On the p-untestability of combinational faults

Suresh k Devanathan

Michael L Bushnell

***Abstract*—*We describe the p-untestability of faults in combinational circuits. They are similar to redundant faults, but are defined probabilistically. P-untestable fault is a fault that is not detectable after N random pattern simulation or a fault, FAN either proves to be redundant or aborts after K backtracks. We chose N to be about 1000000 and K to be about 1000. We provide a p-untestability detectability algorithm that works in about 85% of the cases, with average of about 14% false negatives. The algorithm is a simple hack to FAN and uses structural information and can be easily implemented. The algorithm does not prove redundancy completely but establishes a fault as a probabilistically redundant, meaning a fault with low probability of detection or no detection.*

I. Introduction

It has been known that *automatic test pattern generation* ATPG is a hard problem. One of the most difficult problems that arise in ATPG is proof of untestability. This would offload and do a great benefit to a random pattern generation since a random pattern generator requires an infinite sequence to label a fault as untestable. Although this is generic untestability, we are not concerned with this problem at this moment, as it is very difficult problem. Instead we want to limit our focus on probabilistic untestability which is an easier problem and can easily be proved by a fault simulator by random pattern testing or algorithmic backtrack limits.

II. Definitions

We define p-untestability of a fault to be probability a fault simulator fails to detect it after N patterns of random vector simulation or a fault aborted or proven to be redundant by a deterministic ATPG algorithm , esp. the FAN[6] algorithm after K backtracks. We can call it p-untestability(N,r) or p-untestability(K,b) where r and b stand for random pattern generation or algorithmic backtrack limit. For the sake of simulation, we restricted our search for p-untestability with N=1000000 and K = 1000. We used the ATALANTA[8] atpg tool and HOPE[7] fault simulator for the sake of analysis.

III. Algorithmic identification of p-untestable faults

You will have to modify calls to fan as follows to identify a p-untestable fault. The algorithm has been shown to work in most cases and only provides false negatives in a minority of cases.

A. Symbol meanings

cf stands for current fault.
cf->line is -1 for output stem or an input line for input line fault.
cf->gate is the gate of the current fault.
gate->fn is the function of the gate:AND,OR, NOT,etc.
get_rog_of_rog(g) is a function which returns a random output gate of a random output gate of g.
K is the backtrack limit of the algorithm.
put_fl is the p-untestability fault list.
SA0 means stuck at 0 fault.
SA1 means stuck at 1 fault.
&param denotes a pass-by reference parameter.
g2->wire_input(g) wires the output of gate g into the input of g2.
g2->unwire_input(g) removes gate g from the input of g2.
put_fl->add_fault(f) adds fault f to fault list put_fl.

B. Proc1

Here we describe Proc1 which is internally called by the main p-untestability identification algorithm. The purpose of this procedure is to change the function of the gate depending on whether it is a stem fault or input line fault.

```
procedure PROC1(cf,fg)
    if cf− >line is output stem then
        if cf− >type is SA0 then
            switch fg->fn do
                case AND
                    fg− > fn ← OR
                case NOR
                    fg− > fn ← NAND
        else if cf->type is SA1 then
            switch fg->fn do
                case OR
                    fg− > fn ← AND
                case NAND
                    fg− > fn ← NOR
        end if
    else if cf->line is a input line then
        switch fg->fn do
            case OR
                fg− > fn ← AND
            case NOR
                fg− > fn ← NAND
            case NAND
                fg− > fn ← NOR
            case AND
                fg− > fn ← OR
    end if
end procedure
```

C. Proc2

He we describe Proc2 which is internally called by the main algorithm. The purpose of this procedure is to deal with case where the fault is a input line. This procedure is similar to Proc1's ability to handle input line fault, except it nots the function of the gate.

```
procedure PROC2(cf,fg)
    if cf->line is input line then
        switch fg->fn do
            case OR
                fg-> fn ← NAND
            case NOR
                fg-> fn ← AND
            case NAND
                fg-> fn ← OR
            case AND
                fg-> fn ← NOR
    end if
end procedure;
```

D. Proc3

Here we describe Proc3 which is internally called by the main algorithm. This procedure changes the function of the gate attached to the input line.

```
procedure PROC3 (cf,&fg,&sfn)
    if cf->line is input line then
        *fg ← cf-> gate-> inputs[cf-> line]
        *sfn ← fg-> fn
        Proc1(cf, fg)
    end if
end procedure
```

E. Proc4

Here we describe Proc4 which is internally called by the main algorithm. This procedure rewires the gate to output of a output of the gate of the current fault site.

```
procedure PROC4(cf,&m,&g2)
    g ← cf-> gate
    g2 ← get_rog_of_rog(g);
    m ← 0
    if g is not in g2->inputs then
        m ← 1
        g2-> wire_input(g)
    end if
end procedure
```

F. Identify

Here we describe Identify algorithm which forms the main core of the process. Identify algorithms takes foot once FAN is unable to get test or prove a fault to be untestable. Identify is an approximate procedure. It is really useful in identifying p-untestable faults, as seen in table I. It does however have a false identification rate averaging 14%, as seen in table II. The algorithm works by modifying the function of the gate either at the fault site or around it or it changes the observability criteria of the fault by joining the output of the fault gate to the input of a gate connected to output of one of its output gates.

```
procedure IDENTIFY(cf,&put_fl)
    status ← FAN(cf, K)
    r ← 0
    while (cf not detected | cf is not declared untestable)
& (r < 4) do
        ts ← status
        fg ← cf-> gate
        sfn ← fg-> fn
        switch r do
            case 0
                Proc1(cf, fg)
            case 1
                Proc2(cf, fg)
            case 2
                Proc3(cf, &fg, &sfn)
            case 3
                Proc4(cf, &m, &g2)
        status ← FAN(cf, K)
        if status is not redundant then
            status ← ts
        end if
        fg-> fn ← sfn
        if (r = 3) & m then
            g2-> unwire_input(g)
        end if
        if status is redundant then
            put_fl-> add_fault(cf)
            break
        end if
        r ← r + 1
    end while
end procedure
```

IV. RESULTS

We modified the test generation phase of the ATALANTA FAN algorithm with static learning. We also used the hope fault simulator to test the p-untestability of these faults and applied the original ATALANTA algorithm with backtrack limit of 1000 to test to see if faults declared p-untestable are still declared redundant or aborted or no test is found. Results are reported in table I and II.

TABLE I
P-UNTESTABILITY IDENTIFICATION WITH IDENTIFY ALGORITHM

Ckt.	PutF	N-HOPE Undet.	K-FAN (red+ab)
b01	0	-	-
b02	0	-	-
b03	0	-	-
b04	1	1	1
b08	0	-	-
b10	0	-	-
b11	2	2	2
b12	5	5	5
b13	0	-	-
b14	20	20	20
b15	371	346	347
b17	899	774	806
b18	512	363	393+1=394
b20	100	98	80
b21	117	115	83
b22	122	116	97

PutF: p-untestable faults
N-HOPE Undet: Undetected faults after simulated using HOPE for N random vectors, esp. 1000000
K-FAN (red+tab): Faults aborted or proved redundant by FAN after K-backtrack limit, esp. 1000

TABLE II
P-UNTESTABILITY IDENTIFICATION WITH IDENTIFY ALGORITHM STATISTICS

Amount	Value
Total PutF	2150
Total N-Hope Undet.	1840
Total K-FAN (red+ab)	1835
Success Rate N-Hope	85.62%
Success Rate K-FAN	85.35%
Failure Rate N-Hope	14.38%
Failure Rate K-FAN	14.65%

V. CONCLUSION

In return, we built a new model of untestability, namely p-untestability and showed that it is very useful in identifying untestable faults. We also showcased an approximate algorithm that was able to detect p-untestable faults. Future improvements to the algorithm identification will make it more and more accurate.

ACKNOWLEDGMENT

The authors would like to thank...

REFERENCES

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.
[2] Suresh Kumar Devanathan and Michael L. Bushnell, *Sequential Spectral ATPG Using the Wavelet Transform and Compaction*, VLSI Design, 2006, pp. 407-412
[3] Suresh Kumar Devanathan and Michael L. Bushnell, *Test Pattern Generation Using Modulation by Haar Wavelets and Correlation for Sequential BIST*, VLSI Design, 2007 pp. 485-491
[4] lrith Pomeranz, *Procedures for Static Compaction of Test Sequences for Synchronous Sequential Circuits Based on Vector Restoration*, DATE, 1998
[5] T M Niermann, *HITEC: A test generation package for sequential circuits*, EDA, 1991
[6] H. Fujiwara , *Fan: a fanout-oriented test pattern generation algorithm* ISCAS, 1985
[7] Hyung Ki Lee, Dong Sam Ha , *HOPE: an efficient parallel fault simulator* DAC, 1992, pp. 336-340
[8] H. K. Lee, D. S. Ha , *Atalanta: An Efficient ATPG for Combinational Circuits* ,1993