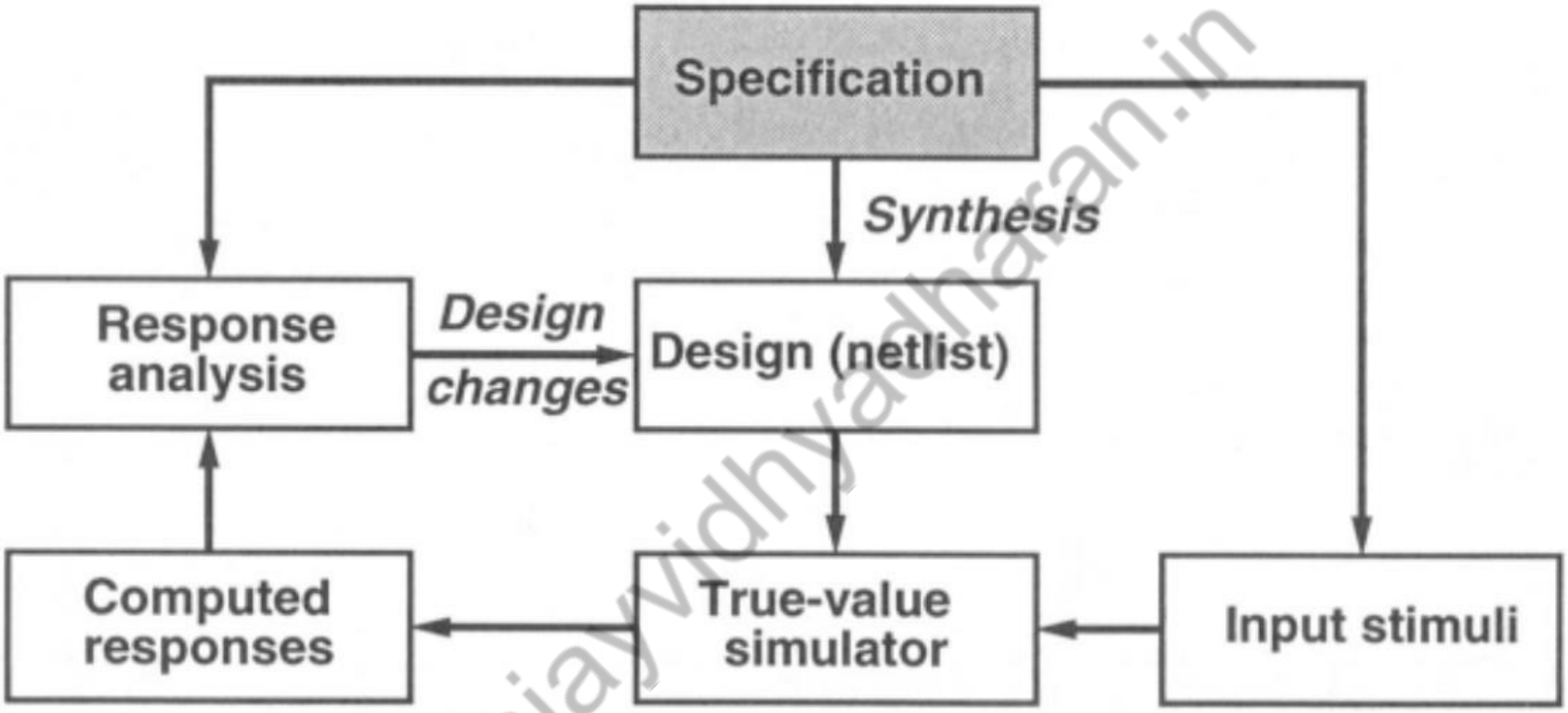
Testability of VLSI

Lecture 4: Logic Simulation

By Dr. Sanjay Vidhyadharan

Simulation for Design Verification



True-value means that the simulator will compute the response for given input stimuli without injecting any faults in the design. The input stimuli are also based on the specification.

A frequently used strategy is to exercise all functions with only *critical* data patterns. This is because the simulation of the exhaustive set of data patterns can be too expensive

Simulation for Design Verification

1. Why learn Design verification now?

2. Many concepts of verification like event driven simulation etc. representation of unknown as X etc. is used in Fault simulation algorithms also.

True Value Simulation

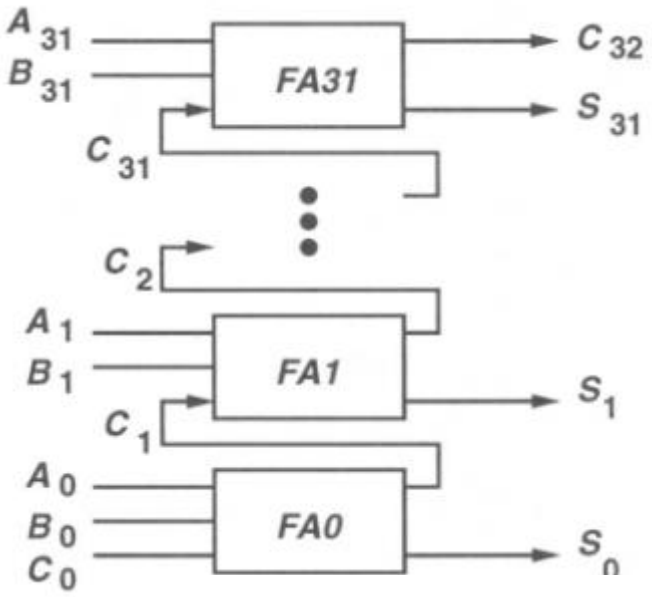
1. A design can be first simulated at a higher behavior level (such as C).
   Netlist not required
   Does not contain the detailed timing information.
   No electrical behavior
2, Once this design is verified, higher-level blocks are replaced by logic-level netlists.
   At this point, a **logic simulator** is used for verification.
3. The process may be repeated by replacing some or all portions by transistor-level or circuit-level implementations.

Simulation is used in this way for verifying very large electronic systems.

The weakness of this method is its dependence on the designer's heuristics used in generating the input stimuli.

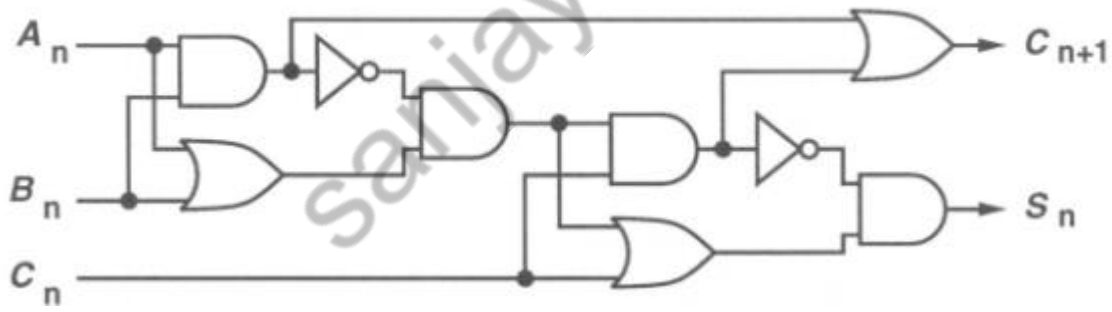
Simulation for Design Verification

Logic design of a 32-bit ripple-carry adder.



How Many Test vectors Required?

Vector no.	Bits: $C_0A_0B_0A_1B_1A_2B_2A_3B_3$	Input $C_nA_nB_n$ to FAn
1	000000000	000 applied to all FAs
2	001010101	001 applied to all FAs
3	010101010	010 applied to all FAs
4	011001100	011 applied to FA0, FA2 & 100 applied to FA1, FA3
5	100110011	100 applied to FA0, FA2 & 011 applied to FA1, FA3
6	101010101	101 applied to all FAs
7	110101010	110 applied to all FAs
8	111111111	111 applied to all FAs

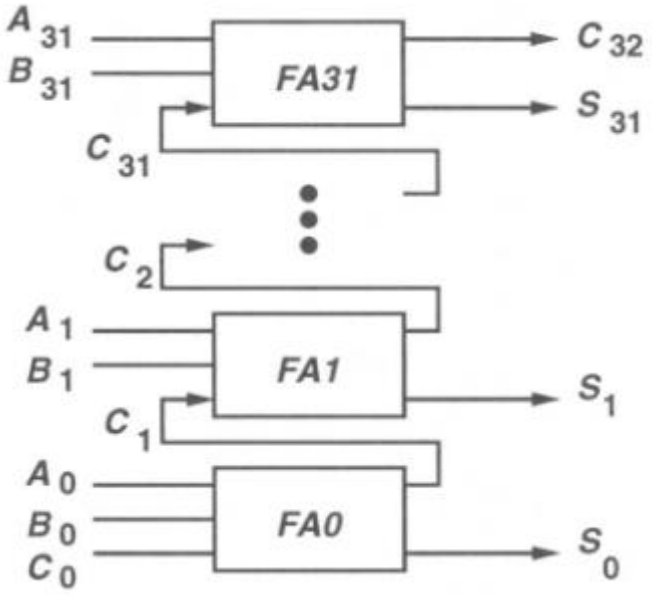


The first seven vectors cover all stuck-at faults. One may, therefore, use only the first seven vectors in the manufacturing test.

Note: This optimization is possible because of same blocks (FA) being used and each test vector verifying similar faults in all blocks

Simulation for Design Verification

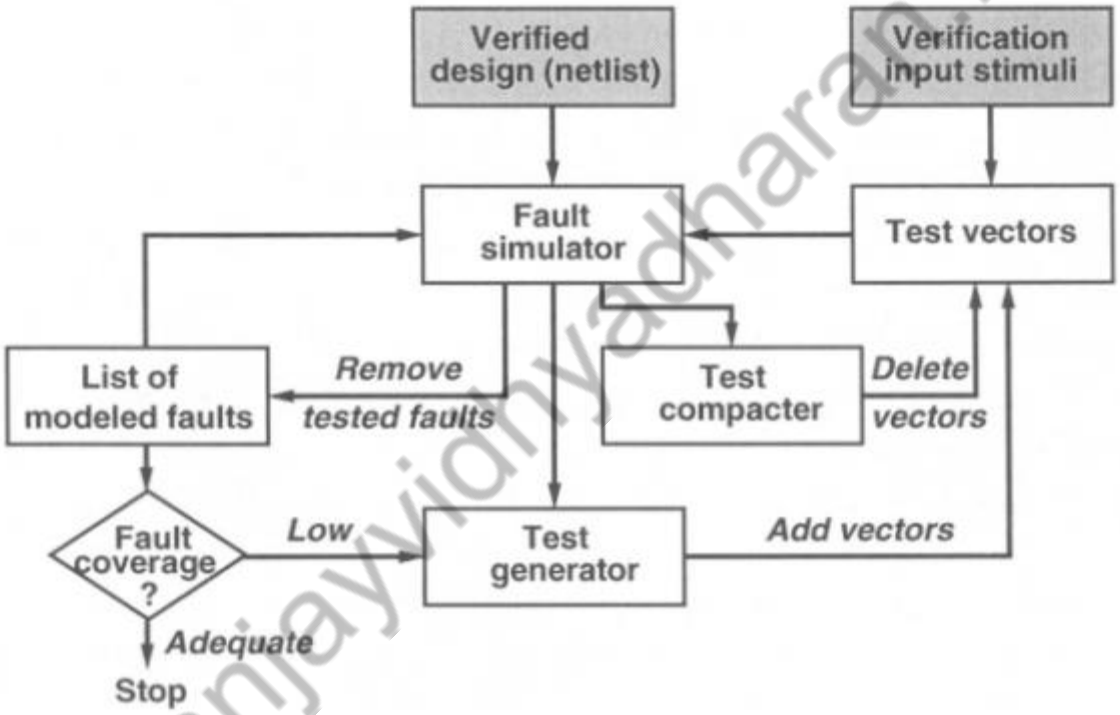
Logic design of a 32-bit ripple-carry adder.



Vector no.	Bits: $C_0A_0B_0A_1B_1A_2B_2A_3B_3$	Input $C_nA_nB_n$ to FAn
1	000000000	000 applied to all FAs
2	001010101	001 applied to all FAs
3	010101010	010 applied to all FAs
4	011001100	011 applied to FA0, FA2 & 100 applied to FA1, FA3
5	100110011	100 applied to FA0, FA2 & 011 applied to FA1, FA3
6	101010101	101 applied to all FAs
7	110101010	110 applied to all FAs
8	111111111	111 applied to all FAs

Timing analysis of 2 followed by 6 or 3 followed by 7 where carry propagates through the chain . Only Possible for Modular Structure

Fault simulation for test generation



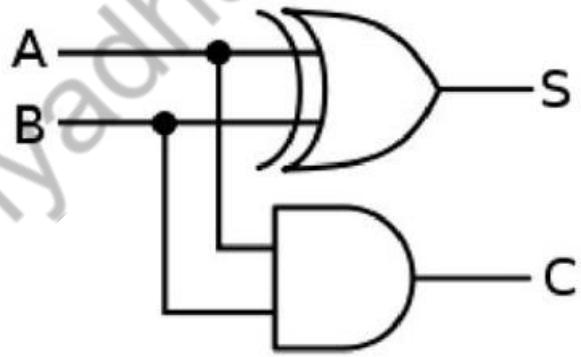
Modeling Circuits for Simulation

1. Function or Behaviour Level

1. Half Adder with dataflow modelling

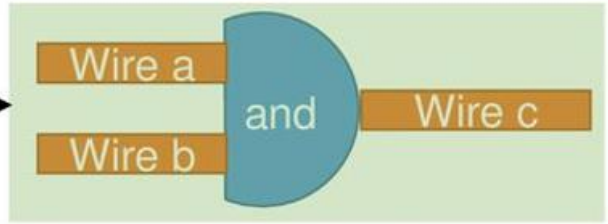
```
module ha(
    input a, b,
    output c, s
     );

assign s = a ^ b;
assign c = a & b;
endmodule
```



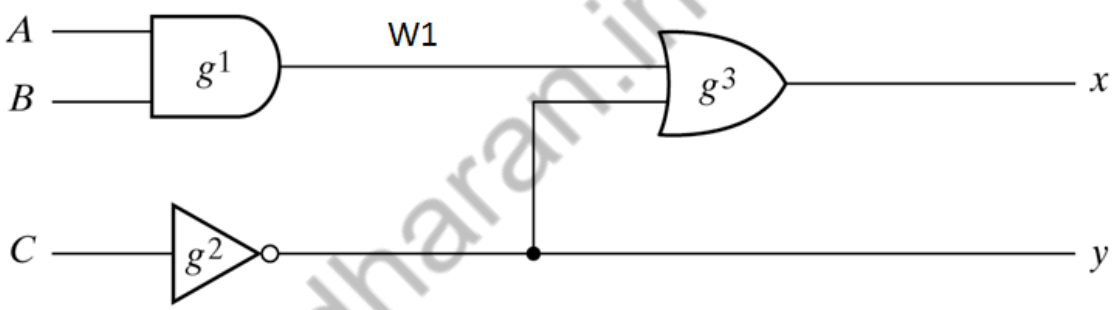
xor

and



Modeling Circuits for Simulation

2. Logic Level



Gate Level Modelling

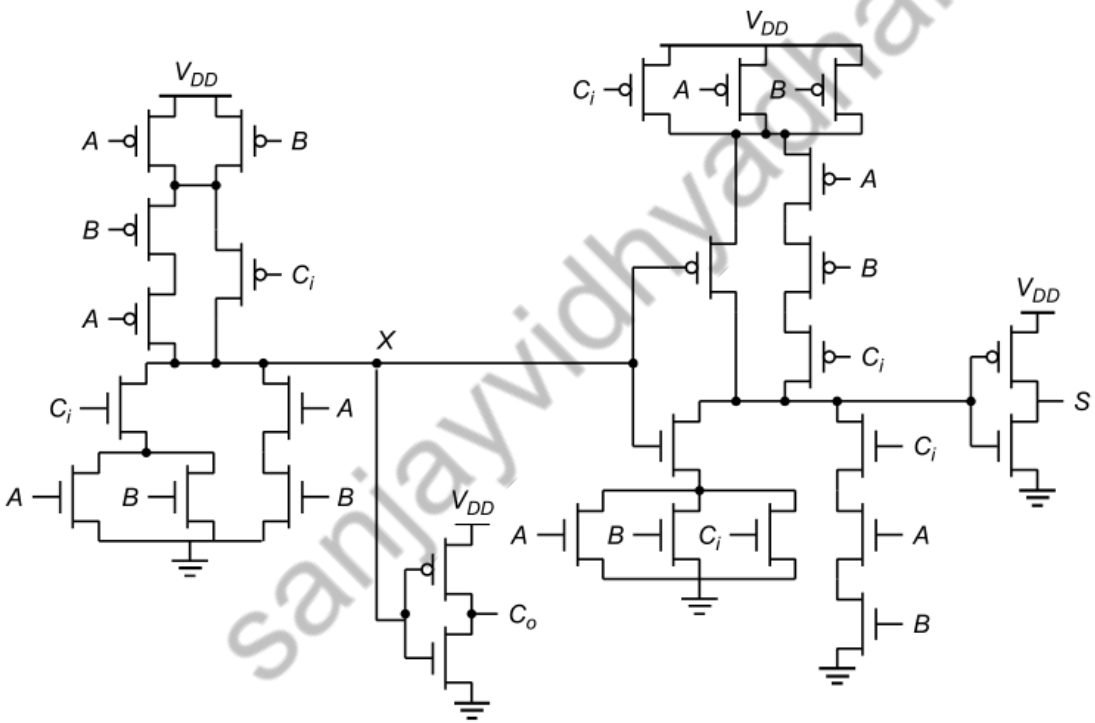
```
module     Simple_circuit (input A, input B, input C,
output x, output y);
wire       w1;

and        g1 (w1,A,B); // and gate instance
not        g2 (y,C);
or         g3 (x,w1,y);

endmodule
```

Modeling Circuits for Simulation

3. Switch Level MOS transistors, which are treated as ideal switches



Modeling Circuits for Simulation

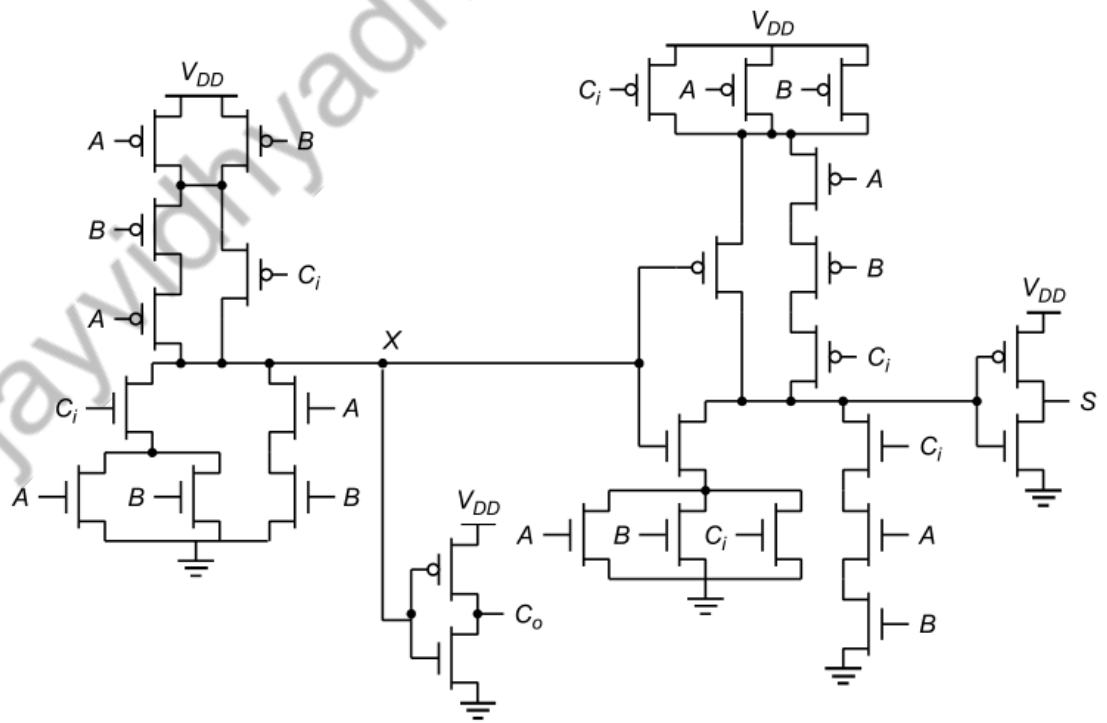
4. Circuit Level

This is the lowest level and represents the ultimate in accuracy for the simulation of electronic systems. The circuit is assumed to be composed of electrical elements such as resistors, capacitors, inductors, and transistors. Equations relating branch or loop currents and node voltages are developed and solved by numerical methods

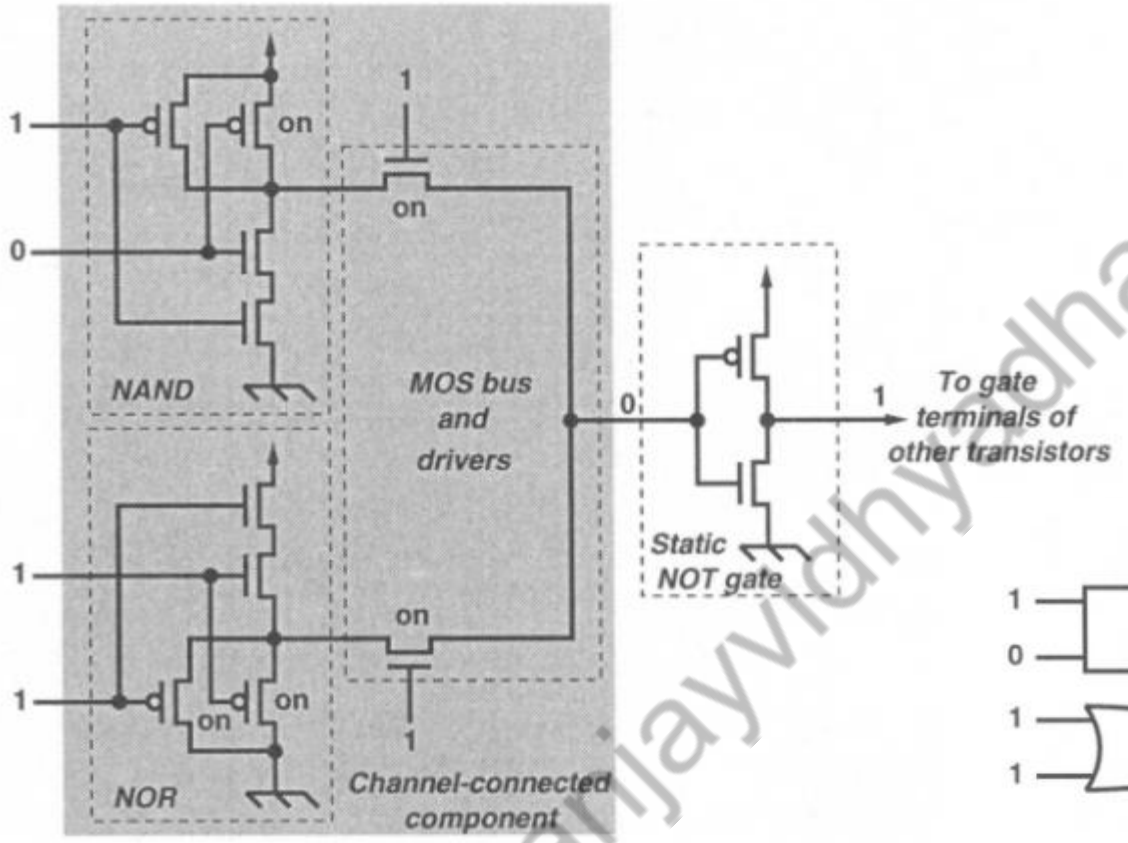
Modeling Circuits for Simulation

5. Timing Level That is, the connectivity of transistors, their sizes and types, and node capacitances are needed. In addition, technology data specifying the transistor voltage-current characteristics are also used to compute charging or discharging currents for the nodes.

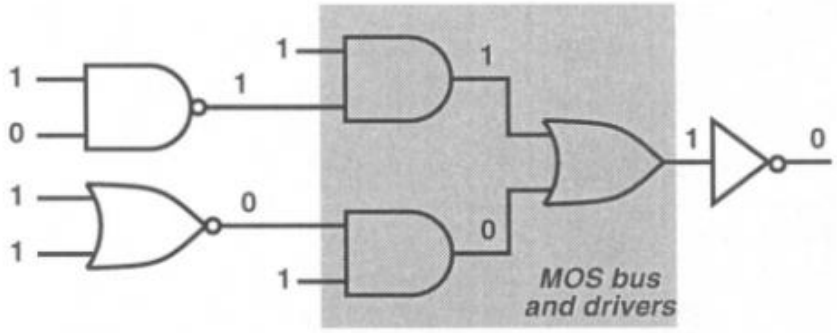
Transistor level Modelling



Why Circuit Level Modeling Is Important



If both control inputs are turned on, Results in High Currents



If both control inputs are turned on, as, the 1 input will dominate.

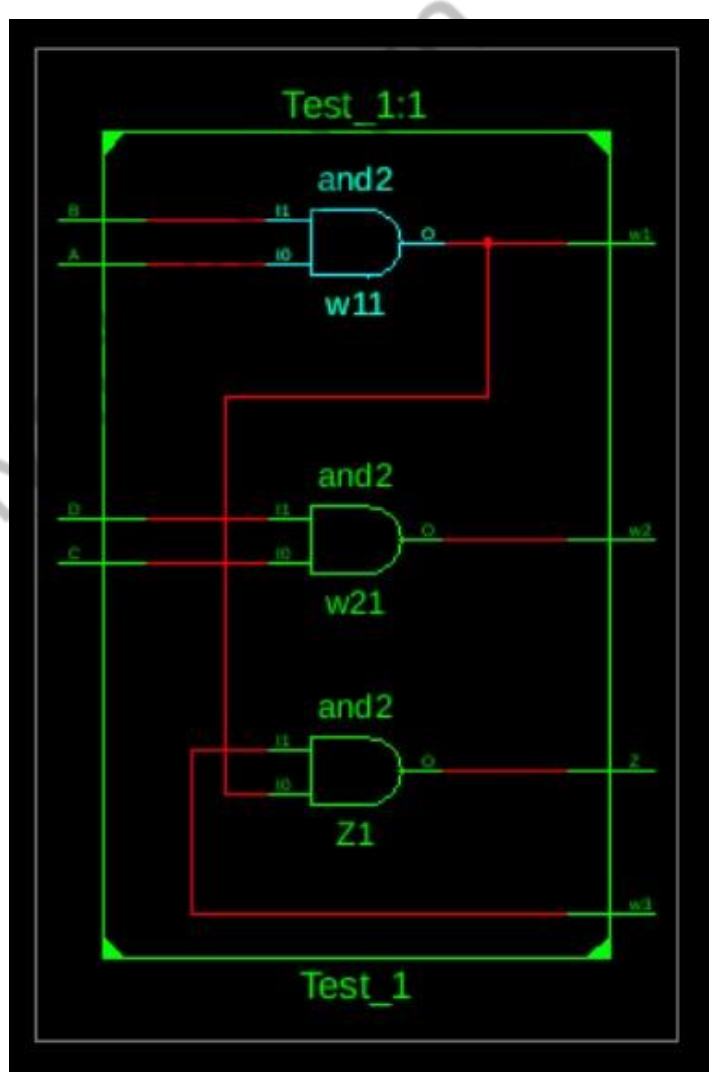
Modeling Signal States

Inputs		Output		
a	b	AND (ab)	OR $(a+b)$	NOT $(\bar{a})$
0	0	0	0	1
0	1	0	1	1
0	X	0	X	1
1	0	0	1	0
1	1	1	1	0
1	X	X	1	0
X	0	0	X	X
X	1	X	1	X
X	X	X	X	X

Example -1

Z = ABCD

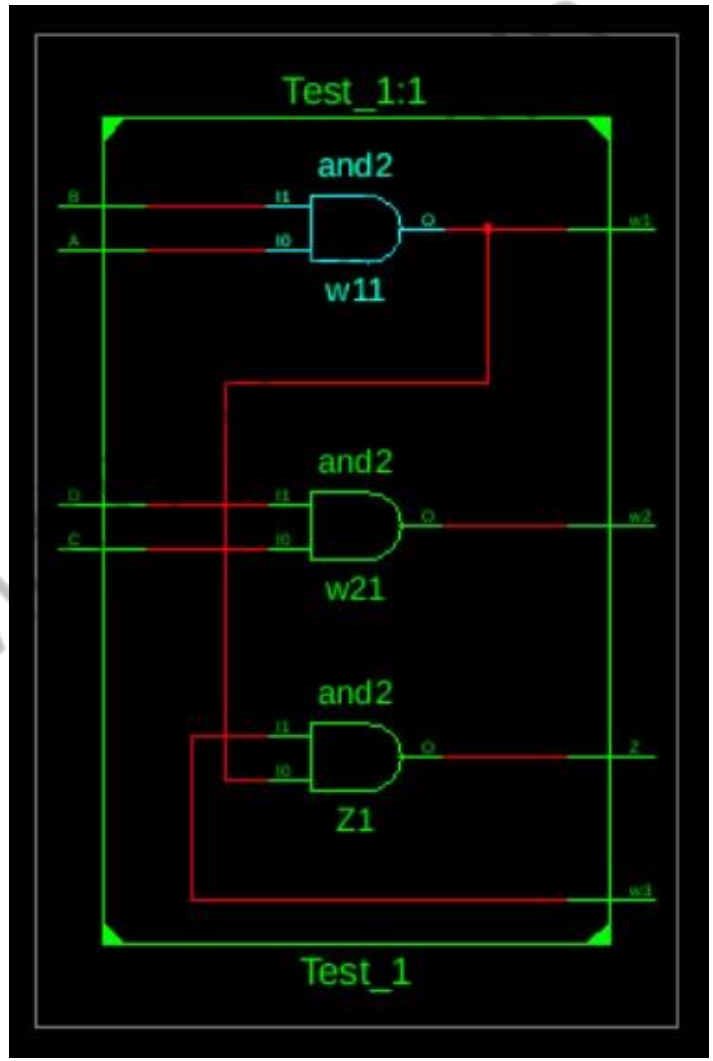
```
module Sample (
    input A, B, C, D,
    output Z,
    inout  w1, w2, w3 );
and g1(w1,A,B);
and g2 (w2, C,D);
and g3 (Z,w1,w3);
endmodule
```



Example -1

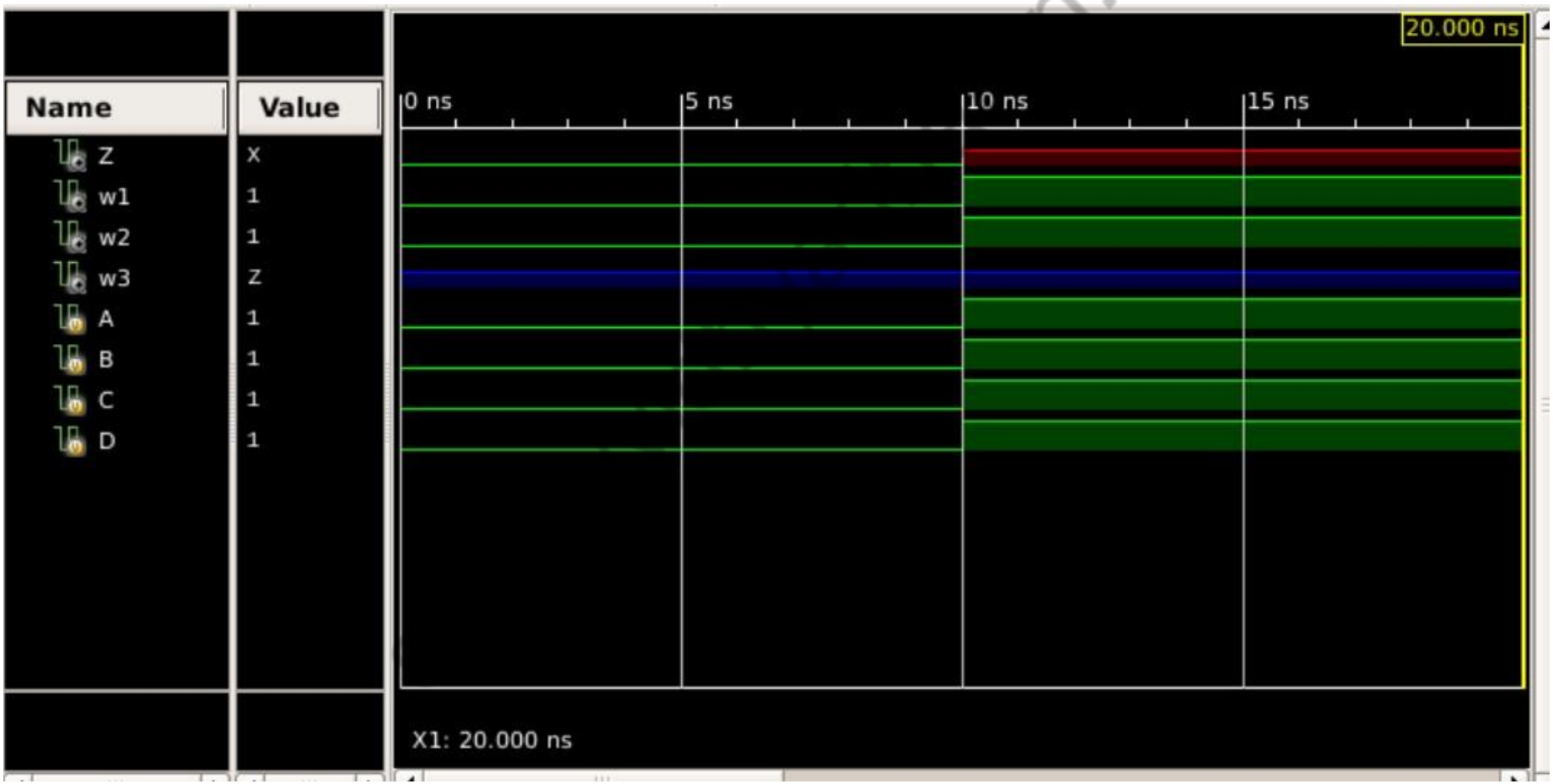
Z = ABCD

```
initial begin
A = 0;
B = 0;
C = 0;
D = 0;
#10;
A = 1;
B = 1;
C = 1;
D = 1;
#10;
$finish;
end
```



Example -1

Z = ABCD

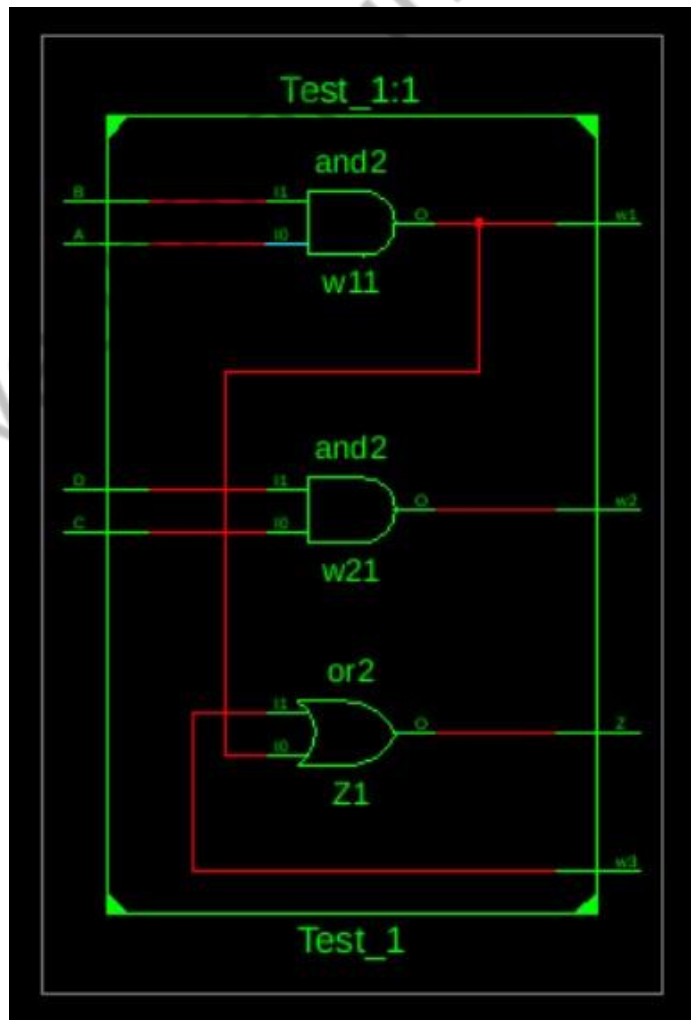


X (in red) stands for Forcing Unknown.

Example -2

Z = AB+CD

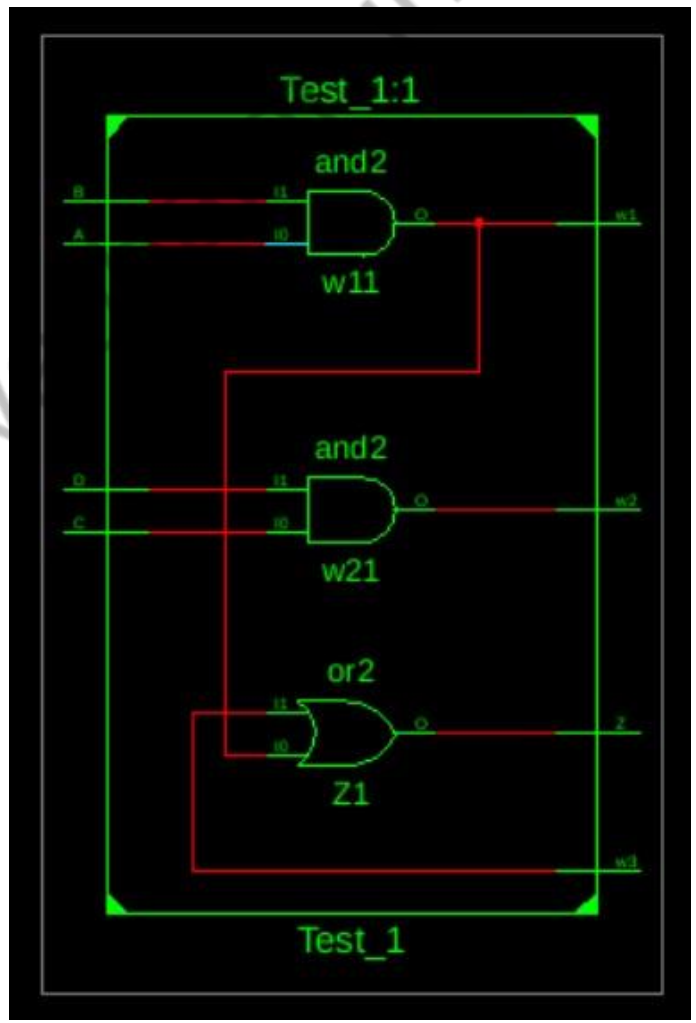
```
module Sample (
    input A, B, C, D,
    output Z,
    inout  w1, w2, w3 );
and g1(w1,A,B);
and g2 (w2, C,D);
or g3 (Z,w1,w3);
endmodule
```



Example -2

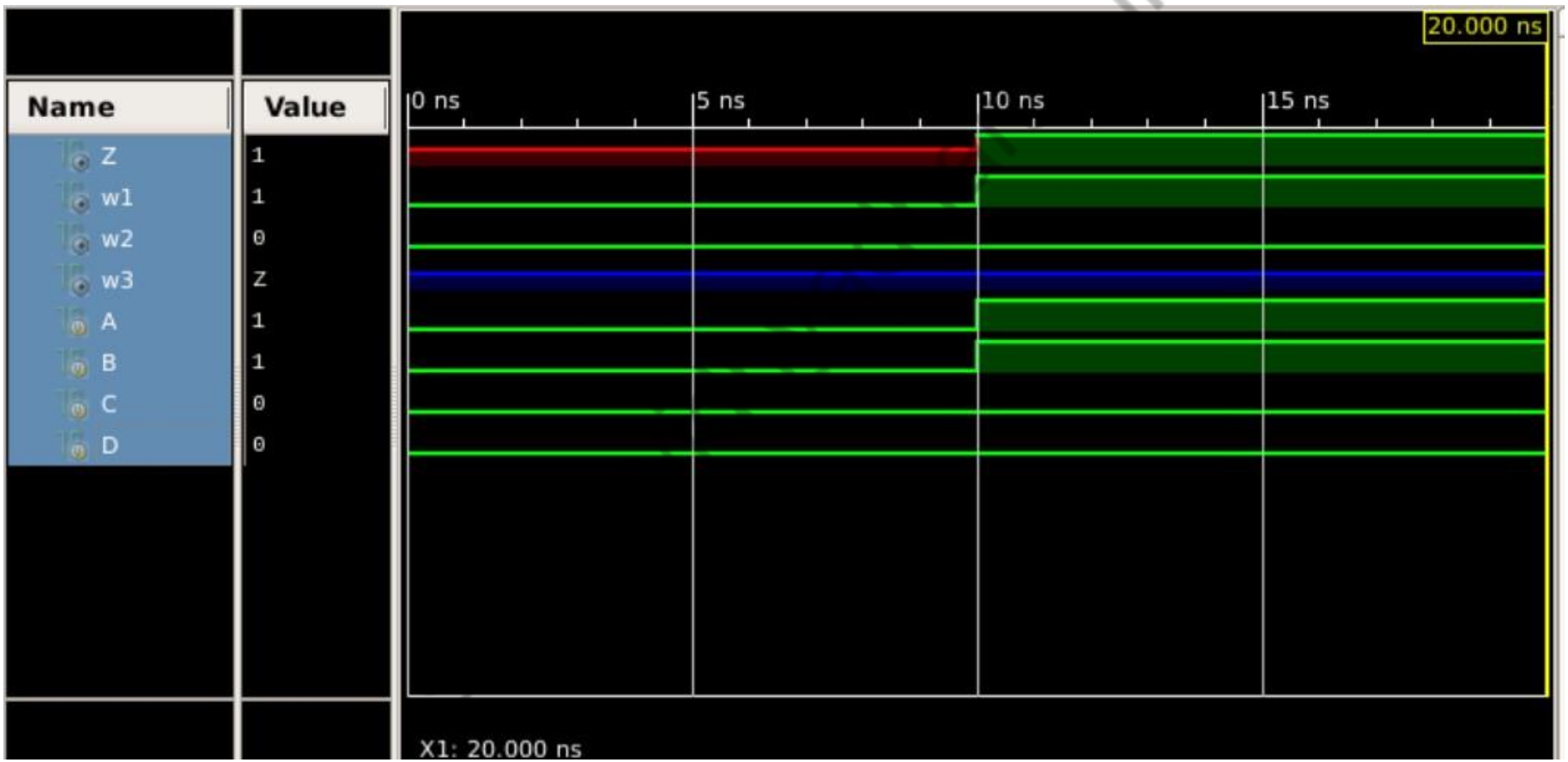
Z = AB+CD

```
initial begin
A = 0;
B = 0;
C = 0;
D = 0;
#10;
A = 1;
B = 1;
C = 1;
D = 1;
#10;
$finish;
end
```



Example -2

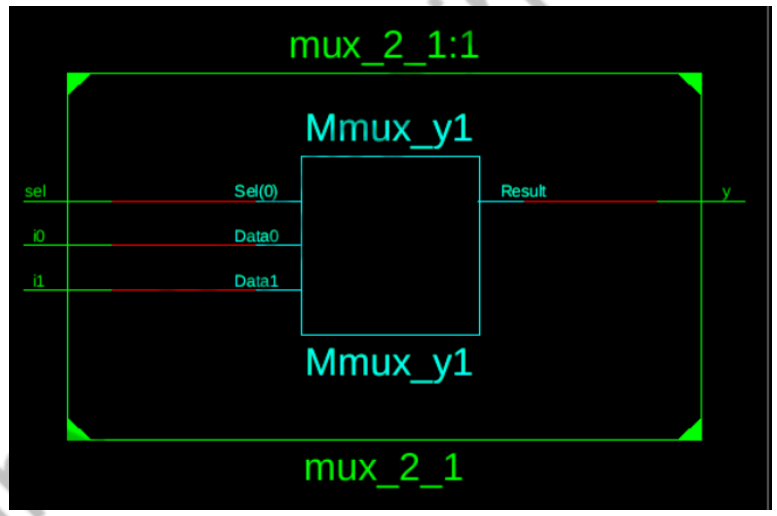
Z = AB+CD

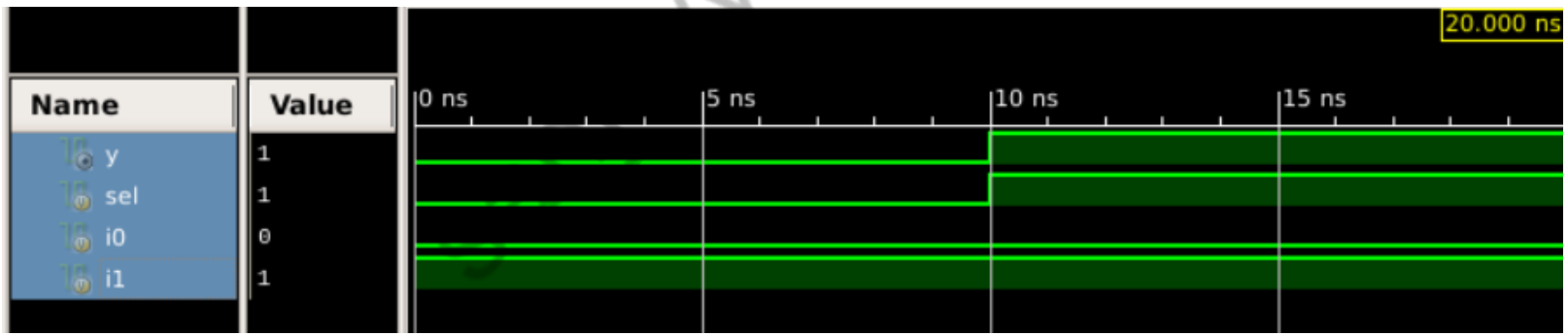


How is the Simulation tool identifying State X ??

Example -3

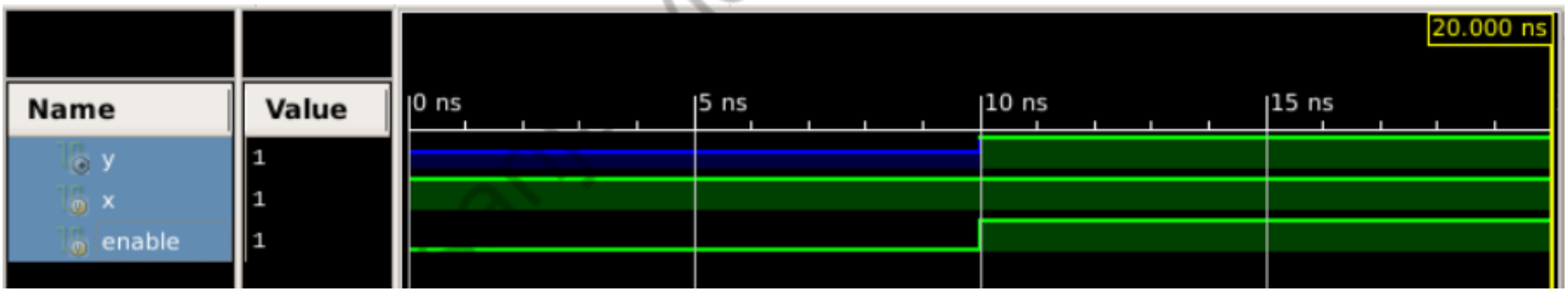
```
module mux_2_1(
  input sel,
  input i0, i1,
  output y);
  assign y = sel ? i1 : i0;
endmodule
```





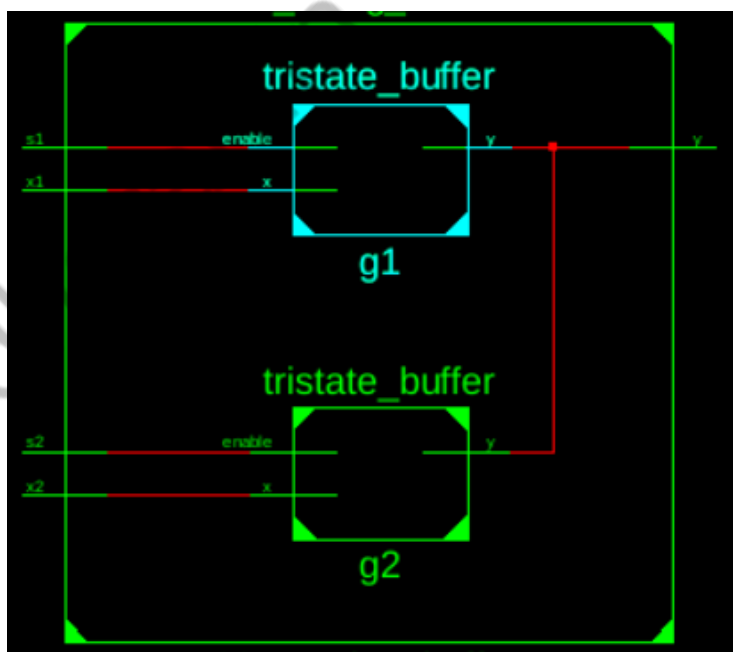
Example -4

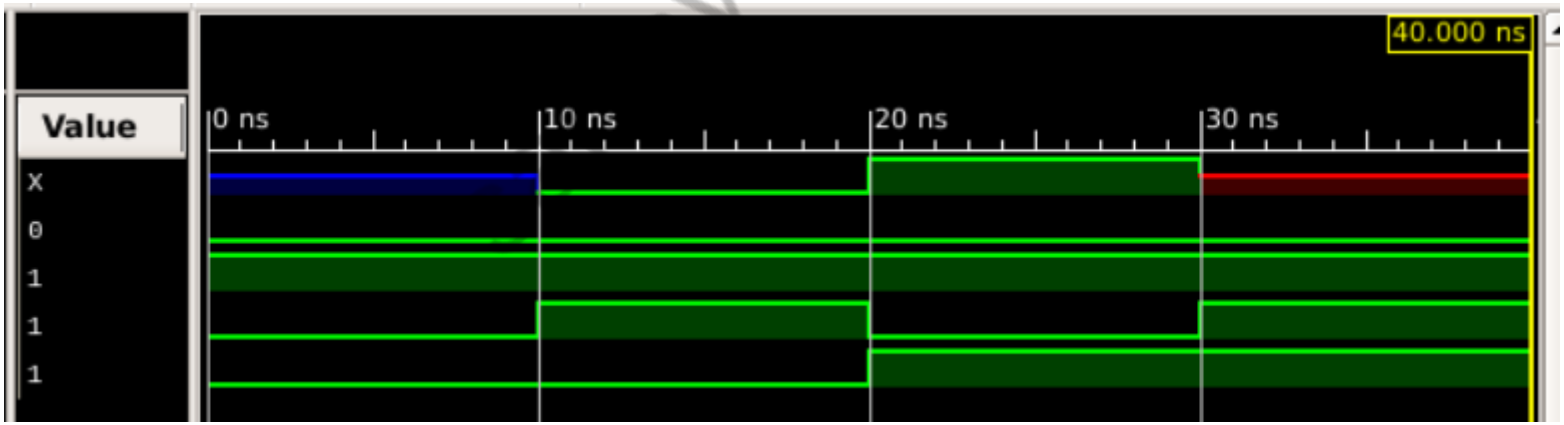
```
module tristate_buffer(
input x,
input enable,
output y);
assign y = enable? x : 'bz;
endmodule
```



Example -4

```
module Mux_using_buffer
(input x1,x2,s1,s2,
output y);
tristate_buffer g1 (x1,s1,y);
tristate_buffer g2 (x2,s2,y);
endmodule
```





Modeling Gates for Z and X inputs

AND	0	1	Z	X
0	0	0	0	0
1	0	1	X	X
Z	0	X	X	X
X	0	X	X	X

OR	0	1	Z	X
0	0	1	X	X
1	1	1	1	1
Z	X	1	X	X
X	X	1	X	X

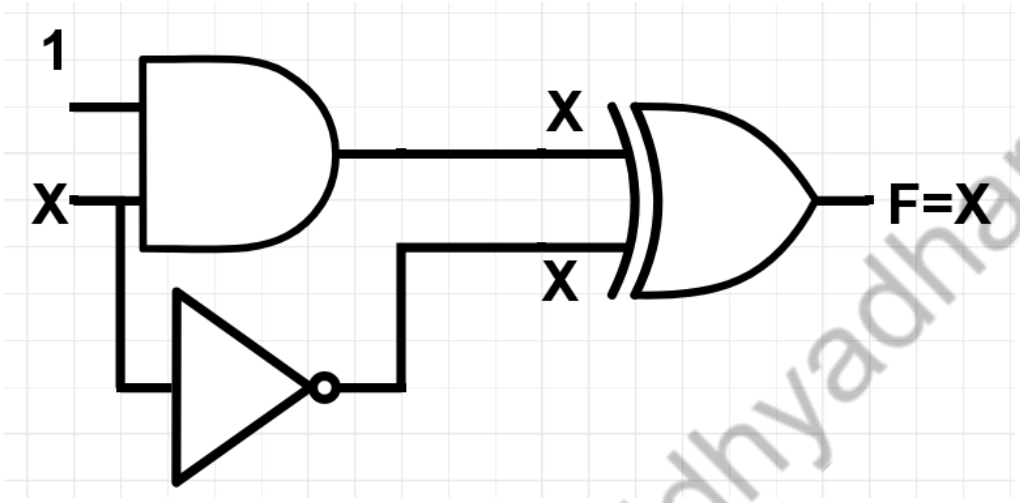
NOT	
0	1
1	0
Z	X
X	X

Modeling XOR/NOR Gate

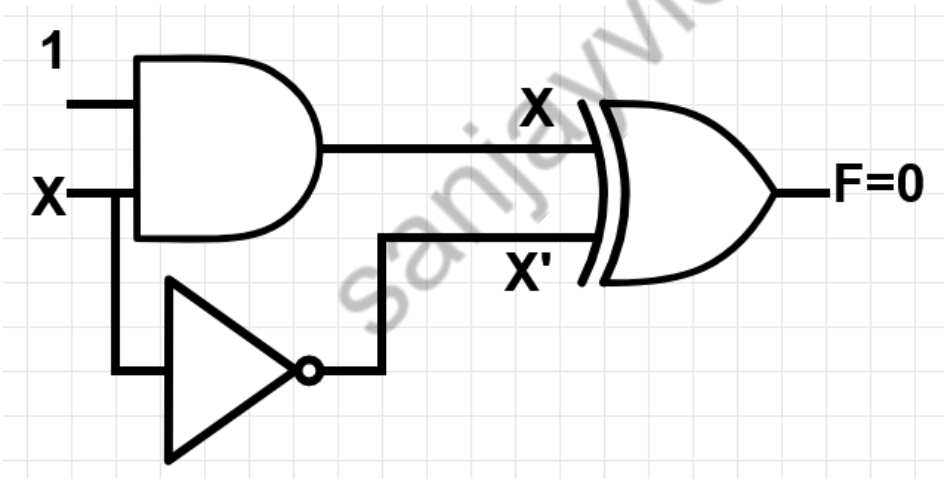
XOR	0	1	Z	X
0	0	1	X	X
1	1	0	X	X
Z	X	X	X	X
X	X	X	X	X

XNOR	0	1	Z	X
0	1	0	X	X
1	0	1	X	X
Z	X	X	X	X
X	X	X	X	X

Limitation in Simulation

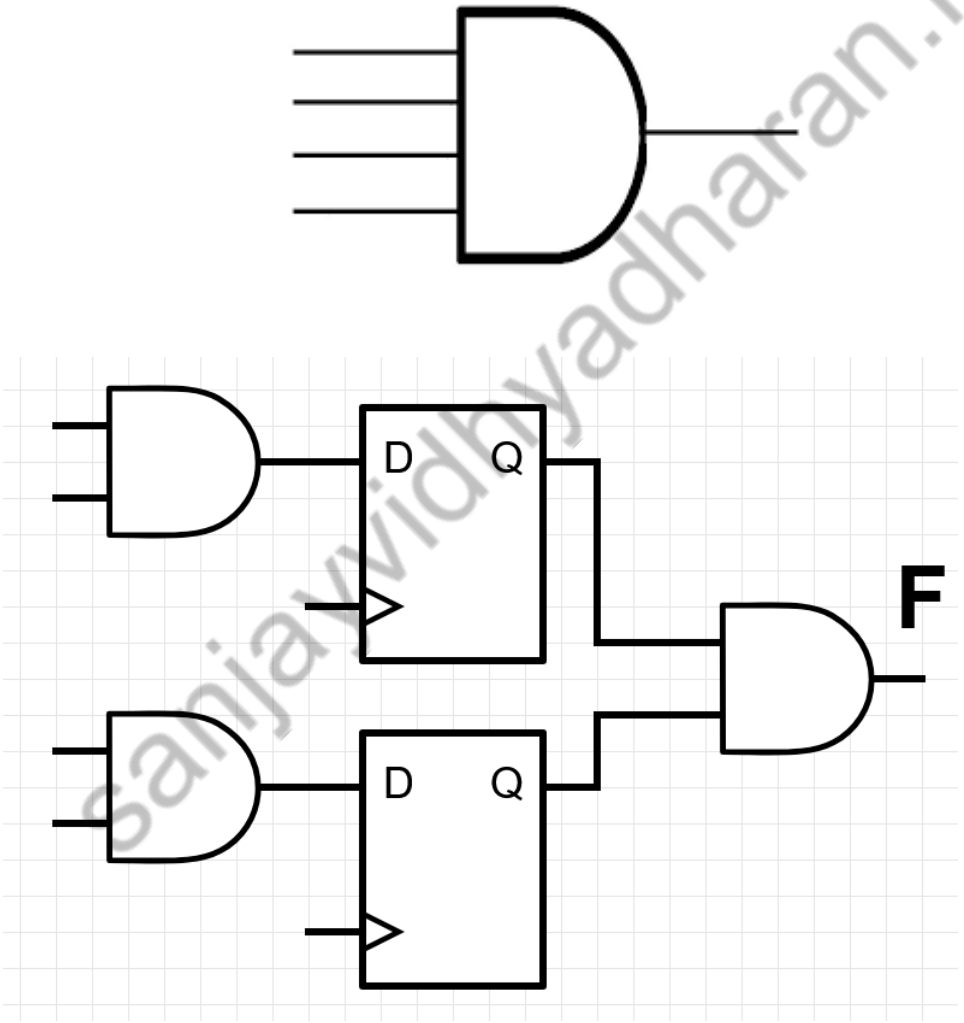


Simulation

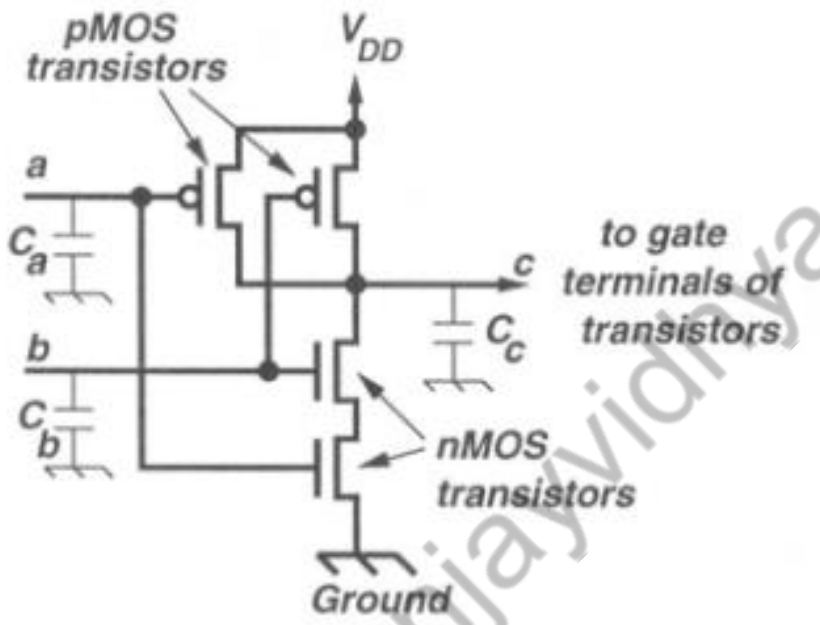


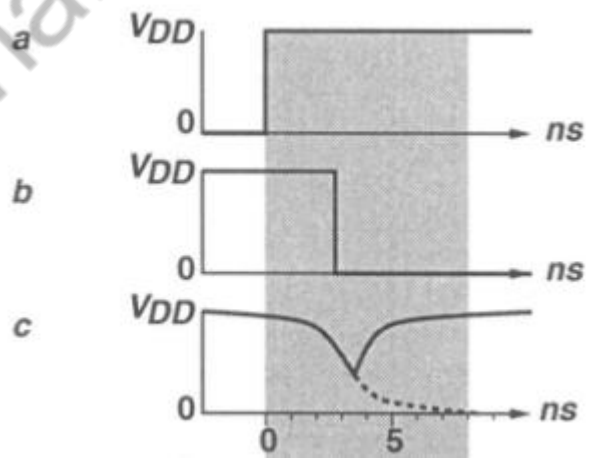
Testing

Design for Testing



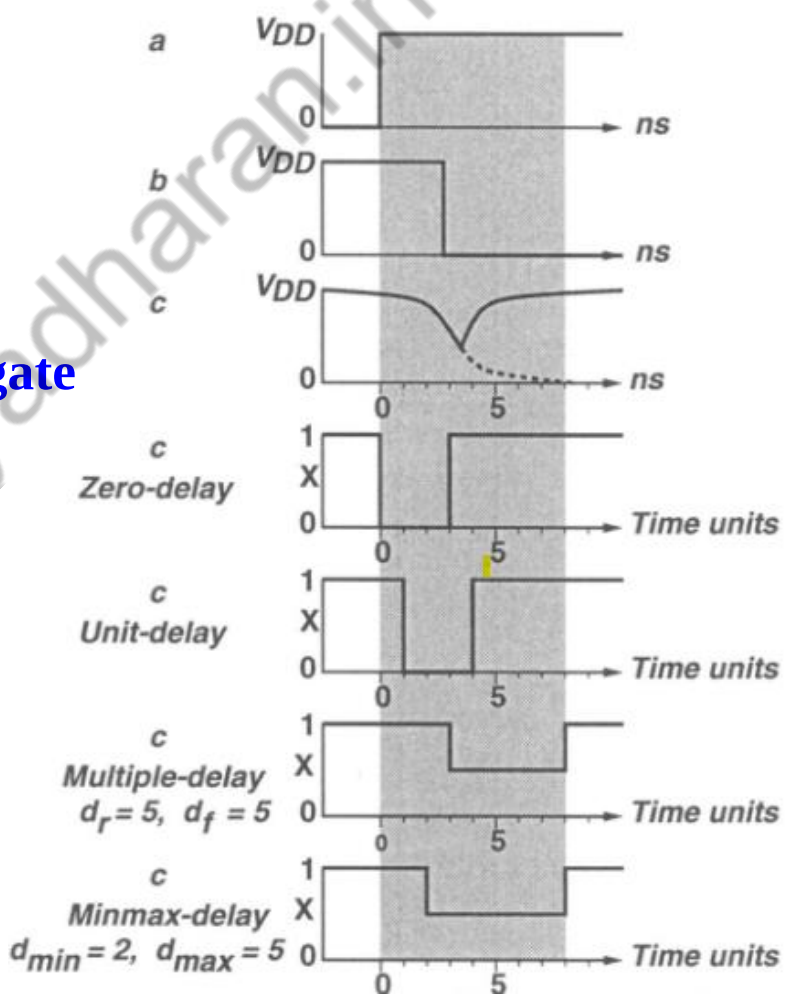
Modeling Circuits for Simulation





Modeling Circuits for Simulation

1. **Zero Delay Model**
2. **Unit Delay Model**
3. **Multiple Delay Model**
   Tr and Tf different for each type of gate
4. **Min. Max Delay**



Digital circuit simulators tend to either ignore the fine grain variations (transients) between those meaningful values or model the transients

Algorithms for True-Value Simulation

- Step 1: Levelize circuit and produce compiled code
- Step 2: Initialize data variables (flip-flops and other memory)
- Step 3: For each input vector
  Set primary input variables
  Repeat until steady-state or maximum iteration-count reached
    Execute compiled-code
    Report or save variable values

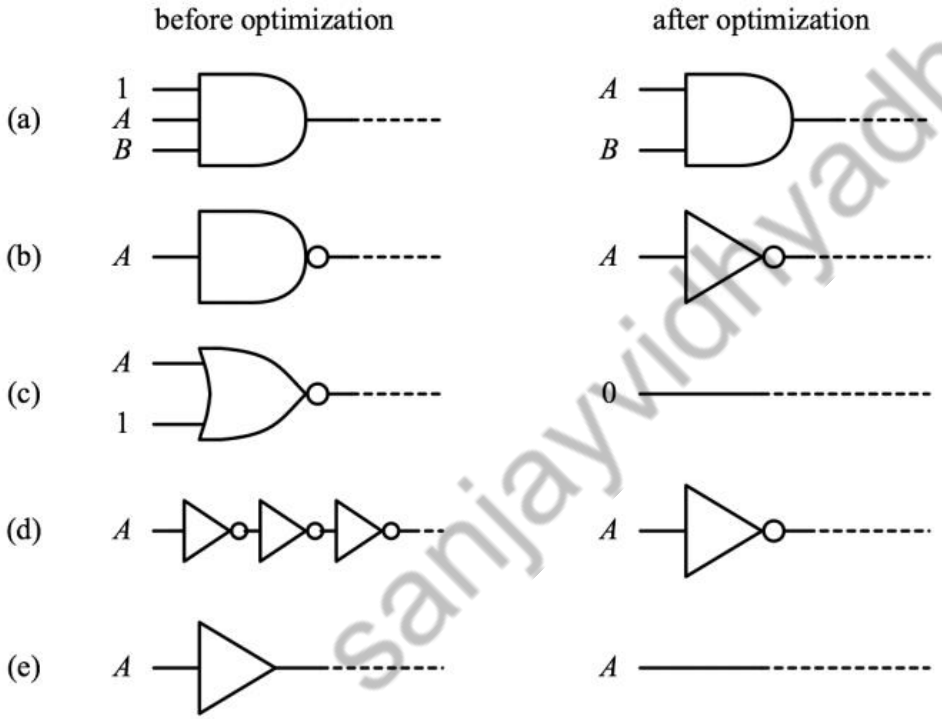
Compiled-Code Simulation

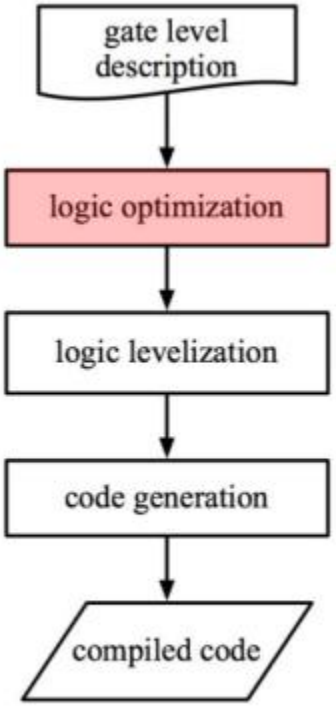
Normally in an HDL such as VHDL or Verilog

1. Circuit Simplification
2. Circuit Levelized
3. Signals are treated as variables in the code
4. For every input vector, the code is repeatedly executed until all variables have attained steady values
5. Compiled-code simulators are very effective where two-state (0,1) simulation suffices
6. Timing are not modeled in a compiled-code simulator

Compiled-Code Simulation

Circuit Simplification





(b) Code generation flow

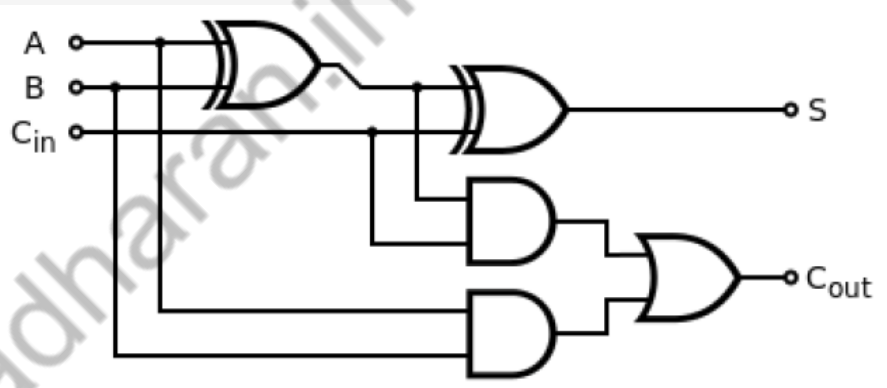
Video lectures by Professor James Chien-Mo Li

Compiled-Code Simulation

Levelisation

```
// Online C++ compiler to run C++ program
#include <iostream>
using namespace std;
void Full_Adder(int A,int B,int C_In){
     int Sum , C_Out , P;
     // Calculating value of sum
    Sum = C_In ^ P;
    P = A ^ B;
    //Calculating value of C-Out
     C_Out = (A & B) | (B & C_In) | (A & C_In);
      // printing the values
    cout<<"Sum = "<< Sum <<endl;
    cout<<"C-Out = "<< C_Out;
   }
   // Driver code
int main() {
     int A = 1;
     int B = 0;
     int C_In = 1;
     // passing three inputs of fulladder as arguments to get result function
     Full_Adder(A, B, C_In);
     return 0;
}

```



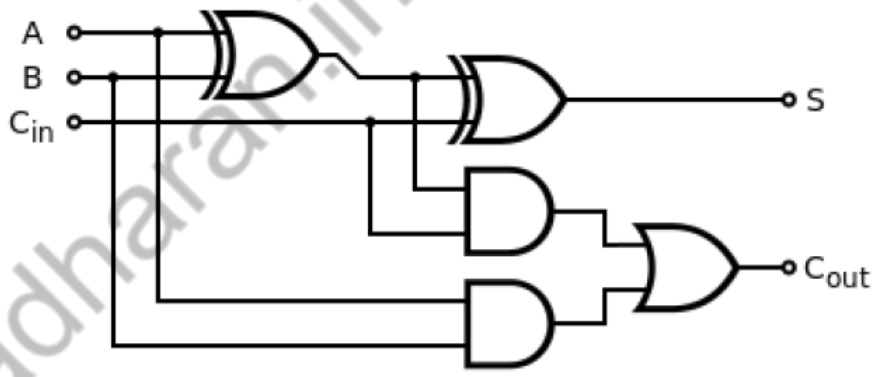
Output

```
/tmp/QmZlkml12E.o
Sum = 1
C-Out = 1
```

Compiled-Code Simulation

Levelisation

```
1  // Online C++ compiler to run C++ program
2  #include <iostream>
3  using namespace std;
4  void Full_Adder(int A,int B,int C_In){
5      int Sum , C_Out , P;
6      // Calculating value of sum
7     P = A ^ B;
8     Sum = C_In ^ P;
9     //Calculating value of C-Out
10     C_Out = (A & B) | (B & C_In) | (A & C_In);
11      // printing the values
12    cout<<"Sum = "<< Sum <<endl;
13    cout<<"C-Out = "<< C_Out;
14   }
15   // Driver code
16 int main() {
17     int A = 1;
18     int B = 0;
19     int C_In = 1;
20     // passing three inputs of fulladder as arguments to get result function
21     Full_Adder(A, B, C_In);
22     return 0;
23 }
24
```



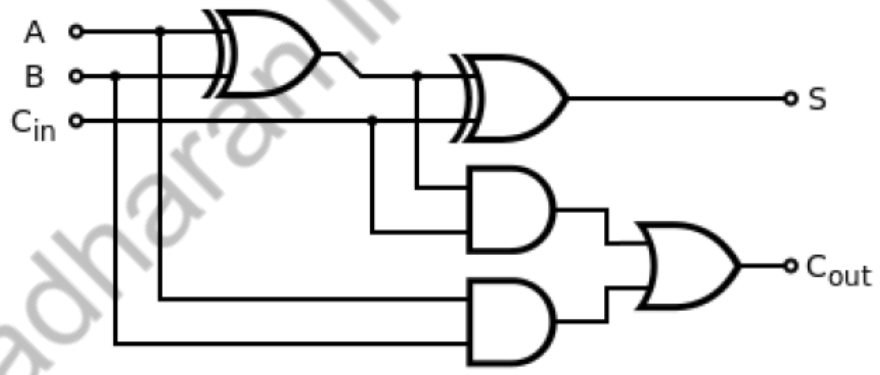
Output

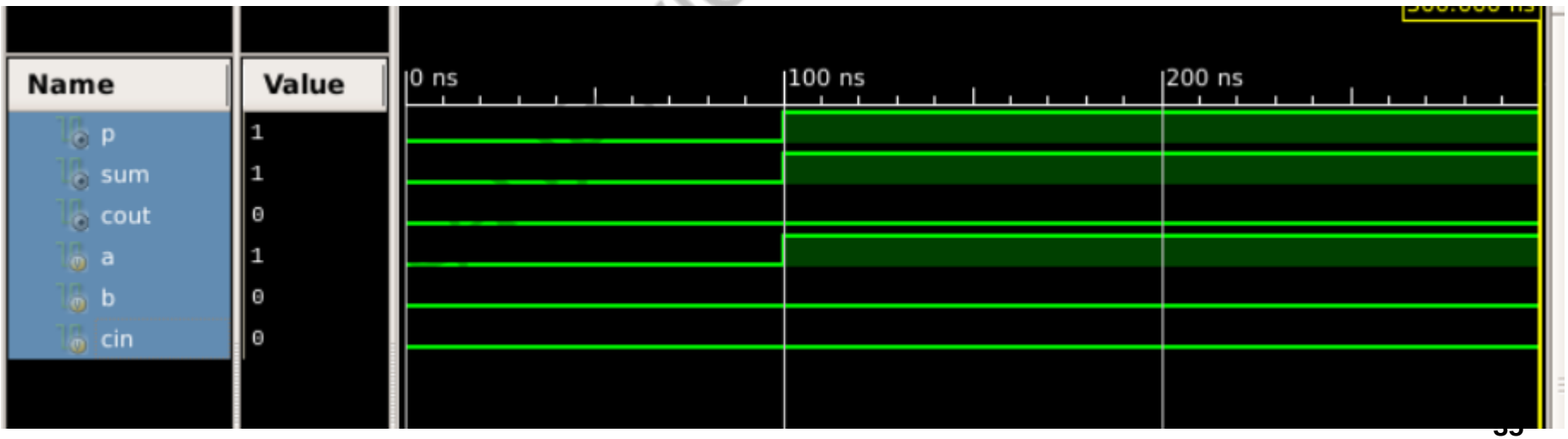
```
/tmp/QmZlkml12E.o
Sum = 0
C-Out = 1
```

Compiled-Code Simulation

Levelisation

```
module Full_adder_df (
input a, b, cin, wire q, r, output p, cout, sum);
assign sum = p^cin;
assign p = a^b ;
assign r = a&b ;
assign q = p&cin ;
assign cout = r | q;
endmodule
```





Compiled-Code Simulation

Compiled-code simulation: convert gates into codes for evaluation

- Optimization: simplifies logic
- Levelization: sort gates in order (*i.e.* topological sort of graph)
- Code generated: 1.high-level, 2.machine, 3.interpreted

☺ Pros

- Simple to implement
- Can speed-up by parallelism
  - see parallel simulation

☹ Cons

- Only *cycle-based accuracy*, no timing (zero gate delay)
- Need to evaluate whole circuit even only small portion changed
  - see event-driven simulation

Event-driven Simulation

* **Zero delay**
* **Nominal delay**
* **Data structure**

Event-driven Faster then CC

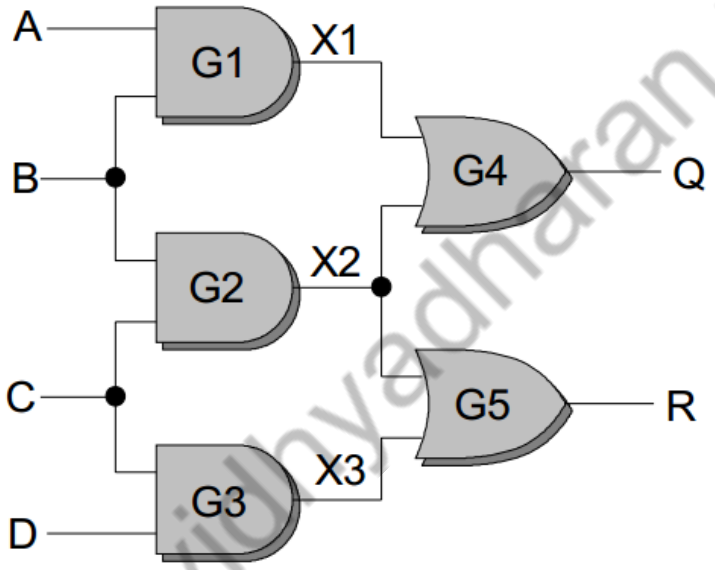
It is based on the recognition that any signal change (event) must have a *cause*, which is also an event. Thus, an event causes new events, which in turn may cause more events. An event-driven simulator follows the path of events.

Gates whose inputs now have events are called *active* and are placed in an *activity list.* The simulation proceeds by removing a gate from the activity list and evaluating it to determine whether its output has an event.

An event-driven simulator only does the necessary amount of work.

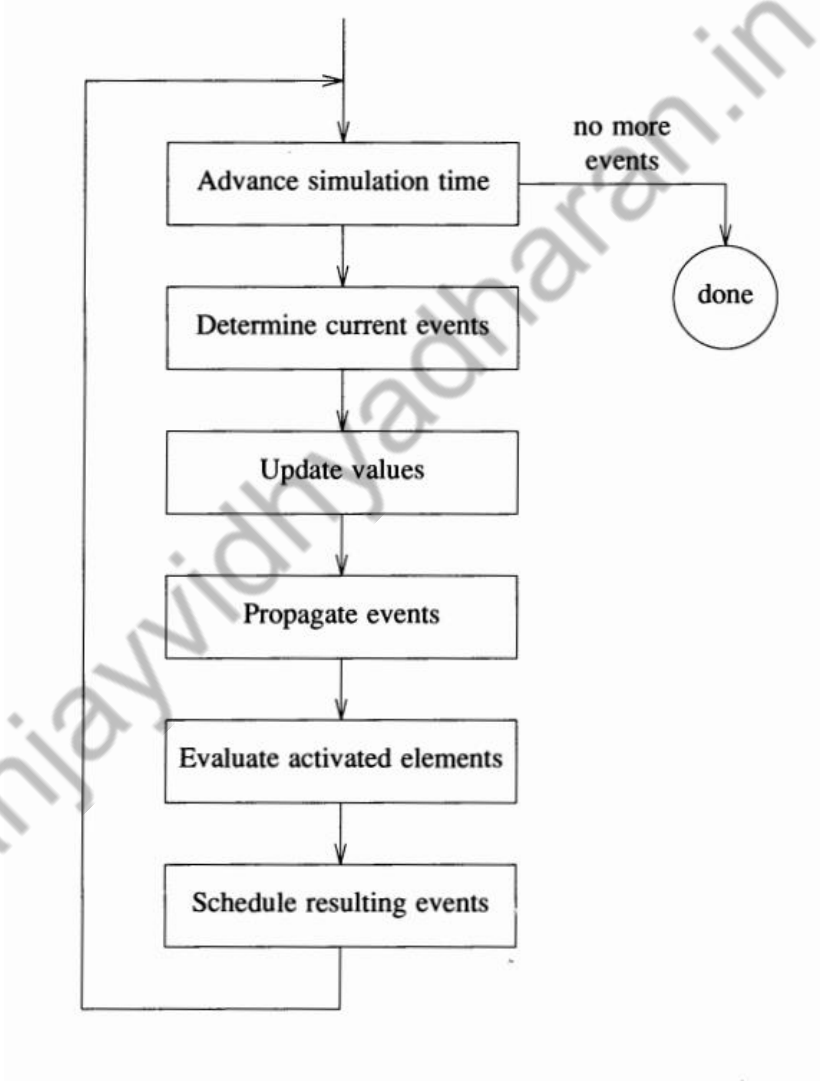
For logic circuits, in which typically very few signals change at a time, this can result in significant savings of computing effort. However, the biggest advantage of this technique is in its ability to simulate any arbitrary delays. This is done by a procedure known as *event scheduling.*

Event-driven Simulation



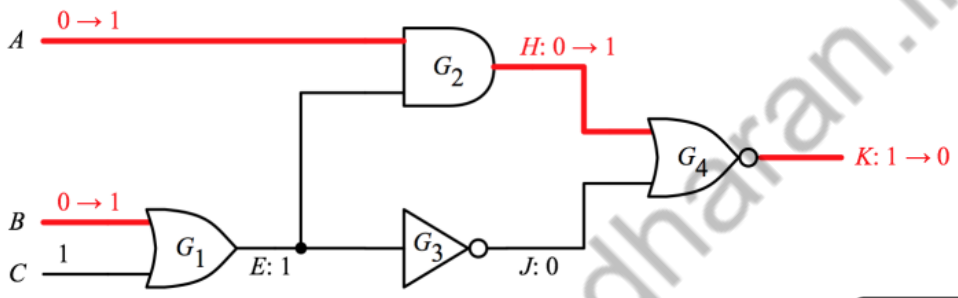
Suppose the circuit is simulated with two consecutive input vectors, (0,0,0,0) and (0,0,0,1). Since A, B, and C have not changed, it is not necessary to simulate gates G1 and G2. Since neither G1 nor G2 have been simulated, it is not necessary to test X1 or X2 for changes. The simulation of G4 can be bypassed without testing X1 or X2.

Event-driven Simulation



Event-driven Simulation

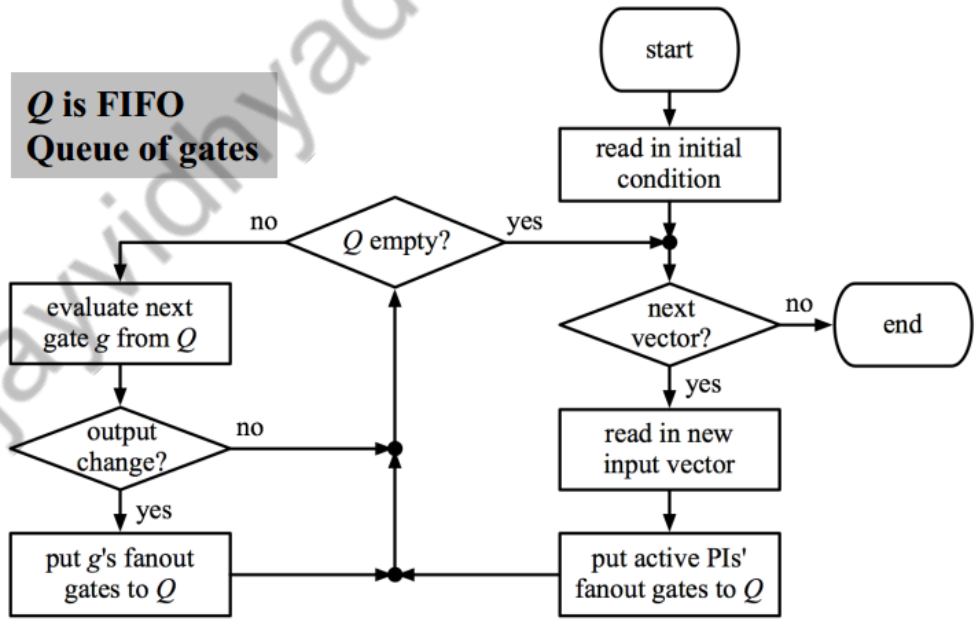
Zero-delay Event-driven Sim



event (g, v_g^+) means gate g changes to v_g^+

Q is FIFO Queue of gates

Executed events	Q
$(B,1)(A,1)$	G_1, G_2
$(G_1, 1)$ -	G_2
$(G_2,1)$	G_4
$(G_4,0)$	-

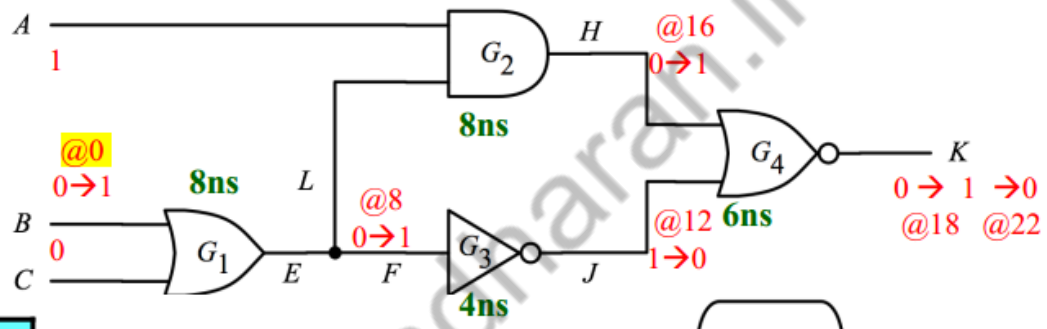


[2]. Video lectures by Professor James Chien-Mo Li

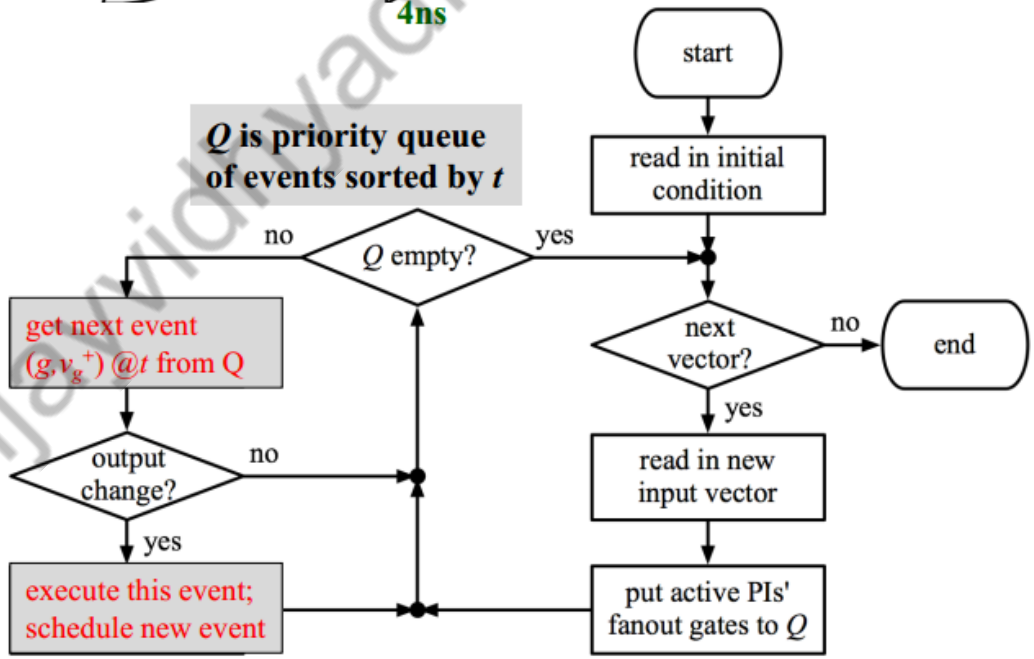
Event-driven Simulation

Nominal delay

event (g, v_g^+) *@ t*
gate *g* output changes to v_g^+ at *time stamp t*



t	Executed events	Q
-		(B,1)@0
0	(B,1)	(G_1,1)@8
8	(G_1,1)	(G_3,0)@12 (G_2,1)@16
12	(G_3,0)	(G_2,1)@16 (G_4,1)@18
16	(G_2,1)	(G_4,1)@18 (G_4,0)@22
18	(G_4,1)	(G_4,0)@22
22	(G_4,0)	--



[2]. Video lectures by Professor James Chien-Mo Li

Event-driven Simulation

t	Executed events	Sch'd event
12	(G_2,1)	-- (G4=0 output no change)
12	(G_3,0)	-- (G4=0 output no change)

G_4=0 @18 Correct

G3 taken as 1

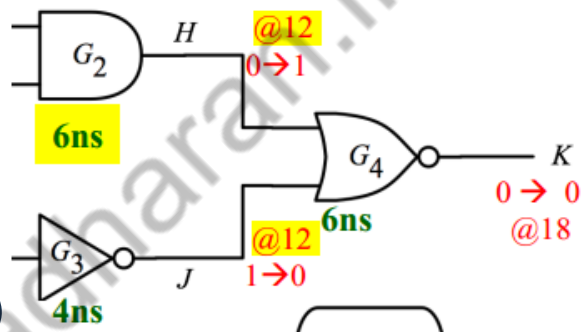
G2 taken as 1

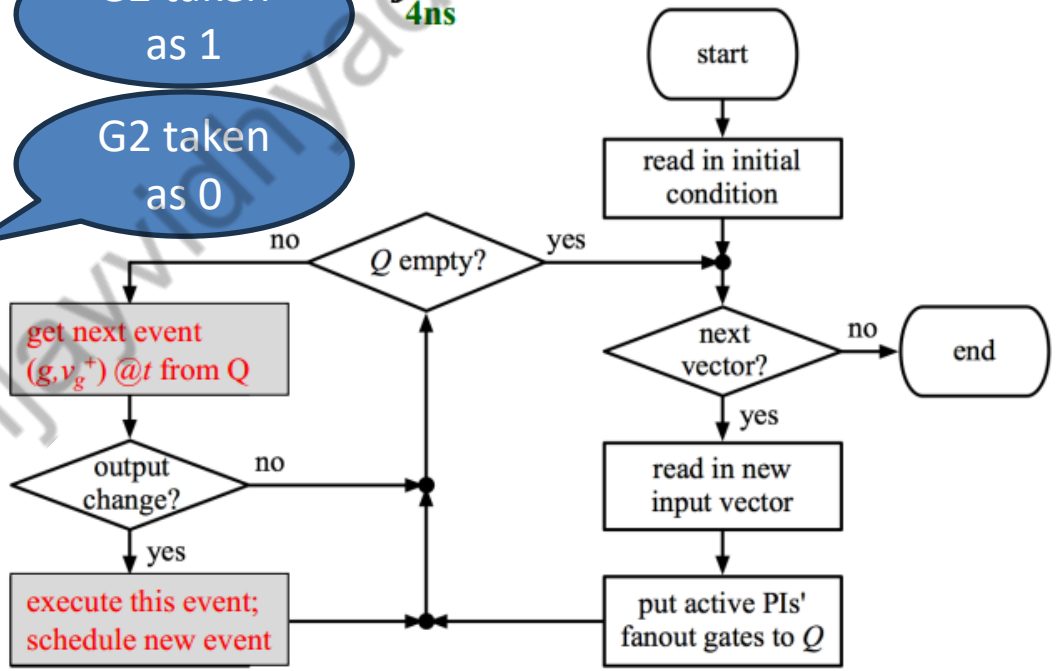
t	Executed events	Sch'd event
12	(G_3,0)	(G_4,1)@18
12	(G_2,1)	-- (G4=0 output no change)
18	(G_4,1)	--

G_4=1 @18 Wrong!

G2 taken as 0

SIC May Results in Different Outputs!





References

1. “Essentials of Electronic Testing, for Digital, Memory and Mixed-Signal VLSI Circuits”, Michael L. Bushnell and Vishwani D. Agrawal, – Kluwer Academic Publishers (2000).

2. Video lectures by Professor James Chien-Mo Li
Lab. of Dependable Systems Graduate Institute of Electronics Engineering
National Taiwan University
https://www.youtube.com/watch?v=yfcoKOUV5DM&list=PLvd8d-SyI7hjk_Ci0zpTqImAtpEjdK5JF&index=1

Thankyou