VLSI testing - Assignment 5

309510133 - Cheng-Cheng Lo

Part.a

Take c17.bench for example. The instructions are described as follows.

Run

1-a. generate patterns for a circuit by running **built-in** ATPG

```
./atpg -output c17.atpg c17.bench
```

1-b. generate patterns for a circuit by running ATPG based on **checkpoint theorem**

```
./atpg -check_point -input c17_atpg c17.bench
```

2-. run fault simulation to obtain fault coverages respectively for original and checlpoint-based ATPG

```
./atpg -fsim -input c17_atpg c17.bench
```

Results

The right two columns shows the fault coverage for built-in and checkpoint-based ATPG respectively, with the number of patterns shown inside ( ).

circuit	original	checkpoint
c17.bench	100% (8)	100% (7)
c499.bench	96.99% (79)	96.99% (80)
c2670.bench	96.29% (167)	96.29% (159)
c5315.bench	99.45% (200)	99.45% (181)
c7552.bench	98.42% (361)	98.43% (342)

Part.b

Run

To generates random patterns, use what we have done in hw2.

```
./atpg -pattern -num 1000000 -output c17_pattern c17.bench
```

The number of faults per pass is defined in line 18 in the file `typeemu.h`.

```
const unsigned PatternNum = 16;
```

Modify the number and run

```
./atpg -fsim -input c17_pattern c17.bench
```

Results

CPU-time (sec)	1	8	16	32	64
c17.bench	0.35	0.35	0.35	0.35	0.35
c499.bench	18.68	17.85	17.70	17.05	17.37
c2670.bench	127.96	121.03	115.73	109.92	105.55

Discussion

The case c17.bench is too small to see the changes.

The second and the third case, which are bigger circuits, in the contrary, the execution time are reduced when the number of faults per pass is increased.

Part.c

Run

```
./atpg -bridging_fsim -input [input_pattern_file] [circuit_name]
# for example,
./atpg -bridging_fsim -input c17_atpg c17.bench
```

Algorithm

The key idea is, given a list of bridging faults, can a pattern detect some of the faults. We start with a pattern, run parallel fault simulation and compare the simulation results to remove detected faults.

The fault simulation for bridging faults are described below.

```
for different patterns {
    run fault-free simulation
    single pattern parallel bridging fault simulation (
        set fault-free value for every gates
        for all undetected faults f {
            if f redundant: skip
            if f not activated (*1): skip
            let g be the fault-occur gate
            if g is PO:
                set the fault to de DETECTED
            add the fault to the simulated list and inject (*2) it
        }
        do fault simulation and remove newly-detected faults
    )
}
```

The process running fault simulation is pretty much the same for stuck-at faults and bridging faults. Except for (marked as * above):

1. the condition of fault activation
    - for s-a-x faults: the signal is not x
    - for bridging faults: the two signals have different values
2. the behavior of fault (inject value)
    - for s-a-x faults: the signal is fixed to x
    - for bridging faults: one of the two signals are changed to 0 (AND) / 1 (OR), to make the two signals the same value

Test Cases

Take c17.bench for example. There are 16 bridging faults in the beginning. The changes of undetected faults are shown in the table below.

G1-5	newly-detected faults	Undetected faults
(start)	-	16
11001	3	13
00110	1	12
01111	2	10
10101	6	4
00010	1	3
10001	2	1
11011	1	0
11010	0	0