Digital System Design

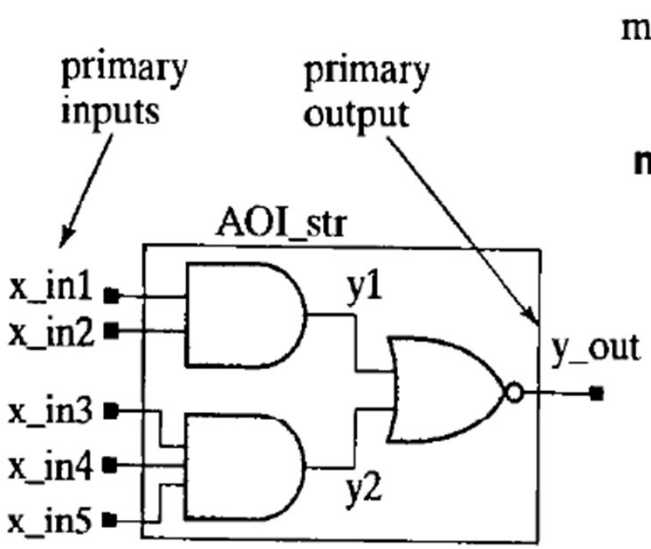
CE 325 L1
SPRING 2024

Instructor : Dr. Syed Arsalan Jawed
Associate Professor of Practice
arsalan.jawed@sse.habib.edu.pk
Room : W-301
Office Hours : Monday 0930-1030, Wednesday 1430-1530, Friday 0930-1030

**Use in conjunction with the whiteboard lecture notes during the class.*

Introduction to Logic Design with Verilog

- Design Encapsulation : MODULE → just like function in C language



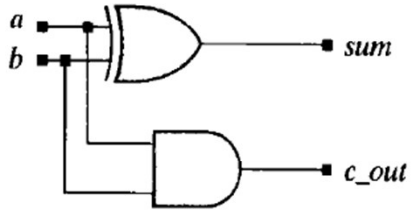
module name — module ports

```
module AOI_str (y_out, x_in1, x_in2, x_in3, x_in4, x_in5);
    output   y_out;
    input    x_in1, x_in2, x_in3, x_in4, x_in5;   ← port modes
    wire     y1, y2;                              ← internal wires establish connectivity
    nor      (y_out, y1, y2);
    and      (y1, x_in1, x_in2);                  ← instantiated primitives
    and      (y2, x_In3, x_in4, x_in5);
endmodule
```

Primitives

- Common Combinational Logic Gates

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




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

Some Verilog Language Rules

4.1.4 Some Language Rules

Verilog is a case-sensitive language, so it matters whether you refer to a signal as *C_out_bar* or *C_OUT_BAR*. Verilog treats these as different names. An identifier (name) in Verilog is composed of a case-sensitive, space-free sequence of upper and lower case letters from the alphabet, the digits (0, 1, ..., 9), the underscore (_), and the $ symbol. The name of a variable may not begin with a digit or $, and may be up to 1024 characters long.[7] White space may be used freely, except in an identifier.

Usually, each line of text in a Verilog description must terminate with a semicolon (;). An exception is the terminating ***endmodule*** keyword. Comments may be imbedded in the source text in two ways. A pair of slashes, //, forms a comment from the text that follows it on the same line; the symbol-pair /* initiates a multiline comment, and must be matched by the symbol-pair */ to terminate the scope of the comment. Multiline comments may not be nested.

Nested Modules

MODELING TIP

Use nested module instantiations to create a top-down design hierarchy.

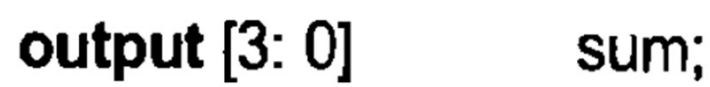
MODELING TIP

The ports of a module may be listed in any order.
An instantiated module must have an instance name.

```
module Add_rca_16_0_delay (sum, c_out, a, b, c_in);
  output [15:0]     sum;
  output            c_out;
  input [15:0]      a, b;
  input             c_in;
  wire              c_in4, c_in8, c_in12, c_out;

  Add_rca_4 M1      (sum[3:0],   c_in4,   a[3:0],   b[3:0],   c_in);
  Add_rca_4 M2      (sum[7:4],   c_in8,   a[7:4],   b[7:4],   c_in4);
  Add_rca_4 M3      (sum[11:8],  c_in12,  a[11:8],  b[11:8],  c_in8);
  Add_rca_4 M4      (sum[15:12], c_out,   a[15:12], b[15:12], c_in12);
endmodule

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

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 Add_half_0_delay (sum, c_out, a, b);
  output            sum, c_out;
  input             a, b;
  xor               M1 (sum, a, b);
  and               M2 (c_out, a, b);
endmodule
```

Vectors

A vector in Verilog is denoted by square brackets, enclosing a contiguous range of bits, e.g., *sum[3:0]* represents four bits from *sum*, which was declared in *Add_rca_16_0_delay* as a 16-bit signal. *The language specifies that, for the purpose of calculating the decimal equivalent value of a vector, the leftmost index in the bit range is the most significant bit, and the rightmost is the least significant bit.* An expression can be the index of a part-select. If the index of a part-select is out of bounds the value **x** is returned by a reference to the variable.[9] For example, if an 8-bit word *vect_word[7:0]* has a stored value of decimal 4, then *vect_word[2]* has a value of 1; *vect_word[3:0]* has a value of 4; and *vect_word[5:1]* has a value of 2.



Structural Connectivity

The logic value of a ***wire*** (net) is determined dynamically during simulation by what is connected to the wire. If a *wire* is attached to the output of a primitive (module), it is said to be driven by the primitive (module), and the primitive (module) is said to be its driver. For example, in Figure 4-5 *y_out* is driven by a ***nor*** gate (primitive). The logic of the gate and the values of its inputs determine *y_out*. In that example we explicitly declared *y1* and *y2* to have type ***wire***, but did not have to do so. *Any identifier that is referenced without having a type declaration is by default of type* ***wire***.[11] Consequently, the input and output ports have default type ***wire*** too, unless we specifically declare them to have a different type (e.g., we will see that a variable may have type ***re***

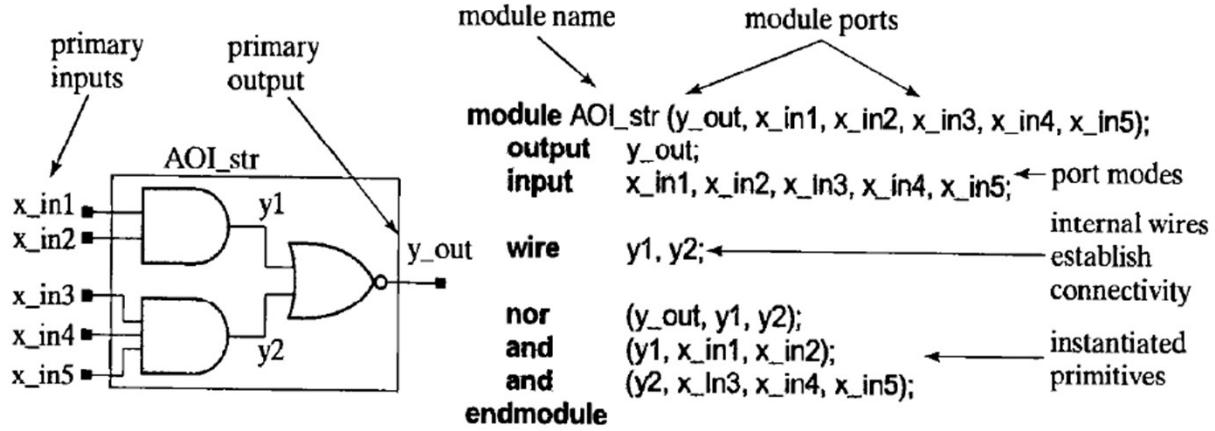


FIGURE 4-5 An AOI circuit and its Verilog structural model.

Four Value Logic

- 0 → low (ground), 1→ high(VDD, power supply)
- X → simulator cannot determine whether 0 or 1
  when a net is driven by un-initialized drivers
- Z→ wire is disconnected from its driver
  needed for multi-driver shared net or bus

Test Methodology and Signal Generators for TestBench

A large circuit must be tested and verified systematically to ensure that all of its logic has been exercised and found to be functionally correct. A haphazard test methodology makes debugging very difficult, creates a false sense of security that the model is correct, and risks enormous loss should a product fail as a result of an untested area of logic. In practice, design teams write an elaborate test plan that specifies the features that will be tested and the process by which they will be tested.

to associate with the underlying circuit. A more clever strategy is to verify that the half adder and fulladder each work correctly. Then, the 4-bit slice unit can be verified exhaustively by applying only 2^9 patterns! Once these simpler design units have been verified, the 16-bit adder can be tested to work correctly for a carefully chosen set of patterns that check the connectivity between the four units. This reduces the number of patterns and focuses the debugging effort on a much smaller portion of the overall circuitry. (See the problems section at the end of the chapter.)

We saw that modeling begins with a complex functional unit and partitions it in a top-down fashion to enable the design of simpler units. Systematic verification proceeds in the opposite direction, beginning with the simpler units and moving to the more complex units above them in the design hierarchy. A basic methodology for verifying the functionality of a digital circuit consists of building a testbench that applies stimulus patterns to the circuit and displays the waveforms that result. The user, or soft-

- Use Procedural Statements inside begin and end of Initial and Always blocks

- Use Delay Control Operator #

The time at which a procedural assignment statement in a ***begin*** ... ***end*** block executes depends on its order in the list of statements, and on the delay time preceding the statement (e.g., #10). The statements execute sequentially from top to bottom and from left to right across lines of text that may contain multiple statements. In this example

each line is preceded by a time delay (e.g., 10 simulator time units) that is prescribed with a delay control operator (#) and a delay value, for example, #10. A delay control operator preceding a procedural assignment statement suspends its execution and, consequently, suspends the execution of the subsequent statements in the behavior until the specified time interval has elapsed during simulation. A single-pass behavior expires when the last statement has executed, but (in general) the simulation does not necessarily terminate, because other behaviors might still be active.

TestBench

```
module t_DUTB_name ();        // substitute the name of the UUT
  reg ...;                    // Declaration of register variables for primary
                              // inputs of the UUT
  wire ...;                   // Declaration of primary outputs of the UUT
  parameter                     time_out = // Provide a value
  UUT_name M1_instance_name (UUT ports go here);
  initial $monitor ( );       // Specification of signals to be monitored and
                              // displayed as text
  initial #time_out $stop;    // Stopwatch to assure termination of simulation
  initial                     // Develop one or more behaviors for pattern
                              // generation and/or
                              // error detection
  begin
                              // Behavioral statements generating waveforms
                              // to the input ports, and comments documenting
                              // the test. Use the full repertoire of behavioral
                              // constructs for loops and conditionals.
  end
endmodule
```

Propagation Delay

- Primitives have 0 default delay.
  - Non zero delay can be associated to make them realistic.
  - Simulation with unit delay is done to expose time sequence of signal activity

The primitives within *Add_full* and *Add_half* are shown below with annotation that assigns to them a unit delay. The delay notation #1 has been inserted before the instance name of each instantiated primitive (# denotes a delay control operator). The effect of the delay is apparent in the simulated transitions of *sum* and *c_out* shown in Figure 4-18. Notice that the 0-delay simulation results do not reveal whether *c_out* is formed before or after *sum*. Both, in fact, appear to change as soon as the inputs change. The unit-delay model reveals the time-ordering of the signal activity.

```
module Add_full(sum, c_out, a, b, c_in);
  output      sum, c_out;
  input       a, b, c_in;
  wire        w1, w2, w3;
  Add_half    M1 (w1, w2, a, b);
  Add_half    M2 (sum, w3, w1, c_in);
  or          #1 M3 (c_out, w2, w3);
endmodule

module Add_half (sum, c_out, a, b);
  output      sum, c_out;
  input       a, b;
  xor         #1 M1 (sum, a, b);
  and         #1 M2 (c_out, a, b);
endmodule
```

Delays in the Standard Cells

ASICs are fabricated by assembling onto a common silicon die the logic cells from a standard-cell library. The library cells are predesigned and precharacterized so that their Verilog model includes accurate timing information, and synthesis tools use this information to optimize the performance (speed) of a design. Our focus in this book will be on synthesizing gate-level structures from technology-independent[18] behavioral models of circuits, using either standard cells or field-programmable gate arrays (FPGAs). The timing characteristics of the latter are embedded within the synthesis tools for FPGAs; the timing characteristics of the former are embedded within the

model of the cell and are used by a synthesis tool to analyze the timing of a circuit in conjunction with selecting parts from a cell library to realize specified logic. Circuit designers do not attempt to create accurate gate-level timing models of a circuit by manual methods. Instead, they rely on a synthesis tool to implement a design that will satisfy timing constraints. We will address this topic in Chapter 10.[19]

Inertial Delay

- Coming from RC delay
  - Causing time-limited/time-defined exponential crossing of the switching threshold.

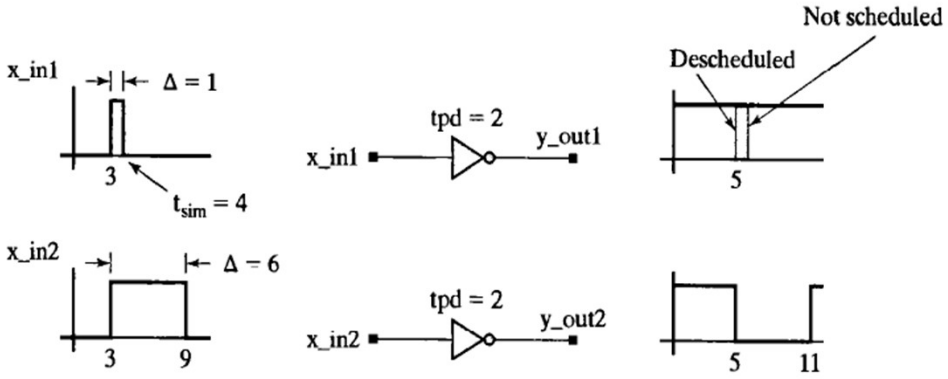


FIGURE 4-20 Event de-scheduling caused by an inertial delay.

In physical circuits, the propagation delay of a logic gate is affected by its internal structure and by the circuit that it drives. The internal delay is referred to as the intrinsic delay of the gate. In a circuit, the driven gates and the metal interconnect of their fanin nets create additional capacitive loads on the output of the driving gate and affect its timing characteristics. The slew rate of the input signal, which represents the slope of a signal's transition between logic values, can also have an affect on the transitions of the output signal. Accurate standard-cell models account for all these effects.

Transport Delay

The time-of-flight of a signal traveling a wire of a circuit is modeled as a transport delay. With this model narrow pulses are not suppressed, and all transitions at the driving end of a wire appear at the receiving end after a finite time delay. In most ASICs, the physical distances are so small that the time-of-flight on wires can be ignored, because at the speed of light the signal takes only .033 ns to travel a centimeter. However, Verilog can assign delay to individual wires in a circuit to model transport delay effects in circuits where it cannot be neglected, such as in a multichip hardware module or on a printed circuit board. Wire delays are declared with the declaration of a wire. For example, ***wire*** *#2 A_long_wire* declares that *A_long_wire* has a transport delay of two time steps.

Types of Assignments

- Blocking Assignment
  - Required for Sequential Execution
  - A=b;
  - Used in all combination procedural statements
- Non-Blocking Assignment
  - Required for concurrent execution of statements
  - A<=b;
  - Used in all sequential procedural statements

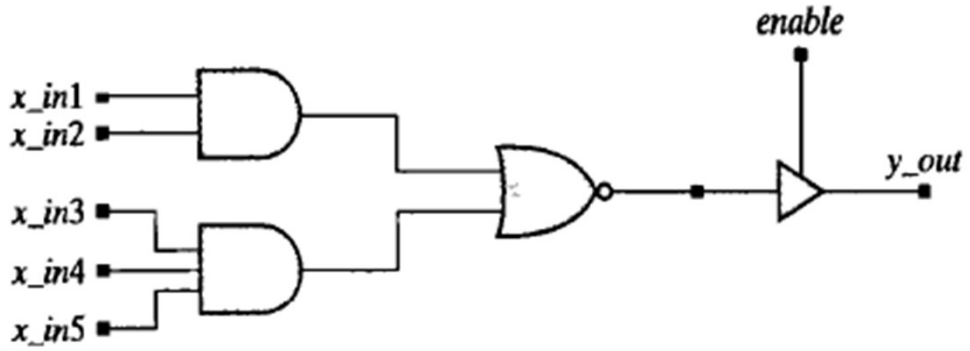
Behavioral Description of Combinations Logic
Assign Statement – versatile and powerful

```
module AOI_5_CA0 (y_out, x_in1, x_in2, x_in3, x_in4, x_in5);
  input          x_in1, x_in2, x_in3, x_in4, x_in5;
  output         y_out;

  assign y_out =  ~((x_in1 & x_in2) | (x_in3 & x_in4 & x_in5));

endmodule
```

Assign Statement

The five-input AOI circuit can be modified to have an additional input, *enable*, and to have a three-state output, as described by *AOI_5_CA1* below.

```
module AOI_5_CA1 (y_out, x_in1, x_in2, x_in3, x_in4, x_in5, enable);
 input          x_in1, x_in2, x_in3, x_in4, x_in5, enable;
 output         y_out;

 assign y_out = enable ? ~((x_in1 & x_in2) | (x_in3 & x_in4 & x_in5)) : 1'bz;

endmodule
```



Propagation Delay and Assign Statement

```
module AOI_5 _CA3 (y_out, x_in1, x_in2, x_in3, x_in4);
  input          x_in1, x_in2, x_in3, x_in4;
  output         y_out;

  wire #1 y1 = x_in1 & x_in2;          // Bitwise and operation
  wire #1 y2 = x_in3 & x_in_4;
  wire #1 y_out = ~ (y1 | y2);         // Complement the result of bitwise OR operation
endmodule
```

Latches and Level Sensitive Circuits in Verilog

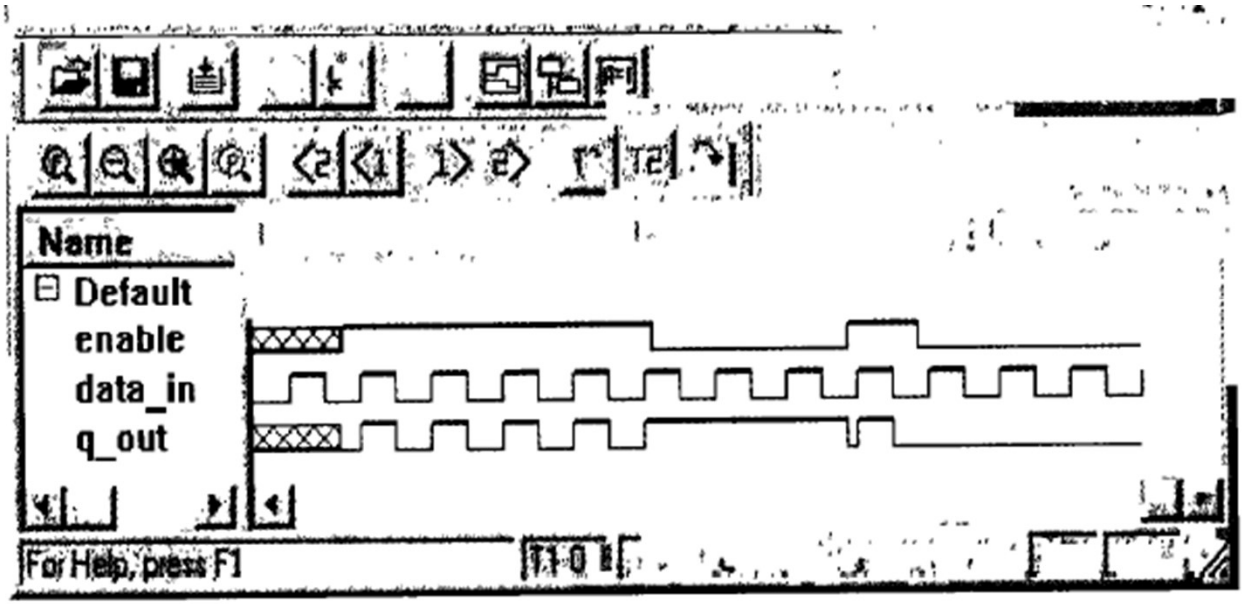
The level-sensitive storage mechanism of a latch (See Chapter 3) can be modeled in a variety of ways. First, note that a set of continuous-assignment statements has implicit feedback if a variable in one of the RHS expressions is also the target of an assignment. For example, a pair of cross-coupled NAND gates could be modeled as

```
assign q = set ~& qbar;
assign qbar = rst ~& q;
```

The implied behavior will still be level-sensitive, but it will correspond to the feedback structure of a hardware latch. However, synthesis tools do not accommodate this form of feedback. But they do support the feedback that is implied by a continuous assignment in which the RHS uses a conditional operator.

```
module Latch_CA (q_out, data_in, enable);
  output        q_out;
  input         data_in, enable;

  assign q_out = enable ? data_in : q_out;
endmodule
```

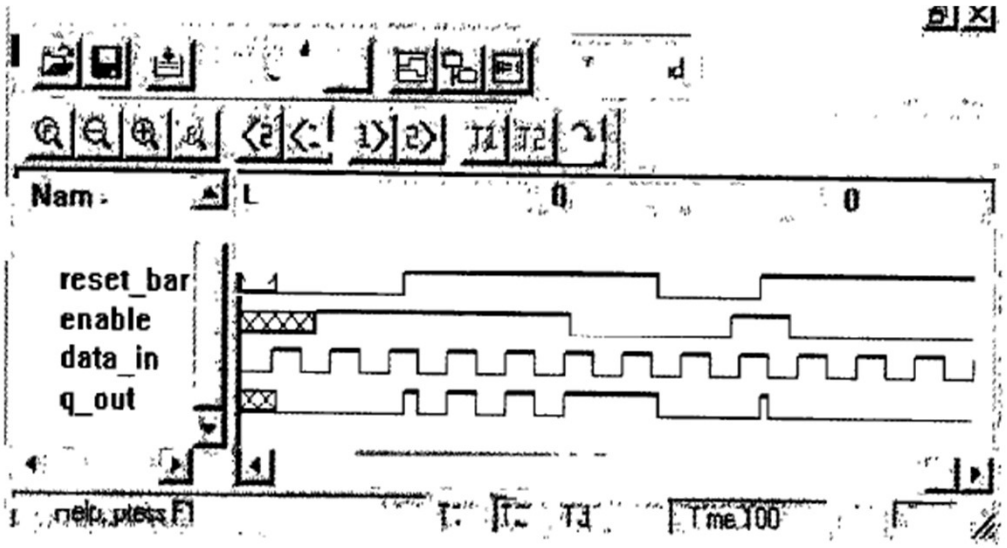
When feedback is used in a continuous-assignment statement with a conditional operator, a synthesis tool will infer the functionality of a latch and its hardware implementation. Chapter 6 will discuss descriptive styles that lead to intentional and accidental synthesis of latches.



Transparent latch with a reset using assign statement

The latch model *Latch_Rbar_CA* below uses a nested conditional operator to add the functionality of an active-low reset to a transparent latch. Simulation of *Latch_Rbar_CA* produces the waveforms shown in Figure 5-5, in which the actions of *enable* and *reset* are apparent.

```
module Latch_Rbar_CA (q_out, data_in, enable, reset);
  output      q_out;
  input       data_in, enable, reset;

  assign q_out = !reset ? 0 : enable ? data_in : q_out;
endmodule
```



Cyclic Behavioral Models – the always block

- Continuous assignment cannot model edge-sensitive behavior
  - Unless we ourselves combine two latches

  - CBM → do not end with their last procedural statement
  - They re-execute

```
module asynch_df_behav (q, q_bar, data, set, clk, reset );
 input          data, set, reset, clk;
 output         q, q_bar;
 reg            q;

 assign         q_bar = ~q;

 always @ (negedge set or negedge reset or posedge clk)
  begin
    if (reset == 0) q <= 0;
      else if (set == 0) q <= 1;
        else q <= data;              // synchronized activity
 end
endmodule
```

```
module tr_latch (q_out, enable, data);
 output q_out;
 input enable, data;
 reg q_out;

 always @  (enable or data)
  begin
    if (enable) q_out = data;
  end
endmodule
```