAULTS_ZDET_T
M5          CE ACTIVE /TEST_BENCH_C2670/FAULTS_PDET_T
M4          CE ACTIVE /TEST_BENCH_C2670/FAULTS_UDET_T
M3          CE ACTIVE /TEST_BENCH_C2670/FAULTS_DET_T
     M6:   ACTIVE /TEST_BENCH_C2670/FAULTS_ZDET_T (value = 5948.0)
    M5:   ACTIVE /TEST_BENCH_C2670/FAULTS_PDET_T (value = 0.0)
     M4:   ACTIVE /TEST_BENCH_C2670/FAULTS_UDET_T (value = 11896.0)
    M3:   ACTIVE /TEST_BENCH_C2670/FAULTS_DET_T (value = 0.0)
     M6:   ACTIVE /TEST_BENCH_C2670/FAULTS_ZDET_T (value = 5948.0)
    M5:   ACTIVE /TEST_BENCH_C2670/FAULTS_PDET_T (value = 0.0)
     M4:   ACTIVE /TEST_BENCH_C2670/FAULTS_UDET_T (value = 11896.0)
    M3:   ACTIVE /TEST_BENCH_C2670/FAULTS_DET_T (value = 0.0)
     M6:   ACTIVE /TEST_BENCH_C2670/FAULTS_ZDET_T (value = 5948.0)
     M5:   ACTIVE /TEST_BENCH_C2670/FAULTS_PDET_T (value = 6202.0)
     M4:   ACTIVE /TEST_BENCH_C2670/FAULTS_UDET_T (value = 3790.0)
     M3:   ACTIVE /TEST_BENCH_C2670/FAULTS_DET_T (value = 1904.0)
     M6:   ACTIVE /TEST_BENCH_C2670/FAULTS_ZDET_T (value = 5948.0)
     M5:   ACTIVE /TEST_BENCH_C2670/FAULTS_PDET_T (value = 6312.0)
     M4:   ACTIVE /TEST_BENCH_C2670/FAULTS_UDET_T (value = 2340.0)
     M3:   ACTIVE /TEST_BENCH_C2670/FAULTS_DET_T (value = 3244.0)
     M6:   ACTIVE /TEST_BENCH_C2670/FAULTS_ZDET_T (value = 5948.0)
     M5:   ACTIVE /TEST_BENCH_C2670/FAULTS_PDET_T (value = 5768.0)
     M4:   ACTIVE /TEST_BENCH_C2670/FAULTS_UDET_T (value = 2143.0)
     M3:   ACTIVE /TEST_BENCH_C2670/FAULTS_DET_T (value = 3985.0)
     M6:   ACTIVE /TEST_BENCH_C2670/FAULTS_ZDET_T (value = 5948.0)
```

```
M5:   ACTIVE /TEST_BENCH_C2670/FAULTS_PDET_T (value = 5513.0)
M4:   ACTIVE /TEST_BENCH_C2670/FAULTS_UDET_T (value = 2066.0)
M3:   ACTIVE /TEST_BENCH_C2670/FAULTS_DET_T (value = 4317.0)
```

Vita

Christopher A. Ryan

Christopher A. Ryan received his Bachelor of Science Degree in Electrical Engineering, Magna Cum Laude, from North Carolina Agricultural & Technical State University in Greensboro, North Carolina on June 25, 1985. After graduation, Mr. Ryan was employed with General Electric as an Electrical Engineering. He received a fellowship from the Microelectronics Center of North Carolina in order to pursue graduate study and completed his Master of Science Degree in Electrical Engineering from North Carolina Agricultural & Technical State University on May 5, 1989. In August, 1989 he received a fellowship from the President of Virginia Polytechnic Institute & State University in order to pursue graduate study. After completion of the Ph.D. degree, Mr. Ryan hopes to pursue a career in teaching.

Mr. Ryan is a member of the National Society of Professional Engineers, the Professional Engineers of North Carolina, the Institute of Electrical and Electronic Engineers, and the Association for Computing Machinery.