FUNCTIONAL FAULT MODELING AND SIMULATION FOR VLSI DEVICES

Anil K. Gupta and James R. Armstrong

Department of Electrical Engineering
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061

ABSTRACT

Functional fault modeling and simulation for VLSI devices is described(*). A functional fault list is compiled using model perturbation and mapping of circuit defects into functional faults. A set of test vectors is then derived which detects all faults in the functional fault list. This same test vector set is then applied to a gate level model of the device. For the test case analyzed, a very high level of equivalent gate coverage was achieved. Conclusions are drawn as to the effectiveness of the technique and how amenable it is to automation.

INTRODUCTION

The progress in fabrication technology has brought us into an era of VLSI circuits. Circuit densities have increased manifold. Speeds have also increased tremendously because of faster devices and close proximity on the layout. The increased density results in increased number of faults per unit area, creating problems in testing.

Logic simulation has been used traditionally for obtaining test patterns and fault signatures for existing systems and more recently, for design verification and system validation. An important aspect of simulator design is the efficiency, which is defined as the ratio of the host CPU time to the real logic time. Efficiency depends on the level of simulation[1,4] and generally decreases with the increase in the complexity of representation.

---

* This work was supported by a grant from IBM, Manassas. Contract no. YD 190121.

Gate level simulation techniques used to obtain test vectors for the less populated LSI chips have become very cumbersome and slow for VLSI circuits. Such simulations have been seen to run for days on dedicated mainframes[1].

Functional level digital logic simulation is a more viable and useful approach in the context of these new developments. Functional level simulations are performed on those 'descriptions' of the digital logic which are a level or two higher than the gate level 'descriptions'. This makes them much faster and less complicated than the gate level simulators.

For gate level simulators, increasing size and complexity plague the fault simulation with respect to computer run time (RT) which depends on the gate count (n) as :

$$RT = K\, n^2 \qquad (1)$$

Even the efficiency goes down tremendously, being of the order of 10^6 - 10^8. In comparison, the efficiency for functional level simulators is of the order of 10^3 - 10^4.

In order to make the simulation of faults in IC's meaningful, an appropriate fault model is necessary. A fault model is the mapping from physical defects to the simulated faults. It shows how a circuit could fail in terms of the fault. More importantly, it aims at imposing a limit on the number of test vectors required, while at the same time keeping the coverage high.

At the gate level, a widely used fault model is the stuck-at fault model. Although, many physical defects can be modeled as stuck-at faults, the mapping from physical defects to stuck-at faults is not complete. The problem has become worse due to the evolution of new fabrication technologies e.g. HNMOS,

0738-100X/85/0720$01.00

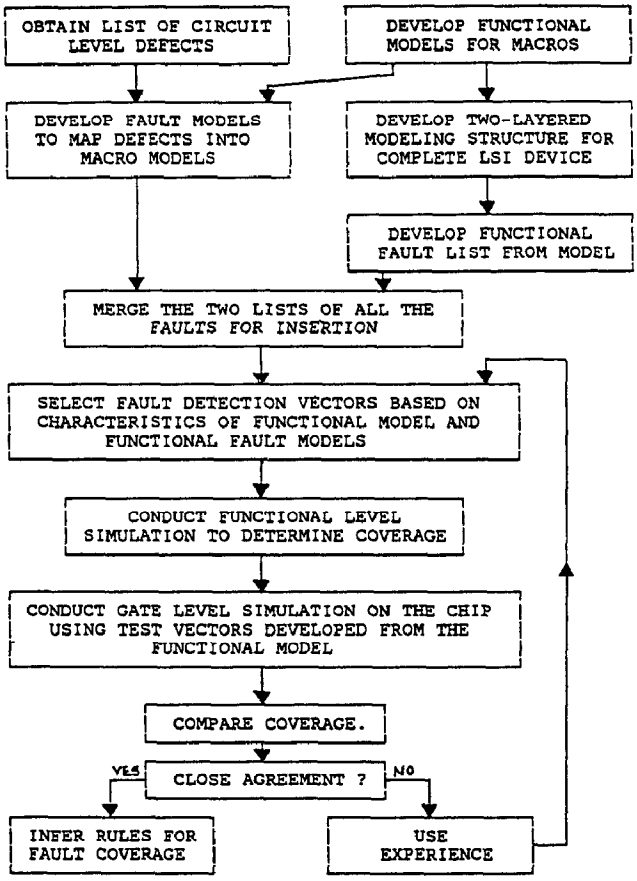

Fig.1 Functional Fault Modeling Methodology for the AMAC chip.

CMOS and the shift from LSI to VLSI systems. However, because of the lack of a proven better method and because of the fact that the stuck-at model has allowed generation of effective chip tests and resulted in a low defect level, its popularity remains.

To date, a generalized and effective functional fault model corresponding to the gate level stuck-at model has eluded the researchers. The difficulty lies in the higher level of representaion of the functional level descriptions. In this paper we furnish the details of the functional fault modeling work done on some typical digital circuits to establish the generality and the effectiveness of the functional fault model [FM] we have proposed[1,2,4].

Fault Simulation Methodology

The flow-chart of fig.1 shows the approach taken during the research involving the modeling and simulation of faults and the generation of functional level test vectors. As shown in the figure, two different approaches were tried for obtaining a list of functional

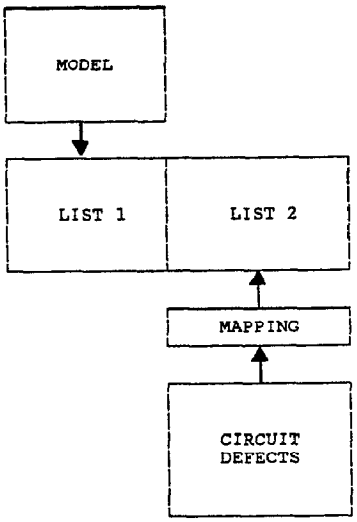

Fig.2 Fault Lists from Different Levels

faults:

1. Functional level defects obtained solely from the model description, i.e. model perturbation, and

2. Circuit level defects mapped into functional level defects.

The procedure resulted in two fault lists; List 1 consisted of faults from the models and List 2 consisted of circuit level defects mapped onto functional level defects [fig.2]. The two lists were merged and faults were injected into the functional descriptions of the circuits under test. Functional level simulations were then performed with several different inputs to obtain the fault detection vectors. These functional level test vectors were subsequently run on the gate level models furnished by IBM, to determine the equivalent gate level coverage. Comparison of the gate and functional coverage levels allowed inference of rules for functional fault modeling, and in some cases the experience gained required further iterations of the process.

For the purpose of modeling, a section [fig.3] of IBM's AMAC [Add Multiply and ACcumulate] chip was used. It consists of a 9-bit and two 17-bit LSSD Registers, three Parity checkers, an Adder and an 8x8 modified Dadda-type Multiplier. The chip is designed based on the LSSD principle, wherein the sequential logic is in the form of combinational circuits interspersed with sets of Shift Register Latches (SRL) connected serially[7]. The gate count for the section of logic mode led was 3700 gates.

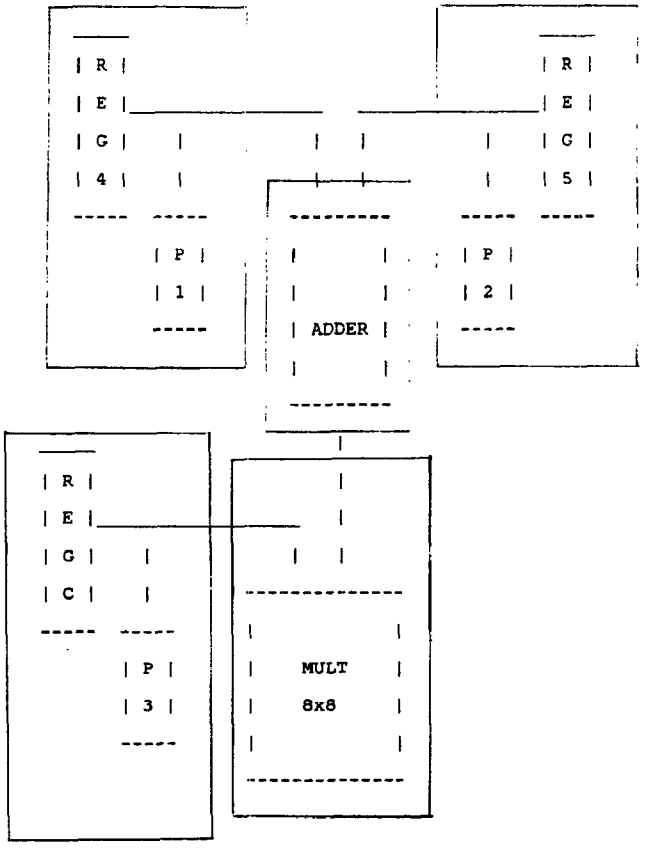

Fig.3 Functional units for AMAC model.

Accurate functional models were prepared for the unit under test [UUT], based on timing specifications, layouts, and other logic level information[6]. Three separate models were prepared, one each for the adder, the multiplier, and the register with parity. The three registers and their odd parity generators work in the same way and hence, only one model was developed for the registers. But, for the purpose of simulation, three copies of the same model were used with the required modifications.

The Adder

The Adder is a 16-bit ALU consisting of four 4-bit slices. Each slice is a TI 74181 ALU. The carry-look-ahead circuit provided with the four ALU's obtains a substantial speedup. The adder can potentially perform 48 operations, but in the AMAC unit it is used only in one particular mode to realize 32 functions. Table 1 gives the ALU functions used.

The model for the adder was prepared as a single functional unit. Each of the 32 functions performed by the adder is defined as a set of micro-operations of the model. The control inputs are decoded to jump to the section of the code containing the corresponding set of micro-operations. These micro-operations manipulate 16-bit data at a time instead of 4-bit slices of data. The carry-look-ahead function was modeled implicitly with the data manipulation. Thus, the model was not based on the physical structure of the circuit. This was done to obtain a higher level description which also speeds up the simulation to a great extent.

TABLE 1: TI 74181 ALU Functions used in AMAC chip

Selection				*M = L (Arithmetic Operations)*	
S3	*S2*	*S1*	*S0*	*Cnbar = H*	*Cnbar = L*
L	L	L	L	F = A	F = A plus 1
L	L	L	H	F = A + B	F = (A + B) plus 1
L	L	H	L	F = A + B'	F = (A + B') plus 1
L	L	H	H	F = -1 (2's compl)	F = zero
L	H	L	L	F = A plus AB'	F = A plus AB' plus 1
L	H	L	H	F = (A+B) plus AB'	F = (A+B) plus AB' plus 1
L	H	H	L	F = A - B - 1	F = A minus B
L	H	H	H	F = AB' - 1	F = AB'
H	L	L	L	F = A plus AB	F = A plus AB plus 1
H	L	L	H	F = A plus B	F = A plus B plus 1
H	L	H	L	F = (A+B') plus AB	F = (A+B') plus AB plus 1
H	L	H	H	F = AB - 1	F = AB
H	H	L	L	F = A plus A*	F = A plus A plus 1
H	H	L	H	F = (A+B) plus A	F = (A+B) plus A plus 1
H	H	H	L	F = (A+ B') plus A	F = (A+B') plus A plus 1
H	H	H	H	F = A minus 1	F = A

The Multiplier

The AMAC chip has a fully combinational, very high speed Multiplier. It uses a modified Dadda-type scheme[11] to generate the 16-bit product of two 8-bit numbers. The numbers are represented in signed 2's-complement notation. The multiplier uses a set of input and output latches to store the data temporarily. Summands are obtained from the latched inputs and are used in groups of two or three to generate partial sums and carry's. Three stages of full adders are used to obtain the final sum and carry. A carry-look-ahead circuit then hastens the calculation of the final product.

The flow of control and data in the model for the multiplier was modified very little from the one in the actual layout. In this sense the model is based on the structure, but most of the individual functions have been defined non-structurally, using the two methods outlined above.

The Register with Parity Generator

The registers in the circuit are dual latches with serial-shift capability for scan. These are popularly known as LSSD registers[3]. Scan capability is added to sequential circuits in order to simplify testing. In the test-mode of operation, the

registers are loaded serially with the desired patterns of binary values. These patterns are then passed through one stage of combinational logic, the results of which are latched to the subsequent stage of registers. The output is serially shifted out and checked for faults in the preceding stage of logic.

An odd-parity generator provided with each register in the UUT is used to check the parity of the data latched into the register.

Each register in the chip was modeled with its scan operations and its parity-check circuit[8]. The model was prepared using a combination of both the techniques specified above.

FUNCTIONAL LEVEL MODELING

GSP: The Simulation Language

The functional fault models were prepared using GSP (General Simulation Program)[1,2,4,5,7]. GSP is a general purpose, two-valued (1,0) simulation language which was developed at Virginia Tech specifically to perform the simulation of VLSI devices at the chip level[5]. Its most useful application is the modeling and simulation of complicated VLSI circuits and microprocessors. The language has been used extensively for modeling functional-level faults in simple and complex VLSI devices. It also has the capability to model such interface timing specifications as setup time, hold time and minimum pulsewidth[1,4,5].

Modeling in GSP is done in an assembly language with special instructions for hardware description. The instruction set is illustrated in the examples given below [figs.6,7]. The GSP manual[7] contains detailed explanation on the usage of each instruction.

The structure for the GSP simulation system is shown in fig.4. Each module description file is assembled to obtain the microcode file. The microcode files are merged together with the states into the LINK file. The DATA file has the information on module interconnections, initializations and inputs. The simulator reads the data file at the beginning of simulation and executes the microcode during simulation, generating the outputs.

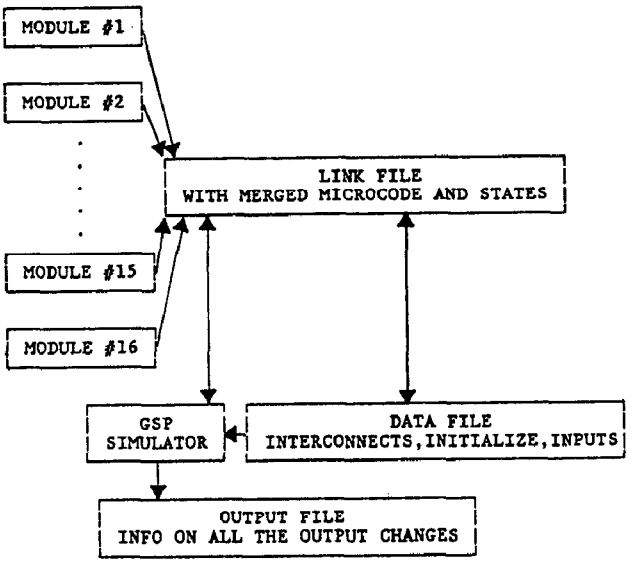

Fig.4 GSP Simulation Structure

Methods of Functional Modeling

The two general methods for modeling digital devices at the functional level are shown in fig.5.

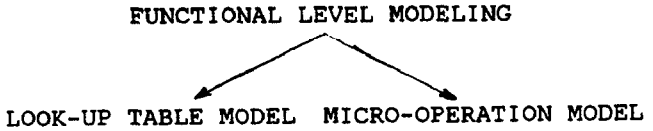

Fig.5

Look-up Table Model

In this method, the functional unit is represented in the form of a truth-table (combinational logic) or a state-table (sequential logic). In order to access a particular value in the truth-table, the inputs to the functional unit are decoded to point to the location containing that value in the 'Look-up Table'. This is a very simple approach to modeling. Several such truth-tables for the different functions are put together to form the model for the whole device. In GSP, the decoding constructs are used to perform this operation. The example in fig.6 describes the 'look-up table' model for an And-Or-Invert function of three inputs, F(x1,x2,x3) = (x1x2 + x2x3 + x3x1)'. As can be seen from the figure, the number of bits of the input register that are to be decoded, are moved into one of the index registers (index register 1, in the example). The index register is used as the pointer to the locations of a table (table AOI, in the example), and the value contained in the location pointed by the contents of the index register is

AND-OR-INVERT (AOI)

```
; registers for the model
;
REG(3)    OLDX
;
; pins for the module  X1,X2,X3 : 1,2,3 ; AOI : 4
;
PIN       X1X3(1,3),OUT(4)
;
; delays for the module
;
EVW       DEL1(40)
;
; module description
;
       BNE   X1X3,OLDX,PROC
       EXR                        ; EXIT
;
PROC:  MOV   X1X3,OLDX            ; STORE FOR NEXT CHECK
       IDX   OLDX(0),3,1          ; STARTING WITH 0TH BIT,
                                  ; MOVE 3 BITS INTO INDEX REG.1
       MOV(DEL1) AOI@1,OUT        ; MOV THE CONTENTS OF LOCATION
                                  ; POINTED BY INDEX REG.1 TO THE
                                  ; OUTPUT, AOI, AFTER DEL1.
       EXR                        ; EXIT AND RESTART
;
;LOCATIONS 0  1  2  3  4  5  6  7
AOI : BYT #1,#1,#1,#0,#1,#0,#0,#0
;
END
```

Fig.6 "Look-up Table" Model.

then moved out to the destination (pin OUT) after a delay of 40 ns.(DEL1).

Micro-Operation Model

Here, the functional unit is defined as a sequence of model micro-operations, using the constructs of the modeling language. The example in fig.7 describes the 'micro-operation' model

AND-OR-INVERT (AOI)

```
; registers
REG(1)    OLDX1,OLDX2,OLDX3
REG(1)    AND12,AND23,AND31
REG(1)    ORBUF
; pins
PIN       X1(1),X2(2),X3(3),OUT(4)
; delays
EVW       DEL1(40)
;
; description
;
       BNE  X1,OLDX1,PROC    ; BRANCH IF VALUE OF X1,X2,X3
       BNE  X2,OLDX2,PROC    ; HAS CHANGED.
       BNE  X3,OLDX3,PROC
       EXR                   ; EXIT
;
PROC:  MOV  X1,OLDX1         ; FOR COMPARISON ON NEXT SIGNAL
       MOV  X2,OLDX2         ; CHANGE ON X1, X2, X3.
       MOV  X3,OLDX3         ;
;
       AND  X1,X2,AND12          ; (AND12) = (X1) . (X2)
       AND  X2,X3,AND23
       AND  X3,X1,AND31
       OR   AND12,AND23,ORBUF
       OR   AND31,ORBUF,ORBUF    ; DESTINATION REG. IS ORBUF
       COM  ORBUF,ORBUF          ; INVERT
       MOV(DEL1) ORBUF, OUT
       EXR                       ; EXIT AND RESTART
END
```

Fig.7 "Micro-operation" Model.

for the And-Or-Invert function of three inputs, similar to the one in fig.6. GSP modeling constructs like AND, OR, and NOT are used in a sequence of micro-operations which yield the final output. The functional model can be viewed as a nodal graph with two kinds of edges interconnecting the nodes. Each node is a set of model micro-operations with control and data being transferred from one node to another.

The functional models were prepared in a non-structural way i.e., the model descriptions were not based on the actual physical structure of the device layout. This way, a truly functional level description is obtained for the circuit.

FAULT MODELING AT THE FUNCTIONAL LEVEL

The functional fault model proposed in[1,3] and described in this document is independent of the technology used for fabrication. Due to the higher level of representation, simulation runs are very fast. The validity and simplicity of using these models for complex VLSI circuits was studied during this research.

There are two basic approaches to functional fault-modeling. In one, the physical structure of the device is given importance and circuit level defects are mapped onto functional level faults [fig.2]. While in the other, the correct functional model description/procedure is faulted to obtain an incorrect version of the procedure. The fault itself MAY or MAY NOT be directly related to any specific circuit-level defect. Once faults are selected, various input vectors are tried to obtain test vectors that can detect those faults.

Previous research in this respect shows that the first approach yields good results for gate-level simulation because of the closeness of the model description to the physical layout of the device. But the same does not hold for the case of functional-level simulation wherein, the physical structure of the device is not explicitly defined in the model.

As such, for functional-level fault simulation, the second approach is favored. We shall call it the Model-Perturbation [MP] approach. However, for the AMAC functional unit, simulations were performed using both the methods, with greater emphasis on Model-Perturbation.

Model-Perturbation can be defined in the following way. If the correct model-procedure is C(x) and the set of faulty model procedures is F(x), where 'x' is a set of inputs, then the transformation from the correct model procedure to the faulty one, can be represented as :

$$F(x) = [A]\ C(x)$$

where the transfer function, [A], is a set of faults injected into C(x).

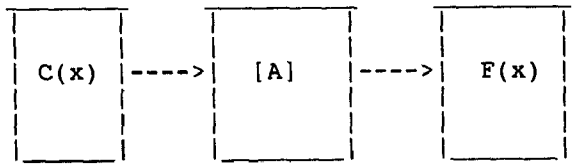

Fig.8 MODEL-PERTURBATION

Our research dealt with the finding of the operator [A] in order to maximize coverage of faults at the functional level. The extent of coverage is defined in terms of the gate-level coverage for the present, but we hope to obtain an independent definition based on the results of our work in this area.

The models can be 'perturbed' in certain ways. In an earlier report[2] we described these as the "Truth-Table Modification" and "Micro-Operation Modification" procedures [fig.9]. In the modeling and simulation performed on the AMAC functional unit, both these methods were adopted.

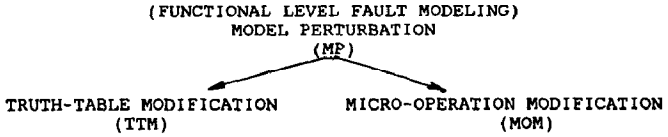

Fig.9

Truth-Table Modification (TTM)

Detectable faults in a block of functional logic result in truth-tables with modified outputs. Hence, the truth-table ('look-up table' in GSP) can be modified in many ways to simulate different faults. This is illustrated in fig.10 for the previously cited AOI example. AOI table now contains certain incorrect values and for corresponding combinations of the input, incorrect values get moved to the output.

It should be noted that the tables are not exhaustively modified because it would result in too many possible combinations, with each combination being tried for each run. For example, a table with 8 (three inputs, 2**3) entries can be modified in (2**8 - 1) ways! It would take very long to run all these simulations even at the functional level. The fault list for table modification was obtained from a study of the VLSI layouts for circuit level faults. The circuit-level faults

FAULT-INJECTED AND-OR-INVERT

```
; registers for the model
;
REG(3)    OLDX
;
; pins for the module  X1,X2,X3 : 1,2,3 ; AOI : 4
;
PIN       X1X3(1,3),OUT(4)
;
; delays for the module
;
EVW       DEL1(40`
;
; module description
;
       BNE   X1X3,OLDX,PROC
       EXR                        ; EXIT AND RESTART
;
PROC:  MOV   X1X3,OLDX            ; STORE FOR NEXT CHECK
       IDX   X1X3(0),3,1          ; STARTING WITH 0TH BIT, MOVE
                                  ; 3 BITS INTO INDEX REG. 1
       MOV(DEL1) AOI@1,OUT        ; MOV THE CONTENTS OF LOCATION
                                  ; POINTED BY INDEX REG. 1 TO THE
                                  ; OUTPUT, AOI, AFTER DEL1.
       EXR                        ; EXIT AND RESTART
;
; table is modified from the previous one, for locations 2,5,7
;
;LOCATIONS  0  1  2  3  4  5  6  7
AOI :  BYT #1,#1,#0,#0,#1,#1,#0,#1
;
END
```

Fig.10 Truth-Table Modification Technique.

were then mapped onto functional faults.

The extent of coverage using this method is dependent on the complexity of the model because for a simple function, there are not many different ways in which the look-up table can be faulted/modified. The functions employed in the modeling were simple. As a result, the number of test vectors obtained is not high and the few test vectors obtained are capable of detecting most of the injected faults.

One important aspect of this process is the method of mapping circuit level defects to the functional level faults. For the AMAC model, each circuit level defect chosen was studied to see how it affects the behavior of the corresponding function. This was done 'manually' and can be a serious limitation in terms of automating the fault simulation process. Inspite of its limitations, this method is attractive as it is very simple and easy to inject faults. The process of table modification and reassembly of the model description for each such modification can be automated.

Micro-Operation Modification (MOM)

Every model description consists of a sequence of micro-operations which define the behavior of the simulated device. The micro-operation model is especially well-suited for the method of fault modeling discussed here. In this method, the correct model of the device is taken to be an entity in itself and

any relationship between the model and the physical circuit it represents, is transparent to the fault modeling process. Thus, for all purposes of fault simulation, the model is the actual circuit.

The micro-operation model is prepared from the control and data-manipulation constructs of the modeling language. It should be noted that the control type micro-operations of the model need not necessarily correspond to the control signals of the simulated device. The *control type micro-operations of the model* include conditional and unconditional branches, loops, decoding functions like CASE or computed GOTO statements, and jumps to subroutines. All other micro-operations constitute the *data-manipulation micro-operations of the model.*

Functional level fault modeling is done in terms of modifying these model micro-operations to make them faulty, one at a time. To obtain the test vectors for the AMAC functional unit, the GSP modeling constructs like AND, OR, XOR, ADD, and SUB were modified. In addition, control constructs such as BEQ and BNE were also changed. Modifications like replacing each occurrence of AND by OR, OR by AND, ADD and SUB by XOR, ADD and SUB by OR, and changing conditional-branches to unconditional-branches and vice-versa, the latter in terms of the immediately preceding operation, resulted in the generation of a substantial number of test vectors. Thus most of the micro-operations were replaced by their logical duals to perform functional fault simulation. This technique of failing the model to the 'incorrect' mode, which is the LOGICAL DUAL (*) of the 'correct' mode of operation, yielded very encouraging results in terms of fault coverage.

The biggest argument in support of this method is its simplicity and regularity, which make it amenable for automation, apart from keeping the coverage high.

RESULTS AND CONCLUSIONS

'Model Perturbation', as developed here, is a simple, feasible and effective technique to inject functional level faults. The simplicity of the TTM and MOM procedures for fault injection makes them viable for automation.

The results for the simulation of AMAC functional unit are very encouraging. In all, 857 functional faults were injected into the models and 110 test vectors detected all these faults. The functional level coverage was 100% of the functional fault list, while the equivalent gate level coverage was 88.60% of the 8919 gate level faults.

We believe that further work in this direction will yield improved coverage and help us come up with a well defined classification of functional level faults and an independent definition of functional fault coverage. The work reported here was performed 'manually'. Research is being done on the automation of the whole process of fault simulation based on the model perturbation approach. Several algorithms are being studied for optimum solutions. The results on these will be reported in the future.

---

* Some Logical Duals are : XOR vs. Equivalence, OR vs. AND.

ACKNOWLEDGEMENTS

The authors would like to extend their gratitude and thanks to Mr. Anil D. Savkar of IBM, Manassas, who continually provided us with useful information in the form of discussions and design & layout details on the project. His support was a great help in completing the research successfully.

REFERENCES

[1] James R. Armstrong, "Chip Level Modeling of LSI Devices", *IEEE Transactions on Computer-Aided Design, Vol. CAD-3, No. 4,* October 1984.

[2] James R. Armstrong et. al., "Interim Report for IBM Contract : Functional Fault Modeling for VLSI Devices", *Dept. of Electrical Engineering, VPI&SU,* Blacksburg, May 1984.

[3] James R. Armstrong, "Interim Report for IBM Contract : Functional Fault Modeling for VLSI Devices", *Dept. of Electrical Engineering, VPI&SU,* Blacksburg, December 1983.

[4] James R. Armstrong, "Chip Level Modeling and Simulation", *SIMULATION,* October 1983.

[5] James R. Armstrong and D. E. Devlin,"GSP: A Logic Simulator for LSI", *18th IEEE Conference on Design Automation,* 1981.

[6] "Information and Specifications provided by IBM on AMAC chip", Fall 1983.

[7] "The GSP Manual", *Dept. of Electrical Engineering,VPI&SU,* Blacksburg, December 1982.

[8] Se June Hong, Daniel L. Ostapko, " A Simple Procedure to Generate Optimum Test Patterns for Parity Logic Networks ", *IEEE Transactions on Computers, Vol. C-30, No. 5,* May 1981.

[9] E. B. Eichelberger and T. Williams, "A Logic Design Structure for LSI Testing"; *Proc. 14th Design Automation Conference, New Orleans,* June 1977.

[10] James B. Stewart, Unpublished work, *Dept. of Electrical Engineering, VPI & SU,* Blacksburg, August 1984.