VLSI Testing and Design Testability Assignment 6

311510173 魏子翔

1. How to compile codes

   Go to the folder /podem, then enter “make” in command line to compile all codes.

2. The algorithm and idea of my codes

   In problem d, I adjusted ***void CIRCUIT::Atpg()*** code to generate random pattern, and added the while loop in this code to compute the fault coverage and the number of patterns.

   In problem e, to deal with the bridging fault list in PODEM program, I modified these codes ***bool FaultEvaluate(FAULT* fptr), ATPG_STATUS Podem(FAULT* fptr, unsigned &total_backtrack_num), ATPG_STATUS SetUniqueImpliedValue(FAULT* fptr), GATEPTR TestPossible(FAULT* fptr)*** so that I can check the two different values in bridging wires.

3. Several case results

   a. Generate test vectors for b17.bench and set backtrack limit

Number of backtrack	Number of patterns	Fault coverage(%)	CPU run times(s)	Actual backtrack number
1	41647	55.00	1234.98	68413
10	72511	82.09	1802.57	376982
100	83711	90.01	2202.96	1821691
1000	86025	91.62	5278.58	12664818

We can see that when the number of backtrack increases, the fault coverage becomes higher. A larger back track limits means the PODEM code can try more to achieve the generation of vectors.

   b. Verifying ATPG results

Fault list	Number of patterns	Fault coverage(%)	CPU run times(s)
PODEM	86025	97.28	321.70
Checkpoint	86025	84.13	239.35

A test set can detect all single stuck at fault, when it can detect all single stuck at fault at check points. However, PODEM cannot cover all stuck at fault at check points, so we cannot get higher coverage.

c. PODEM implementation

net 17 s.a.0				
Meaning	Objective	PI assignment	Forward Implication	D-frontier
Fault activation	net17=1	G1=0	net17=D	G16
Fault propagation	net18=1	G2=0	net18=1, G16=D’	X
Finish	X	X	X	X
n60 s.a.1				
Meaning	Objective	PI assignment	Forward Implication	D-frontier
Fault activation	net60=0	G2=1	net60=D’	G17
Fault propagation	net25=0	G3=0	net14=1, net17=1, net18=0, net25=0, G16=0, G17=D	X
Finish	X	X	X	X

d. ATPG with random patterns

Circuit	Number of patterns		Coverage(%)		CPU times(s)	
	Origin	Random	Origin	Random	Origin	Random
b17	83711	88945	90.01	93.00	2202.96	16126.51
s35932_com	77	1000	89.64	89.69	76.77	80.04
s38417_com	1373	2349	99.68	99.68	6.21	21.89
s38584_com	856	1873	95.57	95.57	7.53	19.51

The random patterns coverage is little higher than original one. But random patterns need to more pattern than origin.

e. Test generation for bridging faults

Circuit	Number of patterns	Coverage(%)	CPU times(s)
c17	5	100	0
c432	52	24.13	0.23
c880	62	62.00	1.23
c1355	239	51.85	0.27
c5315	140	89.95	4.20
c7552	2338	68.26	53.69

We cannot ensure that every detectable defect would be collected by my codes, so the bridging defects are not efficient.