EE-421: Digital System Design

Assignment Statements:
Blocking (=) vs Non-Blocking (<=)

Dr. Rehan Ahmed [rehan.ahmed@seecs.edu.pk]

NUST

SCHOOL OF ELECTRICAL ENGINEERING &
COMPUTER SCIENCE (SEECS)

Statements Ordering in an Always Block

Always Block: Statements Order

- An important property of the always block is that:
  - the statements it contains are evaluated in the order given in the code:
    - [**STATEMENTS ORDER IS IMPORTANT**]
  - [!!!] This contrasts with the continuous assignment statements (recall assign statement), which are evaluated concurrently and hence have no meaningful order!

Example: Statement Ordering inside an Always block

```
module mux (w0, w1, s, f);
  input w0, w1, s;
  output reg f;
  always @(w0, w1, s)
  begin
    f = w0;
    if (s == 1)
      f = w1;
  end
endmodule
```

Still a MUX

```
module mux (w0, w1, s, f);
  input w0, w1, s;
  output reg f;
  always @(w0, w1, s)
  begin
    if (s == 1)
        f = w1;
    f = w0;
  end
endmodule
```

Not a MUX Anymore

Verilog semantics specify that a signal assigned multiple values in an always construct retains the last assignment.

Verilog Procedural Assignment Statements

Procedural Assignment Statements

- There are two kinds of assignments in an always block:
  1. Blocking assignments,
     - denoted by the **=** symbol
  2. Non-blocking assignments,
     - denoted by the **<=** symbol.

Blocking and Non-Blocking Assignments

Blocking (=)

```
always @ (a)
begin
   a = 2’b01;
// a is 2’b01
   b = a;
// b is now 2’b01 as well
end
```

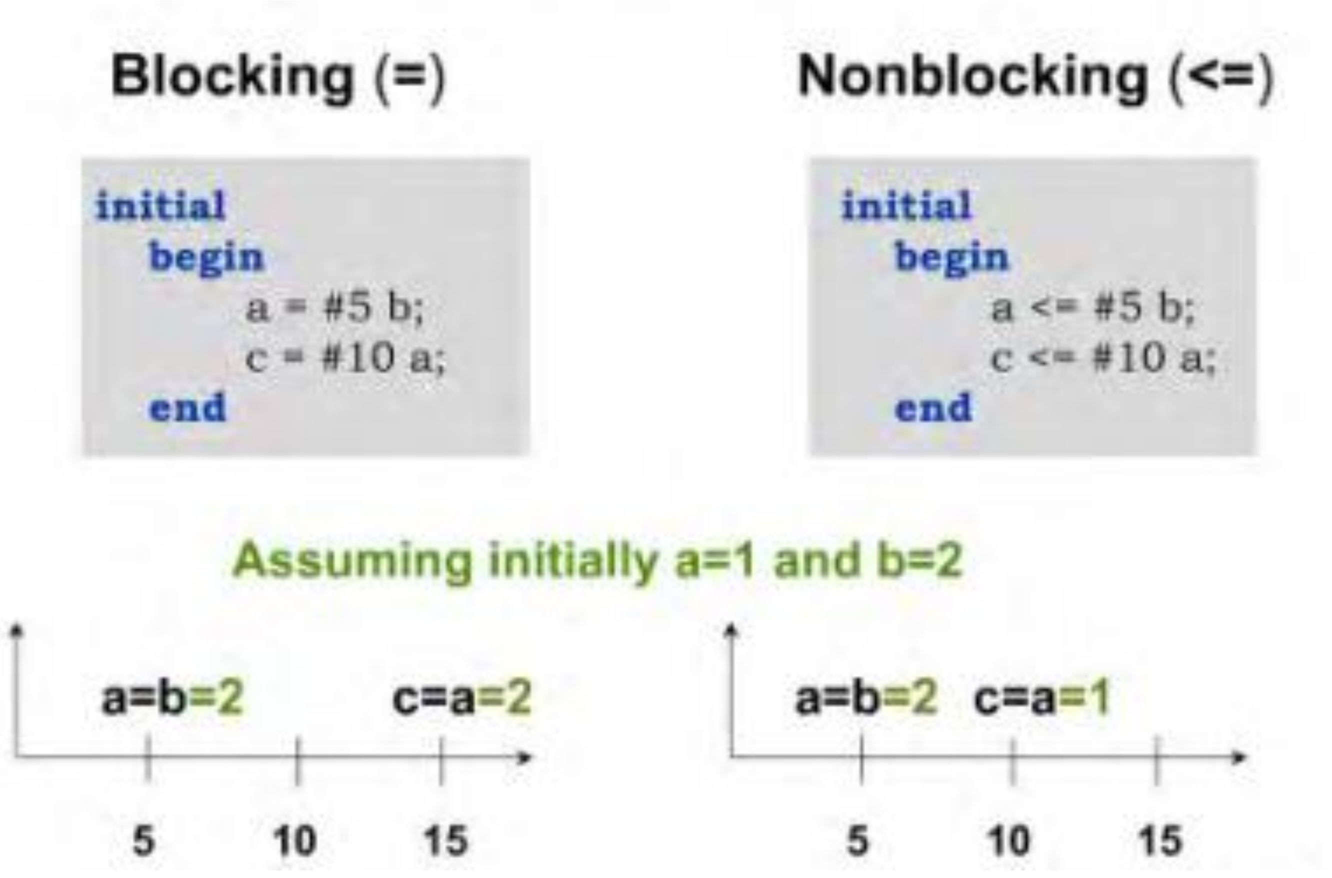
- Each assignment is made immediately
- Process waits until the first assignment is complete, it blocks progress

Non-blocking (<=)

```
always @ (a)
begin
   a <= 2’b01;
   b <= a;
// all assignments are made here
// b is not (yet) 2’b01
end
```

- All assignments are made at the end of the block
- All assignments are made in parallel, process flow is not-blocked

Example:
Blocking and Non-Blocking Assignments



Blocking (=)

```
initial
   begin
         a = #5 b;
         c = #10 a;
   end
```

Nonblocking (<=)

```
initial
   begin
         a <= #5 b;
         c <= #10 a;
   end
```

Assuming initially a=1 and b=2

Credit: Altera Tutorial – Verilog HDL Basics

Example: Blocking Assignment

- Assume all inputs are initially ‘0’

```
always @ ( * )
  begin
    p    = a ^ b ;         // p    = 0  1
    g    = a & b ;         // g    = 0  0
    s    = p ^ cin ;       // s    = 0  1
    cout = g | (p & cin) ; // cout = 0  0
  end
```

- If a changes to ‘1’
  - All values are updated in order they are specified!

The Same Example: Non-Blocking Assignment

- Assume all inputs are initially ‘0’

```
always @ ( * )
  begin
    p    <= a ^ b ;          // p    = 0  1
    g    <= a & b ;          // g    = 0  0
    s    <= p ^ cin ;        // s    = 0  0
    cout <= g | (p & cin) ; // cout = 0  0
  end
```

- If a changes to ‘1’
  - All assignments are concurrent
  - When s is being assigned, p is still 0

Summary: Blocking and Non-Blocking Assignments

- There are two kinds of assignments in an always block:
  1. Blocking assignments,
     - denoted by the **=** symbol
     - Executed in the order they are specified in a sequential block
  2. Non-blocking assignments,
     - denoted by the **<=** symbol
     - Allow scheduling of assignments without blocking execution of the statements that follow in a sequential block

Synthesis of Blocking vs Non-Blocking Assignments

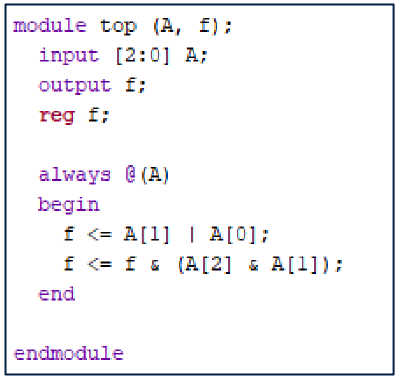
Example: Blocking vs Non-Blocking

- Draw the circuit (netlist) corresponding to each of the following code snippets:

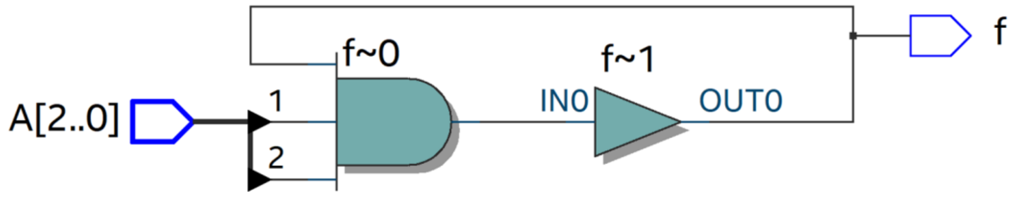
```
module top (A, f);
  input [2:0] A;
  output f;
  reg f;

  always @(A)
  begin
    f <= A[1] | A[0];
    f <= f & (A[2] & A[1]);
  end

endmodule
```

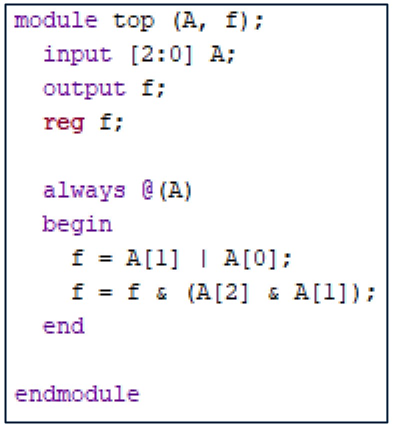
```
module top (A, f);
  input [2:0] A;
  output f;
  reg f;

  always @(A)
  begin
    f = A[1] | A[0];
    f = f & (A[2] & A[1]);
  end

endmodule
```

Example: Blocking vs Non-Blocking

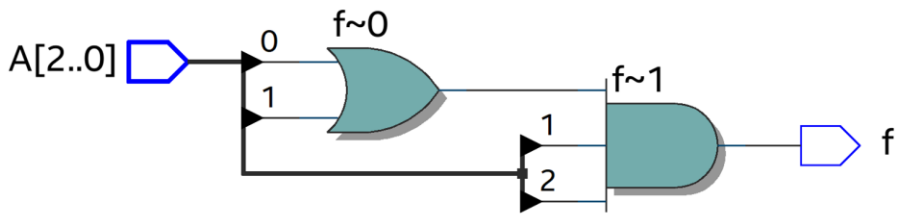


```
module top (A, f);
  input [2:0] A;
  output f;
  reg f;

  always @(A)
  begin
    f <= A[1] | A[0];
    f <= f & (A[2] & A[1]);
  end

endmodule
```





```
module top (A, f);
  input [2:0] A;
  output f;
  reg f;

  always @(A)
  begin
    f = A[1] | A[0];
    f = f & (A[2] & A[1]);
  end

endmodule
```



- There are two key aspects of the Verilog semantics relevant to this code:
  - 1. The results of non-blocking assignments are visible only after all of the statements in the always block have been evaluated.
  - 2. When there are multiple assignments to the same variable inside an always block, the result of the last assignment is maintained.

Blocking vs Non-Blocking: Avoid Confusion

- Notion of "current time step" is tricky in synthesis, so to guarantee that your simulation matches the behavior of the synthesized circuit, follow these rules:
  1. Use blocking assignments to model combinational logic within an always block.
  2. Use non-blocking assignments to implement sequential logic.
  3. Do not mix blocking and non-blocking assignments in the same always block.
     - It is not possible to model both a combinational output and a sequential output in a single always block.
  4. Do not make assignments to the same variable from more than one always block:
     - Would yield multi-driver synthesis error; example ahead
       - not possible electrically, by the way!

THANK YOU