

The Always Block

Joe Crop

ECE 474 – VLSI System Design

Oregon State University

What always@ blocks used for

1. Modeling Sequential logic
 - Flip Flops, state machines, etc.
2. Combinational logic with more complex behavior
 - Higher level of abstraction (if/else, case, etc.)

always@ blocks (Sequential Logic)

- Always block structure

```
always @(sensitivity list) <begin> <procedural statements> <end>
```

- Example

```
module flip_flop(  
    output q,  
    input d,  
    input clk  
);  
  
    reg q;  
  
    always @(posedge clk)  
    begin  
        q <= d;  
    end  
  
endmodule
```

always@ blocks (Combinational Logic)

- Always block structure

`always @(sensitivity list) <begin> <procedural statements> <end>`

- Example

```
module and_or(  
    output y,  
    input a, b, c  
)  
    reg x;  
    reg y;  
    → always @(a, b, c)  
        → begin  
            x = a & b;  
            y = c | x;  
        → end  
endmodule
```

- Whenever a variable in the sensitivity list changes, the always block wakes up and executes its enclosed statements.
- If variables are omitted from the sensitivity list, the block will not wake up when you want it to!
- <begin>,<end> will be needed if multiple statements exist.
- Variables on the LHS inside the always block must be of type reg. ↵

Be Careful!

- Reg variables retain only the last assigned values within always

```
reg tmp1, tmp2;
always @(a,b,c,d,e,f)
begin
    y = a & b;
    y = c & d;
    y = e & f;
end
```

- y only retains the value computed by y = e & f; when always block concludes.

Blocking vs. Non-Blocking

- Two types of procedural assignment statements are used within always

- Non-Blocking

$y \leq a;$

- Use this to model **sequential** logic only

- Blocking

$y = a \& b;$

- Use this to model **combinational** logic only

Non-Blocking Example

- All “<= ” assignments happen at the same time
 - Assignments don’t “block” each other

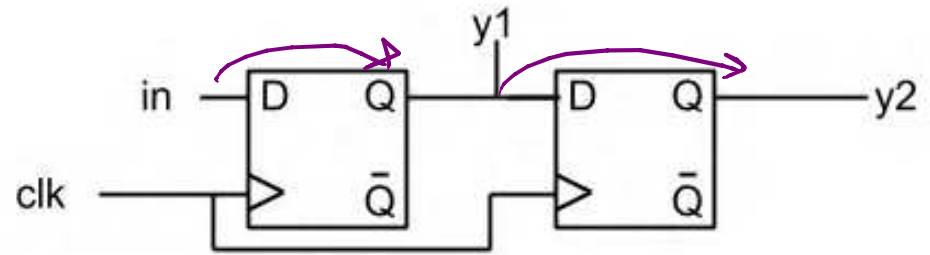
```
always @(posedge clk)
```

```
begin
```

```
    y1 <= in;
```

```
    y2 <= y1;
```

```
end
```



Blocking Example

- All “=” assignments happen one after another
 - Assignments do “block” each other

```
always @(posedge clk)
```

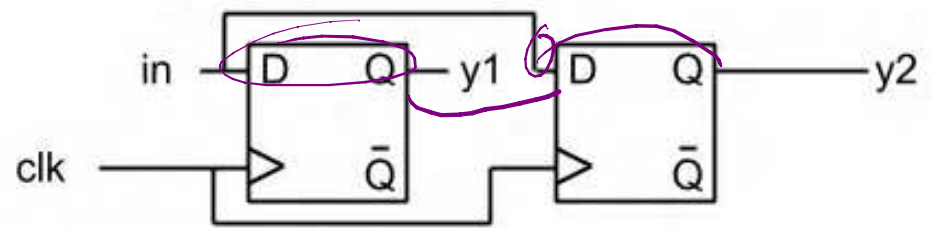
```
begin
```

```
    y1 = in;
```

```
    y2 = y1;
```

```
end
```

always @ (posedge clk)



$$y_1 = i \cdot \frac{1}{j}$$

Blocking vs. Non-Blocking

- **Blocking:**




- The blocking assignment operator is the equals sign "="
- So named because blocking assignments must evaluate RHS and complete the assignment without interruption from any other statement.
- The blocking assignment also blocks other following assignments until the current one is done.
- Blocking statements execute in the order they are specified.

- **Non-blocking:**

- The non-blocking assignment operator is the less-than-or-equals-to operator "<="
- The non-blocking assignment evaluates the RHS at the beginning of a time step and schedules the LHS update for the end of the time step.
- Between the evaluation of the RHS and update of LHS, other statements may execute.
- The non-blocking statement does not block any other statements from being evaluated.

Coding Guidelines

Thanks to Cliff Cummings, Sunburst Design

- When modeling sequential logic, use non-blocking assignments. 
- When modeling latches/FlipFlops, use non-blocking assignments. 
- When modeling combo logic with an always block, use blocking assignments. 
- When modeling both sequential and combo logic within the same always block, use non-blocking assignments.
- Do not mix blocking and non-blocking assignments in the same always block.
- Do not make assignments to the same variable from more than one always block.

For more information, see:

Nonblocking Assignments in Verilog Synthesis, Coding Styles That Kill!

Available at: <http://www.sunburst-design.com>