VLSI testing - Assignment 2

309510133 - Cheng-Cheng Lo

Part.1

The command for logic simulation on a circuit given a pattern is

```
./atpg -logicsim -input <input_pattern> <circuit>
```

simulation of circuit c17.bench given test pattern c17.input

```
./atpg -logicsim -input c17.input ../../circuits/iscas85/c17.bench
```

```
Start parsing input file
Finish reading circuit file
Run logic simulation
01110 00
10101 11
00101 01
01000 11
10001 01
00011 01
00111 00
00000 00
total CPU time = 0.000114
```

Part.2-a

Random Number Generation

I use linear congruential generators (LCG) to generate random number. The pseudorandom sequence is defined as follows.

$$X_{n+1} = (aX_n + c) mod M$$

In Numerical Recipes, a = 1664525, c = 1013904223 and M = 2^32 is used.

Results

c17.bench (total 14 gates)

number of patterns	CPU times(sec)	maximum memory usage (KB)
100	0.000548	808
1000	0.00367	832
10000	0.038851	844

c7552.bench (total 5994 gates)

number of patterns	CPU times (sec)	maximum memory usage (KB)
100	0.015249	2468
1000	0.053402	2480
10000	0.406807	2492

When the number of patterns becomes bigger, the time take becomes longer, roughly the same as the multiple of number of patterns . The memory usage has only slight difference.

Part.2-b

A straightforward approach to encode `0`, `1` and `x` would be represent `0` as 00, `1` as 11 and `x` as 01. Using this encoding, and/or operations works fine !

However, when we invert `x`, 01 becomes 10. Inverted-`x` should also be unknown, so we represent `x` as 01 or 10 instead.

But new problems pop out ! X AND inverted-X should be still X since the two Xs might come from different places with different actual values. Hence, some modification need to be done to solve this.

The only problem comes from the invert. So I revise the original invert function, which shown as follows.

a	representation of a	a'	representation of a'
0	00	1	11
1	11	0	00
X	01	X	**01**
X	10	X	01

In the begining, all unknows are encoded as 01.

By doing so, and, or, nand, nor works fine without further modification !

Note that there are 3 places to modify in the code, which shown as follows:

- enum VALUE {S0, X, X1, S1 in typeemu.h
  - for input pattern `0`, `1` and `X` and stored as 00, 11 and 01 respectively
- VALUE CIRCUIT::Evaluate(GATEPTR gptr) in sim.cc
  - the original code uses table to comupute tha value; we directly compute the value here
  - x and y: `x & y`
  - x or y: `x | y`
  - not x: `if x != 01 then x else 01`
- void CIRCUIT::PrintIO() in sim.cc
  - convert the encoded output to original encoding
  - 00 -> `0`, 11 -> `1`, either 01 or 10 -> `X`

Build

```
make
./atpg ... (depends on different usage)
```

Reference

https://en.wikipedia.org/wiki/Linear_congruential_generator