TRACEFILE

(Understanding and Generation)

The generic testbench used in Quartus for simulation and scan-input.py which generates input file for scan-chain, both makes use of a file which is TRACEFILE.txt. This file contains input test vectors and corresponding expected output test vectors for comparison. There is one more set of value called as mask(mask bits) which is used for output validation. This file is to be written/generated in a particular format which is to be strictly followed else results in errors.

INPUT: set of 0s and 1s, total number of bits should be equal to the total number of input bits mentioned in Testbench/DUT.

OUTPUT: set of 0s and 1s, total number of bits should be equal to the total number of output bits mentioned in Testbench/DUT.

MASK: set of 0s and 1s, total number of bits should be equal to the number of output bits mentioned in Testbench/DUT. The output bit for which the corresponding mask bit is set/written as ‘1’ will only be considered while validating the output. While making the tracefile for combinational circuits, all the mask bits(corresponding to all the output bits) should be made ‘1’.

Tracefile format

<input bits><single space><output bits><single space><mask bits>
<new line>
<input bits><single space><output bits><single space><mask bits>
<new line>

Eg. :

let the number of input bits = 3
let the number of output bits = 2
then, number of mask bits = 2

our TRACEFILE.txt should look like:

000 00 11
001 01 11
010 01 11
011 10 11
100 01 11
101 10 11
110 10 11
111 11 11

One can easily figure out that the above example is for full-adder.

Types of Tracefiles

1. Based on Test check:

Exhaustive: If the number of input bits are less(maybe <=10), then one can go exhaustive testing, it means that the design will be tested for all the possible combination of inputs and the same should be present in the Tracefile.

Selective/Corner cases: If the number of input bits are more(maybe >10), then it is recommended to use this approach to avoid excess test time both for simulation and scan-chain. However while using this approach one has to take care to identify corner/critical cases or else can even go for random generation for input test vectors. Try to make sure that the design gets tested atleast for toggle of every input bit and every output bit.

2. Based on Generation:

Manual: If the number of input bits are less(<=4) or if unfamiliarity with coding then this approach can be used, taking care that the format of tracefile is strictly followed. This approach is not recommended.

Coding: If the number of input bits are more(>4), this approach is to be compulsorily used. Code can be written in C, C++, Python or in any other language you are familiar with. Very basic file operations are required together with very simple logical coding for the algorithm implemented in the design.

Some useful notes: while implementing the design(which is implemented in HDL) in C/Python code for generating the corresponding output for every input in tracefile, try to use different logic/algorithm for implementation so as to avoid any bugs/false/incorrect design in the hardware, using this approach will cross-verify or double check the implementation when implemented using two different approaches.

Code Examples:

Combinational Circuits:

1. Full adder using C

code file (test_gen.c) attached

2. Full adder using Python(2.7)

code file (test_gen.py) attached

Commands:

C

compile: gcc test_gen.c -o test -lm
run: ./test

Python

python test_gen.py

Sequential Circuits:

In case of sequential circuits, things do not go straight forward, first of all, together with the other inputs, a “clock” signal is included. For our design we assume that everything is edge – triggered/sensitive, so the inputs are fetched at rising edge of clock(when clock signal is ‘1’) and then processed in the positive level of clock to generate the output, we take the convention of reading/validating the output at low level of the clock(after negative edge when clock signal is ‘0’), this makes sure that the output is generated and stable(settled) and hence avoid timing violations.

In order to follow the above convention, following changes are to be made/included in the TRACEFILE.txt

1. Input contains a “clock” signal and let it be LSB.
2. Number of input test vectors gets doubled(for writing in TRACEFILE.txt and not for verification)(= 2*(2^number of input bits(excluding the “clock” signal)))
3. Output is validated(MASK bit set as ‘1’ for output) only when clock is ‘0’
4. The input and output bits are to be kept same for two consecutive vectors while changing only “clock” and “MASK” bits

Eg. :

let the number of input bits = 3(2 input and 1 clock)
let the number of output bits = 1
then, number of mask bits = 1
and number of test vectors = 8

our TRACEFILE.txt should look like(some random I/O are considered):

001 1 0
000 1 1
101 0 0
100 0 1
111 0 0

110 0 1
011 1 0
010 1 1

Try to modify the codes provided for combinational circuits to generate TRACEFILE.txt for sequential circuits.