

Behavioral Modeling

- 2 Kinds of Behavioral Keywords
initial **always**
- Keywords Followed by a Statement of Block
- Block is Group of Statements Enclosed by **begin** and **end** Keywords
- Can Have More Than One **initial** or **always** in a module
- Statements within a Block Execute Sequentially Unless You Control them with Delays
- Unlike Statements not Occurring in a Block; they can Execute in ANY Order

initial Blocks/Statements

- Scheduled to Simulate at time=0 Only
- Used to Initiate a Simulation
- We Have Used These in testbenches to Supply Input Signal Values
- We also Used Explicit Delays to Control When Each Input Signal Value Changed

Verilog Example with Testbench

```
// Stimulus for simple circuit
module stimcrct;
  reg A, B, C;
  wire x, y;
  circuit_with_delay cwd(A, B, C, x, y);
  initial
    begin
      A=1'b0; B=1'b0; C=1'b0;
      #100
      A=1'b1; B=1'b1; C=1'b1;
      #100 $finish;
    end
endmodule

// Description of circuit with delay
module circuit_with_delay (A,B,C,x,y);
  input  A,B,C;
  output x,y;
  wire e;
  and # (30) g1 (e,A,B);
  not # (20) g2 (y,C);
  or # (10) g3 (x,e,y);
endmodule
```

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

always Blocks/Statements

- Simulator Attempts to Schedule Statement(s) in **always** Block at Each Simulator Time Event
- Often Controlled with Delays or User Defined Events
- The @ Character is Used in Verilog for a User Defined Event
- Syntax:

@ (*event control expression*)

Example always Blocks/Statements

```
always f = A & (~B | C);  
scheduled to simulate at every time unit  
always #10 f = A & (~B | C);  
scheduled to simulate at every 10 time units  
always @(A) f = A & (~B | C);  
scheduled to simulate everytime A changes  
always @(A or B or C)  
    f = A & (~B | C);  
scheduled to simulate everytime (A or B or  
C) collectively changes  
always @(A or B or Reset)  
    begin  
        f = A & (~B | C);  
        #5 g = ~f^A;  
    end  
first statement scheduled to simulate everytime (A or  
B or Reset) collectively changes, second statement in block  
scheduled 5 time units after (A or B or Reset) changes
```

Module Structure

- Keep Circuit Under Design in Separate Module for Automatic Synthesis
- Use a Testbench to Supply Input Vectors AND Supporting Circuitry
- Synchronous Sequential Circuits Require a Clock
- Put the Clock Generator in the testbench Module

Example Clock Generator

```
initial  
  begin  
    clock = 1'b0;  
    repeat (30)  
      #10 clock = ~clock;  
  end
```

- **repeat** keyword DOES NOT set up a “loop” as in sequential programming language
- **repeat** causes the following statement to be repeated in the code 30 times
- This clock generator will only provide 15 clock periods (Why not 30?)
- What is the clock period?

Another Example Clock Generator

```
initial  
  begin  
    clock = 1'b0;  
    #300 $finish;  
  end  
always  
  #10 clock = ~clock;
```

- **initial** block sets clock signal to 0
- **always** block causes clock signal to toggle every 10 time units
- **\$finish** scheduled to occur at 300 time units halting the simulation

More on Event Control

```
always @ (event control expression)  
  begin  
    procedural assignment statements ;  
    ...  
  end
```

- LHS of Proc. Assign. Statements is **reg** type, RHS can be **reg** or **wire** as can event control expression
- event control expr. also known as “sensitivity list”
- Level-sensitive versus Edge-sensitive events
- Use Keywords **posedge** and **negedge**

```
always @(posedge clock or negedge reset)
```

Procedural versus Continuous Assignment

- Procedural Assignment Stmtns. Occur within Blocks
- Continuous Occur Outside Blocks – Executed Whenever Event causes Scheduling (“dataflow” modeling)
- Two Types of Procedural Statements
 - 1) Blocking
 - a) Use = for Assignment Operator
 - b) Executed Sequentially in Order of Appearance
 - 2) Non-blocking
 - a) Use <= for Assignment Operator
 - b) Make LHS Assignment After ALL RHS Expressions Evaluated First

Blocking/Non-blocking Example

- Assume Initially **A=1, B=2, C=3**
- Assume the Following are Within Blocks:

```
B = A;  
C = B + 1;
```

Blocking

```
B <= A;  
C <= B + 1;
```

Non-Blocking

- When Blocking statements finish Execution:

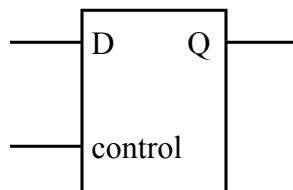
A=1, B=1, C=2

- When Non-Blocking statements finish Execution:

A=1, B=1, C=3

Data (D) Latch

```
//HDL Example 5-1  
//-----  
//Description of D latch (See Fig.5-6)  
module D_latch (Q,D,control);  
    output Q;  
    input D,control;  
    reg Q;  
    always @ (control or D)  
        if (control) Q = D;    //Same as: if (control = 1)  
endmodule
```



D FLIP-FLOPS

```
//HDL Example 5-2
//-----
//D flip-flop
module D_FF (Q,D,CLK);
    output Q;
    input D,CLK;
    reg Q;
    always @ (posedge CLK)
        Q = D;
endmodule
```

D-FF with Single Synchronous Input

```
//D flip-flop with asynchronous reset.
module DFF (Q,D,CLK,RST);
    output Q;
    input D,CLK,RST;
    reg Q;
    always @(posedge CLK or negedge RST)
        if (~RST) Q = 1'b0;
// Same as: if (RST = 0)
        else Q = D;
endmodule
```

D-FF with Single Synchronous Input and Asynchronous reset (RST)

JK and T FLIP-FLOPS

$$Q(t+1) = Q(t) \oplus T$$

$$Q(t+1) = J\bar{Q}(t) + \bar{K}Q(t)$$

```
//T flip-flop from D
// flip-flop and gates
module TFF (Q,T,CLK,RST);
    output Q;
    input T,CLK,RST;
    wire DT;
    assign DT = Q ^ T;
//Instantiate the D flip-flop
    DFF TF1 (Q,DT,CLK,RST);
endmodule
```

```
//JK flip-flop from
// D flip-flop and gates
module JKFF (Q,J,K,CLK,RST);
    output Q;
    input J,K,CLK,RST;
    wire JK;
    assign JK = (J & ~Q)
                | (~K & Q);
//Instantiate D flipflop
    DFF JK1 (Q,JK,CLK,RST);
endmodule
```

JK FLIP-FLOP

```
//HDL Example 5-4
//-----
// Functional description of JK flip-flop
module JK_FF (J,K,CLK,Q,Qnot);
    output Q,Qnot;
    input J,K,CLK;
    reg Q;
    assign Qnot = ~ Q ;
    always @ (posedge CLK)
        case ({J,K})
            2'b00: Q = Q;
            2'b01: Q = 1'b0;
            2'b10: Q = 1'b1;
            2'b11: Q = ~ Q;
        endcase
endmodule
```

Description Based on Characteristic Table Directly

JK FLIP-FLOP

```
// Functional description of JK flip-flop
module JK_FF (J,K,CLK,Q,Qnot,RST,PST);
    output Q,Qnot;
    input J,K,CLK;
    reg Q;
    assign Qnot = ~ Q ;
    always @ (posedge CLK or negedge RST or negedge PST)
        if (~RST and ~PST)
            Q = 1'bx;
        else if (~RST and PST)
            Q = 1'b0;
        else if (~PST and RST)
            Q = 1'b1;
        else
            case ({J,K})
                2'b00: Q = Q;
                2'b01: Q = 1'b0;
                2'b10: Q = 1'b1;
                2'b11: Q = ~ Q;
            endcase
endmodule
```

D FLIP-FLOP

```
// Functional description of JK flip-flop
module dFlop (preset, clear, q, clock, d);
    input  preset, clear, clock, d;
    output q;
    reg    q;

    always @(preset or clear)
        begin
            if (!clear)
                #10 assign q = 0;
            else if (!preset)
                #10 assign q = 1;
            else
                #10 deassign q;
        end

    always @(negedge clock)
        q = #10 d;
endmodule
```

FSM Behavioral Modeling

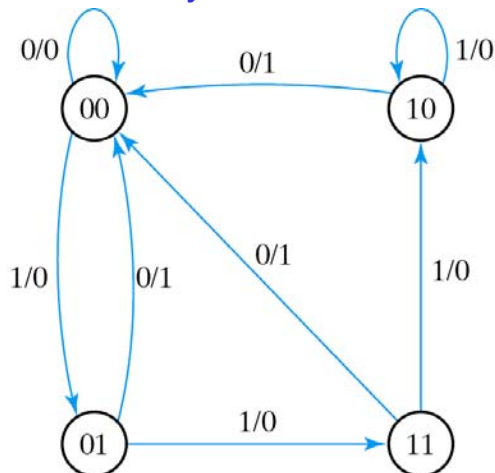
- State Diagrams (SDs) Describe Behavior of FSMs
- Translating Directly from SDs to Verilog is Advantageous
 - No Worry about Mistake in Logic Simplification
 - No Tedious Tables to Create
 - Automatic Tools (synthesis) Create the Schematic Directly
 - Synthesis Tools can Handle Very Large FSMs (100s even 1000s of DFFs)
 - Can EASILY Change State Assignment

FSM Behavioral Modeling

- Basic Construct is **case** Statement within an **always** Block
- Internally Keep two **regs**, **Prstate** and **Nxtstate**
- Use Three **always** blocks (one way to do it)
 - Interact by causing events in one another
 - first **always** block handles asynchronous and synchronous events
 - second **always** block determines the next state
 - third **always** block evaluates output signals

Verilog FSM Behavioral Model Example

Mealy Machine



© 2002 Prentice Hall, Inc., Fig. 5-16 State Diagram of the Circuit of Fig. 5-15
M. Morris Mano
DIGITAL DESIGN, 3e.

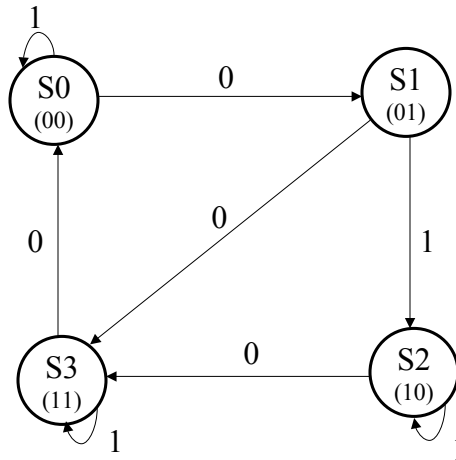
Verilog FSM Behavioral Model Example

```
//HDL Example 5-5
//-----
//Mealy state diagram (Fig 5-16)
module Mealy_md1 (x,y,CLK,RST);
  input x,CLK,RST;
  output y;
  reg y;
  reg [1:0] Prstate, Nxtstate;
  parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;
  always @ (posedge CLK or negedge RST)
    if (~RST) Prstate = S0; //Initialize to state S0
    else Prstate = Nxtstate; //Clock operations
  always @ (Prstate or x) //Determine next state
    case (Prstate)
      S0: if (x) Nxtstate = S1;
      S1: if (x) Nxtstate = S3;
           else Nxtstate = S0;
      S2: if (~x)Nxtstate = S0;
      S3: if (x) Nxtstate = S2;
           else Nxtstate = S0;
    endcase
endmodule
```

Verilog FSM Behavioral Model Example

```
    always @ (Prstate or x) //Evaluate output
      case (Prstate)
        S0: y = 0;
        S1: if (x) y = 1'b0; else y = 1'b1;
        S2: if (x) y = 1'b0; else y = 1'b1;
        S3: if (x) y = 1'b0; else y = 1'b1;
      endcase
endmodule
```

Verilog FSM Behavioral Model Example Moore Machine



Verilog FSM Behavioral Model Example Moore Machine

```

//HDL Example 5-6
//-----
//Moore state diagram (Fig. 5-19)
module Moore_mdl (x,AB,CLK,RST);
  input x,CLK,RST;
  output [1:0]AB;
  reg [1:0] state;
  parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;
  always @ (posedge CLK or negedge RST)
    if (~RST) state = S0; //Initialize to state S0
    else
      case (state)
        S0: if (~x) state = S1; else state = S0;
        S1: if (x) state = S2; else state = S3;
        S2: if (~x) state = S3; else state = S2;
        S3: if (~x) state = S0; else state = S3;
      endcase
  assign AB = state; //Output of flip-flops
endmodule
  
```

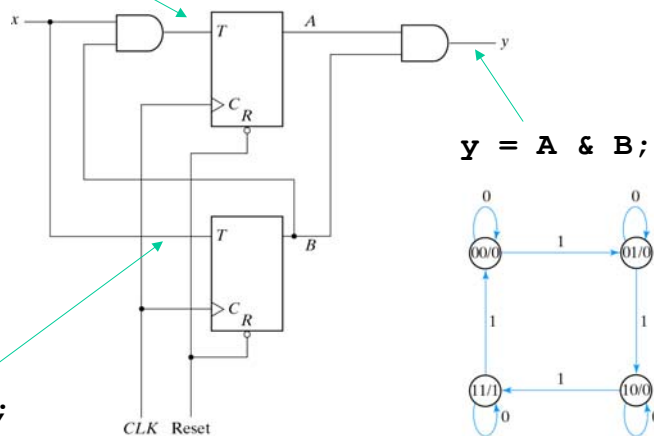
FSM Structural Modeling

- Can Gate-level Primitives or dataflow statements for combinational part
- “Netlist” Model
 - Excitation Circuits
 - Output Circuits
- Use behavioral description for individual FFs
- This Method Requires
 - Initial State Assignment
 - Generation of Simplified Logic
 - Advantage is that Easier to Model Delays Accurately
 - Back-annotation
 - Commercial Tools Can Generate Automatically

FSM Structural Modeling Example

$TA = x \& B;$

$TB = x;$



(a) Circuit diagram

(b) State diagram

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.

Fig. 5-20 Sequential Circuit with T Flip-Flops

FSM Structural Modeling Example

```
//HDL Example 5-7
//-----
//Structural description of
// sequential circuit
//See Fig. 5-20(a)
module Tcircuit (x,y,A,B,CLK,RST);
    input x,CLK,RST;
    output y,A,B;
    wire TA,TB;
//Flip-flop input equations
    assign TB = x,
           TA = x & B;
//Output equation
    assign y = A & B;
//Instantiate T flip-flops
    T_FF BF (B,TB,CLK,RST);
    T_FF AF (A,TA,CLK,RST);
endmodule
```

```
//T flip-flop
module T_FF
(Q,T,CLK,RST);
    output Q;
    input T,CLK,RST;
    reg Q;
    always @ (posedge
CLK
           or negedge RST)
        if (~RST) Q =
1'b0;
        else Q = Q ^ T;
endmodule
```

FSM Structural Example Testbench

```
//Stimulus for testing sequential circuit
module testTcircuit;
    reg x,CLK,RST; //inputs for circuit
    wire y,A,B; //output from circuit
    Tcircuit TC (x,y,A,B,CLK,RST); // instantiate circuit
    initial
        begin
            RST = 0;
            CLK = 0;
            #5 RST = 1;
            repeat (16)
                #5 CLK = ~CLK;
            end
        initial
            begin
                x = 0;
                #15 x = 1;
                repeat (8)
                    #10 x = ~ x;
                end
            endmodule
```

FSM Structural Example Simulation Output

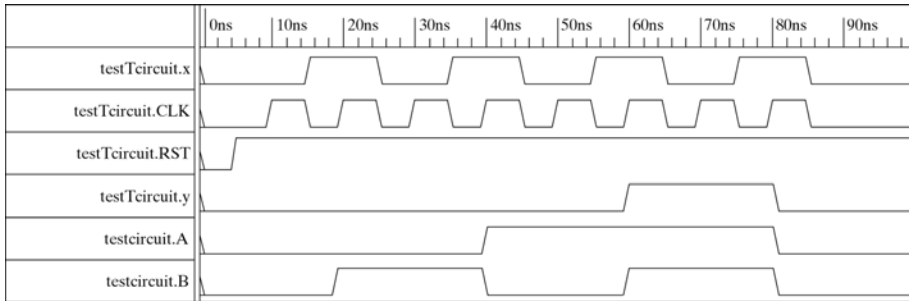


Fig. 5-21 Simulation Output of HDL Example 5-7

© 2002 Prentice Hall, Inc.
M. Morris Mano
DIGITAL DESIGN, 3e.