# Diagnosis

# 1. Introduction

The goals of fault diagnosis are to ascertain whether faults are present (fault detection) in the CUT and to identify them (fault location). Fault detection can be done by comparing logic simulation results with fault simulation results, and fault location is usually performed with the aid of a fault dictionary which is usually exhaustive. But here I'm using a non-exhaustive method to locate the fault of CLB as described below.

# 2. The non-exhaustive method

For the dictionary-based method, the dictionary corresponding to a input vector is a full dictionary (where we do simulation for fault 0 to fault 255 for every CLB) but actually it is not necessary and this is where we can improve. A CLB is fully represented by its Boolean Switching Function. What we all have to do is to determine the eight rows of a CLB (or as many as we can determine).

Imagine a CLB with three PIs (say 001) as inputs, for a certain test vector T1, we are using only one row of its BSF (because PI has no fault and inputs will be fixed 001 for this T1). So we don't have to insert and simulate fault 0 to 255 for this CLB.

For a more general case, more than one input is from outputs of other CLBs, say inputs are 111. Since our project assumption is that we only have one faulty CLB, so this inputs might be 011, 101 or 110, if inputs are from three different outputs ( if two or three of inputs come from the same branch, this input vector could be any value from 000 to 111). Actually we can focus on the output of a CLB and this way we don't even have to worry about branches. A fault is the output of a CLB going from 0 to 1 or from 1 to 0. With the one faulty CLB assumption, if we assume the output is wrong then inputs of this CLB (outputs of its upstream CLBs) must be right and fixed for a given test input vector. This brings much convenience. I implement the method with this conclusion and two lemmas below.

Lemma 1: Two same circuits with same inputs, will have same outputs.

Lemma2: Two circuits with same inputs get same outputs, they might and might not be the same circuits.

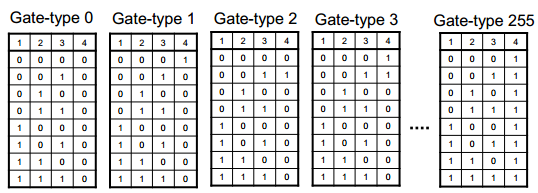
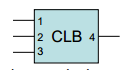
So for a test vector from ATE, I do logic simulation and compare with ATE response,

CASE1:

if they match, this CUT might be correct and might not (lemma2). This will help us find faults, if exist, that cannot propagate to the POs (because the POs match logic simulation results ). To find them, we flip one CLB output at a time and do fault simulation, if ATE response and fault simulation results (1) match, then this insertion is a possible fault because it does not cause any conflicts; (2)mismatch, then this insertion is a impossible fault on this CUT(contrapositive of lemma1). This actually determines only one row (say Row1)of a CLB BSF( at the beginning the 8 rows of a CLB in this CUT are all unknown for us). So any fault that has the same Row1 is possible fault. It's clear that there will be 128 (half of 256) possible faults for this CLB after this vector testing. We actually can just record the BSF rows or bits.

CASE2:

if they mismatch, then we can narrow down the faulty area: cone insertion. And CLBs in the intersection will be targets to flip the outputs. The rest is the same as CASE1.



# 3. Steps of the code.

1. Read netlist.txt

2. Levelize (logic simulation is level by level)

3. Read input\_output.txt

4. Logic simulation and compare with ATE

5. Cone insertion (if PO mismatches)

6. Fault injection

7. Fault simulation (only simulate the down cone of this fault, time reduced)

8. Compare faulty outputs with ATE, store and print results.

9. Flip the fault back and simulate down cone (to reset to logic simulation results)

10. Do step 4 to step 9 for every test vector in the input\_output.txt

# 4. Some results

For test case 2:

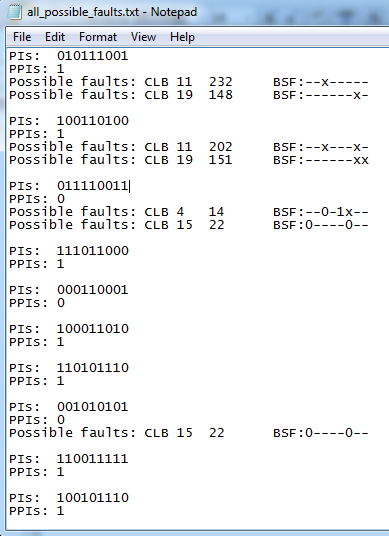
input\_output.txt which I'm using(deleted \*\*\* here):

3 1  
010111001--00  
\*\*\*  
3 1  
100110100--01  
\*\*\*  
3 0  
011110011--01  
\*\*\*  
3 1  
111011000--10  
\*\*\*  
3 0  
000110001--10  
\*\*\*  
3 1  
100011010--11  
\*\*\*  
3 1  
110101110--11  
\*\*\*  
3 0  
001010101--01  
\*\*\*  
3 1  
110011111--11  
\*\*\*  
3 1  
100101110--11  
\*\*\*

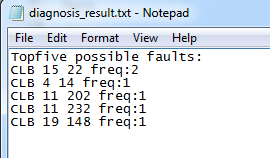
‘x’:likely fault;

‘-’: not tested;

'0' or '1': as it is in the fault free CLB



Top five possible faults(ranked by the frequency that it gets tested):



I still produced the full dictionary for reference.

