Lecture 3: Modeling Sequential Logic in Verilog HDL

Blocking(=) vs Nonblocking (<=)

### Assignment

```verilog
always @ (b)
begin
    a = b;
    c = a;
end
```

**final value:**

- `a = b`  
- `c = old a`

**blocking statement:** executed sequentially within an always block

```verilog
always @(posedge clk)
begin
    a <= b;
    c <= a;
end
```

**final value:**

- `a = b`

**nonblocking statement:** executed in parallel at the end of the always block

Use non-blocking for flip flop inference

posedge/negedge require nonblocking

```verilog
always @ (posedge clock)
begin
    b <= a;       // swap a and b
    a <= b;
end
```

If using blocking for flipflop inference, we may not get what you want.

```verilog
always @(posedge clock)
begin
    b = a;       // b = a, a = a
end
```

### Procedural assignments

- **Blocking assignment (=)**
  - Regular assignment inside procedural block
  - Assignment takes place immediately
  - LHS must be a register

- **Nonblocking assignment (<=)**
  - Compute RHS
  - Assignment takes place at end of block
  - LHS must be a register
**Execution order**

at Q1 (in any order)
- Evaluate RHS of all non-blocking assignments
- Evaluate RHS and change LHS of all blocking assignments (execution of all blocking assignments is in the order it is encountered)
- Evaluate RHS and change LHS of all continuous assignments
- Evaluate inputs and change outputs of all primitives

at Q2
- Change LHS of all non-blocking assignments at the same time

All non-blocking assignments are evaluated using the values the variables have when your design entered the always block.

**Non-blocking assignment** is also known as an RTL assignment if used in an always block triggered by a clock edge all flip-flops change together

```plaintext
always @(posedge clk)
begin
    B = A;
    C = B;
    D = C;
end
//what does this circuit do?
B=A, C=A, D=A
```

```plaintext
always @(posedge clk)
begin
    B <= A;
    C <= B;
    D <= C;
end
//What does this circuit do?
a shift register
```

**Finite State Machine Review**

A finite state machine has
- a state register that holds current state
- some combinational logic that calculates the next state and outputs based on the current state and/or the current inputs

A finite state machine is a feedback system which updates on each clock.

**Moore machine:** output based on current state

**Mealy machine:** based on current state and inputs
Comparison of Mealy and Moore machines

- Mealy machines tend to have less states
  - different outputs on arcs (n') rather than states (n)
- Moore machines are safer to use
  - outputs change at clock edge (always one cycle later)
- In Mealy machines, input change can cause output change as soon as logic is done
  - a big problem when two machines are interconnected – asynchronous feedback may occur if one isn’t careful
- Mealy machines react faster to inputs
  - react in same cycle – don’t need to wait for clock
  - in Moore machines, more logic may be necessary to decode state into outputs – more gate delays after clock edge

Finite State Machine Design

Determine inputs/outputs
Determine machine types (Mealy vs. Moore)
Determine machine flow (state diagram)
State assignment
Implementation

A Moore Machine

Using D flip flops next state logic
- \( D_0 = Q_3x' + Q_2x + Q_0x \)
- \( D_1 = Q_3x' \)
- \( D_2 = Q_0x' + Q_1x \)
- \( D_3 = Q_3x + Q_3x' \)

Output logic
- \( Z_2 = Q_2; Z_1 = Q_1; Z_0 = Q_3 + Q_0 \)

Write a complete structural Verilog model of this simple Moore machine.

module DFF(q, d, clk, reset, preset);
  input d, clk, reset, preset;
  output q;
  reg q;

  always @(posedge clk or negedge reset or negedge preset)
  begin
    if (~reset)
      q <= 0;
    else if (~preset)
      q <= 1;
    else
      q <= d;
  end
endmodule
module clock (clk);
  output clk;
  reg clk;

  initial
  #1 clk=0;

  always begin
  #10 clk = ~clk;
  end
endmodule

module OneHotNS(D, Q, x);
  input x;
  input [3:0] Q;
  output [3:0] D;
  reg [3:0] D;

  always @(Q or x)
  begin
    D[2] =(Q[0] & ~x) | (Q[1] & x);
    D[0] = (Q[1] & ~x) | (Q[0] & x) | (Q[3] & x);
  end
endmodule

module OneHotOutput(Z, Q);
  input [3:0] Q;
  output [2:0] Z;
  reg [2:0] Z;

  always @ (Q) begin
    Z[2] = Q[2];
    Z[1] = ~Q[2];
    Z[0] = Q[2] | Q[0];
  end
endmodule

module OneHotFSM(Z, x, clk, reset);
  input x, clk, reset;
  output [2:0] Z;
  wire [3:0] D, Q;

  DFF    OneHotD3(Q[3], D[3], clk, reset, 1'b1),
          OneHotD2(Q[2], D[2], clk, reset, 1'b1),
          OneHotD1(Q[1], D[1], clk, reset, 1'b1),
          OneHotD0(Q[0], D[0], clk, 1'b1, reset);

  OneHotNS     NextStateUnit(D, Q, x);
  OneHotOutput   OutputUnit(Z, Q);
endmodule
module testbench (Z, x, clk, reset);
  input [2:0] Z;
  input clk;
  output x, reset;
  reg x, reset;
  reg[11:0] data;
  integer i;

  initial begin
    $monitor ($time,, "Z =  %b, x =  %b, clock=  %b", Z, x, clk);
    reset = 0;
    data = 12'b001001001011; // x = 110100100100 (from L to R)
    #1 reset = 1;
    for (i = 0; i <= 11; i = i + 1) begin
      x = data[i];#20;
    end
    #1 $finish;
  end
endmodule

module system
  wire clk, reset, x;
  wire [2:0] Z;

  testbench testunit (Z, x, clk, reset);
  clock clockunit (clk);
  OneHotFSM FSMunit(Z, x, clk, reset);
endmodule

0 Z = 011, x = x, clock= x
1 Z = 011, x = 1, clock= 0
10 Z = 011, x = 1, clock= 1
20 Z = 011, x = 1, clock= 0
30 Z = 011, x = 1, clock= 1
40 Z = 011, x = 1, clock= 0
41 Z = 011, x = 0, clock= 0
50 Z = 101, x = 0, clock= 1
60 Z = 101, x = 0, clock= 0
61 Z = 101, x = 1, clock= 0
70 Z = 010, x = 1, clock= 1
80 Z = 010, x = 1, clock= 0
81 Z = 010, x = 0, clock= 0
90 Z = 010, x = 0, clock= 1
100 Z = 010, x = 0, clock= 0
110 Z = 010, x = 0, clock= 1
120 Z = 010, x = 0, clock= 0
121 Z = 010, x = 1, clock= 0
130 Z = 011, x = 1, clock= 1
140 Z = 011, x = 1, clock= 0
141 Z = 011, x = 0, clock= 0
150 Z = 101, x = 0, clock= 1
160 Z = 101, x = 0, clock= 0
170 Z = 010, x = 0, clock= 1
180 Z = 010, x = 0, clock= 0
181 Z = 010, x = 1, clock= 0
190 Z = 101, x = 1, clock= 1
191 Z = 011, x = 1, clock= 0
200 Z = 101, x = 1, clock= 0
201 Z = 101, x = 0, clock= 0
210 Z = 010, x = 0, clock= 1
220 Z = 010, x = 0, clock= 0
230 Z = 011, x = 0, clock= 1
240 Z = 011, x = 0, clock= 0
Another finite state machine given the state transition diagram

```verilog
module sfsm (clk, reset, in, out);
  input clk, reset, in;
  output out;
  parameter zero = 2'b00;
  parameter one1 = 2'b01;
  parameter two1s = 2'b10;
  reg out;
  reg [2:1] state; // state variables
  reg [2:1] next_state;
  always @(posedge clk)
    if (reset) state <= zero; else state <= next_state;
  always @(in or state) // generate Next State
    case (state)
      zero: begin // last input was a zero
        if (in) next_state = one1; else next_state = zero;
      end
      one1: begin // we've seen one 1
        if (in) next_state = two1s; else next_state = zero;
      end
      two1s: // we've seen at least 2 ones
        begin
          if (in) next_state = two1s; else next_state = zero;
        end
    endcase
  endmodule
```

An always block for a combinational function.

A module can have more than one of these always blocks.

Partition a design into several modules
Tools optimize each module separately

Combining several modules into one module, tools optimize this single module.

A Mealy Machine

always @(state)
  case (state)
    zero: out = 0;
    one1: out = 0;
    two1s: out = 1;
  endcase

Binary encoding
state A = 00; state B = 01; state C = 10; state D = 11
module mealyFSM (out, x, clk, reset);
  input x, clk, reset;
  output [1:0] out;
  reg [1:0] out, curstate, nextstate;

  parameter [1:0] A=0, B=1, C=2, D=3;

  // state transition
  always @(posedge clk or negedge reset) begin
    $display(" ***********************");
    if (~reset) begin
      curstate <= 0;
      $display("state 0 ");
    end else begin
      curstate <= nextstate;
      $display("state %d", nextstate);
    end
  end

  // next state logic and output logic combinational
  always @(curstate or x) begin
    case (curstate)
      A: begin
        out = 3;
        nextstate = D;
      end
      B: begin
        out = (x == 1)? 2 : 0;
        nextstate = (x == 1)? A : D;
      end
      C: begin
        out = (x == 1)? 1 : 0;
        nextstate = (x == 1)? B : D;
      end
      D: begin
        out = (x == 1)? 1 : 0;
        nextstate = (x == 1)? C : D;
      end
    endcase
  end
endmodule

module clock(clk);
  output clk;
  reg clk;
  initial
    clk = 0;
  always
    #10 clk = ~clk;
endmodule

module testbench (out, x, clk, reset);
  output x, reset;
  input [1:0] out;
  input clk;
  reg x, reset;
  reg[11:0] data; integer I;
  initial begin
    $monitor($time,, "clk = %b, x = %b, out = %d, clk, x, out");
    reset = 0; // set state to A
    data = 12'b100101101111;
    #1 reset = 1;
    for (i = 11; i >= 0; i = i -1) begin
      x = data[i];
      #20;
    end
    $finish;
  end
endmodule
module system;
    wire [1:0] out;
    wire x, clk, reset;
    mealyFSM M1(out, x, clk, reset);
    clock timer(clk);
    testbench test(out, x, clk, reset);
endmodule

state 0
  0 clk = 0, x = x, out = 3
  1 clk = 0, x = 1, out = 3
  ***********************
state 3
  10 clk = 1, x = 1, out = 1
  20 clk = 0, x = 1, out = 1
  21 clk = 0, x = 0, out = 0
  ***********************
state 3
  30 clk = 1, x = 0, out = 0
  40 clk = 0, x = 0, out = 0
  ***********************
state 3
  50 clk = 1, x = 0, out = 0
  60 clk = 0, x = 0, out = 0
  61 clk = 0, x = 1, out = 1
  ***********************
state 2
  70 clk = 1, x = 1, out = 1
  80 clk = 0, x = 1, out = 1
  ***********************
state 1
  90 clk = 1, x = 1, out = 2
  100 clk = 0, x = 1, out = 2
  101 clk = 0, x = 1, out = 1
  ***********************
state 3
  110 clk = 1, x = 0, out = 0
  120 clk = 0, x = 1, out = 1
  ***********************
state 2
  130 clk = 1, x = 1, out = 1
  140 clk = 0, x = 1, out = 2
  141 clk = 0, x = 0, out = 0
  ***********************
state 3
  150 clk = 1, x = 0, out = 0
  160 clk = 0, x = 0, out = 0
  161 clk = 0, x = 1, out = 1
  ***********************
state 2
  170 clk = 1, x = 1, out = 1
  180 clk = 0, x = 1, out = 1
  ***********************
state 1
  190 clk = 1, x = 1, out = 2
  200 clk = 0, x = 1, out = 2
  ***********************
state 3
  210 clk = 1, x = 1, out = 3
  220 clk = 0, x = 1, out = 3
  ***********************

state 3
  230 clk = 1, x = 1, out = 1
  240 clk = 0, x = 1, out = 1
  ***********************
copied from
Silos output window