3. LOGIC SIMULATION

Fernando Gonçalves ©

1

Introduction

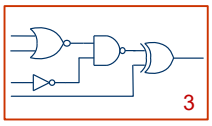
Levels of simulation

```
always @(negedge clk)
begin
  if (reset)
    data_out <= 0;
  else if (enable)
    data_out <= data_in;
end
```

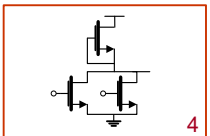
1

BEHAVIOR level

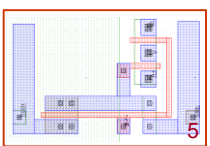
FUNCTIONAL or **REGISTER TRANSFER (RTL)** level



GATE level



TRANSISTOR level



PHYSICAL or **GEOMETRICAL** level

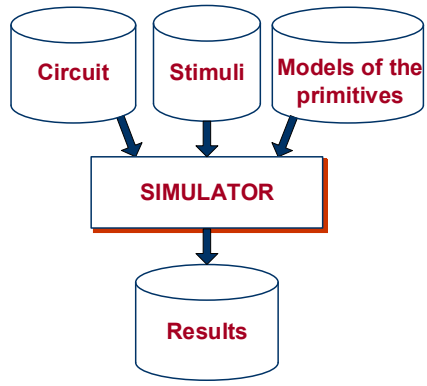
2

Objectives of the Logic Simulation

- **Project validation and verification** – detects project errors, initialization problems, errors caused by propagation delays, spikes, etc.
  - Logic verification (functional)
  - Timing analysis
- **Fault analysis**
  - Fault coverage
  - Fault diagnosis
- **Estimation of the circuit activity in order to evaluate the power consumption**
- **Code coverage evaluation**

3

Inputs/Outputs of a Simulator



4

Characteristics of a Logic Simulator

- Major problems
  - How to generate the stimuli?
  - How to know the correct responses?
  - How to evaluate the quality of the stimuli?

Performed by the designer, using the specifications of the product

Use metrics: code coverage and node toggling

- Types of simulations
  - **Digital**: used at the behavioral, functional or gate levels
  - **Analog**: used at the transistor level (Spice-like simulators)
  - **Mixed**: used for mixed analog/digital circuits
- Techniques to drive the simulation
  - by compiled code
  - by events (event-driven)

5

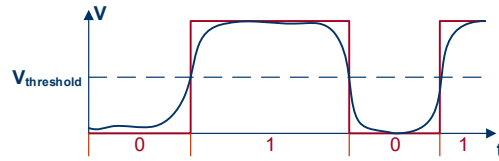
Characteristics of a Logic Simulator

- Main features of a logic simulator
  - Accuracy of the results
  - Performance
  - Universality of the models
- Typical cost of a commercial simulator > 50 000€
- Requirements of a logic simulator
  - Logic values
  - Models of the logic elements
  - Technique to drive the simulation

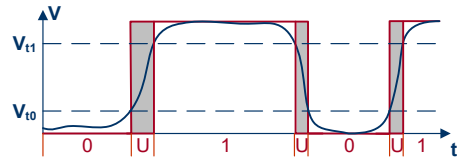
6

Logic Values

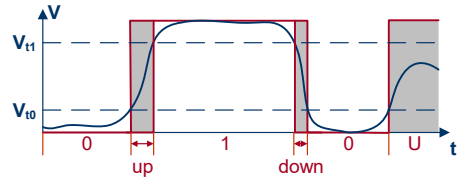
2 logic values:
0 and 1



3 logic values:
0, 1 and U



5 logic values:
0, 1, U, up and down



7

Unknown Logic Value

The initial state of the memory elements is unknown

U = Unknown logic value ⇔ {0,1}

Boolean Operations

AND	OR	NOT
AND (0, U) = AND ({0}, {0,1}) = {AND (0,0), AND (0,1)} = {0,0} = 0	OR (0, U) = OR ({0}, {0,1}) = {OR (0,0), OR (0,1)} = {0,1} = U	NOT (U) = NOT ({0,1}) = {NOT (0), NOT (1)} = {1,0} = U

8

Three-Valued Algebra

Boolean operations using three logic values

AND	0	1	U
0	0	0	0
1	0	1	U
U	0	U	U

NOT	0	1	U
OUT	1	0	U

OR	0	1	U
0	0	1	U
1	1	1	1
U	U	1	U

9

Primitive Cubes in Three-Valued Logic

Difference between X and U:

X = don't care

U = unknown

x_1	x_2	x_3	Z
X	1	0	**0**
1	1	X	**0**
X	0	X	**1**
0	1	1	**1**

10

Primitive Cubes in Three-Valued Logic

Given the values v_1, v_2, v_3, where $v_i \in \{0,1,U\}$, determine $Z(v_1, v_2, v_3)$

1. Create the cube $\mathbf{v_1\ v_2\ v_3} \mid \mathbf{X}$
2. Intersect that cube with the primitive cubes of **Z**
3. If the intersection is consistent, then the value of **Z** is in the rightmost position
4. If the intersection is not consistent, then assign **Z = U**

Examples:

1U0 | X not consistent with any cube ⇒ **Z = U**

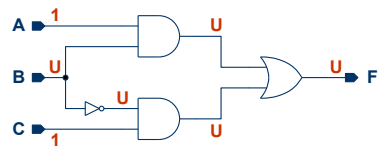
10U | X consistent with X0X | 1 ⇒ **Z = 1**

Intersection operator

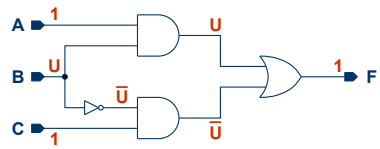
∩	0	1	X	U
0	0	∅	0	∅
1	∅	1	1	∅
X	0	1	X	U
U	∅	∅	U	U

Three-Valued Logic ⇒ Lost of Information

If B = U, then $\overline{B}$ = U (B e $\overline{B}$ are not complementary)

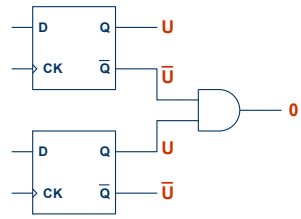


⇒ F = U pessimistic result (correct is F = 1)



Shall we use the $\overline{U}$ value? such that: U . $\overline{U}$ = 0 and U + $\overline{U}$ = 1

Problems with the Unknown Value



Incorrect result because U and $\bar{U}$ from different FFs are not complementary

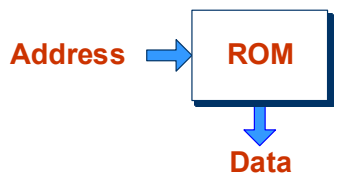
⇒ Better to be pessimistic than incorrect!

Correct solution:

use U_1, U_2, … and propagate expressions ⇒ complexity

13

Unknown Value in Functional Models



If the control values have **K** unknowns, then it is necessary to perform 2^K operations on the individual results

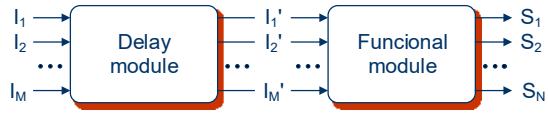
Addresses	Outputs
0000	(*)
0001	(*)
0010	(*)
0011	(*)
0100	(*)
0101	(*)
0110	(*)
0111	(*)
1000	**1100**
1001	(*)
1010	**0100**
1011	(*)
1100	**0110**
1101	(*)
1110	**1110**
1111	(*)
1UU0	**U1U0**

(*) Outputs for these addresses are irrelevant

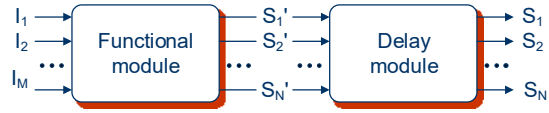
14

Modeling of Logic Elements

Front-end delay model



Back-end delay model

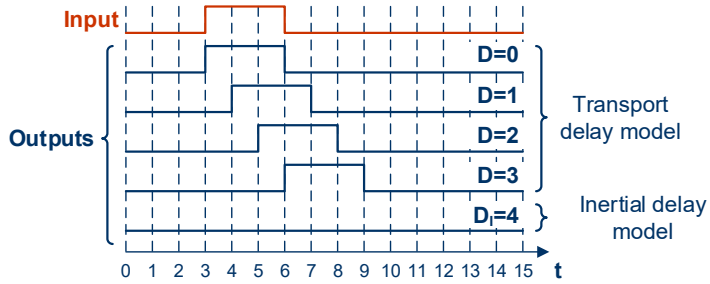


Typically M > N

15

Delay Model

Inertial and transport delay models



Transport delay model: represents a pure propagation delay

Inertial delay model: represents the time for which an input value must be stable before the value is allowed to propagate to the output − filters out unwanted spikes and transients − default mode in Verilog and VHDL

16

Delay Models

Min-Max delay model

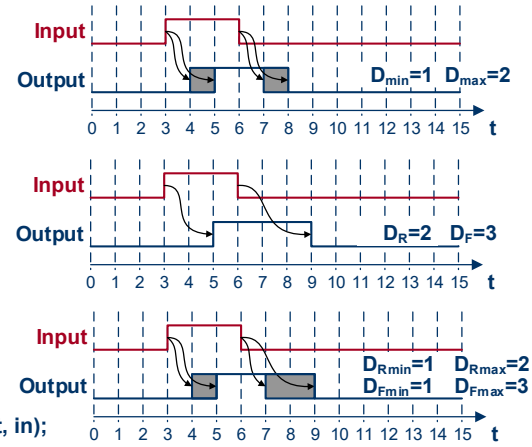
e.g. in Verilog: **buf #(3:5:6) (out, in);**
(min:typ:max)

Rise-fall delay model

e.g. in Verilog: **buf #(2,3) (out, in);**
(rise,fall)

Accurate delay model

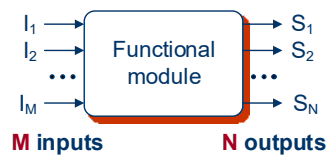
e.g. in Verilog: **buf #(2:3:5,3:4:6) (out, in);**



All these delay models are available in Verilog

Models for a Functional Module

- **Goal**: obtain the result of a logical operation
- Implementation of the model using:
  - Table
  - Algebraic expression
  - Binary decision diagram
  - Procedure



Functional Module

Model defined by tables

AND	0	1	U
0	0	0	0
1	0	1	U
U	0	U	U

NOT	0	1	U
OUT	1	0	U

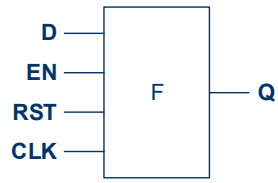
OR	0	1	U
0	0	1	U
1	1	1	1
U	U	1	U

Operations using a three-valued algebra

19

Functional Module

Model defined by a procedure




```
always @(posedge CLK)
begin
  if (RST)
    Q <= 0;
  else if (EN)
    Q <= D;
end
```


20

Techniques to Drive the Simulation

- by compiled code

  Convert the description of the circuit in machine code of the computer (i.e. an executable program)

- by events (event-driven)

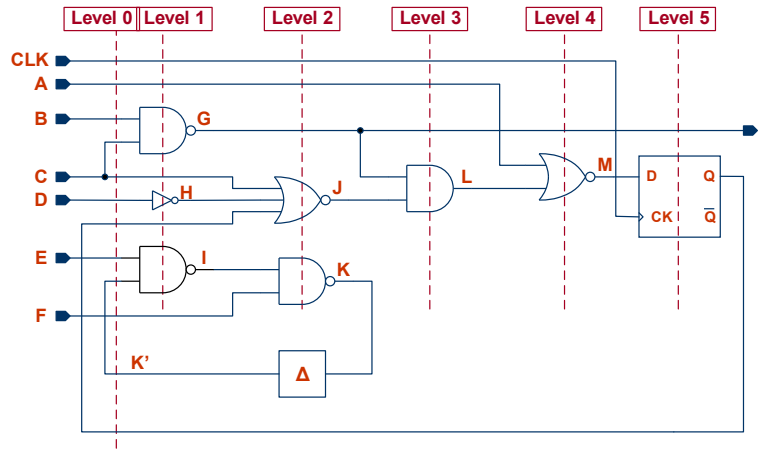
  Convert the description of the circuit in a set of tables

Modern simulators use a mix of both techniques

Compiled Code Simulation: Overview

- Pre-processing steps
  - Translate the circuit to the canonical form
  - Sort the logic elements ⇔ levelize the circuit
  - Generate the code for compilation (typically C code)
  - Compile the code to create the simulator
- Levelize the circuit
  - Level 0 is composed by all primary inputs and by the feedback lines that were previously broken
  - Level K is composed by the elements whose inputs are lines with maximum level (K-1)

Levelize the Circuit



23

Generate the Code for Compilation

```
main ();
{
    K´ = <initial value>;
    Q  = <initial value>;
    do {
        status = read_inputs (CLK, A, B, C, D, E, F);
        if (status == <no more data>) exit();
        do {
            G = ~(B & C);                    /* level 1 */
            H = ~D;                          /* level 1 */
            I = ~(E & K´);                   /* level 1 */
            J = ~(C | H | Q);                /* level 2 */
            K = ~(I & F);                    /* level 2 */
            L = G & J;                       /* level 3 */
            M = ~(A | L);                    /* level 4 */
            if (CLK) Q = M                   /* level 5 */
            if (K´ == K)                     /* asynchronous signal */
                stable = TRUE;
            else
                stable = FALSE;
            K´ = K;
        } while (not stable);
        write_results ();
    } while (true);
}
```

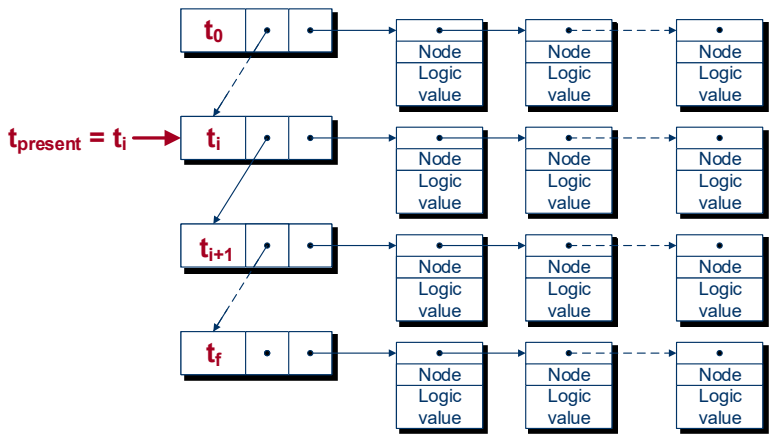

24

Event-Driven Simulation: Overview

- Models of the primitives: gate level or functional level
- Driven by events
- Path selection
- Disadvantages
  - Requires a more complex data structure – dynamic structure
- Advantages
  - Only simulates the elements potentially active (path selection)
  - Allows the simulation of combinatorial and sequential circuits
  - Allows accurate delay processing

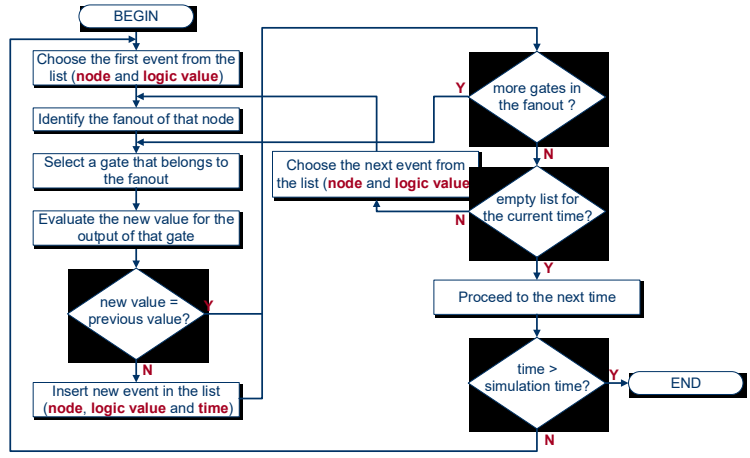
25

Event-Driven Simulation: List of Events



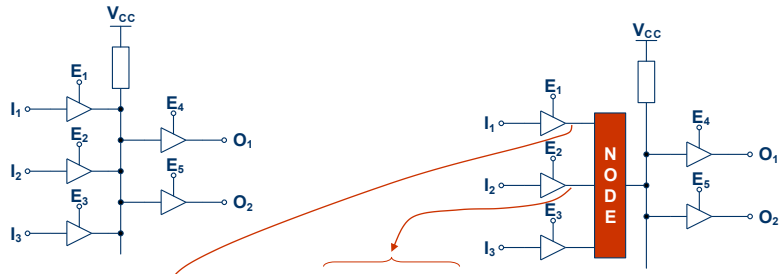
26

Event-Driven Simulation: Algorithm



27

Event-Driven Simulation: Buses



NODE	0	1	U	Z
0	0	U	U	0
1	U	1	U	1
U	U	U	U	U
Z	0	1	U	1

Necessary to include the logic value **Z**

The table processes two inputs at once. In this example, it has to be used twice.

28