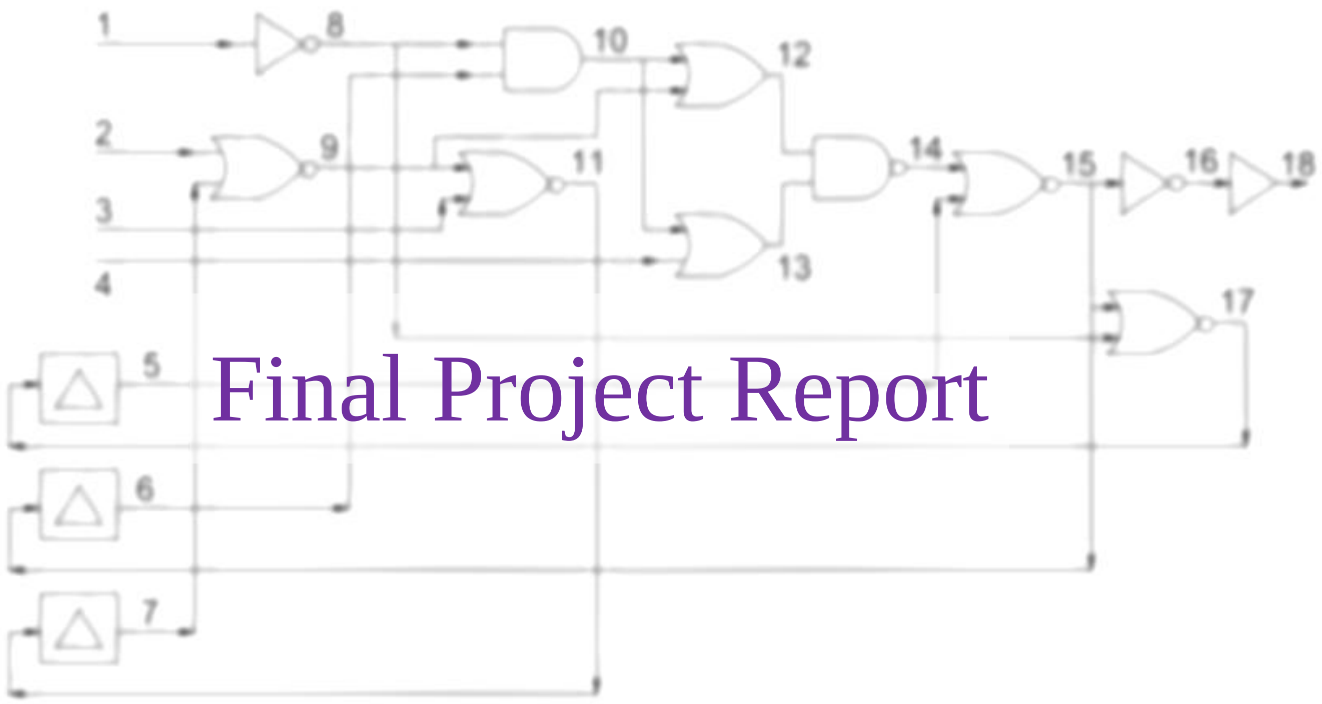
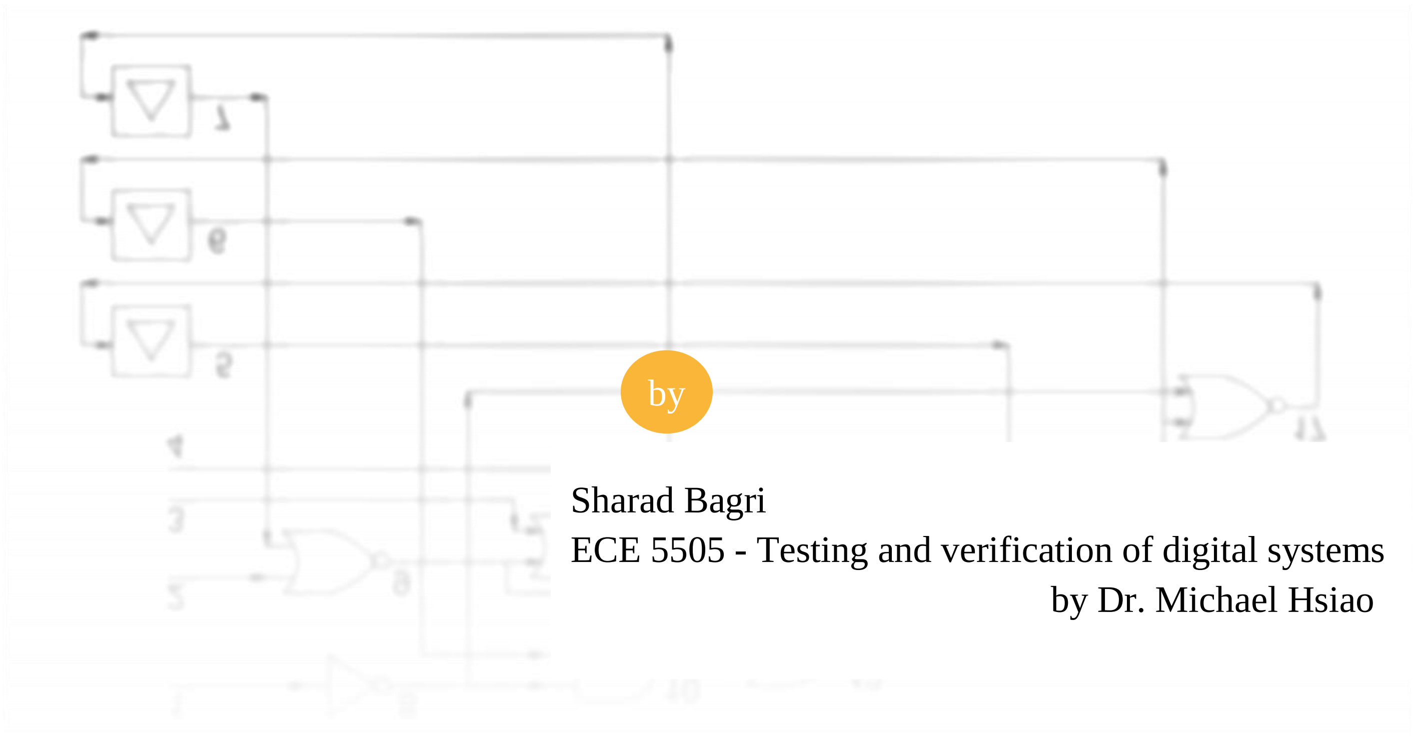


Final Project Report

Sequential circuit simulator with faults



by

Sharad Bagri

ECE 5505 - Testing and verification of digital systems

by Dr. Michael Hsiao

1. INTRODUCTION

Circuits are becoming increasing complex as more and more transistors fit onto a chip. Also, minimum quantity of chips to be fabricated is going up. This results in huge loss if faults in design find their way to tape-out without being detected during design. These undetected faults can cause loss of millions of dollars or more and sometimes challenge the survival of the company itself. Due to such wide reaching effects, Chip design engineers make sure that none of the faults find their way to actual hardware. This requires simulation of faults beforehand and verifying the design correctness. Such scenarios require building fault simulator.

Sequential circuits are most common types of electronic circuits used in real world. Output of sequential circuits depends on it present input as well as past input values which gets saved in flip flops after being operated on by the combinational part of the circuit. They are like circuits with memory of its previous inputs. In terms of functionality they can be divided in synchronous and asynchronous category. In synchronous sequential circuits gate values are updated only at one point during a clock cycle which is fixed and same for each cycle. There are different kinds of flip flop in real use like SR, JK, D etc. flip flop. D flip flop is can be taken as representative of different kinds of flip flops in sequential circuits.

This project discusses about building a fault simulator for sequential circuits. Complexity in building simulator for sequential circuits is orders of magnitude higher than combinational circuits as output of present inputs depends on current inputs and previous inputs whose values gets saved in flip flops after some combinational operations. Additionally, in case of sequential circuits input consists of sequence of vector and not just one vector as in combinational circuit.

Further, number of vectors in a sequence can be of any number. In today's big and complex designs these numbers may run into high values of thousands or even more.

Each circuit can have various kinds of faults. There are numerous sources through which faults can creep into a design. It can be due to variation in manufacturing process parameters, random variations in inputs, deep submicron effects etc. Faults can be of different type like stuck-at fault, delay fault, bridging fault etc. Stuck-at faults are the most widely reported faults in real circuits. Hence, a greater emphasis is placed on simulation of them.

2. PROJECT SPECIFIC BACKGROUND

A fault simulator is expected to simulate all kinds of faults but that is very difficult to do because

A. Some faults are hard to simulate
B. Some faults get covered by other faults, like stuck-at faults can cover bridging faults.

Considering these, this design starts with a basic fault simulator. It simulates single stuck-at fault for synchronous sequential circuits. In the vast array of flip flops available, this simulator simulates only D flip flop. It is expected that design working for D flip flop can easily be modified to run for other kinds of flip flop as well.

ISCAS89 circuits are benchmark circuits widely used by research community for benchmarking and publishing their digital logic circuit simulations or algorithms. It is also useful to benchmark on those circuits as basic result of simulations like fault detected, output of the circuit etc. are easily available. These can be very helpful while the initial code is written where errors are expected to happen. These serve as reference to check the outputs and possibly gain an insight of what could have gone wrong which helps a lot in debugging.

For each faulty circuit simulations following files are required

1) .lev file: It is leveled representation of the actual circuit. Filename starting with s represent sequential circuit while those with c represent combinational circuit.
2) .vec file: It contains vectors to be simulated on the sequential circuit.
3) .eqf file: It contains list of files in circuit where each line separated by ':' represent list of equivalence collapsed faults.

After fault simulation, following files are generated

1) .out file: It contains input, output, flip flop values of the circuit in fault free case.
2) .det file: It contains list of fault which can be detected by the vector in .vec file.
3) .ufl file: It contains list of faults which cannot be detected by the vector in .vec file.

Various types of fault simulation methods are present. Simplest simulator evaluates each gate per cycle. Another kind of simulator is event simulator which simulates a particular gate only when inputs to it change. It maintains an event queue in order to keep track of which gates are to be simulated. This kind of simulator especially works very well for big circuits where simulating each gate irrespective of change in its inputs would be waste of time and computational resources. Simulator designed in this project uses event driven simulator as its simulation engine. Event simulators are tougher to write compared to simplest simulator because it needs to check for conditions on inputs of gates, manage and update event queue. In this design values from previous simulation of one cycle is saved in an array. This facilitates finding which gates are to be simulated at the start of a cycle. It checks for difference in previous and new value of the gate and puts it in event queue only if there's a difference. Mostly Inputs and flip flop are placed in event queue at the beginning of a cycle.

This simulator distinguishes between fault on stem and branch. It simulates both of them in separate fashion. So, a fault on branch doesn't get propagated to other branch while a fault on stem gets propagated to each branch of it.

Three value logic simulation is done where 1 and 0 represent themselves while -1 is used to represent unknown value or 'X'

3. PSEUDO CODE

```
Read circuit file, read faults
Initialize all global variables, create output files
Do fault free simulation
For each vector
    Reset event queue, gate values, update flip flop values from previous run
    Simulate full circuit through event driven simulator
    Save fault free values
Do faulty simulation
For each Fault
    For each vector
    Reset event queue, gate values, update flip flop values from previous run
    Simulate full circuit through event driven simulator
        Faulty gates are simulated separately depending on fault.
    Compare fault free values with faulty values
        If different write to .det file and continue to next fault
All vectors exhausted and fault not detected => write fault to ufl file and continue to next fault.
All faults simulated?
exit
```

4. MODULE DESCRIPTION

This section gives details of sections which are non-trivial and critical to the working of this code

All functions, variables can be broadly divided into three categories. Below is a list of each category and important functions in it.

1) Circuit & file handling related: `circuit()`, `PrintInput_outs()`, `PrintOutput_outs()`, `ReadEqfFile()`
2) Vector related: `ReadVecFile()`, `UpdatePrevNew()`, `SaveFFVal()`, `UpdateFFs()`, `**Vects`, `**FltFreeOpVctrs`, `**FltyOpVctrs`
3) Simulation related: `CmprFltyOps()`, `UpdateEventList()`, `SimOneCyc()`, `SimFaultFree()`, `SimFaulty()`.

4.1 BRIEF DESCRIPTION OF EACH MODULE

1) Circuit & file handling related
   a) `circuit()` – This is constructor of main and lone class of the program. It reads all circuit details from .lev file and update it into member variables of class. It also counts total number of flip flops, inputs, outputs. If reading is successful it creates file handler for .out, .ufl and .det files. In order not to cause confusion of these files from reference files taken from benchmarking sites, output file are created in following format <circuit Number>_S.<Extension of out or ufl or det>. Example: s27_S.out.
   b) `PrintInput_outs()` – These function writes current input vector being applied on the input of the circuit to .out file.
   c) `ReadEqfFile()` – This function reads equivalence collapsed faults. Only first fault in each line is read. It is because if simulator can detect all such faults then there's a very

high probability that it would detect others as well. This function saves fault value in 2D array of `‘FltVct’` where 1st element of each row is faulty gate number. 2nd element tells whether the fault is on input or output and the location of fault. 3rd element specifies the type of fault, whether its stuck at 1 or 0.

2) Vector related:

   a) `ReadVecFile()`: It reads sequence of vectors from *.vec file. First line of .vec file is expected to be number of bits per vector. From next line onwards till finding the keyword “END”, all vectors are read as vectors of same sequence. These values are saved in 2D array `‘Vects’`. This array is allocated dynamically for efficient use of memory as well as helping it in scaling up for bigger applications.

   b) `UpdatePrevNew()`: It updates value of simulation in previous cycle to that in new cycle. This is useful to update event queue at the beginning of simulation.

   c) `SaveFFVal(),UpdateFFs()`: `SaveFFVal()` saves value of flip flop at the end of cycle to carry it to next cycle. `UpdateFFs()` updates the value of flip flop at the beginning of a cycle. This is necessary because at end of each simulation, simulation values are cleared to avoid previous simulation values being used for next cycle. As soon as update is done, local value of flip flop in `UpdateFFs()` is resetted. This is useful because in case of faulty ciruits with flip flop having the fault, these values are updated without even simulation.

   d) `**FltFreeOpVctrs, **FltyOpVctrs`: These 2D vectors saves circuit’s output values in fault free and faulty case respectively. In case of sequential circuit output is produced as sequence of vectors, hence these are 2D array. Each row represents output of one vector.

3) Simulation related:

   a) `CmprFltyOps()`: It compares value of fault free circuit with faulty circuit and if a difference in found in any vector of the sequence. Fault is flagged as detected and simulation for it is terminated.

   b) `UpdateEventList()`: This updates event queue of the simulator. Simulation of any gate calls this function. It adds fanout gates of that gate to the event queue only if the simulated gate's value changes and is changed to either 0 or 1.

   c) `SimOneCyc()`: It handles complete simulation related activity for one full cycle of the circuit. This is the main function called for simulation.

   d) `SimFaultFree(), SimFaulty()`: Both these functions are called by `SimOneCyc()`. `SimFaultFree()` simulates fault free gates. `SimFaulty()` is called for simulating gates with faults. It first checks if fault is on output or input of gate. If fault is on output, gate value is updated to faulty value and simulation is moved to next gate without simulating the gate. If any of the input is faulty, simulation of that gate is done with only that input being updated to faulty value.

RESULTS

The simulator was used to simulate ISCAS89 benchmark circuits of s27.lev, s298.lev, s344.lev, s382.lev, s400.lev, s444.lev, s382.lev, s5378.lev. All except S5378 was simulated on linux Ubuntu OS running on laptop having Intel i5-3210 processor @ 2.5 GHz with 8.00 GB RAM.

Development was done on windows Visual studio 2010 Professional edition. S5378 was tested with debug code and it took around 6 hours on it.

Final run was done on linux after compiling the program with optimization enabled. Command used to build the program is: `g++ -O3 Main.cpp`

Here are the results

Circuit Name	Simulation Time (s)	.out file matching	.det file matching	.ufl file matching
S27	0.003	100%	100%	100%
S298	0.116	100%	100%	100%
S344	0.057	100%	100%	100%
S382	0.739	100%	100%	100%
S400	1.320	Not available(NA)	100%	100%
S444	1.226	NA	100%	100%
S832	6.251	NA	100%	100%
S5378*	6+ hours	NA	100%	100%

*: Simulation was done on windows with unoptimized code

- All Time reported are 'real' time taken for the program to run with time parameter
- For bigger circuits reference .out file wasn't available and so the matching was not done.

IMPORANT LEARNINGS

- Only gate values (1, 0, -1) were created of type int, everything else like Gate Number, No. Of faults, fanin list, number of faults was unsigned int. If there was any place where data values was being compared to or assigned to other values like gate number etc. then a warning was flagged while compiling. This helped a lot in catching potential problems.

- All functions, global variables are put in one class. Previous designs of mine had different classes for simulation, circuit & vectors. In some cases while working on a particular class I had to pass the class as argument. Putting it all in one class saved lot of such transfers.
- Code was written with considerations of debugging. So, functions were made to print different parameters or data. While debugging it was very helpful to put those prints wherever it was needed.
- Macros were defined for different levels of debugging. One macro (`#define DEBUG`) printed all simulation related values like event queue, simulation of each gate, its input and output values, fanout gates scheduled on event queue after each gate's simulation. Other macros printed values specific to fault free and faulty circuit.

CONCLUSIONS AND FUTURE WORK

Building sequential fault simulator was a great learning experience. It helped understand concepts taught on sequential circuits better. It also helped to get a real understanding of complexity involved in building any algorithm on sequential circuits. An increase in no. of circuit elements or number of vectors in each sequence increases simulation time for each pass which cumulates into overall simulation time increasing a lot.

Sequential circuit simulator with faults forms basis of any algorithm related to sequential circuits. The design of this simulator is such that it can easily be enhanced to simulate multiple stuck at faults in circuit. It can also be integrated to find ATPG vectors using Genetic Algorithm. Many other kinds of algorithms or ideas on sequential circuit fault simulator can use this as simulation engine.