

A Robust Logic Simulator Using Dynamic Levelization



Kasi L. K. Anbumony

Dept. of Electrical Engineering
Auburn University, AL - 36849 USA

Motivation for This Work

- **Logic simulator to verify combinational circuits in bench circuit description language.**
- **Logic simulator to handle unlevelized netlists.**
- **Logic simulator to handle hierarchical netlists.**
- **Using Logic simulator as Fault simulator with the help of combinational ATPG & AUSIM.**
- **Gaining popularity to simulate, verify the functionality and diagnose design faults in netlists.**

Outline

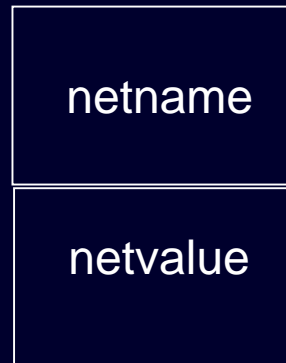
- **Definition**
- **Data structure & Algorithm**
- **Complexity**
- **Dynamic Levelization**
- **Results for Logic Simulator**
- **Diagnosis**
- **Fault Dictionary**
- **Algorithm**
- **Atalanta & AUSIM**
- **Simulation & Discussion**
- **Conclusions**

Logic Simulator

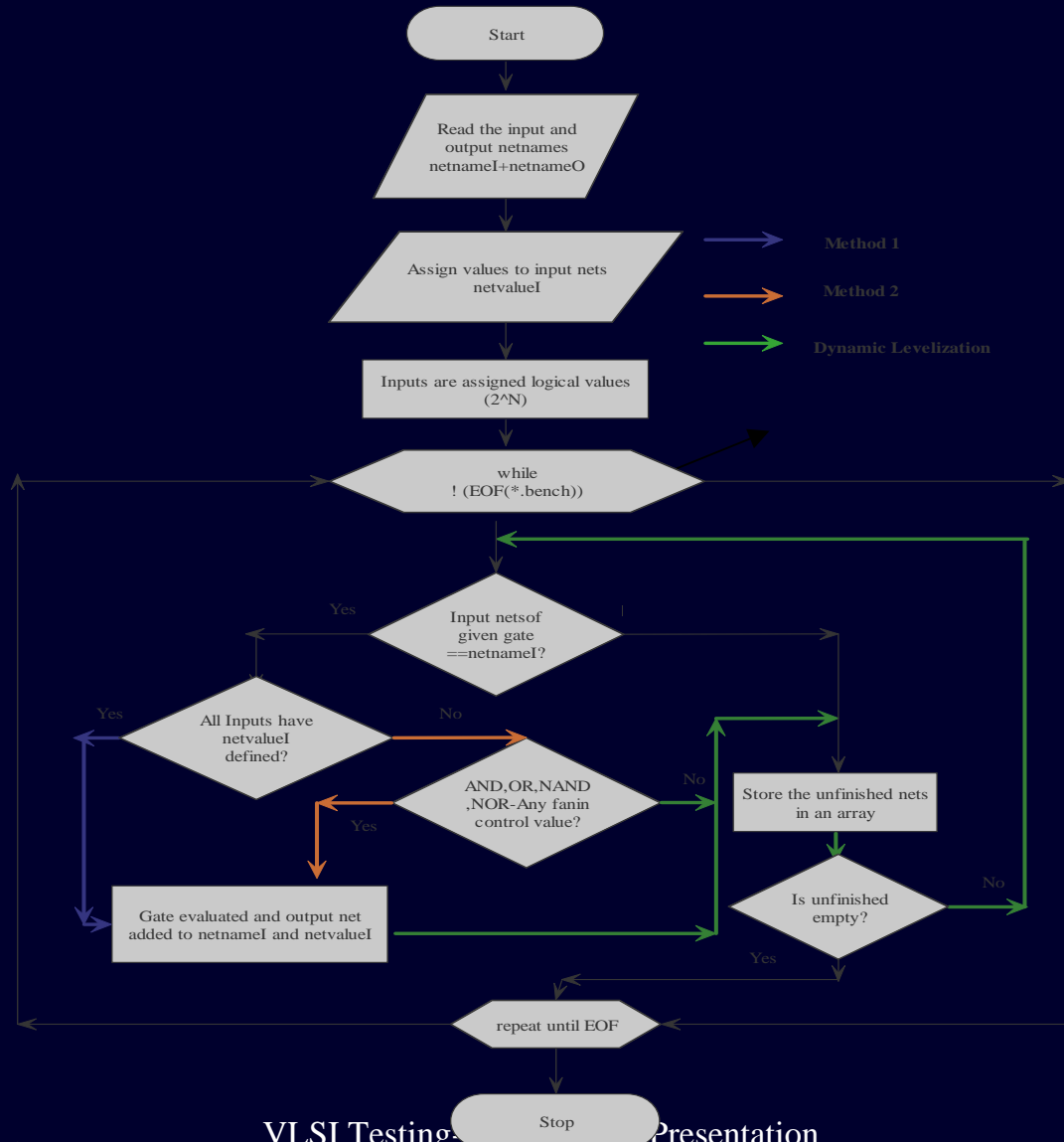
- True value simulator that computes the responses for a given stimuli
- Used for design verification
- Supports AND, OR, NOT, BUFF, NAND, NOR, XOR, XNOR for any number of fan-ins
- Two methods have been tried to optimize the CPU time

Data structure: *Used*

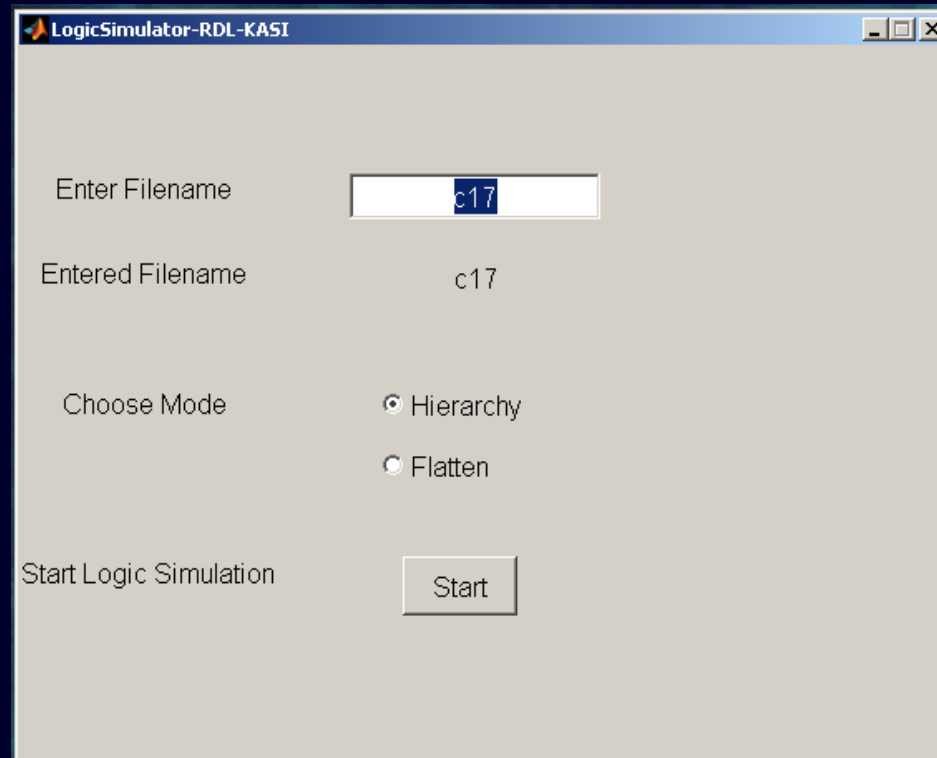
Dynamically growing array with each net having two properties: name and their logic value



Algorithm in detail: *Flowchart*



Logic Simulator: *GUI*



Complexity: *Levelized Circuits*

- Method 1: If the netlist is levelized with n gates and an average fanin of each gate is k , then complexity is given by $O(kn)$
- Method 2: If one of the fanin's of a logic gate is a control value (i.e.) for an AND gate it is 0, then we need not wait for other fanin's for that gate to get stabilized. Saving by a factor of say "k1". Other gates: OR,NOR,NAND.

$$\begin{aligned} &= O_{nand}\left(\frac{kn_1}{k1}\right) + O_{and}\left(\frac{kn_2}{k1}\right) + O_{nor}\left(\frac{kn_3}{k1}\right) + O_{or}\left(\frac{kn_4}{k1}\right) + O_{xor}(kn_5) \\ &+ O_{xnor}(kn_6) + O_{buff}(n_7) + O_{not}(n_8) \end{aligned}$$

Note : $n = n_1 + n_2 + n_3 + n_4 + n_5 + n_6 + n_7 + n_8$

Word on computational savings

A	B	Y=A.B
0	0	0
0	1	0
1	0	0
1	1	1

- 3 out of 4 case, I can apply Method 2 and can get some saving. So I can multiply a weight of $\frac{3}{4}$ for a saving of

$$O_{and}\left(\frac{kn_2(0.75)}{k1}\right)$$

- While remaining $\frac{1}{4}$ will have a complexity of $O_{and}(kn_2(0.25))$

Dynamic Levelization: *Complexity*

$$= O_1(k) + O_6(k) + O_5(k) + O_4(k) + O_3(k) + O_2(k) + M \cdot (O_6(k) + O_5(k) + O_4(k) + O_3(k))$$

- Where M is the number of traversals on the unsolved nodes to get the final response.
- On the right is the worst case depiction with $M = (O_6(k) + (O_6(k) + O_5(k)) + (O_6(k) + O_5(k) + O_4(k))) + (O_6(k) + O_5(k) + O_4(k) + O_3(k)) + (O_6(k) + O_5(k) + O_4(k)) + (O_6(k) + O_5(k)) + (O_6(k))$
 $= 7O_6(k) + 5O_5(k) + 3O_4(k) + 1O_3(k)$

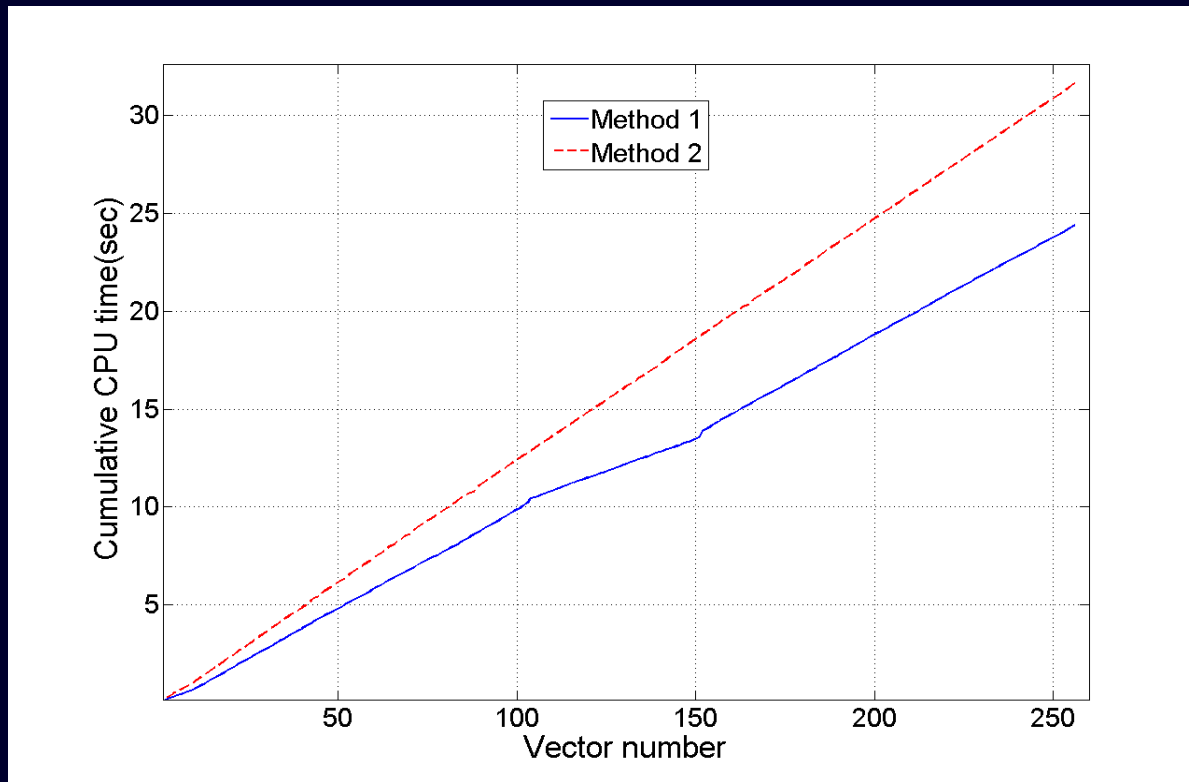
Level 1 node
 Level 6 node
 Level 5 node
 Level 4 node
 Level 3 node
 Level 2 node

CPU time vs. number of vectors:

4-bit adder circuit

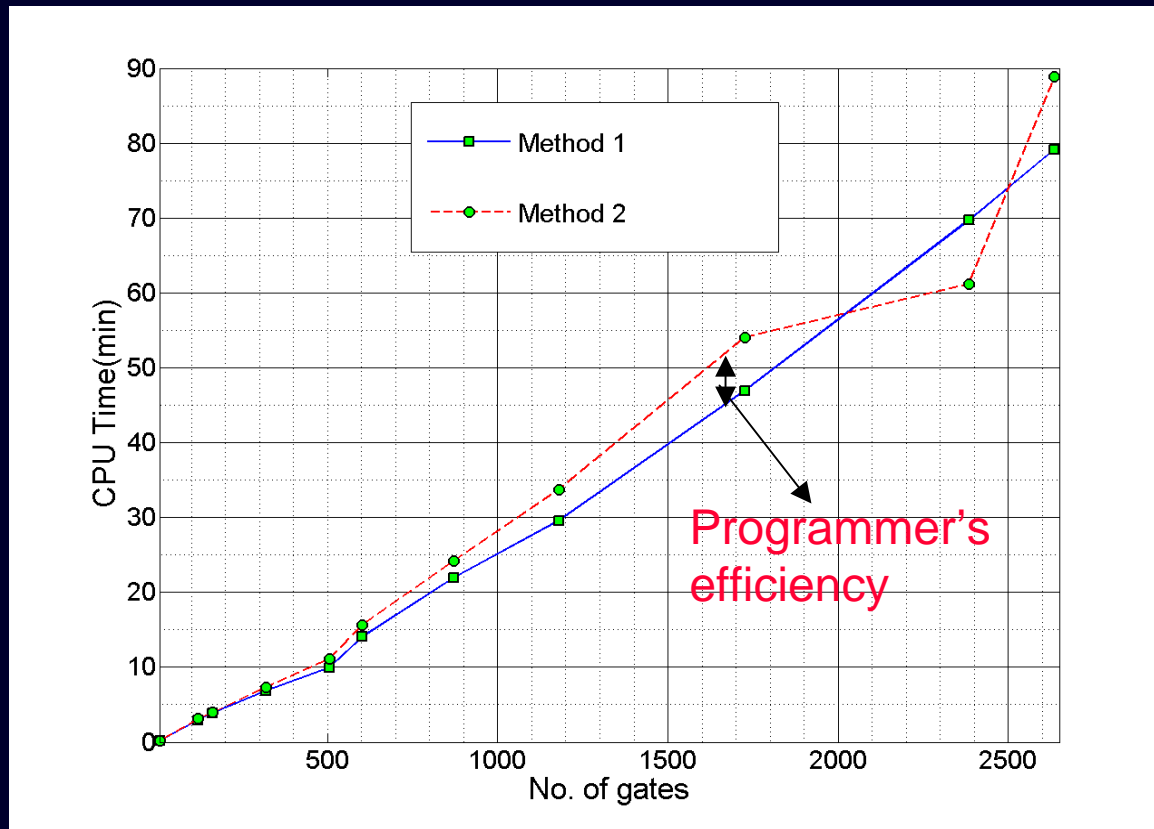
MATLAB Version: 7.0.4.365 (R14) Service Pack 2

Machine: Intel® Celeron™ CPU 1.19 GHz with 384 MB RAM



CPU time vs. number of gates: *ISCAS'85* circuits

Simulated for 1000 random vectors



Diagnosis: *Primitive Stage*



Diagnosis: *Fault Dictionary*

Fault	Test syndrome			
	t_1	t_2	t_3	t_4
No fault	0	0	0	0
a_0, b_0, d_0	0	0	0	1
a_1	1	0	0	0
b_1	0	0	1	0
c_0	0	1	0	0
c_1, d_1, e_1	1	0	1	0
e_0	0	1	0	1

a_0 : Line a stuck-at-0

$t_i = 0$, if T_i passes
 $= 1$, if T_i fails

Algorithm

- Creation of fault dictionary for the specific circuit exhaustively for all stuck-at faults and bridge faults
- Compare the good circuit and faulty circuit response to generate a signature for all test vectors listed in “fault dictionary”

netname_good[] compare with netname_fault[]
netvalue_good[] compare with netvalue_fault[]

- Compare the signature with the signatures in the fault dictionary
- Any match, then higher probability of fault existing of that type or other faults close to that nets

Algorithm-Continued

- Update the fault list dictionary, if any new faults found for the same signature
- Use minimum hamming distance to find a closer fault, if no signature match and verify those nets for possible faults

Stuck-at Fault Generator: *ATALANTA*

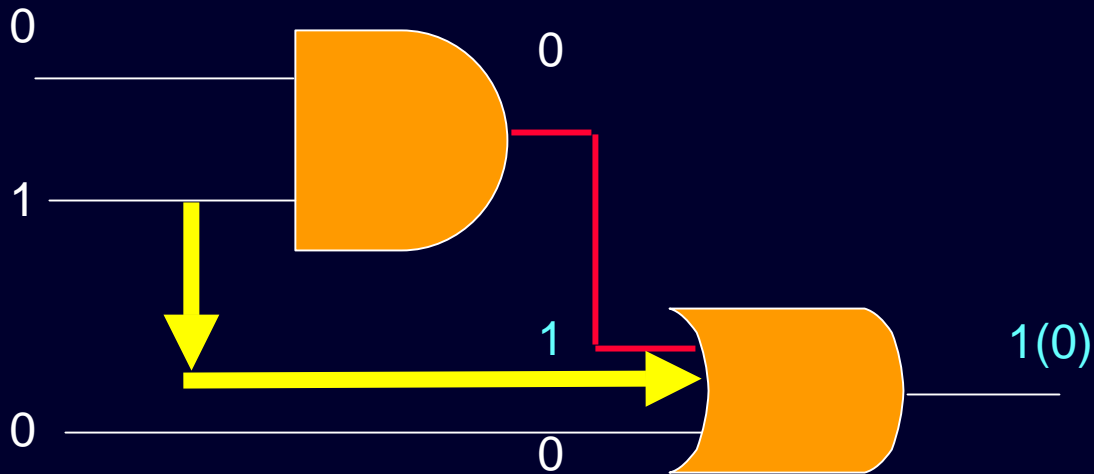
- Use `atalanta` in diagnostic mode
- Command: `atalanta -D 1 *.bench`
- Check `*.test` for the stuck-at fault and the vector detecting it

Bridge Fault Simulator: *AUSIM*

- Generate random vector patterns and store it *.vec
- Convert bench to asl file format
- Generate *.lib and *.ctrl file
- *.ctrl file contents
- ausim *.ctrl
- Check *.det file

```
default <filename>
proc
audit
simul8
bftgen
bftsim
```

Backtracing approach: *Wrong net connection*



Complexity of Diagnosis

- (1) Complexity of Atalanta $O(n^2)$
- (2) Complexity of AUSIM $O(n^2)$ [1]
- (3) Complexity of backtracing is similar to PODEM algorithm $O(n^2)$

[1] Bridge Fault Simulation Strategies for CMOS Integrated Circuits,
Brian Chess Tracy Larrabee,
Computer Engineering Board of Studies University of California, Santa Cruz

Simulation & Discussion

Conclusions

- Thus a robust Logic Simulator is designed.
- Method 2 is able to optimize the time of simulation further.
- Capability to handle hierarchical nodes, with suitable definitions placed in the library.
- Algorithm is simple and can be implemented in any low-level language.
- Primitive model of Diagnosis program is coded and simulation results discussed.

References

- **M. Nemani and F. Najm, '*Towards A High-Level Power Estimation Capability*', in IEEE Trans. on Computer-Aided Design of Integrated Circuits, vol. 15, No. 6, June 1996.**
- **S. Even. Graph Algorithms. Computer Science Press, Rockville, MD, 1979.**

Thank You . . .