VLSI System Design
(Graduate Level)

QuickStart – Verilog Basics

Fall 2008

NCKU EE
LY Chiou

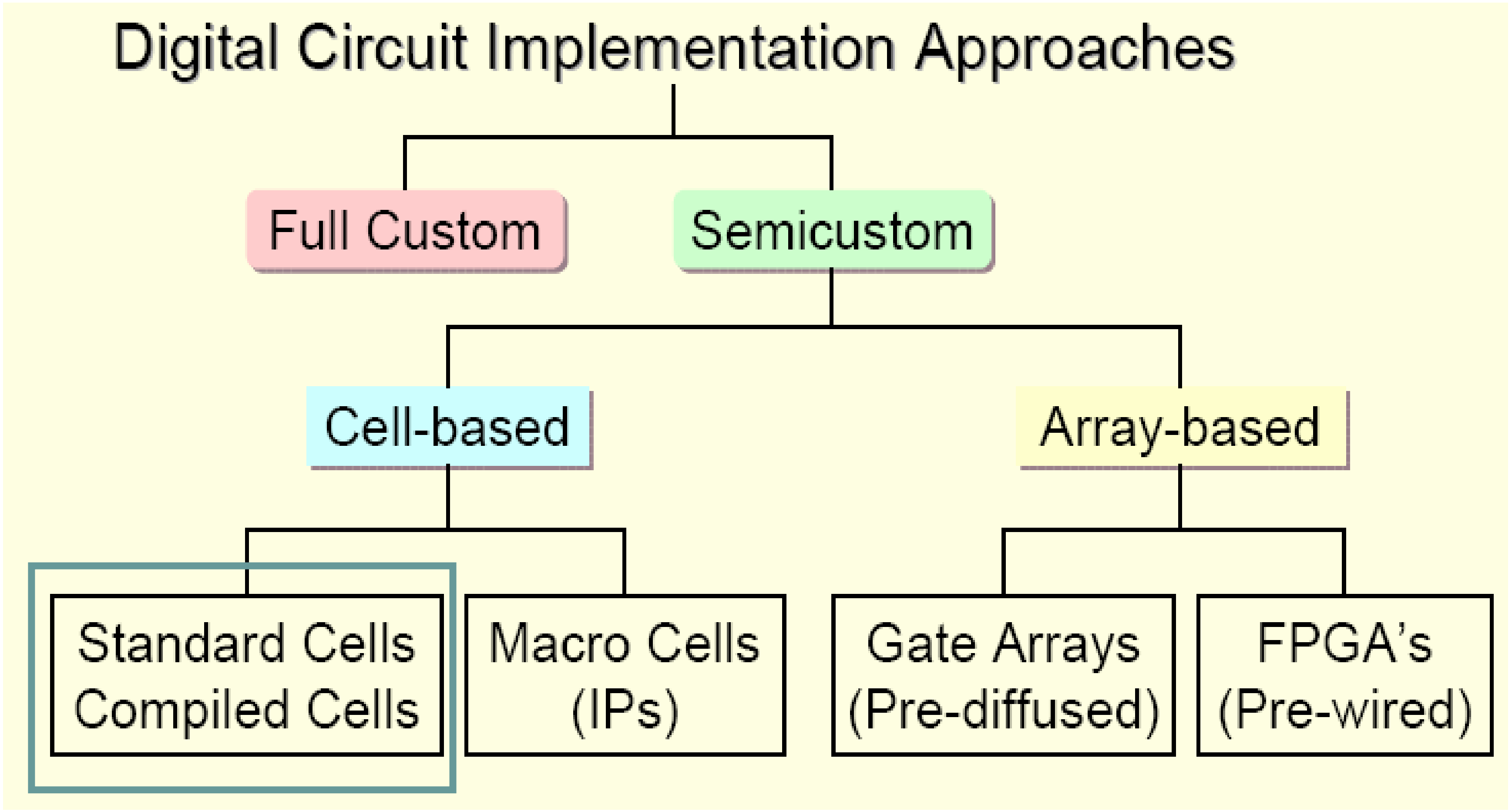
Outline

- Overview
- From RTL to Gate Level
- Quick Start for Verilog
- 9/25 Tutorial
  - Basics of Unix, Environment in SoC Lab, Verilog-XL simulator, Debussy, Design Vision
  - 9:10-10:00 Overview
  - 10:10-noon Hand-on practices @CIC or SoC Lab
    - If your last digit of your student ID is even, please go to CIC.
    - If odd, go to SoC Lab, please.
  - Watch video last year first

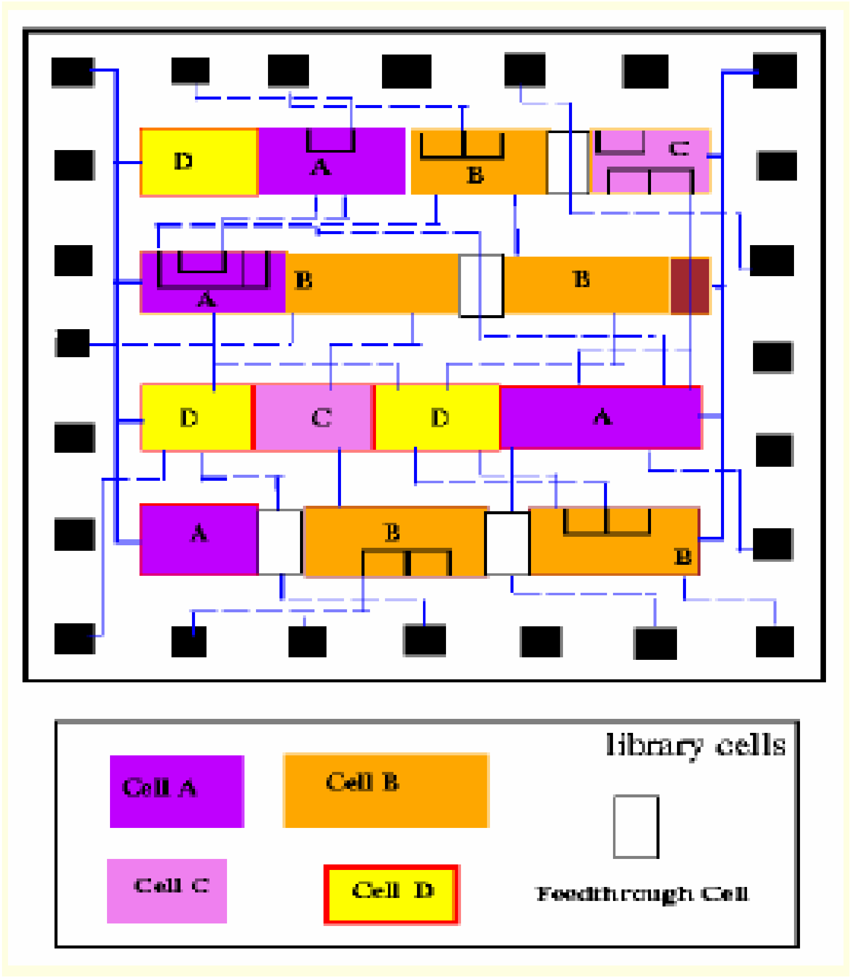
Things to Learn

- What position is the course in the digital design flow?
- What is RTL?
- What is hierarchical design?
- What is testbench?
- What are basic elements of Verilog language?

Target Digital Design Implementation


Digital Circuit Implementation Approaches
Full Custom
Semicustom
Cell-based
Array-based
Standard Cells Compiled Cells
Macro Cells (IPs)
Gate Arrays (Pre-diffused)
FPGA's (Pre-wired)

Standard Cell Design Style

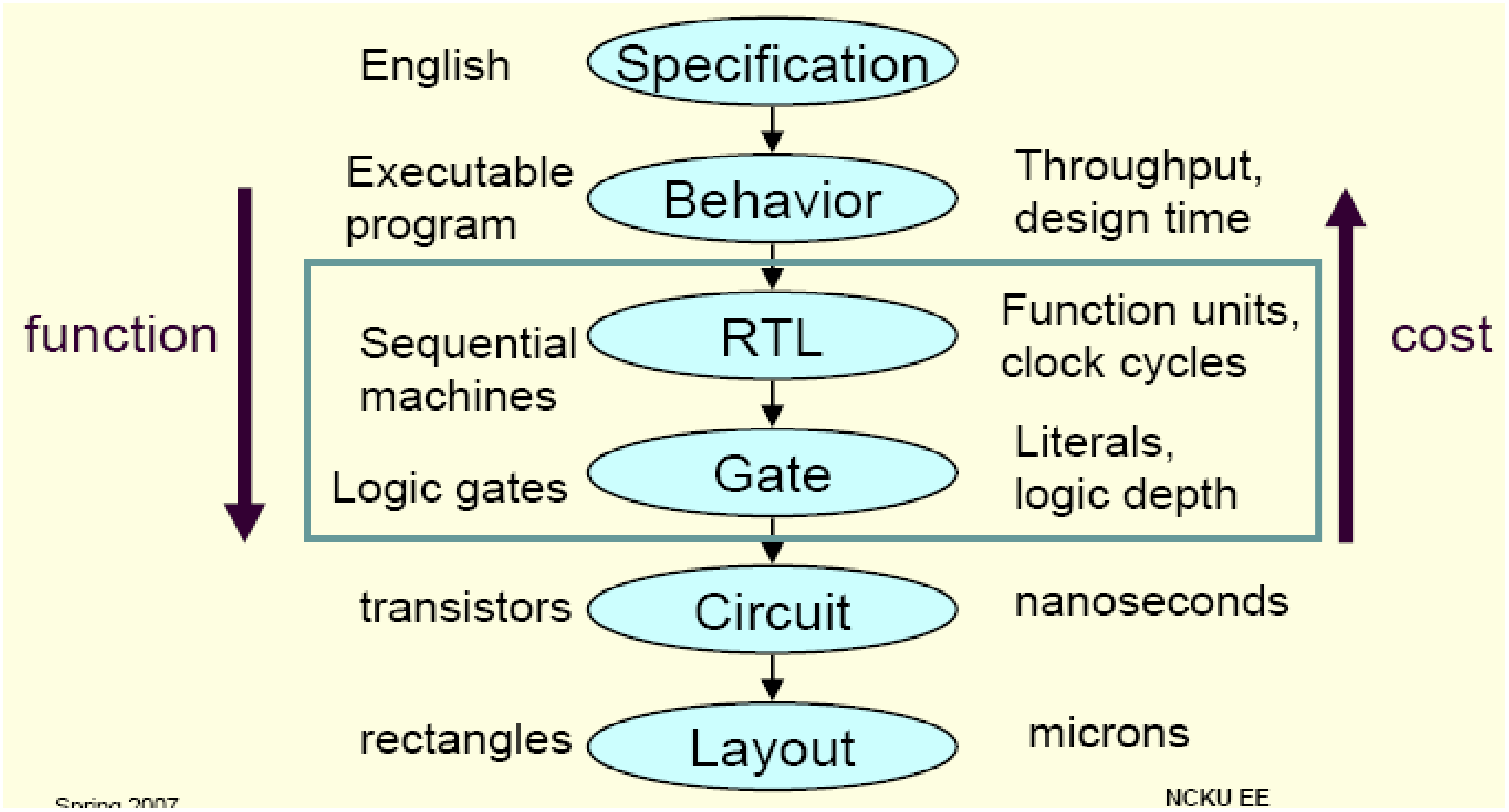


Selects pre-design cells (of the same height) to implement logic

These cells may be

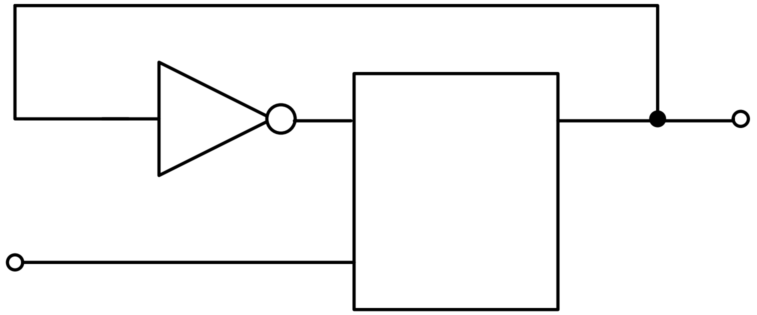
- Logic gates: nand2, NOT, and2, and4, nor2, mux2, decoder etc.
- Latches: latchx1, latchrx2, …
- Flip-flops: pdffx1, pdffx2,…
- Basic blocks: fulladder

Target Levels of Design Abstractions

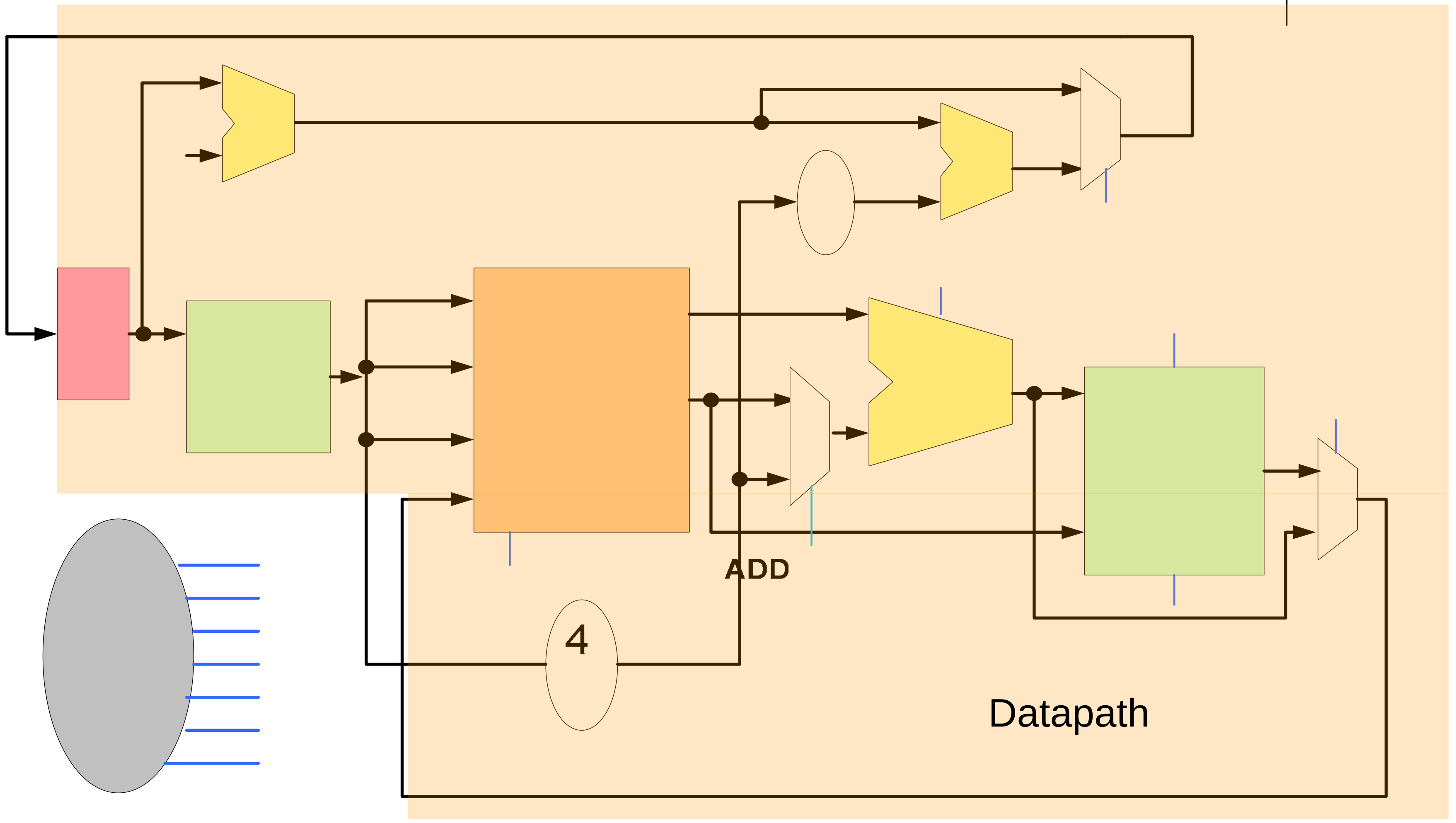


Register-Transfer Level

- RTL description is a way of describing the operation of a synchronous machine.
- The behavior of a machine is defined by the flow of signals (or transfer of data) between hardware registers and the logical operations (+,-, not,….)

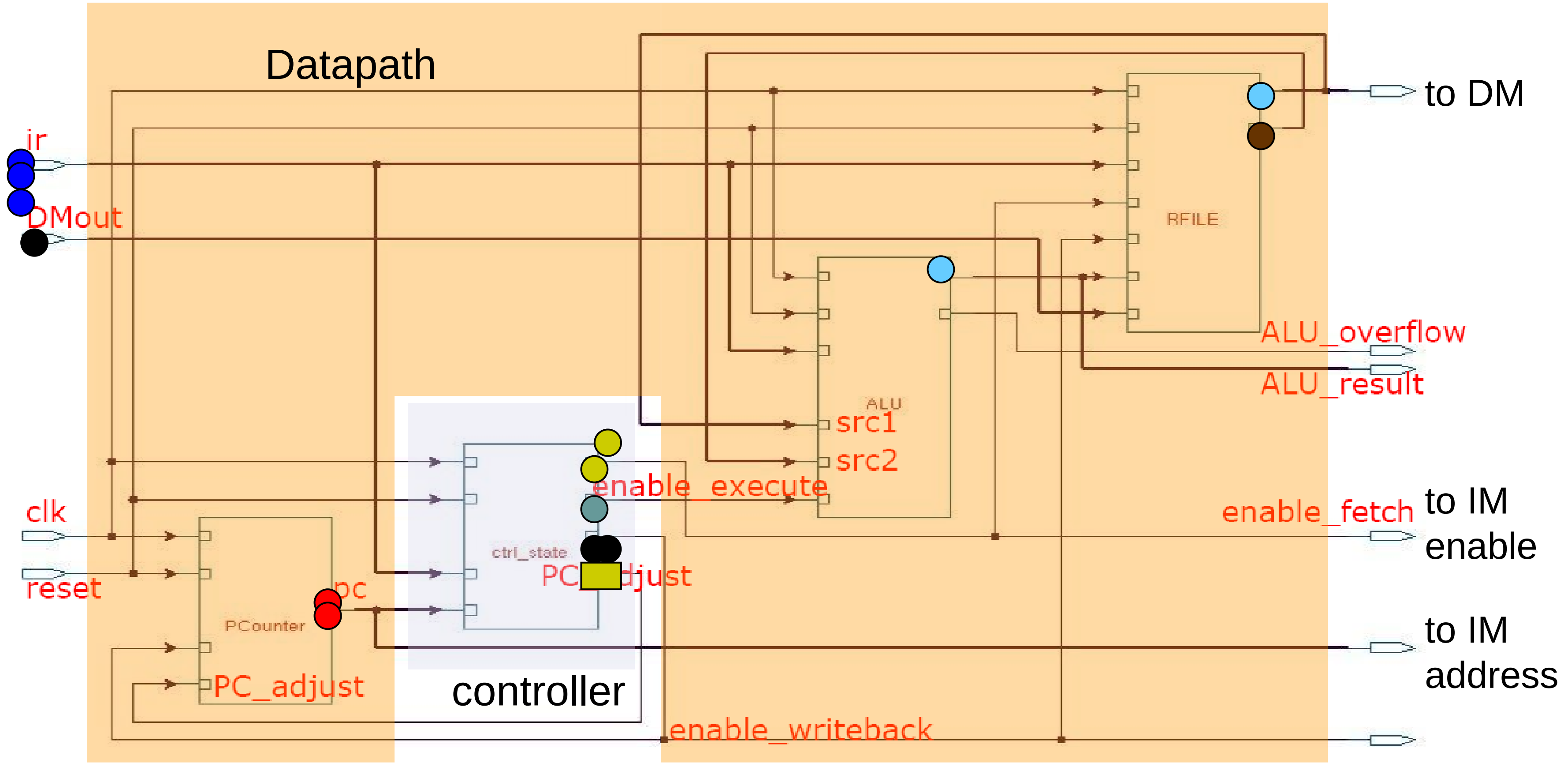

A simple RTL design

A Complex RTL Design -- CPU

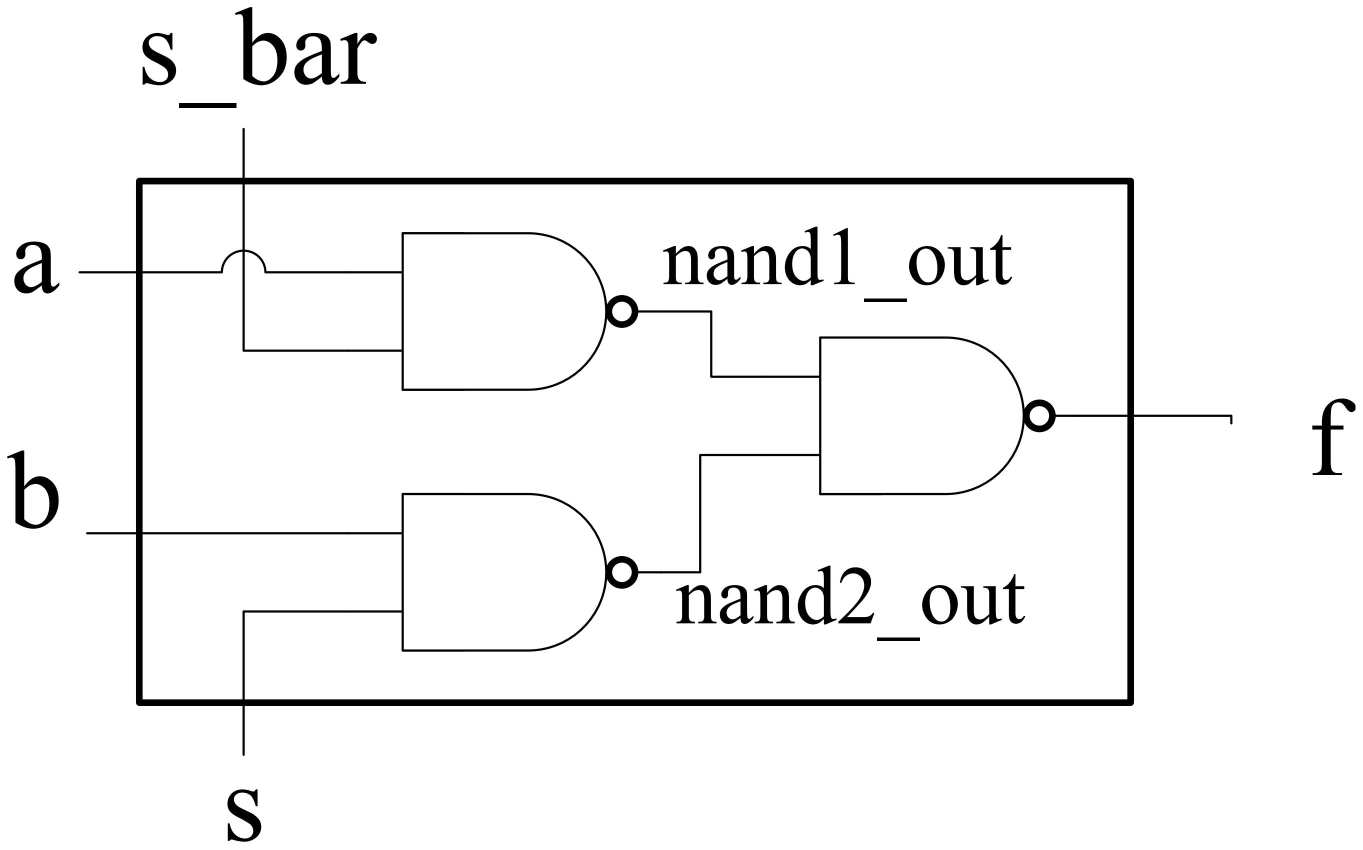


CPU in Action (ALU REG <- REG)

- RF[dst] <- ALU(RF(src),RF(dst))



One-bit Multiplexer



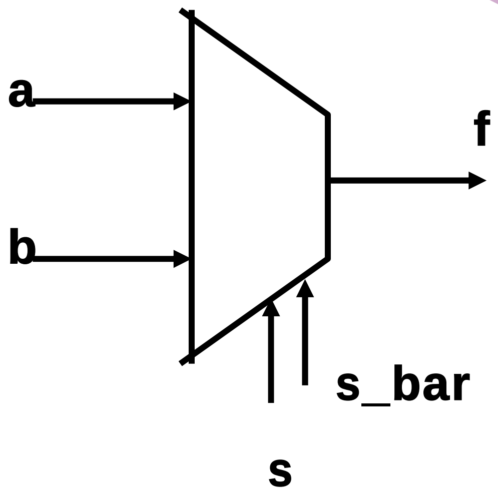
First Look at Verilog (1/2)

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
```



First Look at Verilog (2/2)

Variable declaration

output

Input

wire

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
```

Basic Language Rules

- General
  - A statement shall terminate with semicolon (;), except *endmodule*
  - Comments: (//) for single line and {/*, */} for multiple lines
- Naming
- Number representation
- Primitive operators
- First examples

Naming

- Case sensitive
  - C_out_bar and C_OUT_BAR: two different identifiers
- NO whitespace
- Accepted chars
  - Lower/upper case letters
  - Digits (0,1,…,9)
  - Underscore (_)
  - Dollar sign ($)
  - Max characters: 1024

Number Representation

- May be represented using
    - Binary, octonary, decimal, hexadecimal,
- Format
    - <size>’ <base_format> <number>
    - base_format:
        - b, o, d, h
- Example
    - 4’b1111; -16’d255
    - 23456 (32-bit decimal # by default); ‘hc3 (32 bit)
    - 12’b1111_0000_1010

Primitive Operator

- Most basic functional models of combinational logic gates
- Built in the Verilog

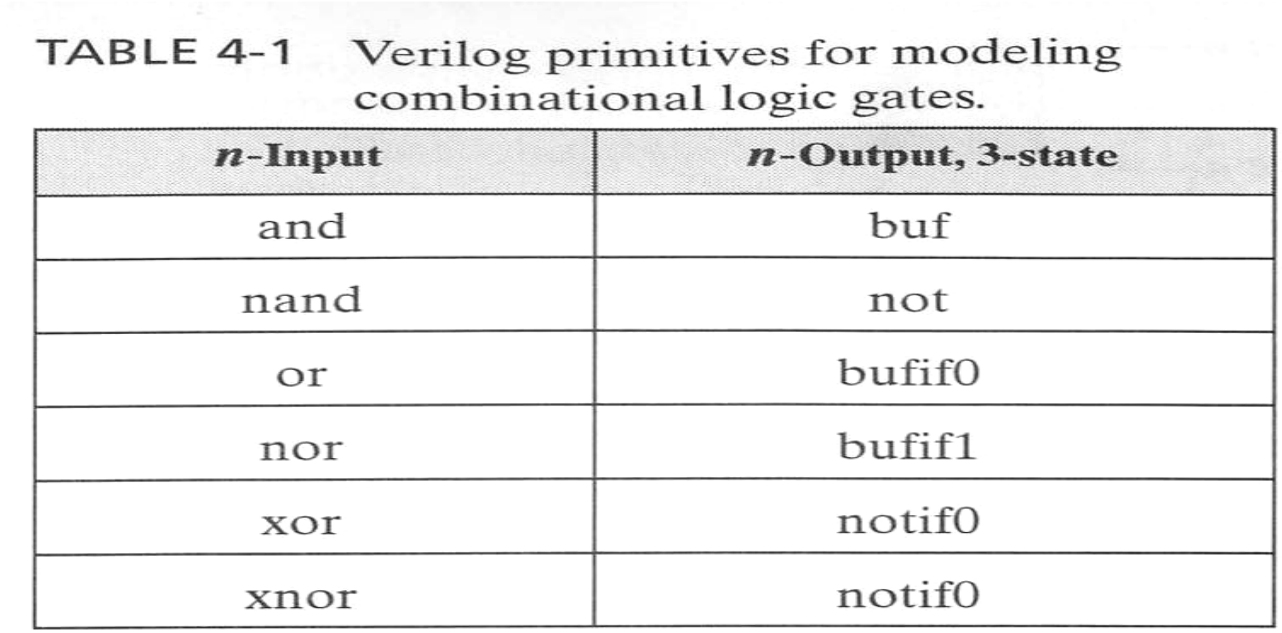
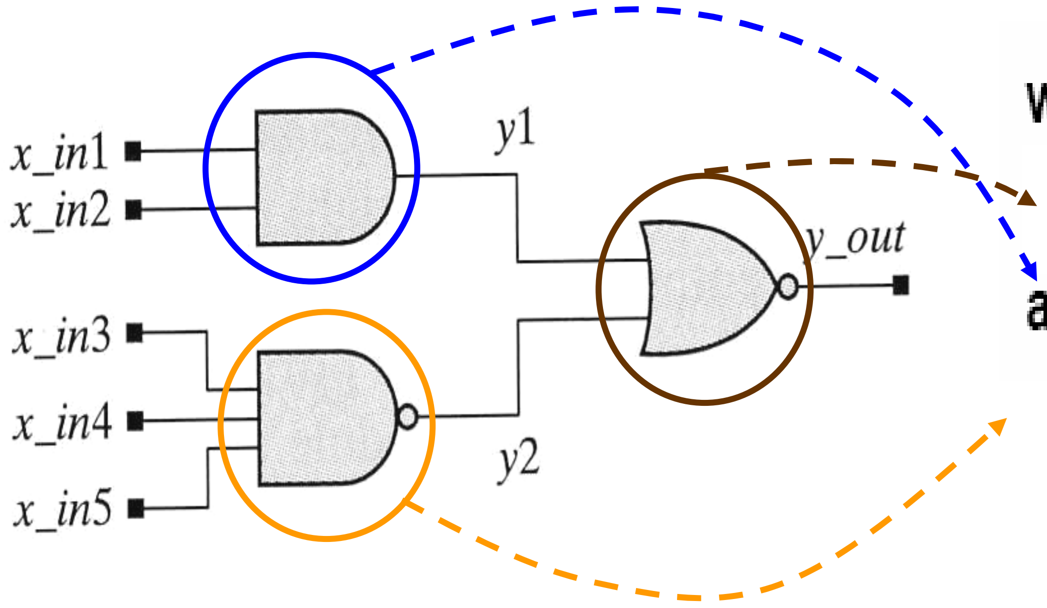


TABLE 4-1 Verilog primitives for modeling combinational logic gates.

n-Input	*n*-Output, 3-state
and	buf
nand	not
or	bufif0
nor	bufif1
xor	notif0
xnor	notif0

Example: AOI Gate



```
wire    y1, y2;

and     (y1, x_in1, x_in2);
```

Output ports of a primitive first, followed by its input port(s)!

Module Ports

- Interface to the environment
- Mode
  - Input
  - Output
  - Inout: bidirectional
- Not order sensitive in declaration

```
module AOI (y_out,x_in1,x_in2,x_in3,x_in4,x_in5);

  input x_in1,x_in2;
  input x_in5,x_in4,x_in3;
  output y_out;

  // ...
endmodule
```

Module Ports

- Order sensitive when used
  - Position sensitive

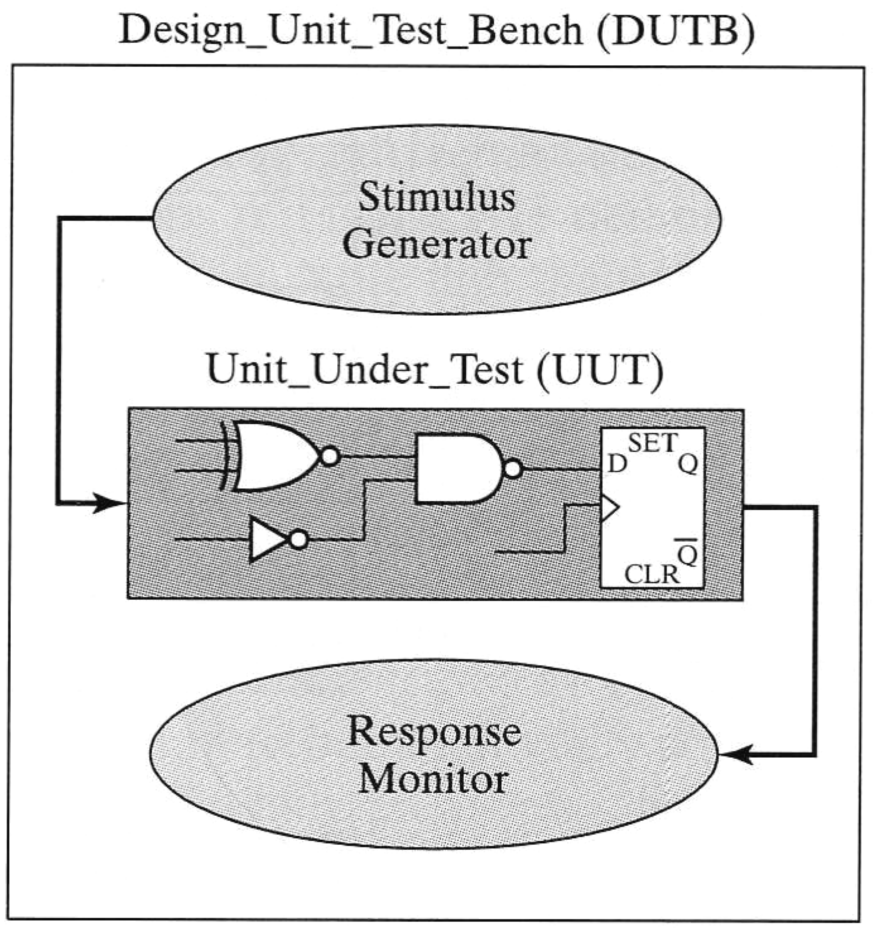
    AOI M1(w1, a1, a2, b1, b2, b3);

  - Explicitly declared

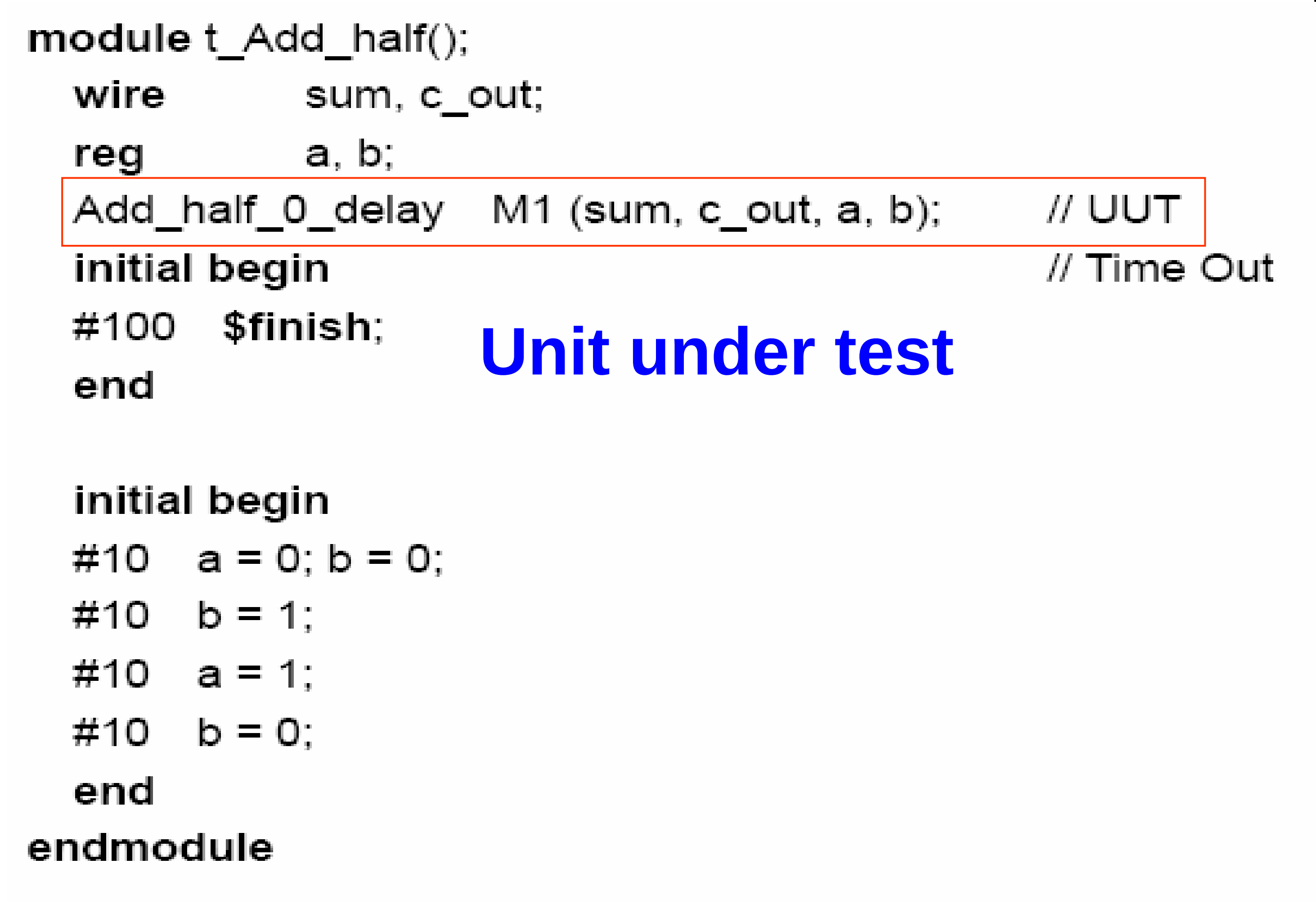
AOI M1 (.x_in3(b1), .y_out(w1), .x_in2(a2), .x_in1(a1),.x_in4(b2), .x_in5(b3));

```
module AOI (y_out,x_in1,x_in2,x_in3,x_in4,x_in5);

  input x_in1,x_in2;
  input x_in5,x_in4,x_in3;
  output y_out;

  // …
endmodule
```

Testbench


Design_Unit_Test_Bench (DUTB)
Stimulus Generator
Unit_Under_Test (UUT)
SET
D
Q
CLR
Q̄
Response Monitor

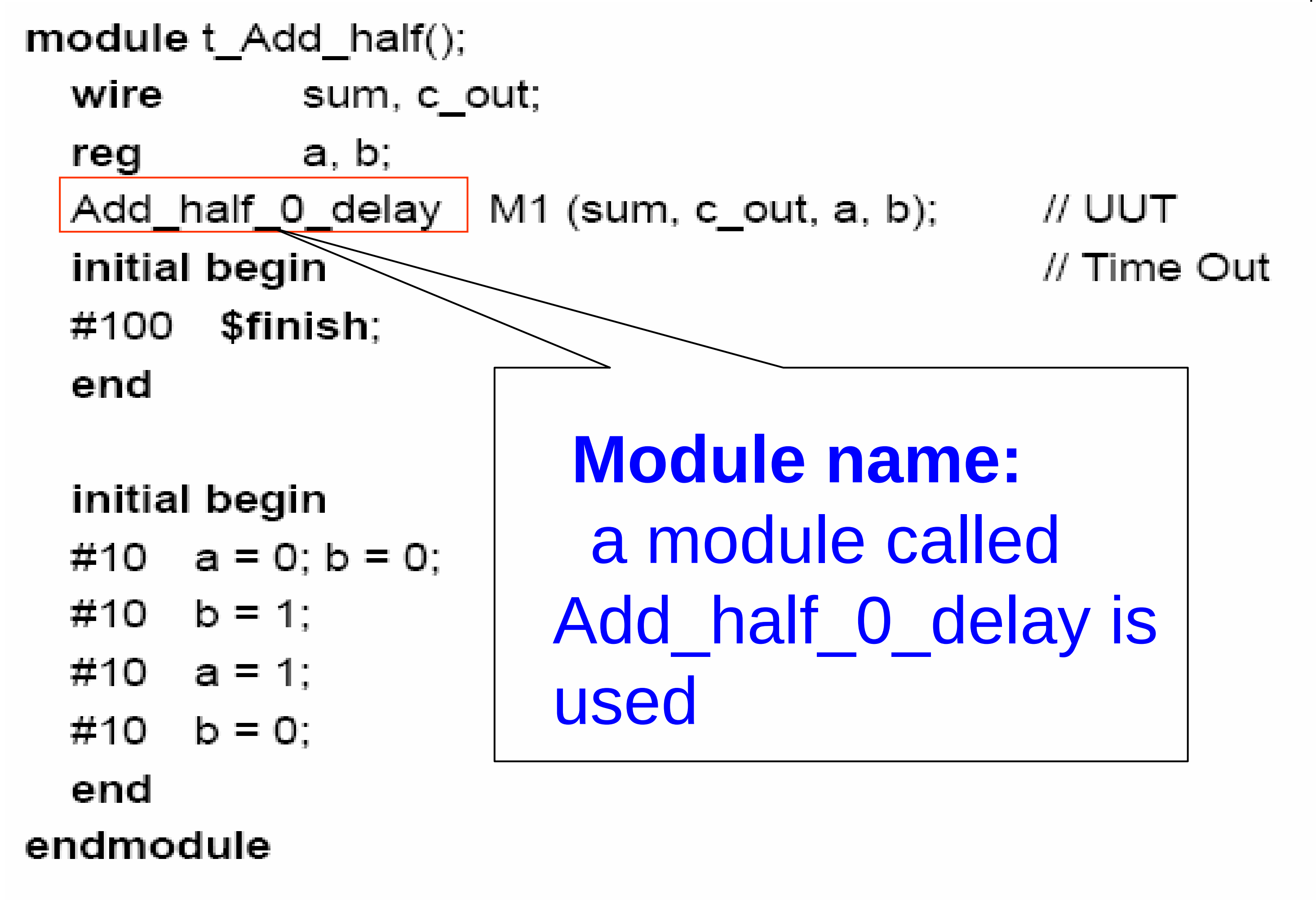
Example 4.7



```
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
```

Unit under test

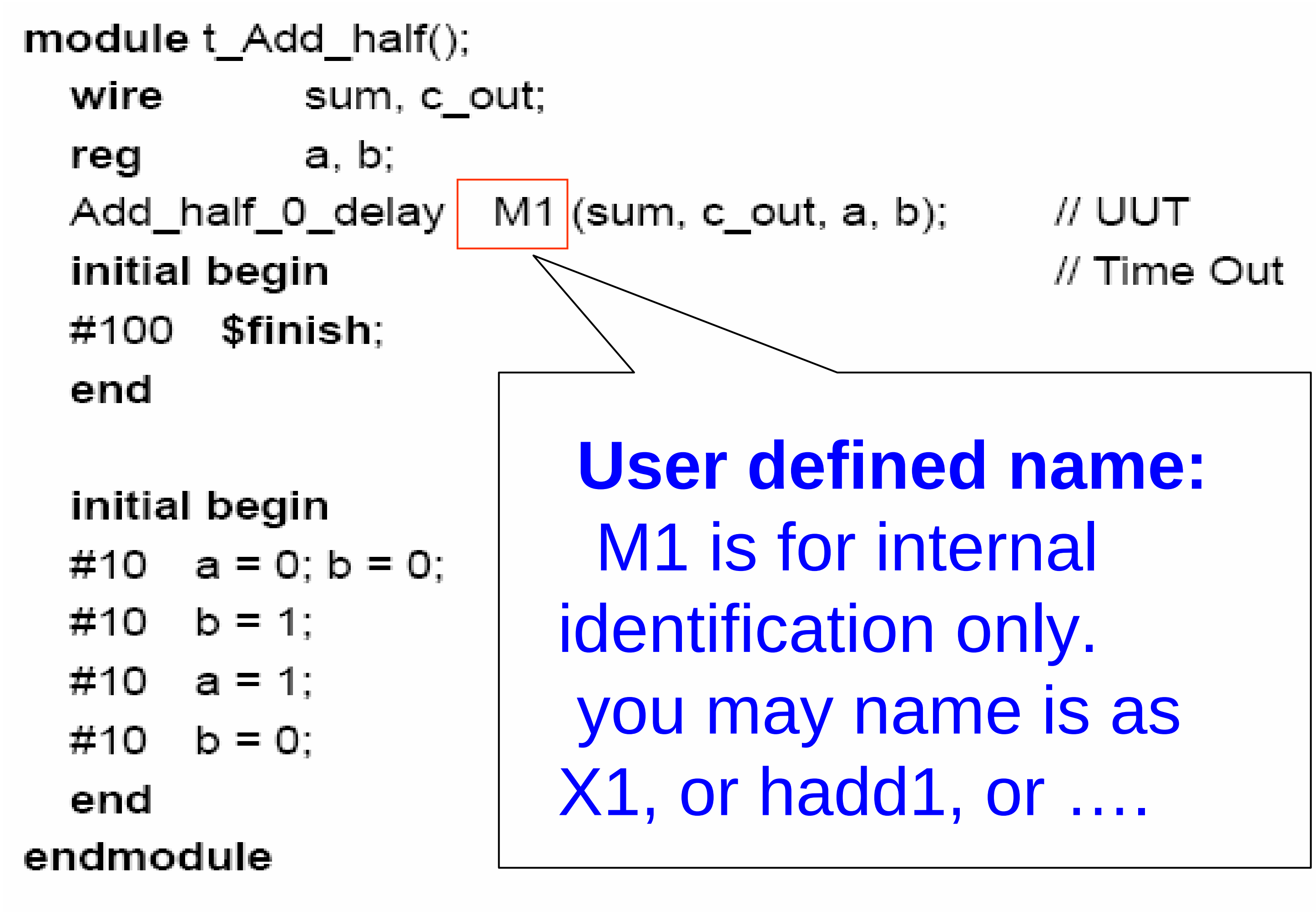
Example 4.7



```
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
```

Module name:
a module called
Add_half_0_delay is
used

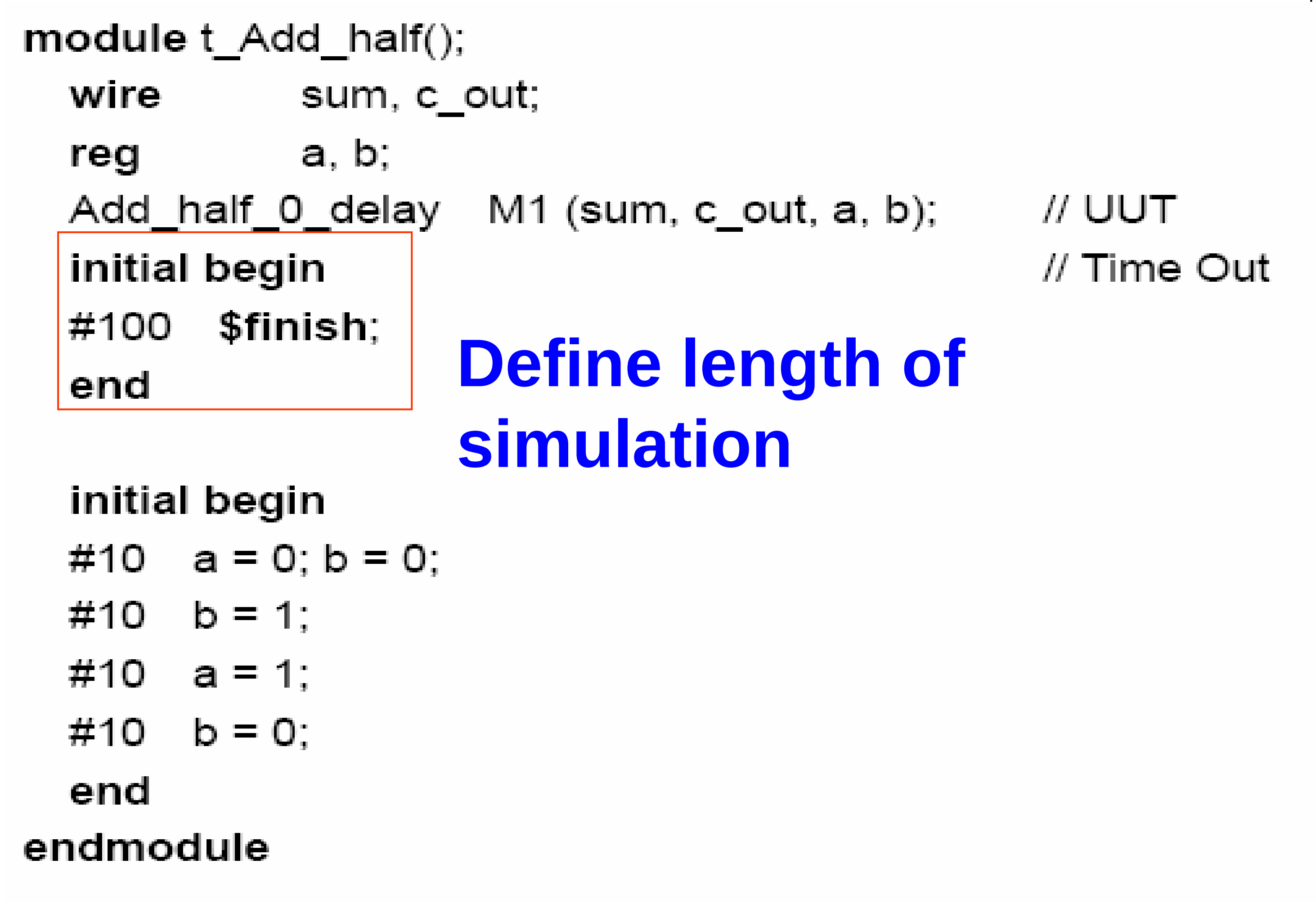
Example 4.7



```
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
```

User defined name:
M1 is for internal identification only.
you may name is as X1, or hadd1, or ….

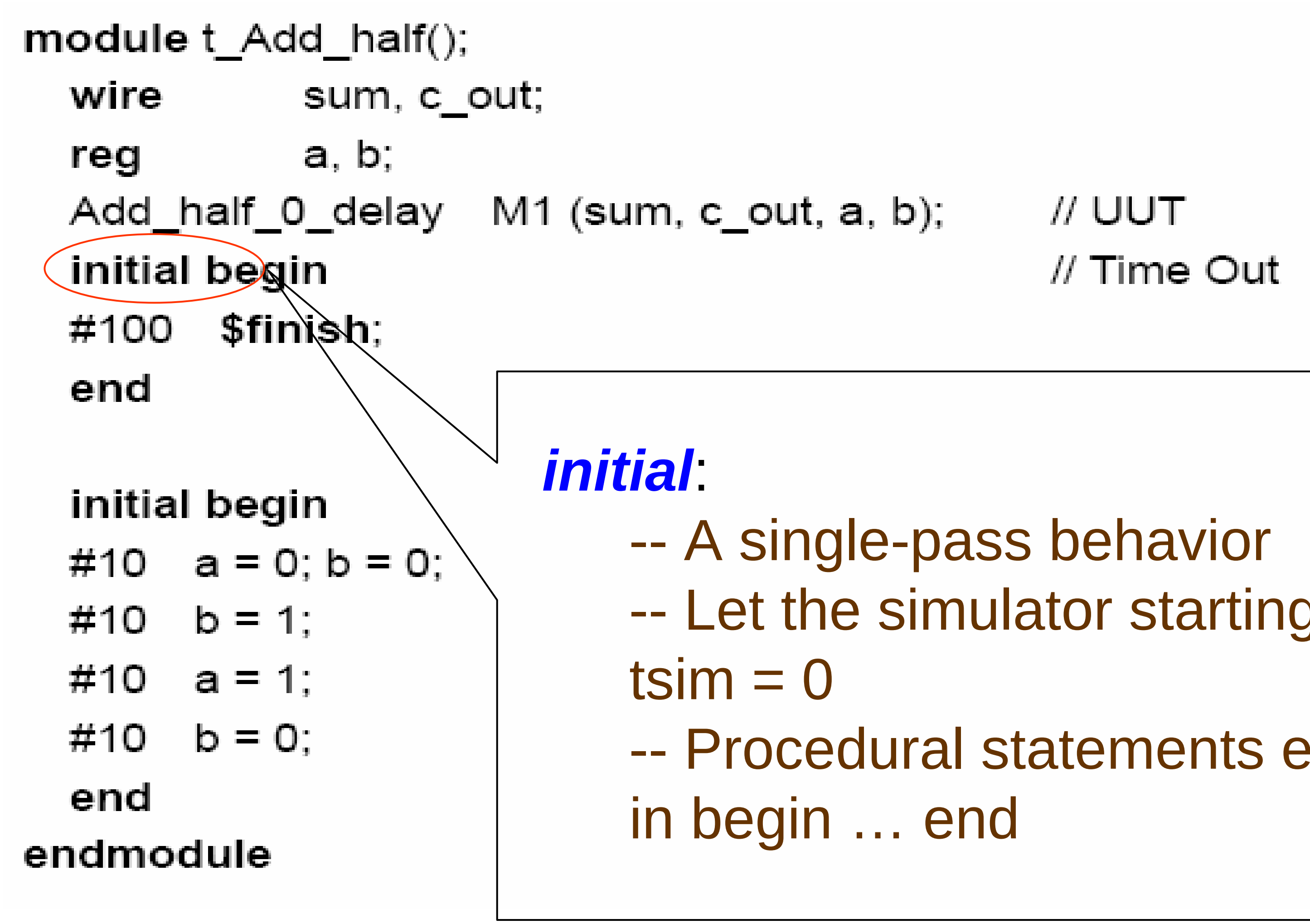
Example 4.7



```
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
```

Define length of simulation

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
```

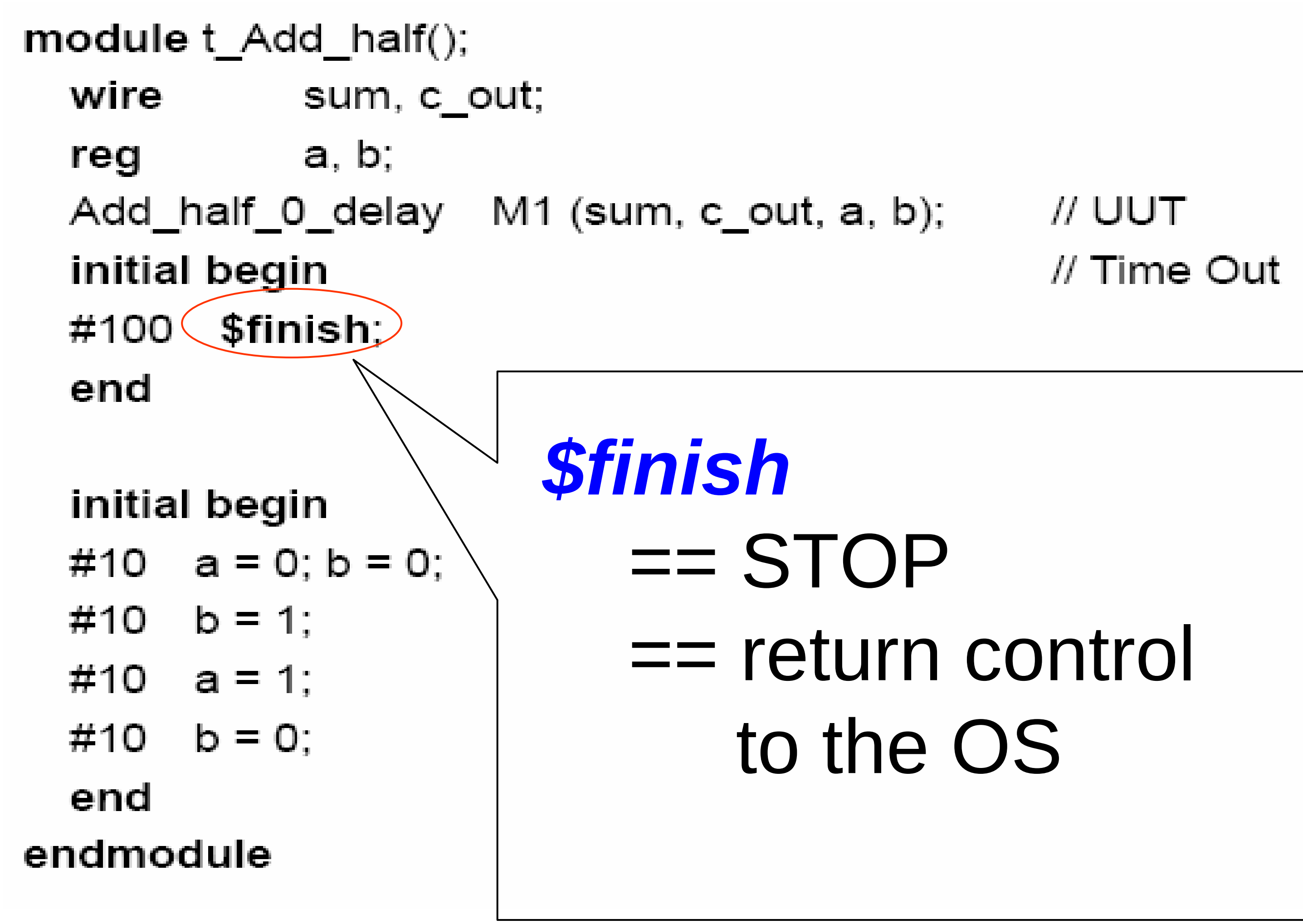
initial:

-- A single-pass behavior

-- Let the simulator starting from tsim = 0

-- Procedural statements enclosed in begin … end

Example 4.7



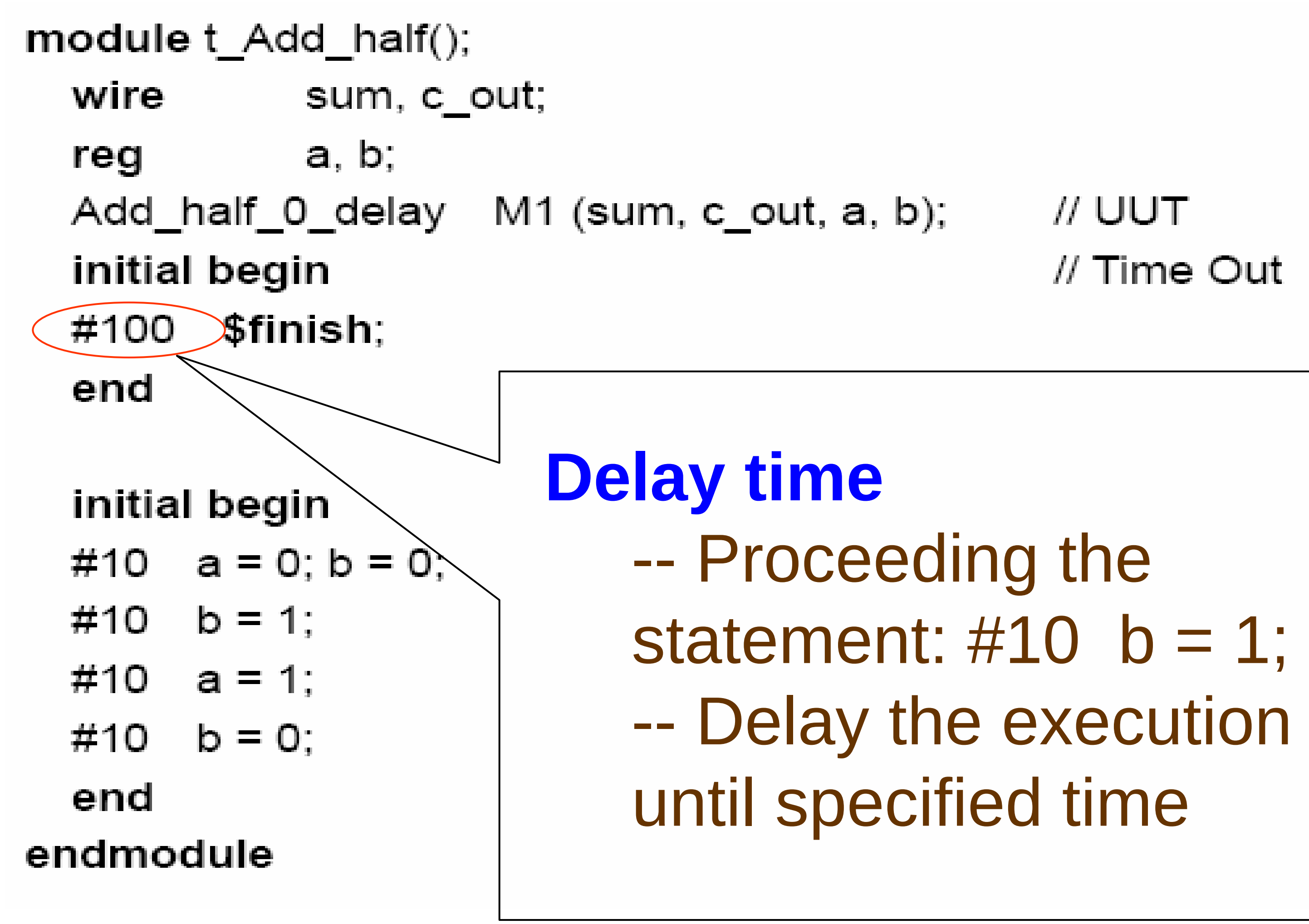
```
module t_Add_half();
   wire            sum, c_out;
   reg             a, b;
   Add_half_0_delay   M1 (sum, c_out, a, b);        // UUT
   initial begin                                    // Time Out
   #100   $finish;
   end

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

$finish

== STOP

== return control
   to the OS

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
```

Delay time

-- Proceeding the statement: #10 b = 1;

-- Delay the execution until specified time

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
```

Define Inputs

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
```

a	0	0	1	1	1	1
b	0	1	1	0	0	0

$\longrightarrow$

time

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***
- ***$finish*** == return control to the OS

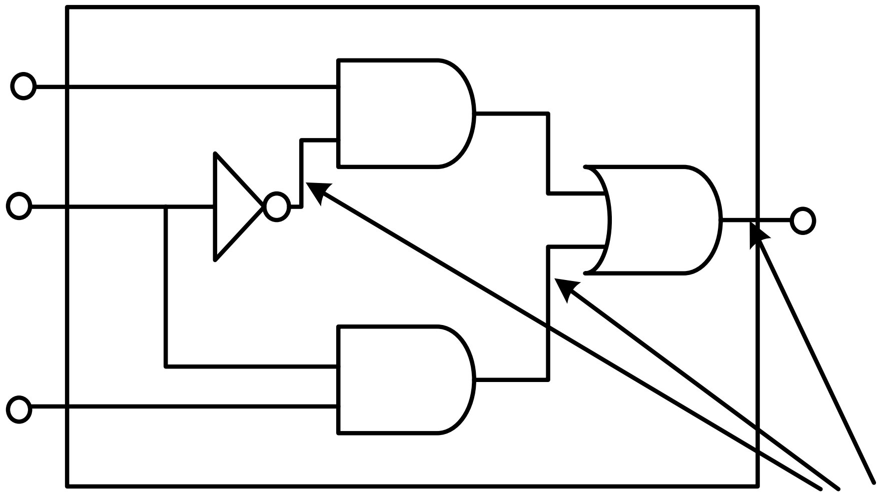
Data Types

- Net: ***wire***
  - Acts like wires in a physical circuit
  - Connects design objects
  - Needs for a driver

- Register: ***reg***, ***integer***
  - Acts like variables in ordinary procedural languages
  - Stores information while the program executes
  - No needs for a driver, changes its value as wish

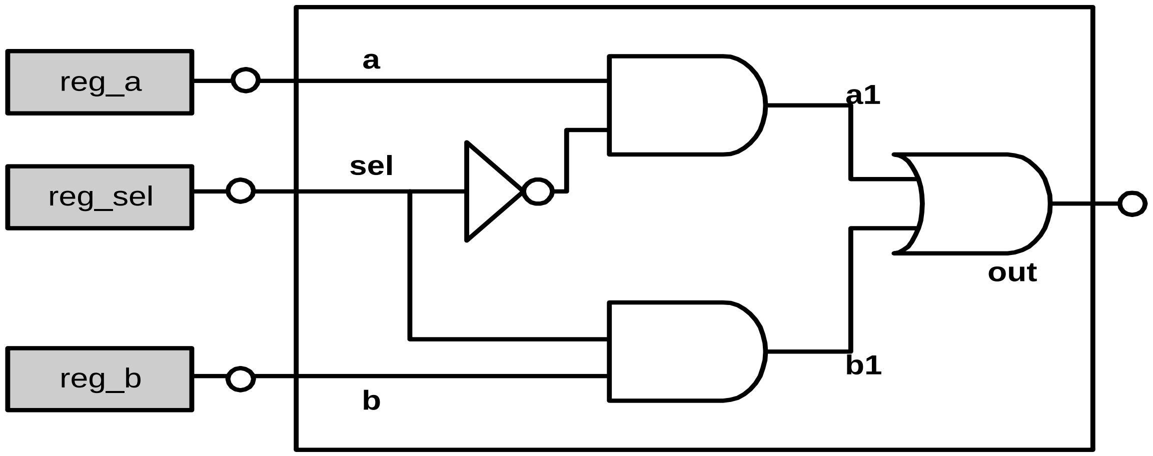
Nets

- Nets are continuously driven by the devices that drive them.
- Verilog automatically propagates a new value onto a net when the drivers on the net change value.



Registers

- A register is merely a variable, which holds its value until a new value is assigned to it. It is different from a hardware register.


reg_a
reg_sel
reg_b
a
sel
b
a1
b1
out