Assignment #4: SAT-Based Path-Delay-Fault ATPG

(Due Date: 2022/06/09 23:59)

Introduction:

ATPG (Automatic Test Pattern Generation/Generator) is an electronic design automation method used to find an input sequence (or test vectors/test pattern) that, when applied to a digital circuit, we are able to distinguish between the correct circuit behavior and the faulty circuit behavior caused by defects.

A path-delay-fault ATPG generates a vector pair (v1, v2) to creates a transition at the beginning of the target path P. Then the corresponding transition is propagated through target path P. Different off-input assignments can result in different path sensitization criterion. Figure 1 shows the basic idea of path sensitization.
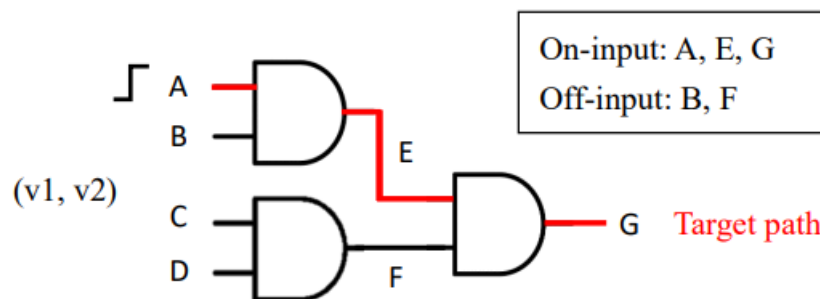

Figure 1: Path Sensitization

An efficient circuit-SAT solver (CSAT) [1][2] is utilized as the fundamental engine of this path-delay-fault ATPG (KF-ATPG).

Objective:

In this assignment, you are asked to write a C++ program to perform Circuit-SAT based ATPG, and generate the test patterns for path-delay-faults.

A Quick Start to KF-ATPG:

(1) *kai_main.cpp*: the main flow is implemented in this part, including input files parsing and creating an instance of a CSAT solver.

(2) *kai_objective.cpp* & *kai_objective.h*: these are the program files you need to finish. The functions *BuildFromPath_R ()* and *BuildFromPath_NR()* will receive a target path pointer, and you need to give appropriate constraint settings for each gate along the path in order to propagate the transition state toward primary output through the path. Also, don't forget to create the transition at the beginning of the target path as well.

(3) *example/*: 2 exemplary test cases are given under this directory, which can be used to test your program.

Implementation Details:

(1) Controlling Values and Non-Controlling Values

A controlling value at a gate input is the value that determines the output value of that gate irrespective of the other input value. The opposite of controlling value is non-controlling value. For example, the controlling value of an and gate is 0; the controlling value of an or gate is 1.

Gate Type	Controlling Value (CV)	Non-Controlling Value (NCV)
AND	0	1
OR	1	0

(2) On-input and off-input

A signal is an on-input of path P if it is on P. A signal is an off-input of path P if it is an input to a gate on P but not an on-input.
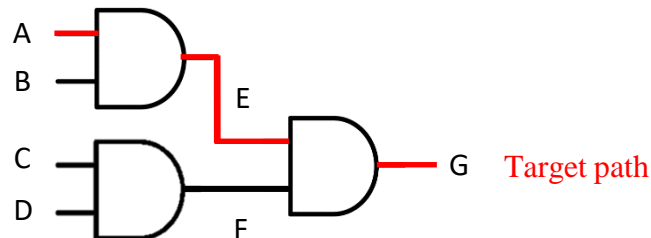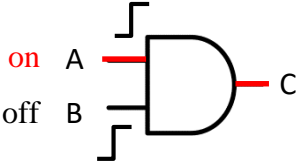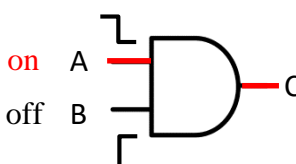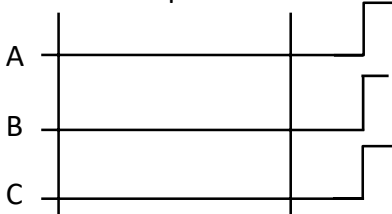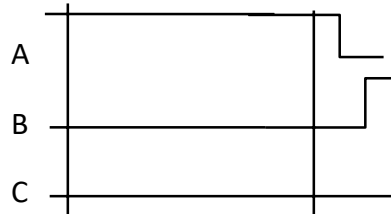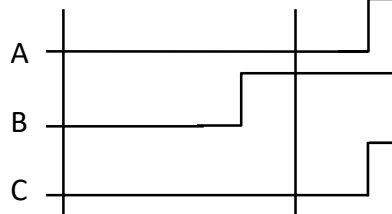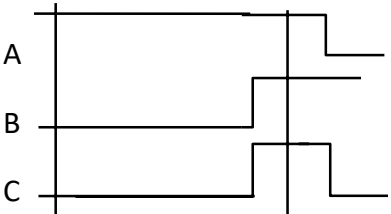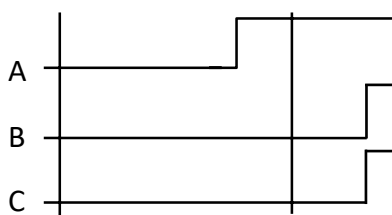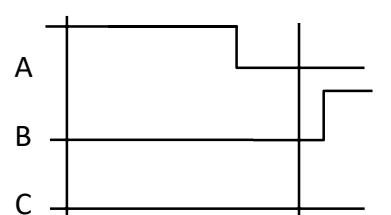

Target path	A→E→G
On-input	A, E, G
Off-input	B, F

(3) Timeframe

Transition Type	Logic value at Timeframe 0	Logic value at Timeframe 1
Rising(0→1)	0	1
Falling(1→0)	1	0

(4)   Robust Test(R) vs. Non-Robust Test(NR)

		Robust Test	Non-Robust Test
Definition		Guarantee to detect the delay fault on the target path. Fault detected or not is independent of all other delays in the circuit.	The detectability of a fault under a non-robust criterion depends on the arrival time of input pins.
Example		on A C off B	on A C off B
Case 1	Both inputs have delay fault	Clock period A B C	Clock period A B C
	Result	Rising transition on A is successfully propagated through C	Falling transition on A is failed to propagate through C
Case 2	Only on-input has delay fault	Clock period A B C	Clock period A B C
	Result	Rising transition on A is successfully propagated through C	Falling transition on A is successfully propagated through C
Case 3	Only off-input has delay fault	Clock period A B C	Clock period A B C
	result	Rising transition on A is successfully propagated through C	Falling transition on A is failed to propagate through C

Summary	Rising transition on A is detectable in every case. **By applying this vector(A:01, B:01), we are running a robust test.**	Falling transition on A is detectable only in case 2. **By applying this vector(A:10, B:01), we are running a non-robust test.**

The definition of robust/non-robust test may look a bit complicated, but here you got a table which sums up all the situations you will encounter. You can easily determine what value to apply on on-input/off-input by looking up the table below.

On-input transition	Off-input assignments	
	Robust(R)	Non-Robust(NR)
cv → ncv	x → ncv	x → ncv
ncv → cv	ncv → ncv	x → ncv

(5) Input files

For each case, you got 2 files as input, which are **xxx.bench** and **xxx.path_not**.

(1)**xxx.bench**: this is a netlist file of a combination circuit.

```
1   # C17.iscas
2   INPUT(g1GAT_0_)
3   INPUT(g2GAT_1_)
4   INPUT(g3GAT_2_)
5   INPUT(g6GAT_3_)
6   INPUT(g7GAT_4_)
7   OUTPUT(g22GAT_10_)
8   OUTPUT(g23GAT_9_)
9   g_169_ = NAND(g1GAT_0_,g3GAT_2_)
10  g_166_ = NAND(g3GAT_2_,g6GAT_3_)
11  g_173_ = NAND(g_166_,g2GAT_1_)
12  g22GAT_10_ = NAND(g_169_,g_173_)
13  g_171_ = NAND(g_166_,g7GAT_4_)
14  g23GAT_9_ = NAND(g_171_,g_173_)
```

(2)**xxx.path_not**: this file indicate the path being targeted. Each path starts with a primary input and ends wih a primary output. The R/F in the front of each path indicates the transition state of the primary input gate. R represents rising and F represents falling.

transition state of the primary input gate

```
3  @ R g6GAT_3_  g_166_  g_173_  g22GAT_10_    →Path 1
4  @ R g3GAT_2_  g_166_  g_171_  g23GAT_9_     →Path 2
5  @ R g6GAT_3_  g_166_  g_173_  g23GAT_9_
6  @ R g6GAT_3_  g_166_  g_171_  g23GAT_9_
7  @ F g6GAT_3_  g_166_  g_173_  g22GAT_10_
8  @ F g3GAT_2_  g_166_  g_173_  g22GAT_10_
9  @ F g6GAT_3_  g_166_  g_173_  g23GAT_9_
10 @ F g3GAT_2_  g_166_  g_173_  g23GAT_9_
11 @ F g6GAT_3_  g_166_  g_171_  g23GAT_9_
12 @ F g3GAT_2_  g_166_  g_171_  g23GAT_9_     →Path 10
```

primary input   primary output

For example, we want to generate the test patterns in order to detect these 10

path delay faults. So we need 10 pairs of input vectors, one for each path.

Language:
C++.

Platform:
You may develop your software on UNIX/Linux.

Usage:
Compile: $ make

Clean up: $ make clean

Execution: $ ./KF_ATPG

-atpg [testing type, R or NR]

-cir example/[testbench file, ex. xxx.bench]

-path_not example/[path file, ex. xxx.path_not]

-output [output patterns generated by ATPG, name it as xxx.pttn]

Exmale: $ ./KF_ATPG -atpg NR -cir example/c17.bench -path_not example/c17.path_not -output c17.pttn

Submission
Please compress all of your source code into a zip file and name it by your ***student ID***. For example, "310510180.zip". Then upload the compressed file to the new E3 website by the deadline (2022/06/09 23:59).

Grading policy:
(1) Example case correctness (60%)
(2) Hidden case correctness (40%)

Delay Penalty:
● Within 1 day: score * 0.8
● Within 2 days: score * 0.7
● Within 3 days: score * 0.6
● More than 3 days: score = 0

Reference:
[1] Feng Lu, Li-C.Wang, Kwang-Ting Cheng, and R. C.-Y. Huang, *A Circuit SAT Solver With Signal Correlation Guilded Learning*. Proc. Design Automation and Test in Europe (DATE), 2003

[2] Feng Lu, L.-C. Wang, K.-T. Cheng, J. Moondanos and Z. Hanna, *A Signal Correlation Guided ATPG Solver and Its Applications for Solving Difficult Industrial Cases*. Proc. Design Automation Conference (DAC), 2003