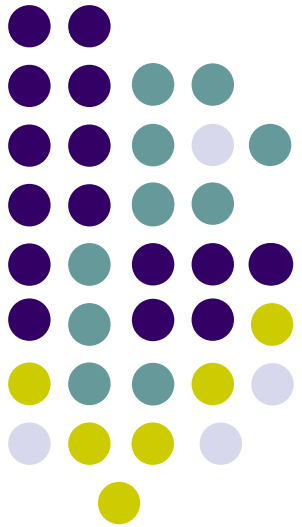
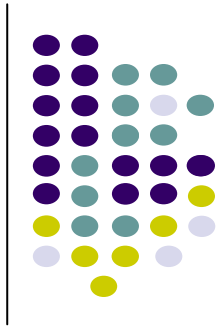
VLSI System Design
(Graduate Level)

Digital Design with HDL-Verilog(1)

[Material adapted from “Advanced Digital Design with the Verilog HDL” by M.D. Ciletti and lecture notes of Prof. KJ Lee]



Outline



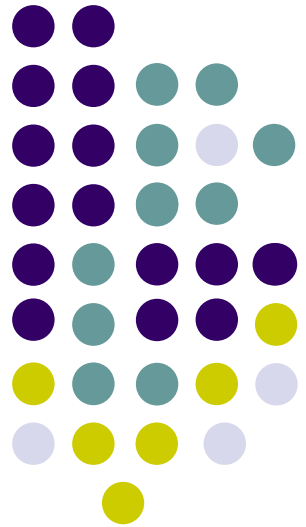
- History of Verilog
- Design-Order Strategy
- Structural Models of Combinational Logic
- Logic Simulation, Design Verification and Test Methodology
- Propagation Delay
- Truth Table Model

[Reading: Ciletti's Chap 4]

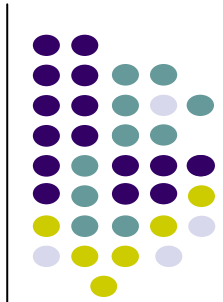
What you shall know after this course

- Basic rules of Verilog language
- Writing a combinational module using
  - Schematic/gates
  - Truth tables
- Writing a testbench to verify a module

History of Verilog

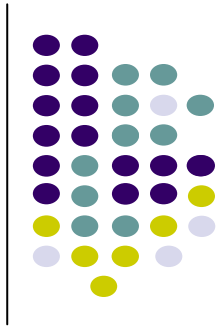


Hardware Description Languages (HDL)



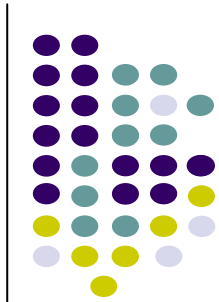
- Similarity and Uniqueness
  - Similar to general-purpose languages like C
  - Additional features
    - Modeling and simulation of the functionality of combinational and sequential circuits
    - Parallel vs. sequential behaviors
- Two competitive forces
  - Verilog: C-like language
  - VHDL: ADA-like language

What HDL can do?



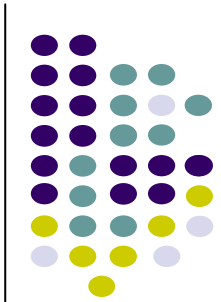
- Simulate the behavior of a circuit before it is actually realized.
  - Describe models in common language
  -
  - Execute the models as if they are hardware
  -
  - Be translated into gate-level designs
  -

Development of Verilog



1984	Gateway Design Automation, Phil Moorby
1986	Verilog-XL: an efficient gate-level simulator
1988	Verilog logic synthesizer, Synopsys
1989	Cadence Data System Inc. acquired Gateway
1990	Verilog HDL is released to public domain
1991	Open Verilog International (OVI)
1994	IEEE 1364 Working group
1995	December : Verilog becomes an IEEE standard (IEEE Std. 1364)
2001	SystemVerilog

Verilog HDL and Verilog-XL



Verilog HDL

A hardware description language that allows one to describe circuits at different levels of abstraction.

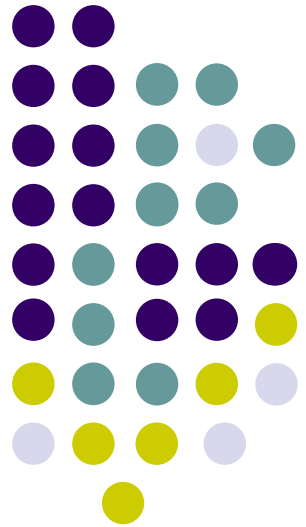
Verilog-XL software:

A high speed, event-driven logic simulator for Verilog HDL

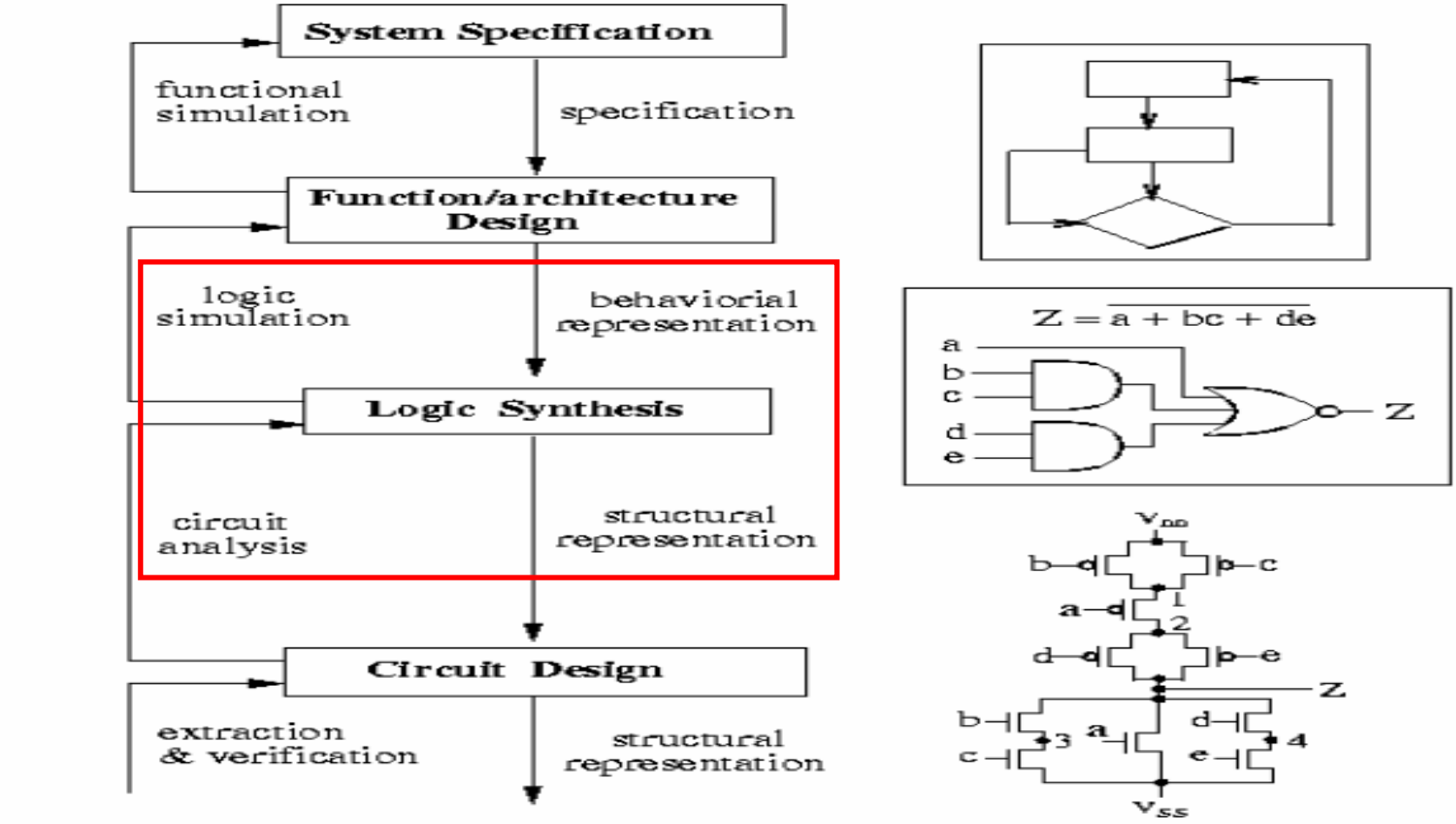
- **Event-driven simulator : Evaluate an element only when its inputs change states. Most practical and widely used simulation algorithm.**

- **Verilog-XL : Incorporates Turbo algorithm, XL algorithm, Switch-XL Algorithm, Caxl algorithm.**

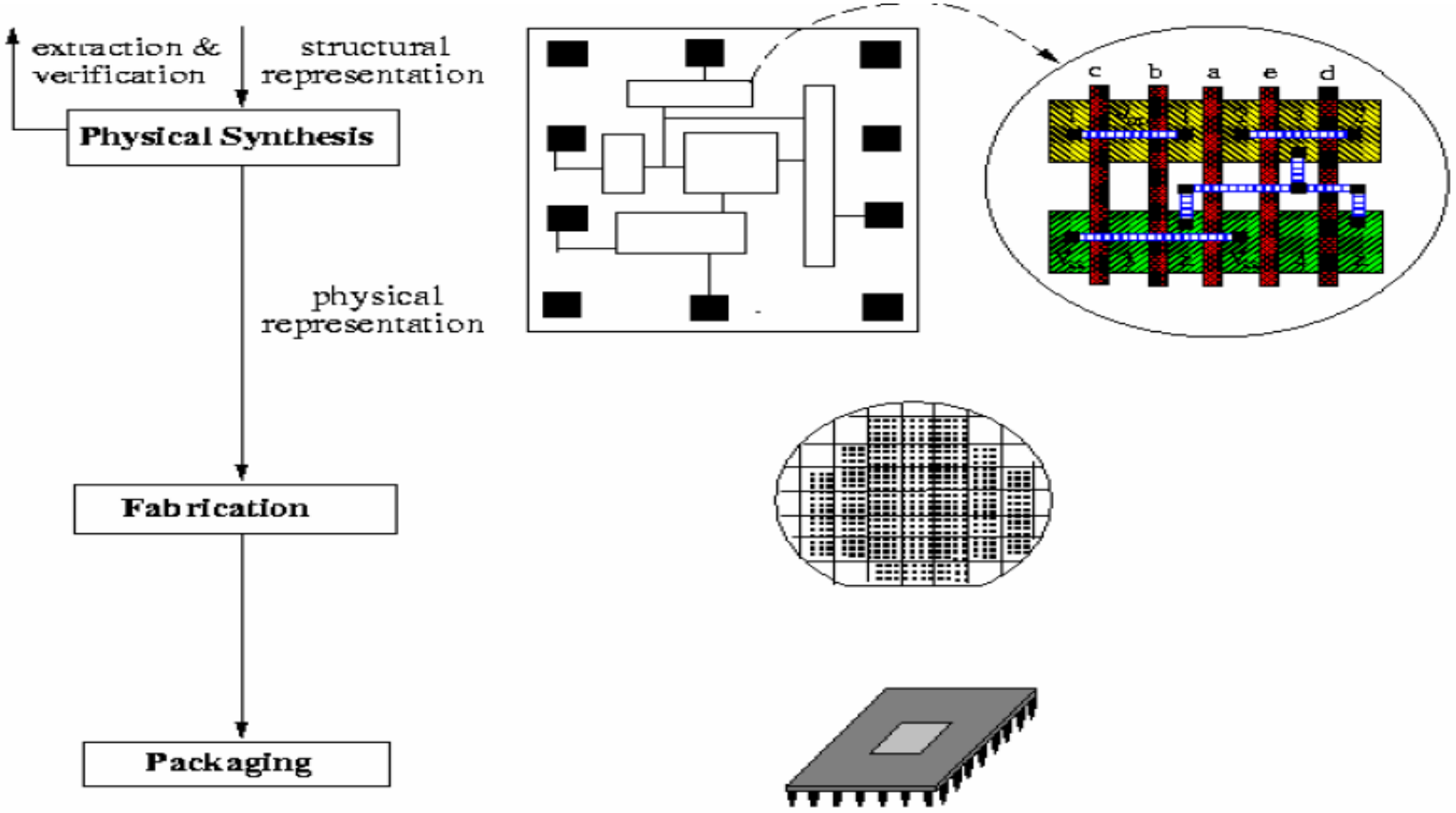
Design Order Strategies



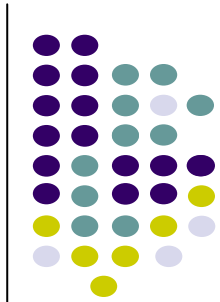
Typical VLSI Design Cycle



Typical VLSI Design Flow (Cont'd)

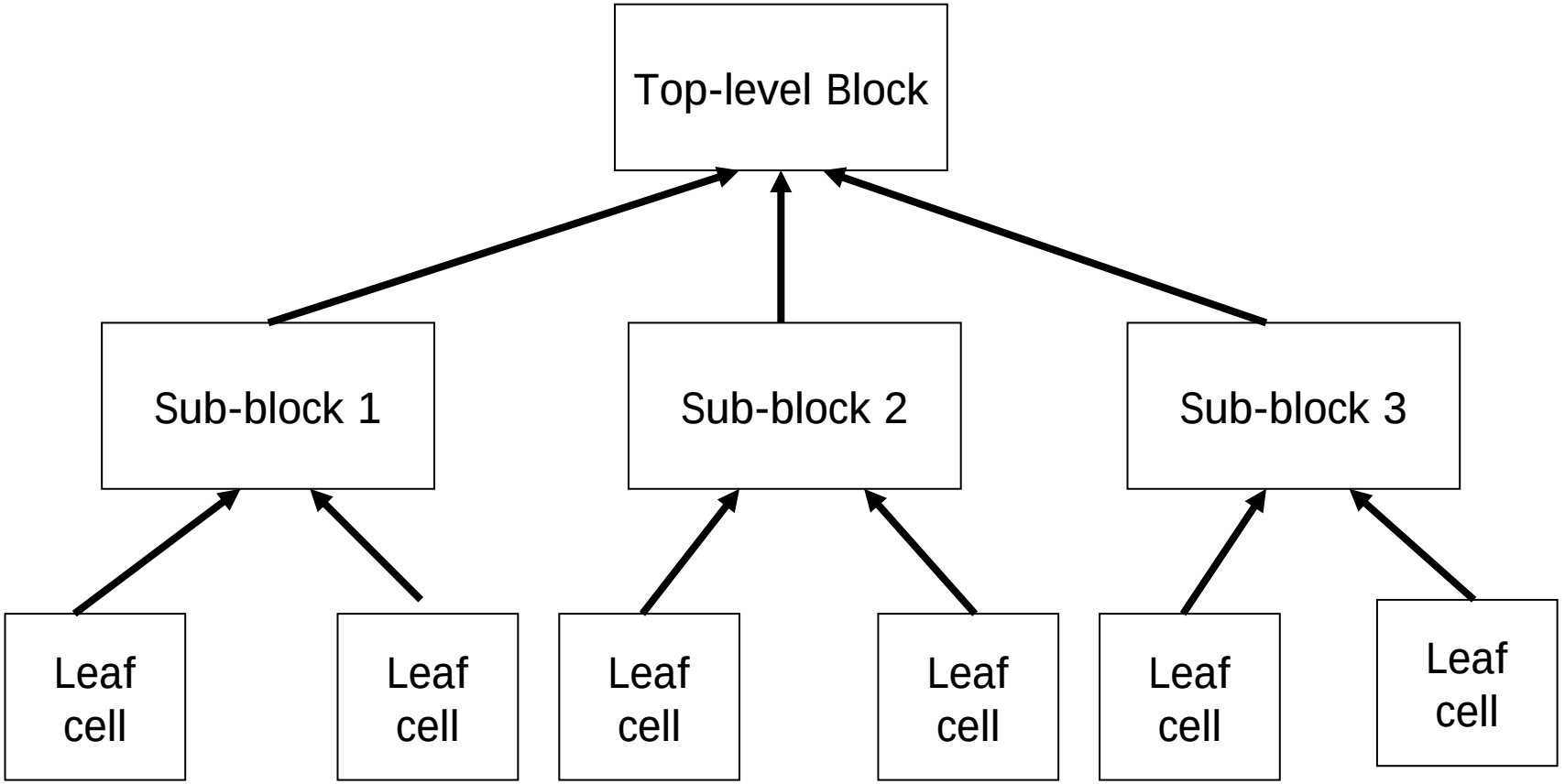


Hierarchical Modeling Concepts

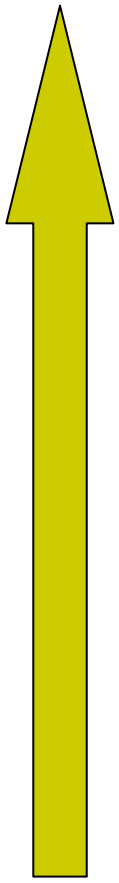


- Bottom-up
  - Identify building blocks available
  - Build bigger cells using these building blocks
  - These cells then used for higher-level blocks
- Top-down design methodologies
  - Define the top-level block first
  - Identify the sub-blocks necessary to build the top-level block
  - Further subdivide the sub-blocks until leaf cells
- Hybrid
  - A combination of top-down and bottom-up flows

Bottom-up Design



Bottom-Up Design Flow

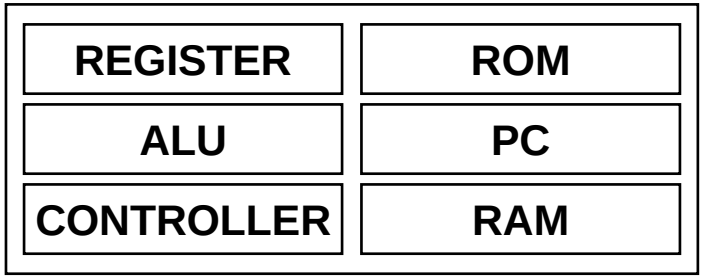


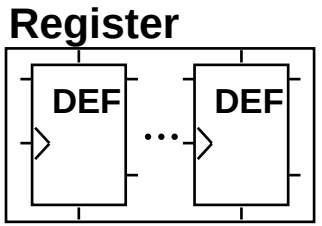
4 **Timing, performance, and testability are analyzed.**

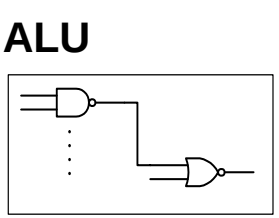
3 **The complete design is modeled and tested.**

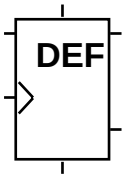
2 **Intermediate level components are modeled at the gate level and tested.**

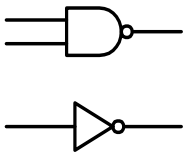
1 **Lowest level components are modeled at the gate or circuit level and tested.**



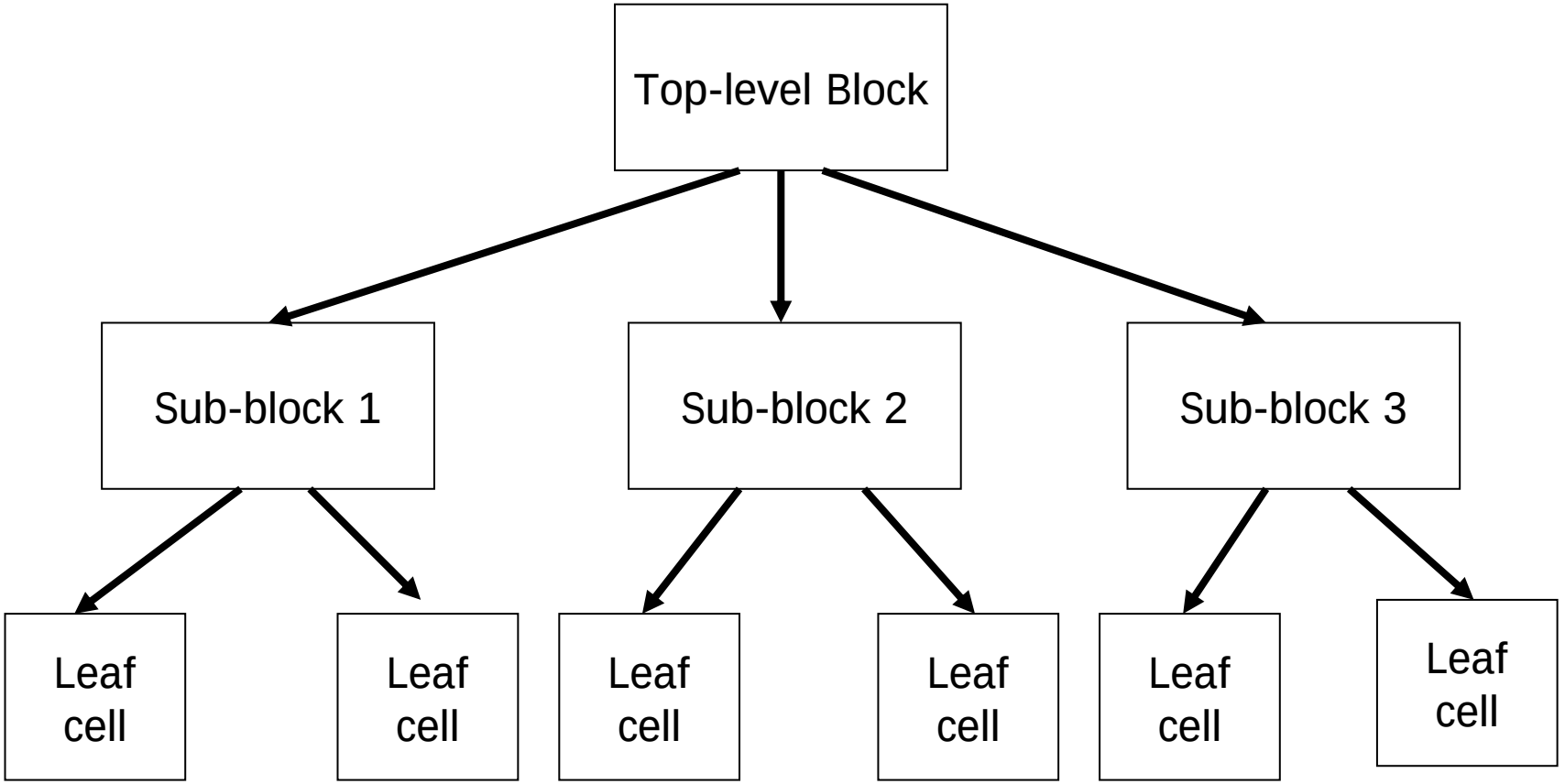




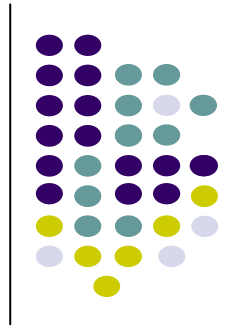


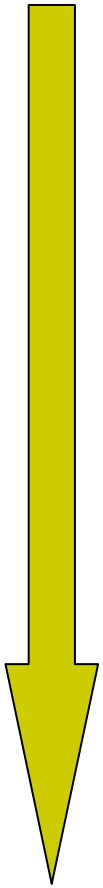


Top-down Design



Top-Down Design Flow

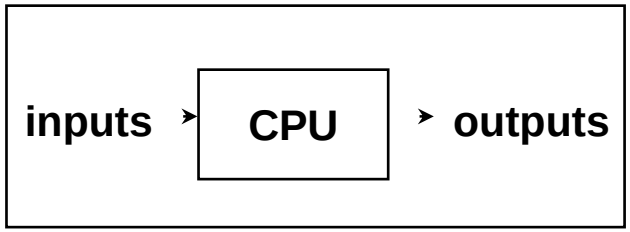


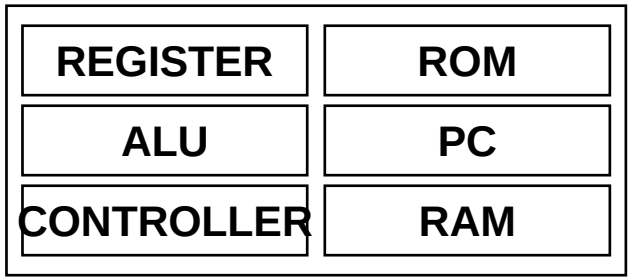


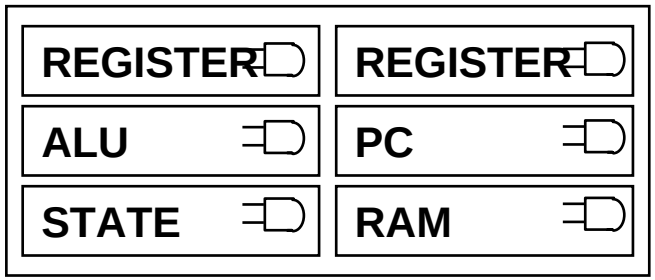
1 **The top-level system is modeled using a high-level behavioral description.**

2 **Each major component is modeled and simulated at the register transfer level.**

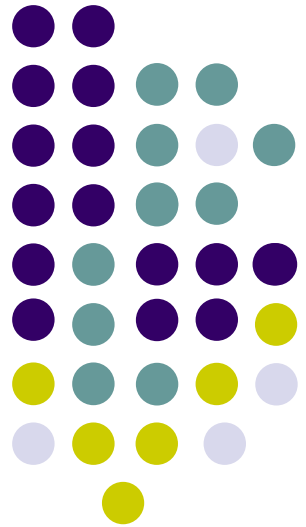
3 **Each major component is modeled and simulated at the gate level.**





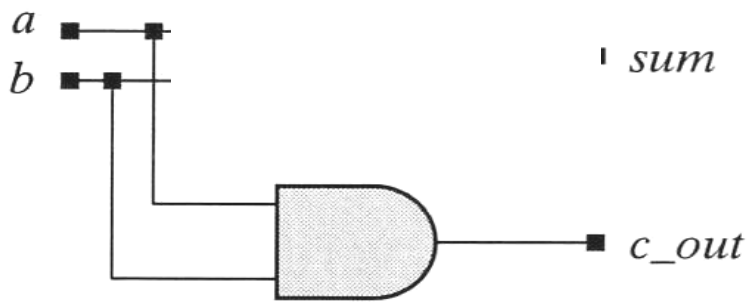


Structural Models of Combinational Logic



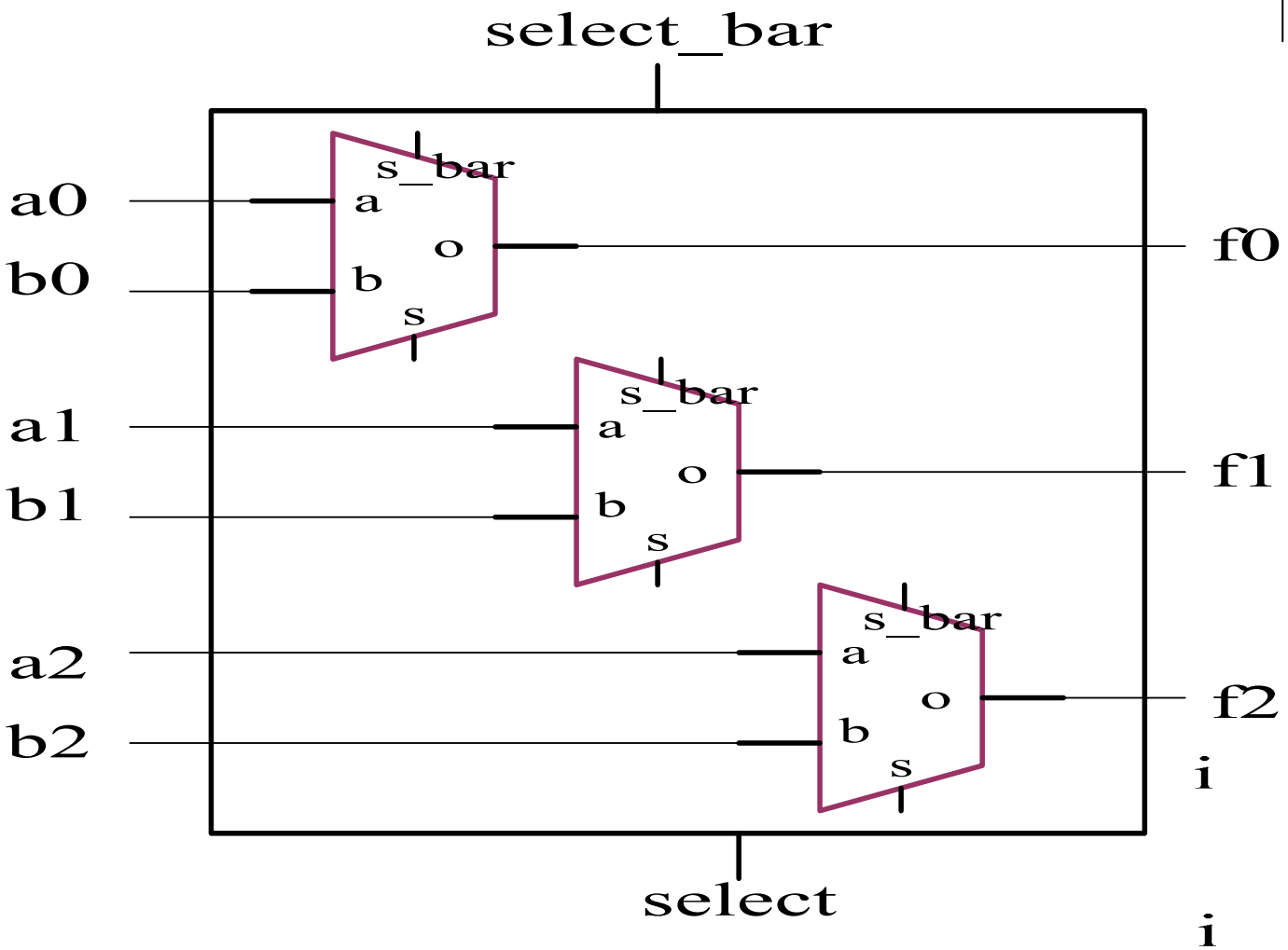
Example: Half-adder

a	b	sum	c_out
0	0		
0	1		
1	0		
1	1		

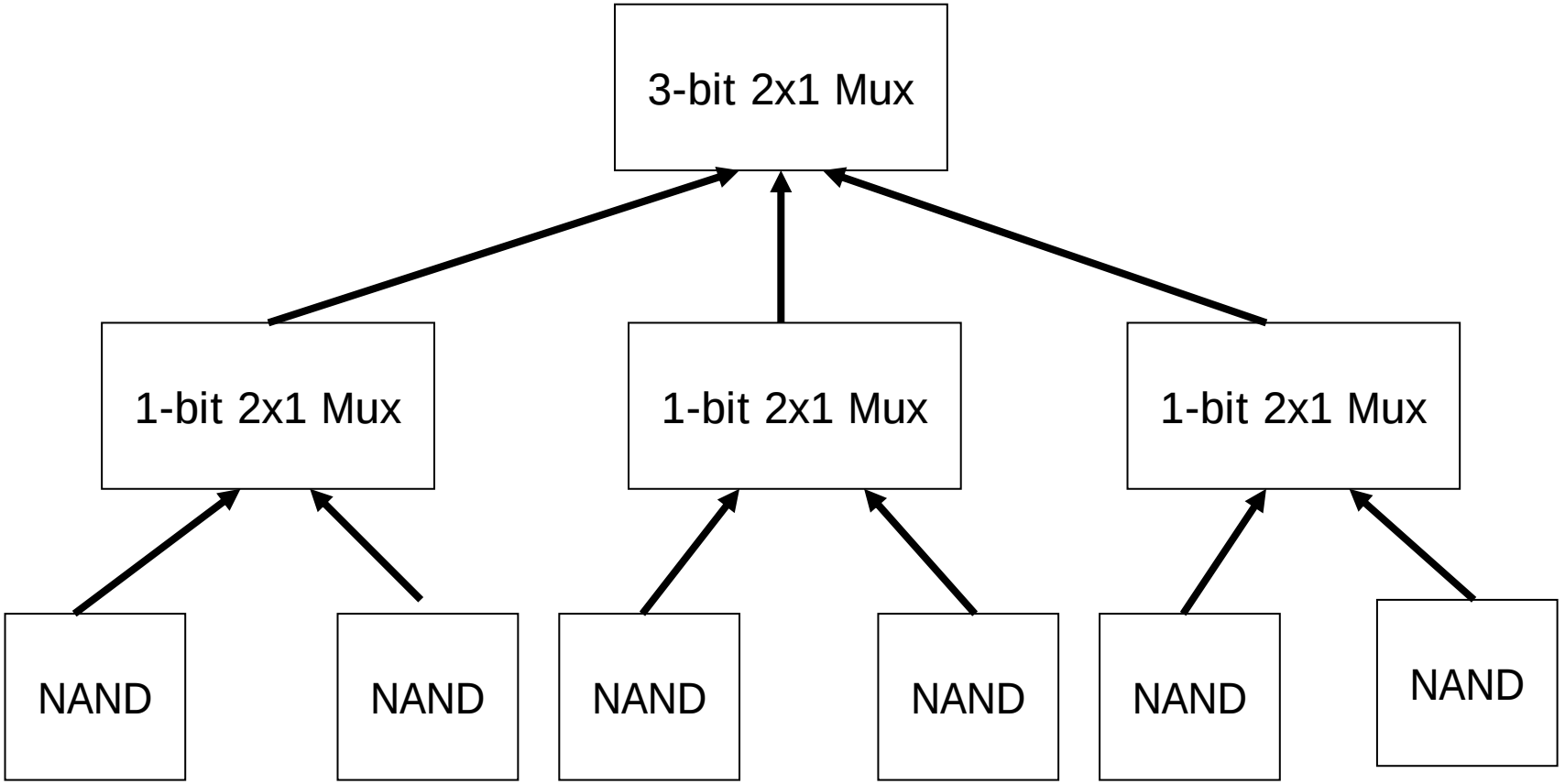



```
module   Add_half (sum, c_out, a, b);
  input      a, b;
  output     c_out, sum;

  and        (c_out, a, b);
endmodule
```

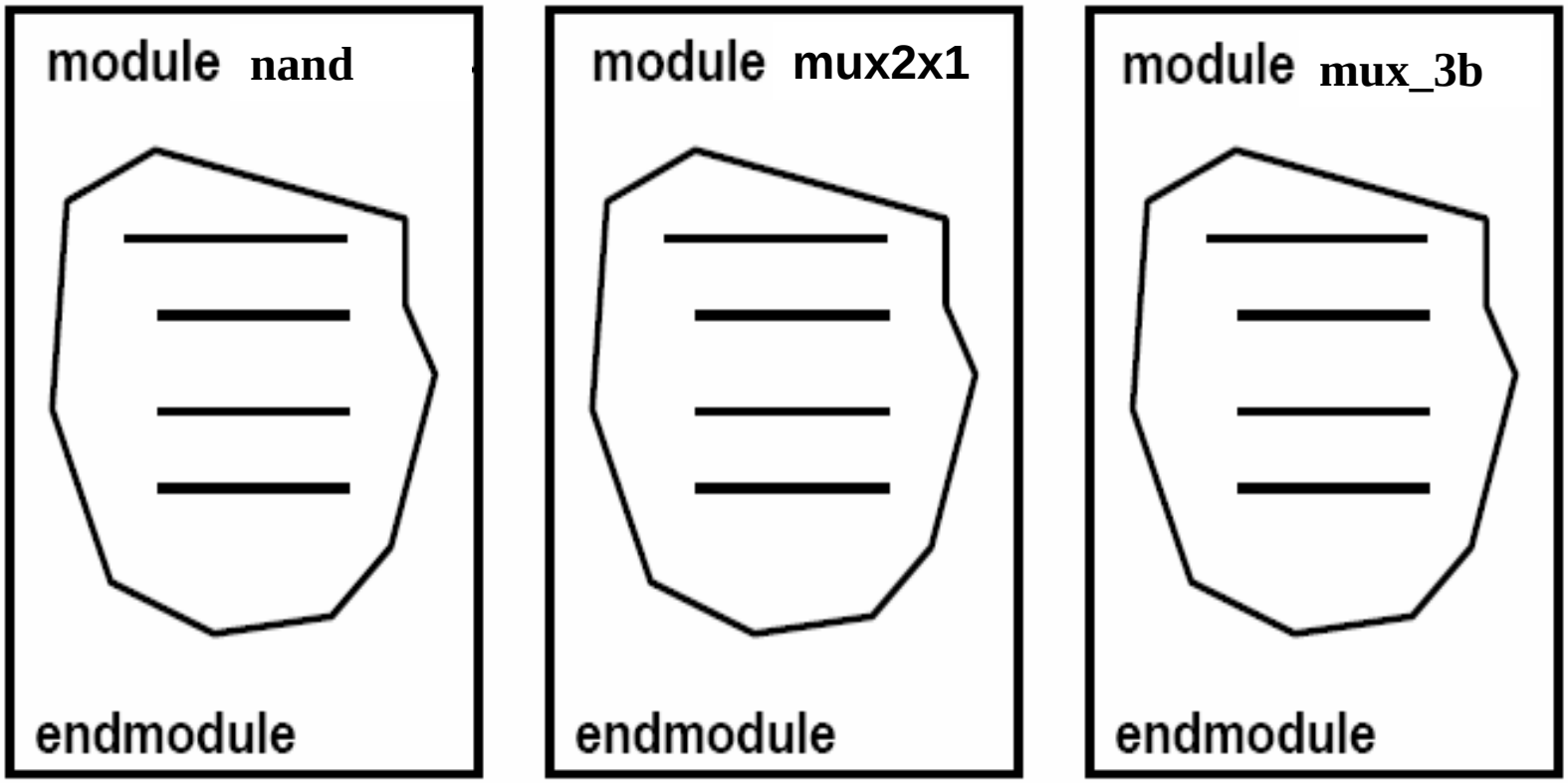
3-bit mux



3-bit 2x1 Multiplexier



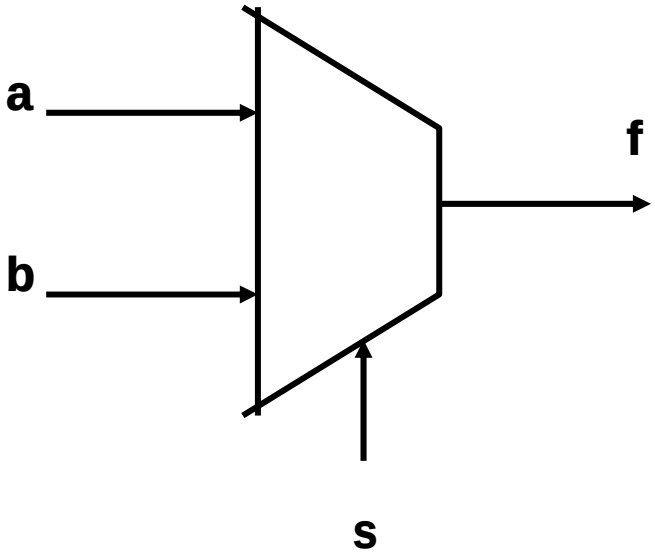
Concepts of Modules



Design of a 2x1 Multiplexier

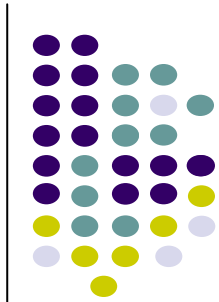
A multiplexier:

select a signal from available inputs



s	a	b	f
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

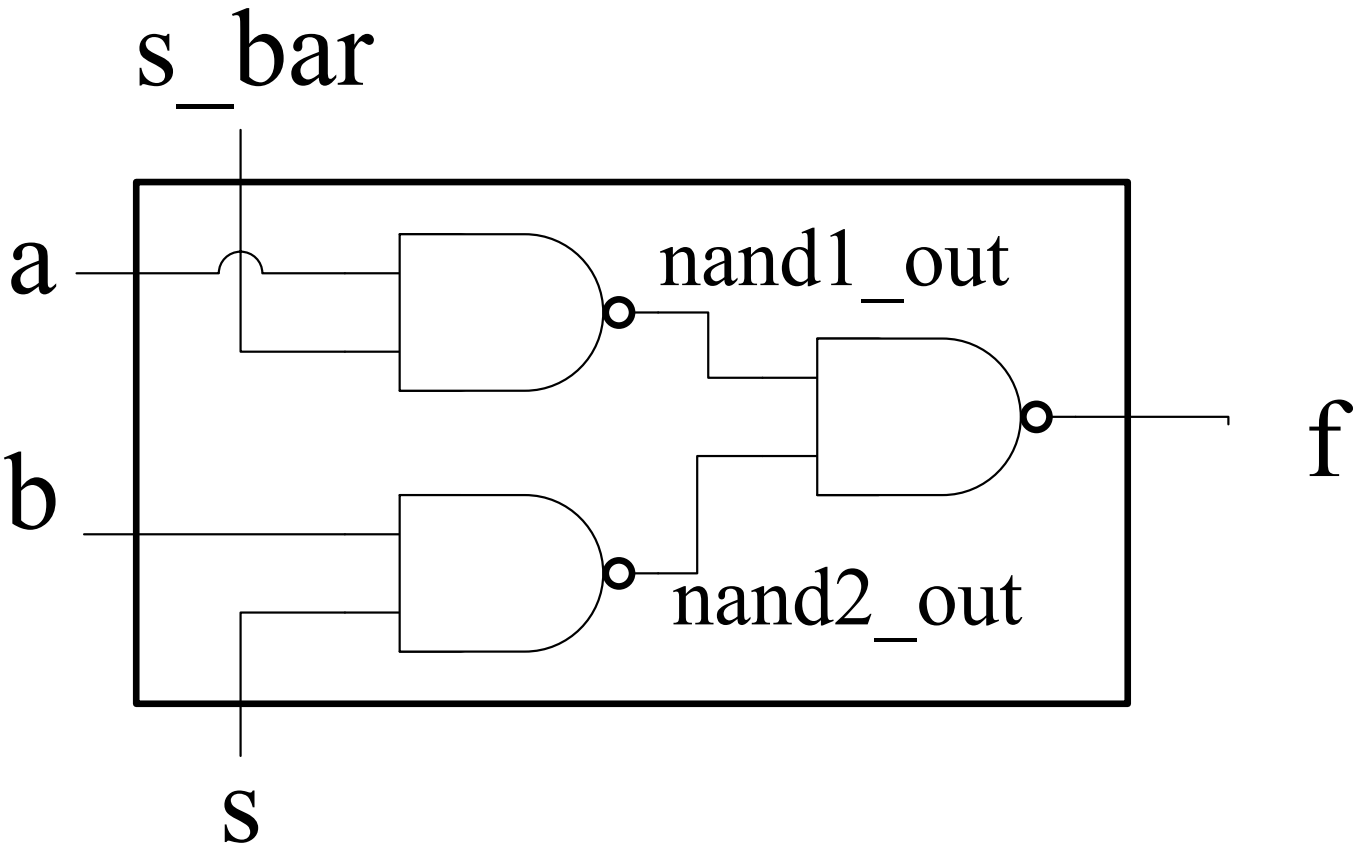
Boolean Eq. of a MUX



- F = s'a + sb
- Implement using NAND gates

- F = ?

One-bit mux

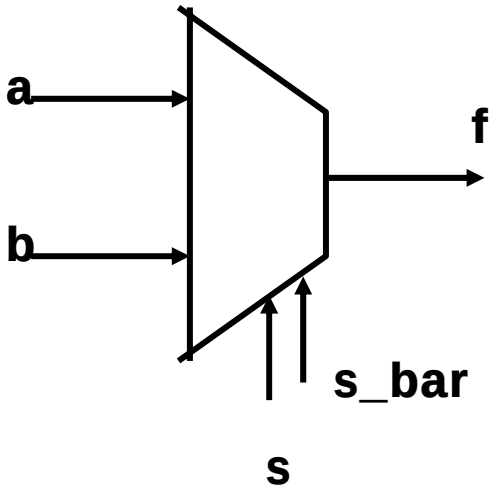


Verilog Implementation

Main frame

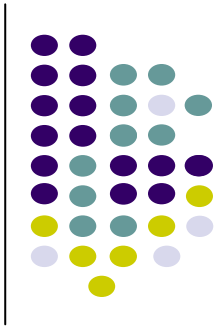
Variable declaration

Main body




```
module mux2x1(f, s ,s_bar, a, b);
    output  f;
    input  s, s_bar;
    input  a, b;
    wire  nand1_out, nand2_out;
  // boolean function
    nand(nand1_out, s_bar, a);
    nand(nand2_out, s, b);
    nand(f, nand1_out, nand2_out);
endmodule
```

Connectivity

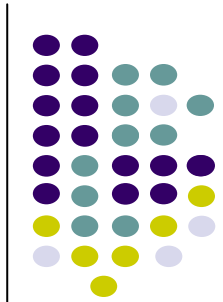


- Wire
  - Its value is determined during simulation
  - The default type for an undeclared identifier
- Port
  - Position sensitive

    Add_half_0_delay M1(w1, w2, a, b);
  - Explicitly declared

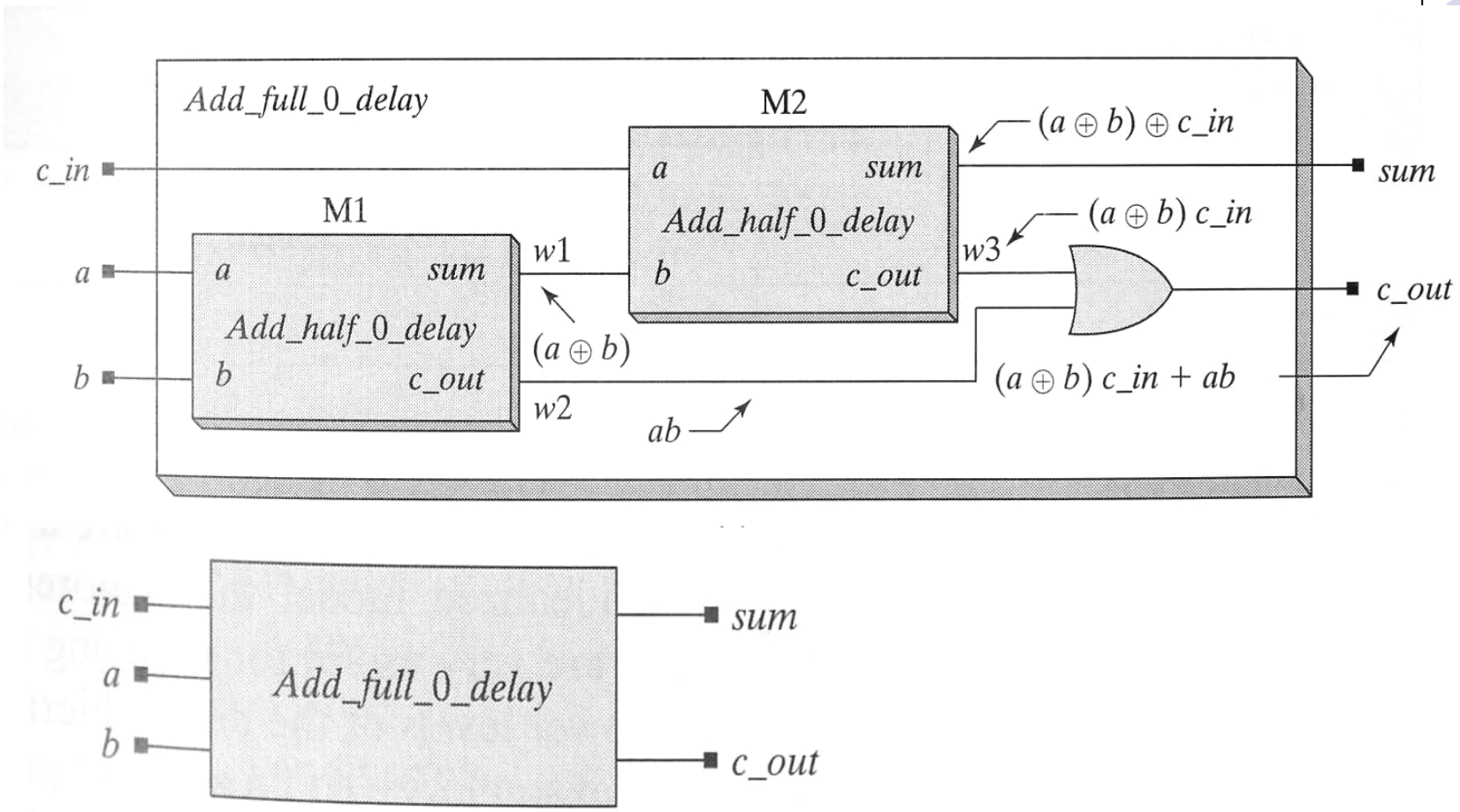
    Add_half_0_delay M1(.b(b), .c_out(w2), .a(a), .sum(w1));

Verilog Implementation



```
module mux_3b (f2,f1,f0,a0,b0,a1,b1,a2,b2,
select,select_bar);

   input a0,b0,a1,b1,a2,b2;
   input select, select_bar;
   output f2,f1,f0;

   mux2x1   M0(f0,select, select_bar, a0,b0);
   mux2x1   M1(f1,select, select_bar, a1,b1);
   mux2x1   M2(f2,select, select_bar, a2,b2);

endmodule
```

Full adder


Add_full_0_delay
M2
(a ⊕ b) ⊕ c_in
c_in
a
sum
Add_half_0_delay
b
c_out
sum
M1
(a ⊕ b) c_in
w3
a
sum
w1
Add_half_0_delay
c_out
b
(a ⊕ b)
(a ⊕ b) c_in + ab
w2
ab
c_in
a
b
Add_full_0_delay
sum
c_out

Coding Notes

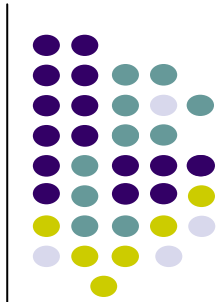
```
module Add_full_0_delay(sum, c_out, a, b, c_in);
   output                sum, c_out;
   input                 a, b, c_in;
   wire                  w1, w2, w3;

   Add_half_0_delay      M1 (w1, w2, a, b);
   Add_half_0_delay      M2 (sum, w3, w2, c_in);
   or                    M3 (c_out, w2, w3);
endmodule
```

Module instance name

Example: 4-bit RCA

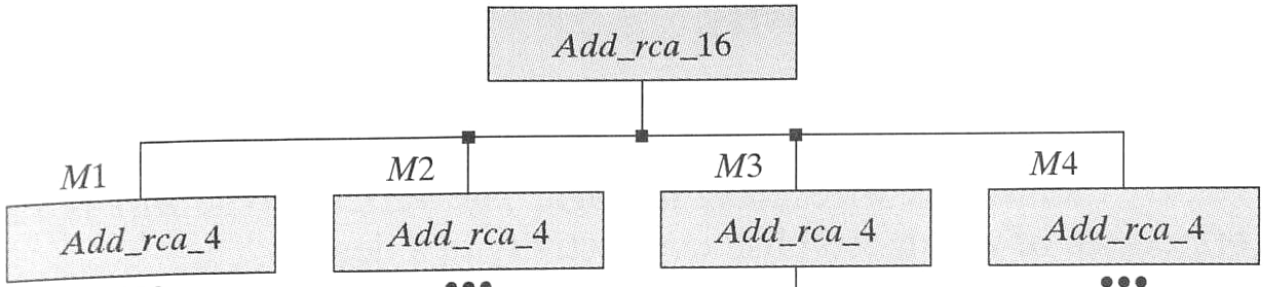
```
module Add_rca_4 (sum, c_out, a, b, c_in);
   output      [3: 0]     sum;
   output                 c_out;
   input       [3: 0]     a, b;
   input                  c_in;
   wire                   c_in2, c_in3, c_in4;

   Add_full M1 (sum[0],   c_in2,   a[0], b[0], c_in);
   Add_full M2 (sum[1],   c_in3,   a[1], b[1], c_in2);
   Add_full M3 (sum[2],   c_in4,   a[2], b[2], c_in3);
   Add_full M4 (sum[3],   c_out,   a[3], b[3], c_in4);
endmodule
```

Vectors in Verilog

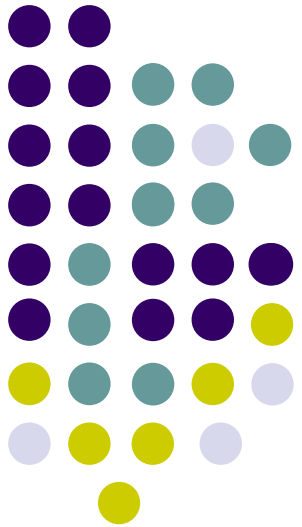


- e.g., output[15:0] sum;
- Leftmost index bit is the most significant bit
- Rightmost index bit is the least significant bit
- If out of bounds, x is returned
  - X: unknown value

Example: 16-bit RCA

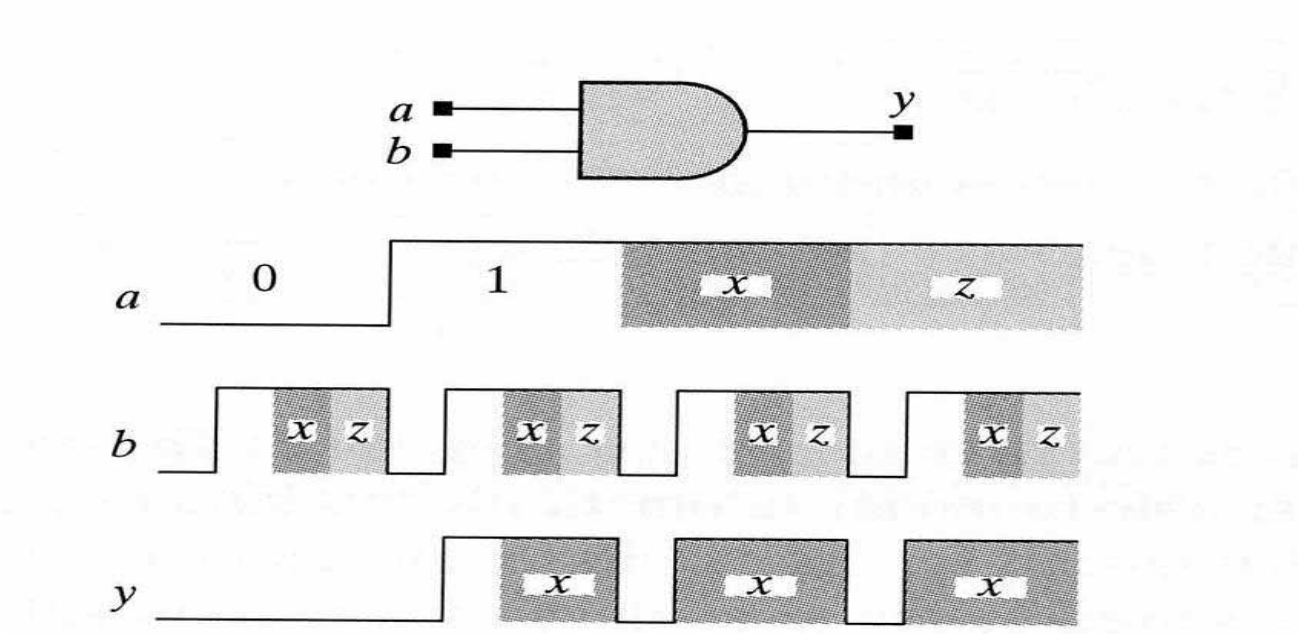


Logic Simulation, Design Verification and Test Methodology

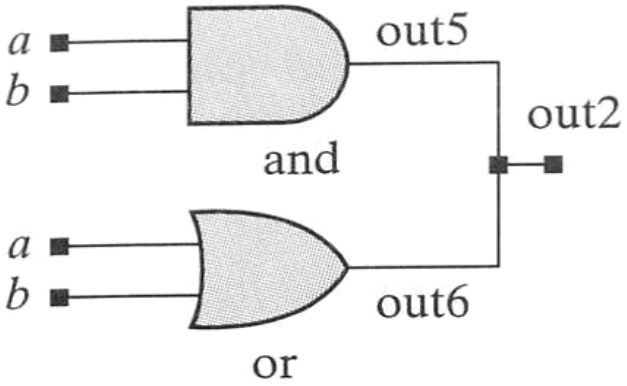


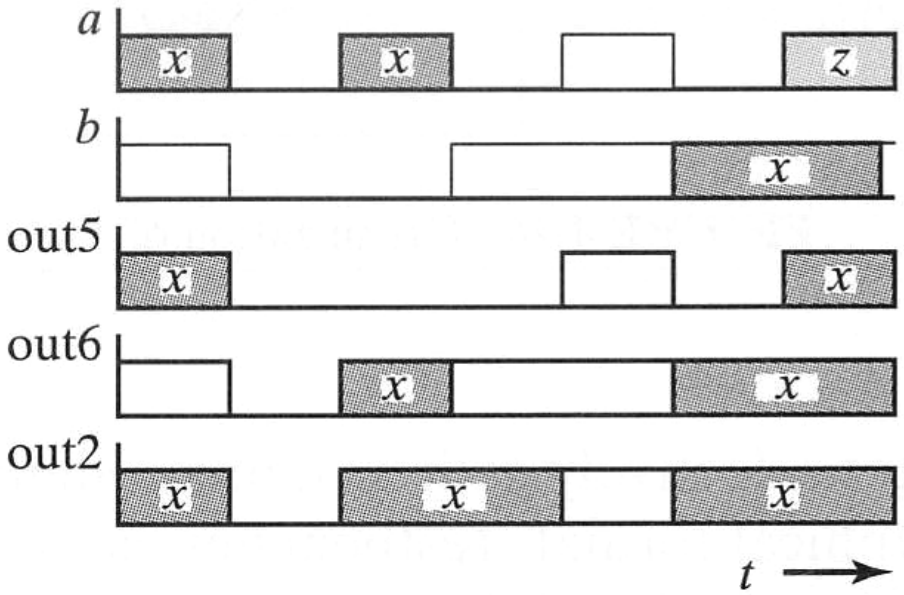
Logic values of the signal

- 4 valued logic: 0, 1, x, and z
  - x: unknow
  - z: high impedance



Signal Contention and Resolution



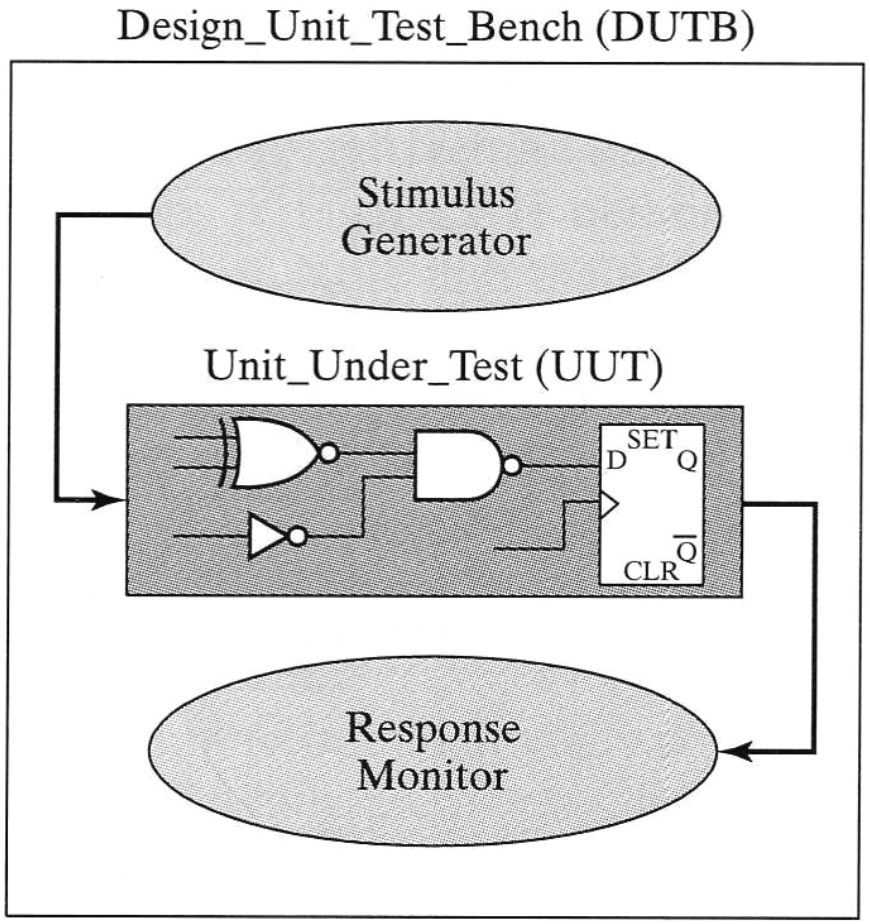


Test Methodology

- Input pins of 16-bit RCA: 33
- Require 2^{33} input patterns ➔ difficult to realize

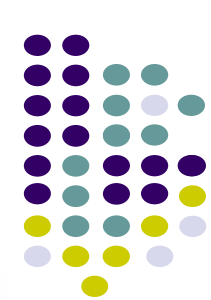
- Suggestions
  - Verify basic components first: half-adder/ fulladder
  - Then composite components: 4-bit slice units
  - Chose set of patterns carefully

Testbench



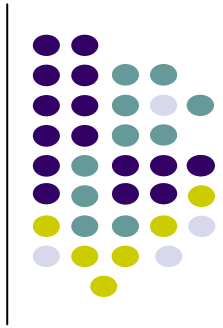
- Generate test patterns
  - Waveforms to inputs
- Timing of applying patters
  - Delay in signals
  - Clock generation
- Start/end time for simulation
  - $initial
  - $finish

Example 4.7



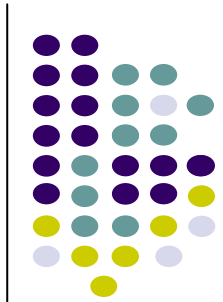
```
module t_Add_half();
   wire          sum, c_out;
   reg           a, b;
   Add_half_0_delay   M1 (sum, c_out, a, b);       // UUT
   initial begin                                   // Time Out
   #100   $finish;
   end

   initial begin
   #10    a = 0; b = 0;
   #10    b = 1;
   #10    a = 1;
   #10    b = 0;
   end
endmodule
```

Signal Generators



- *initial*:
  - A single-pass behavior
  - Let the simulator starting from $t_{sim} = 0$
  - Procedural statements enclosed in begin … end
- Delay time
  - Proceeding the statement: #10 b = 1;
  - Delay the execution until specified time
- Signal type: *reg*
  - Retain its value from the moment assigned by the procedural statement until change by the next statement
  - Initially given the value ***x***
- Stop: *$finish* == return control to the OS

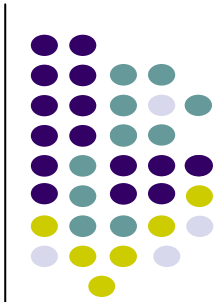
Test Fixture Template



```
module testfixture ;

    // Data type
    declaration


    // Instantiate modules


    // Apply stimulus


    // Display results
endmodule
```

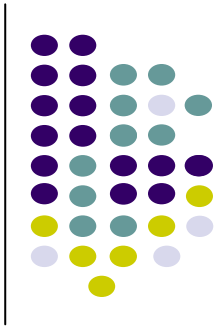
⇨ **There are no ports for test fixture**

Test Fixture—Making an Instance



```
module testfixture ;

// Data type declaration

// MUX instance
      MUX2_1 mux (out, a, b, sel) ;
// Apply stimulus

// Display results

endmodule
```

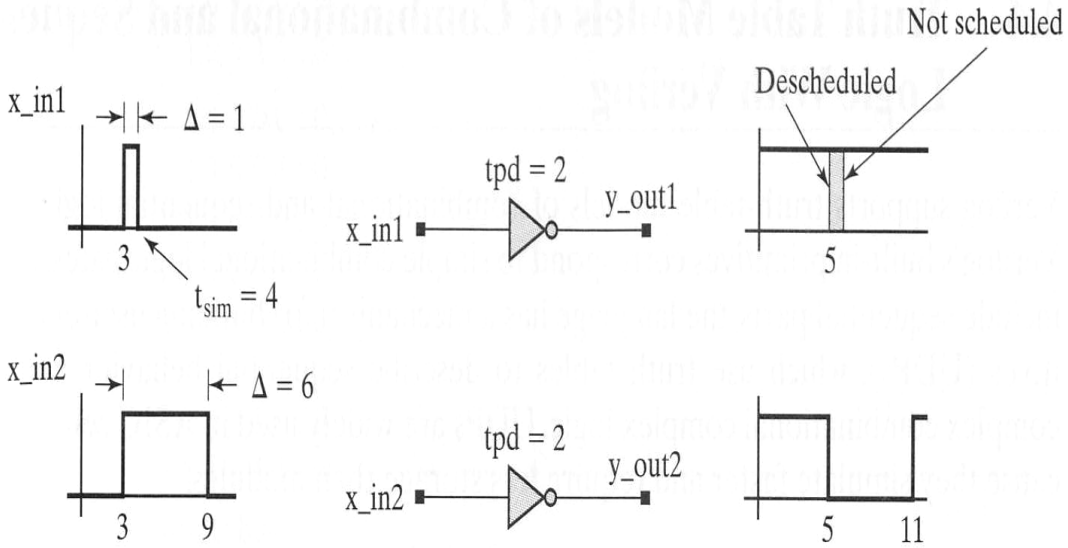
Propagation Delay



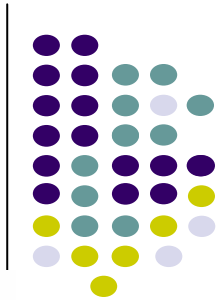
- For functionality, verification with 0 delay is quick.
- For physical logic gate, verification with unit delay can expose the time sequence of signal activity.
- For timing verification, simulation must obtain cell delay to account for real delay.

Types of Delay

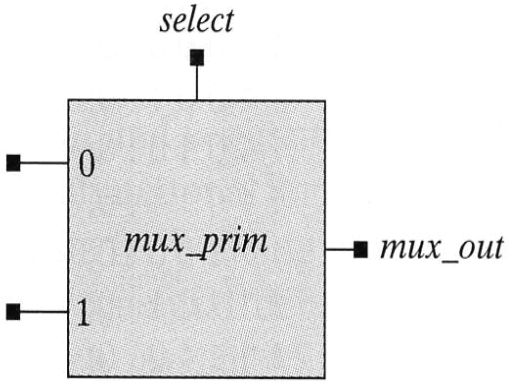
- Inertial Delay
  - Delay due to gate itself
  - Input pulse width must larger than
- Transport delay
  - Delay due to the a wire of circuit



Truth Table Models



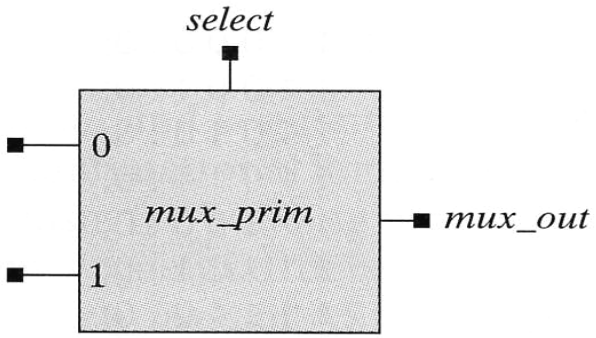
```
primitive AOI_UDP (y, x_in1, x_in2, x_in3, x_in4, x_in5);
   output      y;
   input x_in1, x_in2, x_in3, x_in4, x_in5;

table
// x1 x2 x3 x4 x5
0 0 0 0 0 : 1;
0 0 0 0 1 : 1;
0 0 0 1 0 : 1;
0 0 0 1 1 : 1;
0 0 1 0 0 : 1;
0 0 1 0 1 : 1;
0 0 1 1 0 : 1;
0 0 1 1 1 : 0;
0 1 0 0 0 : 1;
0 1 0 0 1 : 1;
0 1 0 1 0 : 1;
0 1 0 1 1 : 1;
0 1 1 0 0 : 1;
0 1 1 0 1 : 1;
0 1 1 1 0 : 1;
0 1 1 1 1 : 0;
```



- Build user-defined primitives (UDP)

Mux_prim

```
primitive mux_prim (mux_out, select, a, b);
   output      mux_out;
   input       select, a, b;
   table

// select        a        b        :    mux_out

     0           0        0        :    0;
     0           0        1        :    0;
     0           0        x        :    0;
     0           1        0        :    1;
     0           1        1        :    1;
     0           1        x        :    1;

// select        a        b        :    mux_out

     1           0        0        :    0;
     1           1        0        :    0;
     1           x        0        :    0;

     1           0        1        :    1;
     1           1        1        :    1;
     1           x        1        :    1;

     x           0        0        :    0;
     x           1        1        :    1;

     endtable
endprimitive
```



- Reduce ambiguousness in last two rows