CS 223 Digital Systems
Fall 2011
Verilog
Computer Engineering Dept
Bilkent University
Behavioral Models represent functionality of the digital hardware.

It describes how the circuit will operate without specifying hardware.

There are two important keywords for behavioral description.

- **initial** Specifies a single-pass behavior.
- **always** Specifies cyclic behavior.
Behavioral Modelling

- **initial** keyword is followed by a single statement or a `begin.. end` block
- The statement following **initial** is executed once at $t=0$ and it expires.

- **always** keyword is followed by a single statement or a `begin.. end` block
- The statement following **always** is executed repeatedly until simulation is stopped.
Use of initial and always.

初始

begin    // 定义一个时钟周期为20个单位的时钟。
    clock=1'b0;
repeat(30)  // 时钟将在300个单位内运行。
    #10 clock=¬clock;
end
Use of initial and always.

```verilog
initial
begin
    clock=1'b0;
end
initial
    #300 $finish; // clock will operate for 300 units.
always
    #10 clock=¬clock; //defines a clock with 20 unit clock period.
System task $finish causes simulation to terminate.
```
always and event control @

always @(event control expression) begin
    ........
end

@ is called event control operator.

always @(event) waits until the event occurs.

When event takes place begin.. end block will be executed.

Example: wait for positive edge of clock
always @(posedge clock)
always @(posedge clock or negedge reset)
Blocking and Nonblocking Assignments

- Blocking assignments use (=) as assignment operator
Blocking and Nonblocking Assignments

- Blocking assignments use (=) as assignment operator
- Blocking assignments are executed *sequentially*.
Blocking and Nonblocking Assignments

- Blocking assignments use (=) as assignment operator
- Blocking assignments are executed *sequentially*.
- Nonblocking assignments use (<=) as assignment operator
Blocking and Nonblocking Assignments

- Blocking assignments use (=) as assignment operator
- Blocking assignments are executed \textit{sequentially}.
- Nonblocking assignments use (<=) as assignment operator
- Nonblocking assignments are executed \textit{concurrently}.
Blocking and Nonblocking Assignments

- If $A=3$ and $B=5$

$$B=A$$

$$C=B+2 \quad \text{will set } B=3 \text{ and } C=5$$

$$B<=A$$

$$C<=B+2 \quad \text{result } B=3 \text{ and } C=7$$
Blocking and Nonblocking Assignments

- If $A=3$ and $B=5$
- $B=A$

$C=B+2$ ....... will set $B=3$ and $C=5$

$B<=A$
$C<=B+2$
result $B=3$ and $C=7$
Blocking and Nonblocking Assignments

- If A=3 and B=5
- B=A
- C=B+2 .......will set B=3 and C=5
Blocking and Nonblocking Assignments

- If A=3 and B=5
- B=A
- C=B+2 .......will set B=3 and C=5
- B<=A
Blocking and Nonblocking Assignments

- If A=3 and B=5
- B=A
- C=B+2 .......will set B=3 and C=5
- B<=A
- C<=B+2
Blocking and Nonblocking Assignments

- If $A=3$ and $B=5$
- $B=A$
- $C=B+2$ .......will set $B=3$ and $C=5$
- $B<=A$
- $C<=B+2$
- result $B=3$ and $C=7$
D-flip-flop Description

module D_flip_flop (Q, D, CLK);
output Q;
input D, CLK;
reg Q;
always @ (posedge CLK)
Q <= D;
endmodule
D flip-flop Description

```verilog
module D_flip_flop_b (Q, Q_b, D, CLK);
output Q, Q_b;
input D, CLK;
reg Q;
assign Q_b = ~Q;
always @(posedge CLK)
  Q <= D;
endmodule
```
D flip-flop with active-low asynchronous reset

// Description of D flip-flop
// with active-low asynchronous reset
module D_flip_flop_AR (Q, D, CLK, RST);
output Q;
input D, CLK, RST;
reg Q;
always @ (posedge CLK, negedge RST)
if (RST == 0) Q <= 1'b0;
else Q <= D;
endmodule
D flip-flop with active-low asynchronous reset

module D_flip_flop_AR_b (Q, Q_b, D, CLK, RST);
output Q, Q_b;
input D, CLK, RST;
reg Q;
assign Q_b = ~Q;
always @ (posedge CLK, negedge RST)
  if (RST == 0) Q <= 1’b0;
  else Q <= D;
endmodule
D-flip-flop test

module t_D_flip_flops;
wire Q, Q_AR;
reg D, CLK, RST;
D_flip_flop M0 (Q, D, CLK);
D_flip_flop_AR M1 (Q_AR, D, CLK, RST);
initial #100 $finish;
initial begin CLK = 0; forever #5 CLK = ~CLK; end
D-flip-flop test continued

initial fork
D = 1;
RST = 1;
#20 D = 0; #40 D = 1;
#50 D = 0; #60 D = 1;
#70 D = 0; #90 D = 1;
#42 RST = 0; #72 RST = 1;
join
endmodule
fork and join

- fork ....join pair specifies that the statements inside fork and join are to be done in parallel.
- The statement delays are with respect to time 0 and each statement is executed independent of others.
- The common variables are updated by each statement. One should be careful so that race conditions should not take place.
Example State Diagram

- States: S0, S1, S2, S3
- Transitions:
  - From S0 to S1 on input 0
  - From S0 to S3 on input 1
  - From S1 to S0 on input 0
  - From S1 to S2 on input 1
  - From S2 to S0 on input 1
  - From S2 to S3 on input 0
  - From S3 to S0 on input 1
  - From S3 to S2 on input 0
module Moore_Model_1( output [1: 0] y_out, input x_in, clock, reset );
reg [1: 0] state;
parameter S0 = 2’b00, S1 = 2’b01, S2 = 2’b10, S3 = 2’b11;
always @ (posedge clock, negedge reset)
if (reset == 0) state <= S0; // Initialize to state S0
else case (state)
  S0: if (∼x_in) state <= S1; else state <= S0;
  S1: if (x_in) state <= S2; else state <= S3;
  S2: if (∼x_in) state <= S3; else state <= S2;
  S3: if (∼x_in) state <= S0; else state <= S3;
endcase
assign y_out = state; // Output of flip-flops
endmodule
module t_Moore_Model_1;
wire [1: 0] t_y_out;
reg t_x_in, t_clock, t_reset;
Moore_Model_1 M0 (t_y_out, t_x_in, t_clock, t_reset);
initial #200 $finish;
initial begin t_clock = 0; forever #5 t_clock = ∼t_clock; end
initial fork
  t_reset = 0;
  #2 t_reset = 1;
  #87 t_reset = 0;
  #89 t_reset = 1;
  #10 t_x_in = 1;
  #30 t_x_in = 0;
  #40 t_x_in = 1;
Moore FSM test

#50 t_x_in = 0;
#52 t_x_in = 1;
#54 t_x_in = 0;
#70 t_x_in = 1;
#80 t_x_in = 1;
#70 t_x_in = 0;
#90 t_x_in = 1;
#100 t_x_in = 0;
#120 t_x_in = 1;
#160 t_x_in = 0;
#170 t_x_in = 1;
join
endmodule
module Moore_Model_STR_Fig_5_20 (output y_out, A, B, input x_in, clock, reset);
wire TA, TB;
// Flip-flop input equations
assign TA = x_in & B;
assign TB = x_in;
// output equation
assign y_out = A & B;
// Instantiate Toggle flip-flops
Toggle_flip_flop_3 M_A (A, TA, clock, reset);
Toggle_flip_flop_3 M_B (B, TB, clock, reset);
endmodule