CSE-308: Digital System Design

Lecture 5

Blocking & Non-blocking Assignments

- Sequential statements within procedural blocks ("always" and "initial") can use two types of assignments:
  - Blocking assignment
    - Uses the '=' operator
  - Non-blocking assignment
    - Uses the '<=' operator

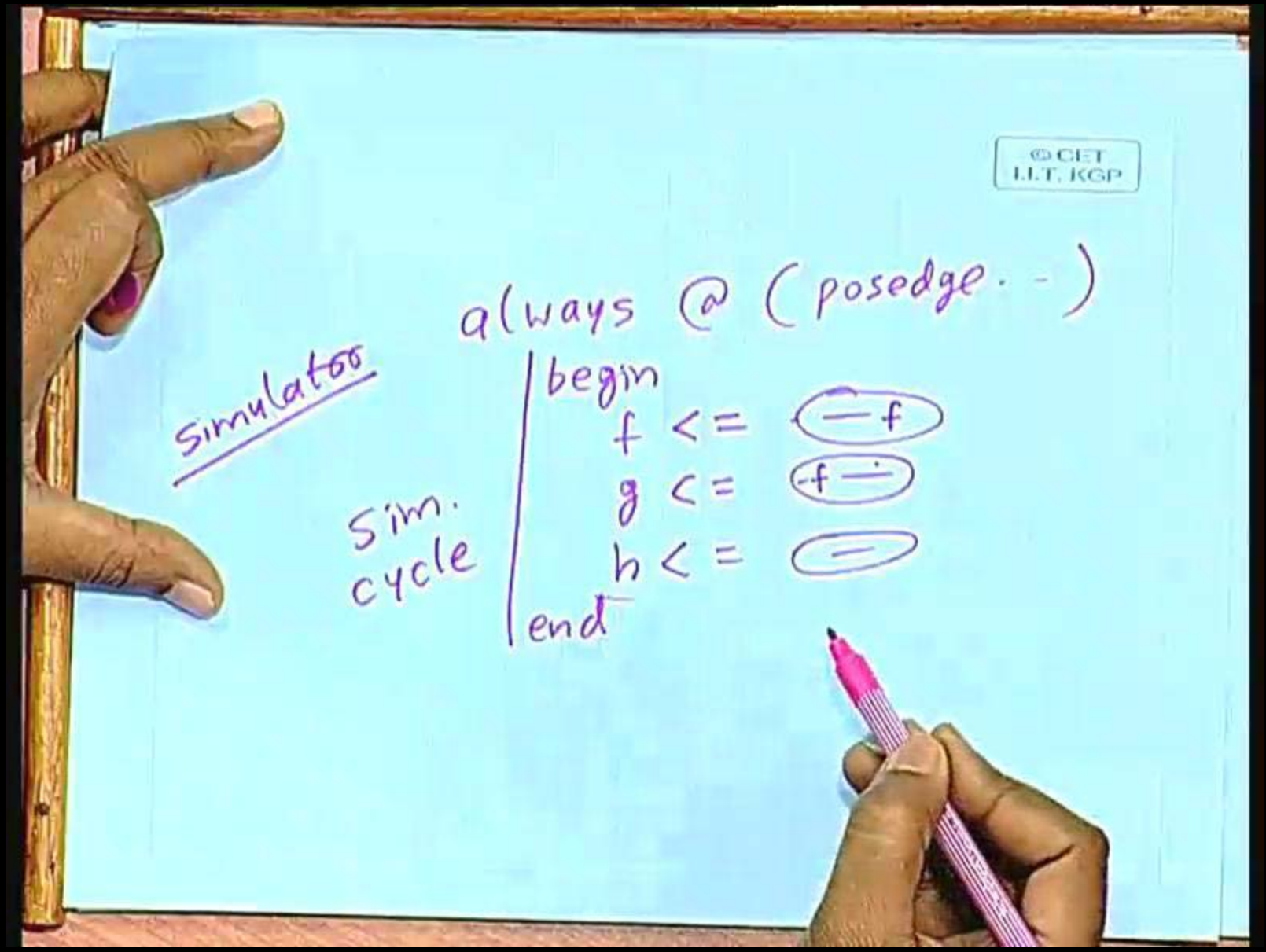
Blocking Assignment (using '=')

- Most commonly used type.
- The target of assignment gets updated before the next sequential statement in the procedural block is executed.
- A statement using blocking assignment blocks the execution of the statements following it, until it gets completed.
- Recommended style for modeling combinational logic.

```
f = a + b;
```

Non-Blocking Assignment (using '<=')

- The assignment to the target gets scheduled for the end of the simulation cycle.
  - Normally occurs at the end of the sequential block.
  - Statements subsequent to the instruction under consideration are not blocked by the assignment.
- Recommended style for modeling sequential logic.
  - Can be used to assign several 'reg' type variables synchronously, under the control of a common clock.


© CET
I.I.T. KGP
always @ (posedge . .)
begin
f <=
g <=
h <=
end
Simulator
Sim.
cycle

Non-Blocking Assignment (using '<=')

- The assignment to the target gets scheduled for the end of the simulation cycle.
  - Normally occurs at the end of the sequential block.
  - Statements subsequent to the instruction under consideration are not blocked by the assignment.
- Recommended style for modeling sequential logic.
  - Can be used to assign several 'reg' type variables synchronously, under the control of a common clock.

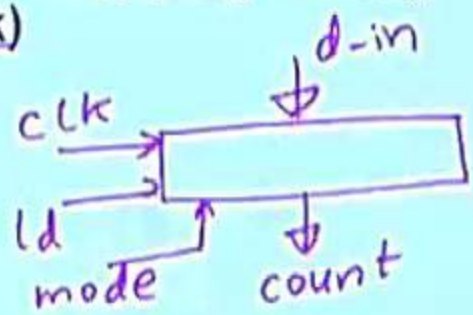
Some Rules to be Followed

- Verilog synthesizer ignores the delays specified in a procedural assignment statement.
  - May lead to functional mismatch between the design model and the synthesized netlist.
- A variable cannot appear as the target of *both* a blocking and a non-blocking assignment.
  - Following is not permissible:

```
value = value + 1;
value <= init;
```

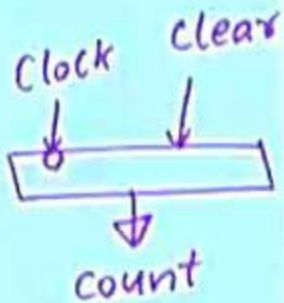
#10
#20

```
// Up-down counter (synchronous clear)

module  counter (mode, clr, ld, d_in, clk, count);
   input mode, clr, ld, clk;     input [0:7] d_in;
   output [0:7]  count;          reg [0:7]  count;
   always  @ (posedge  clk)
      if  (ld)
         count  <= d_in;
      else  if  (clr)
               count  <= 0;
            else  if  (mode)
                     count  <=  count + 1;
                  else
                     count  <=  count + 1;
endmodule
```



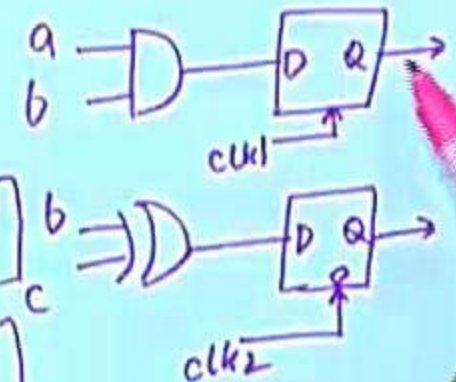
// Parameterized design:: an N-bit counter

```
module  counter (clear, clock, count);
   parameter  N = 7;
   input  clear, clock;
   output [0:N]  count;        reg [0:N]  count;

   always  @ (negedge  clock)
      if  (clear)
          count  <= 0;
       else
          count  <=  count + 1;
endmodule
```

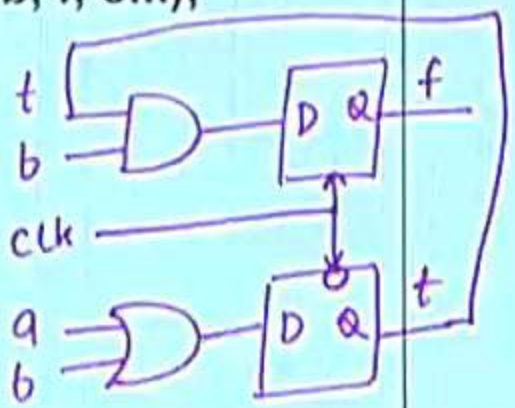


// Using more than one clocks in a module

```
module multiple_clk (clk1, clk2, a, b, c, f1, f2);
   input clk1, clk2, a, b, c;
   output f1, f2;
   reg f1, f2;
   always @(posedge clk1)
      f1 <= a & b;
   always @(negedge clk2)
      f2 <= b ^ c;
endmodule
```

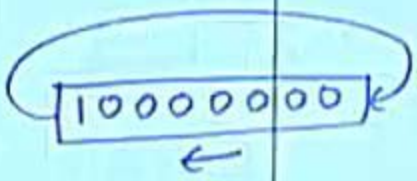


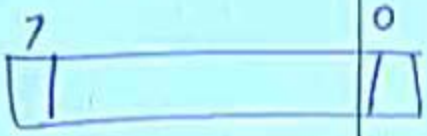
```
// Using multiple edges of the same clock

module multi_phase_clk (a, b, f, clk);
   input  a, b, clk;
   output  f;
   reg  f, t;
   always  @ (posedge  clk)
      f  <=  t & b;
   always  @ (negedge  clk)
      t  <=  a | b;
endmodule
```



A Ring Counter Example


```
module ring_counter (clk, init, count);
   input clk, init;       output [7:0] count;
   reg [7:0] count;
   always @ (posedge clk)
   begin
      if (init)
         count = 8'b10000000;
      else begin
               count    = count << 1;
               count[0] = count[7];
            end
   end
endmodule
```





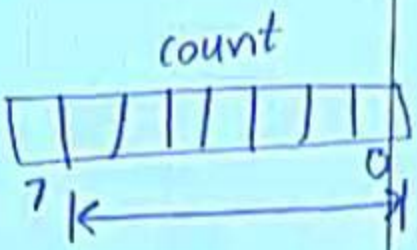
A Ring Counter Example (Modified--1)


```
module ring_counter_modi1 (clk, init, count);
   input clk, init;       output [7:0] count;
   reg [7:0] count;
   always @(posedge clk)
   begin
      if (init)
         count = 8'b10000000;
      else begin
               count    <= count << 1;
               count[0] <= count[7];
           end
   end
endmodule
```

10000000

A Ring Counter Example (Modified – 2)

```
module ring_counter_modi2 (clk, init, count);
   input clk, init;        output [7:0] count;
   reg [7:0] count;
   always @ (posedge clk)
   begin
      if (init)
         count = 8'b10000000;
      else
         count = {count[6:0], count[7]};
endmodule
```

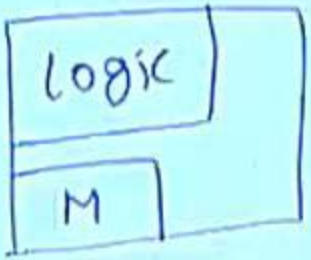


About "Loop" Statements

- Verilog supports four types of loops:
  - 'while' loop ✓
  - 'for' loop ✓
  - 'forever' loop ✓
  - 'repeat' loop ✓
- Many Verilog synthesizers supports only 'for' loop for synthesis:
  - Loop bound must evaluate to a constant.
  - Implemented by unrolling the 'for' loop, and replicating the statements.

Modeling Memory

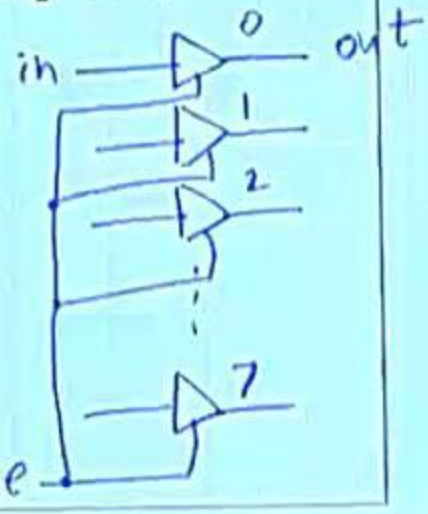
- Synthesis tools are usually not very efficient in synthesizing memory.
  - Best modeled as a component.
  - Instantiated in a design.
- Implementing memory as a two-dimensional register file is inefficient.



```
module  memory_example (en, clk, adbus, dbus,
                                                rw);
    parameter  N = 16;
    input  en, rw, clk;
    input  [N-1:0]  adbus;
    output [N-1:0]  dbus;
    ..............
    ROM  Mem1  (clk, en, rw, adbus, dbus);
    ..............
endmodule
```

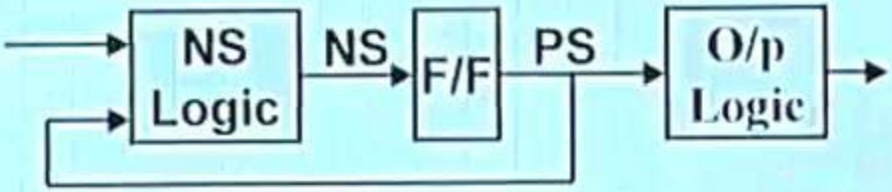
Modeling Tri-state Gates


```
module  bus_driver  (in, out, enable);
   input  enable;          input [0:7]  in;
   output [0:7]  out;      reg [0:7]  out;

   always  @ (enable or in)
      if  (enable)
         out = in;
      else
         out = 8'bz;
endmodule;
```

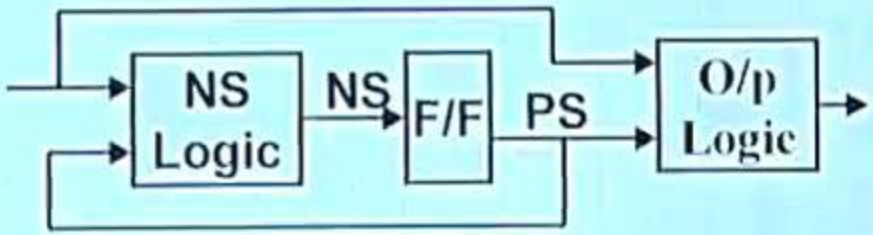


Modeling Finite State Machines

- Two types of FSMs
  - Moore Machine



  - Mealy Machine



Output logic :

* Latched
* Non-latched