EE-421: Digital System Design

Blocking vs Non-Blocking Revisited (in Sequential Ccts)

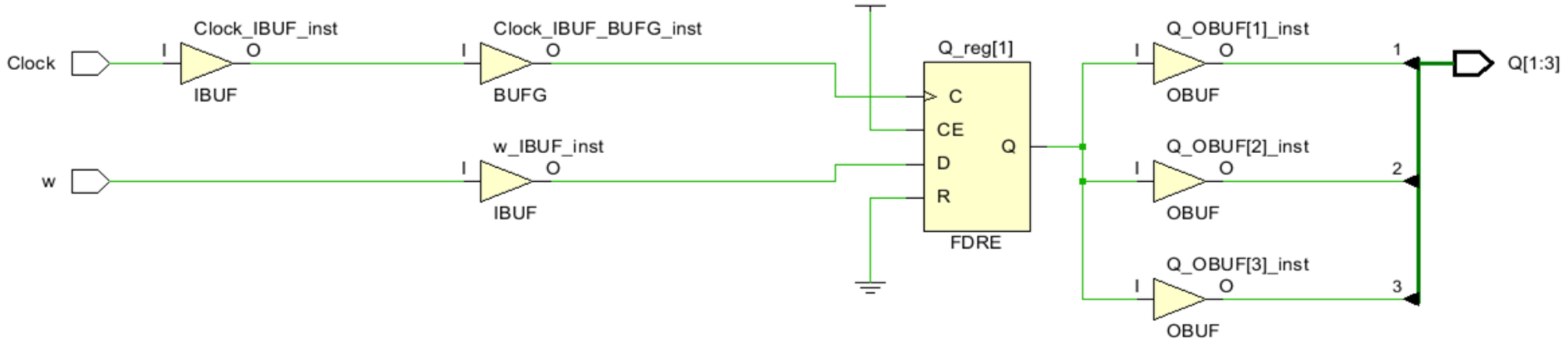
Dr. Rehan Ahmed [rehan.ahmed@seecs.edu.pk]

NUST
SCHOOL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCE (SEECS)

Blocking vs Non-Blocking Revisited (in Sequential Ccts)

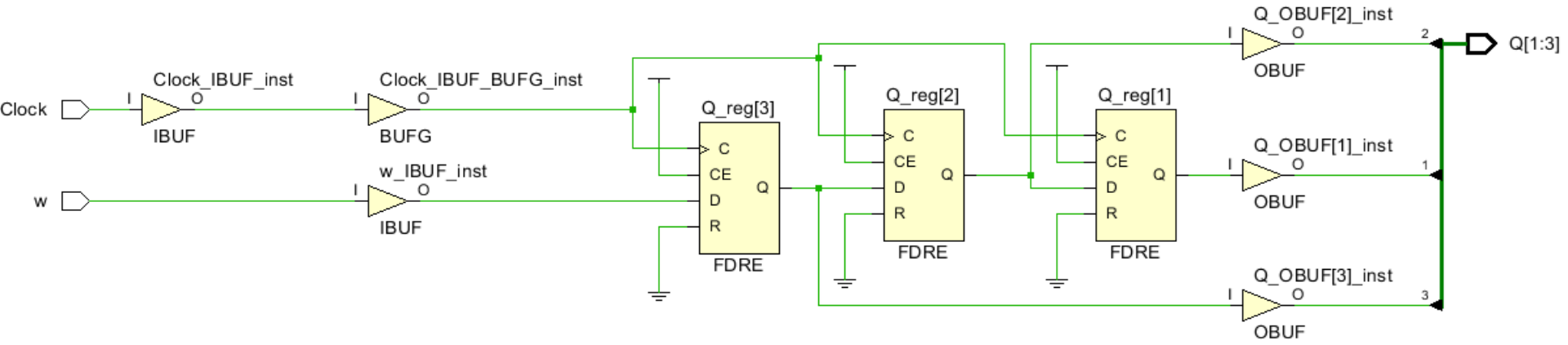
Review (1): Blocking vs Non-Blocking Statements and Statement Ordering

```
module shift3 (w, Clock, Q);
  input w, Clock;
  output reg [1:3] Q;
  always @(posedge Clock)
  begin
    Q[3] = w;
    Q[2] = Q[3];
    Q[1] = Q[2];
  end
endmodule
```



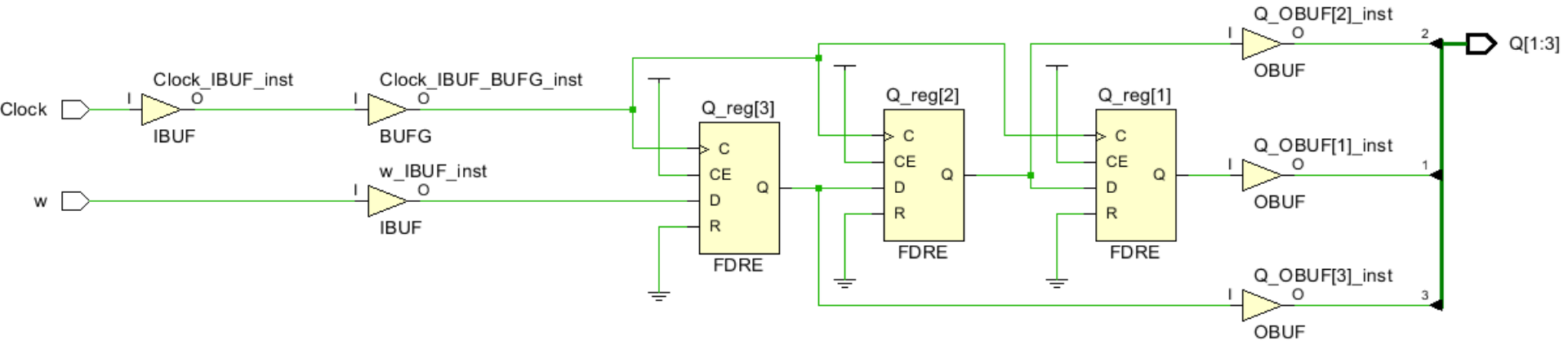
Review (2): Blocking vs Non-Blocking Statements and Statement Ordering

```
module shift3 (w, Clock, Q);
  input w, Clock;
  output reg [1:3] Q;
  always @(posedge Clock)
  begin
     Q[1] = Q[2];
     Q[2] = Q[3];
     Q[3] = w;
  end
endmodule
```



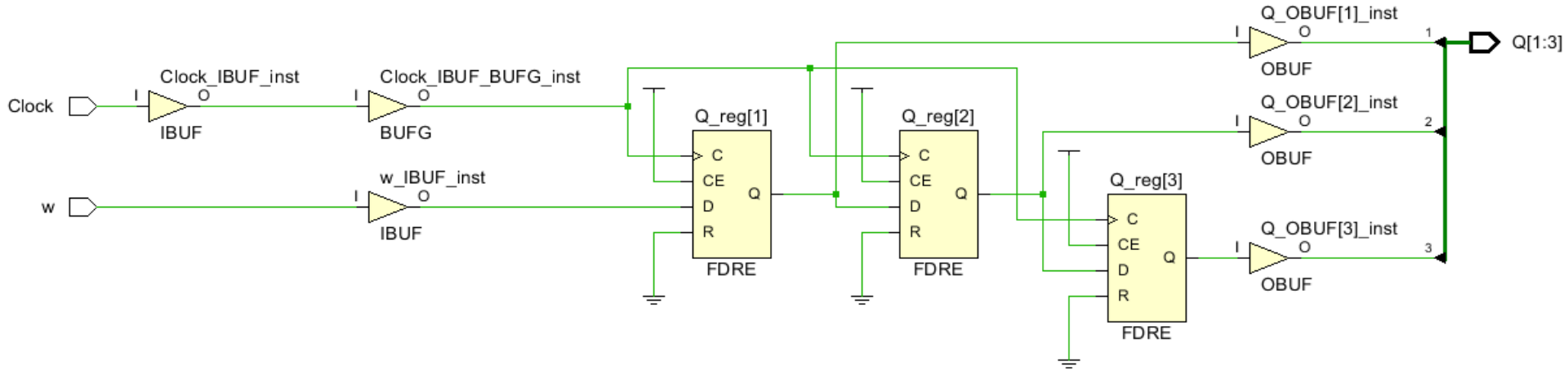
Review (3): Blocking vs Non-Blocking Statements and Statement Ordering

```
module shift3 (w, Clock, Q);
  input w, Clock;
  output reg [1:3] Q;
  always @(posedge Clock)
  begin
     Q[1] <= Q[2];
     Q[2] <= Q[3];
     Q[3] <= w;
  end
endmodule
```



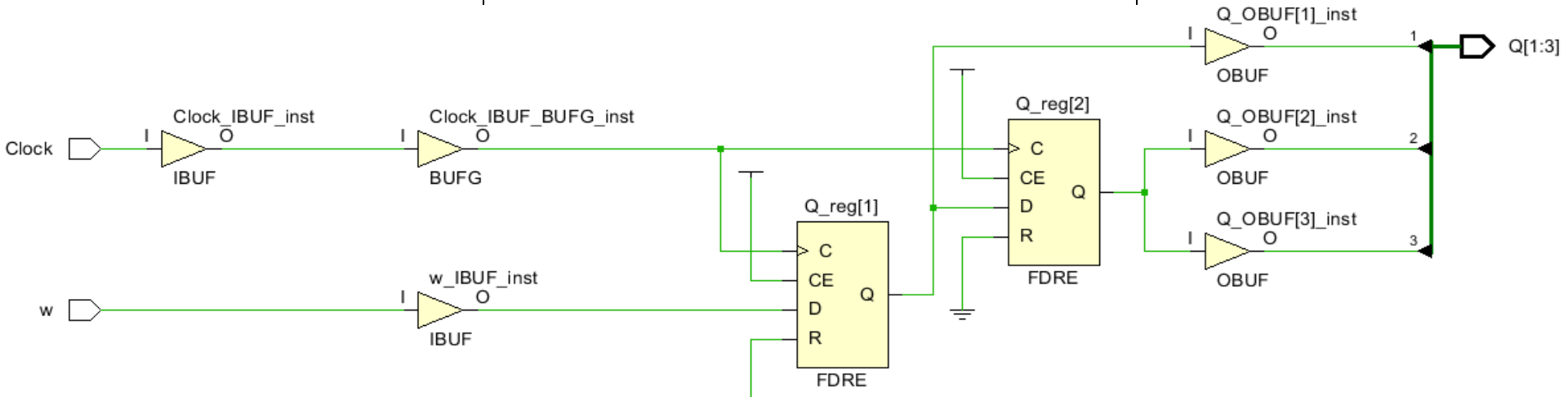
Review (4): Blocking vs Non-Blocking Statements and Statement Ordering

```
module shift3 (w, Clock, Q);
  input w, Clock;
  output reg [1:3] Q;
  always @(posedge Clock)
  begin
     Q[2] <= Q[1];
     Q[3] <= Q[2];
     Q[1] <= w;
  end
endmodule
```



Review (5): Blocking vs Non-Blocking Statements and Statement Ordering

```
module shift3 (w, Clock, Q);
  input w, Clock;
  output reg [1:3] Q;
  always @(posedge Clock)
  begin
     Q[2] = Q[1];
     Q[3] = Q[2];
     Q[1] = w;
  end
endmodule
```



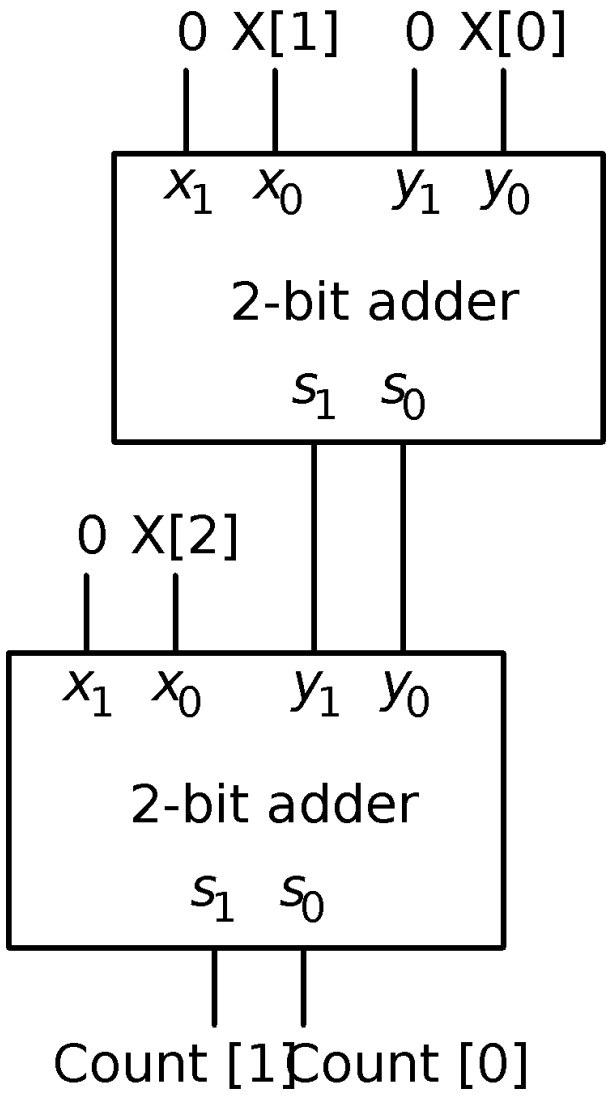
Takeaway: Blocking vs Non-Blocking: Avoid Confusion

- 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

Class Activity

- Design a circuit that counts the number of bits in an n-bit input X that have the value 1.



Blocking vs Non-Blocking

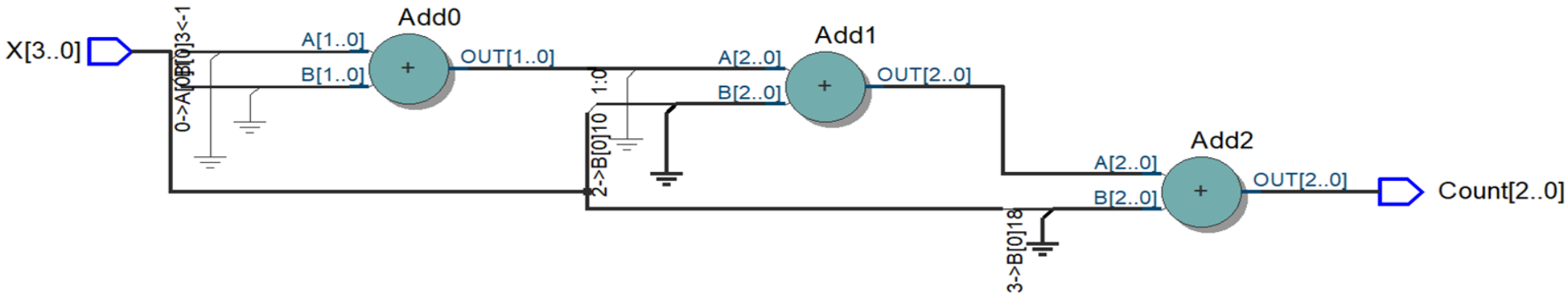
```
module bit_count (X, Count);
  parameter n = 4;
  parameter logn = 2;
  input [n-1:0] X;
  output reg [logn:0] Count;
  integer k;

  always @(X)
  begin
    Count = 0;
    for (k = 0; k < n; k = k+1)
      Count = Count + X[k];
  end
```

counts the number of bits in the n-bit input X that have the value 1

Loop unrolled as:

```
Count = Count + X[0];
Count = Count + X[1];
Count = Count + X[2];
……
```



Blocking vs Non-Blocking

```
module bit_count (X, Count);
  parameter n = 4;
  parameter logn = 2;
  input [n-1:0] X;
  output reg [logn:0] Count;
  integer k;

  always @(X)
  begin
    Count <= 0;
    for (k = 0; k < n; k = k+1)
      Count <= Count + X[k];
  end
endmodule
```

Loop unrolled as:

Count <= Count + X[0];

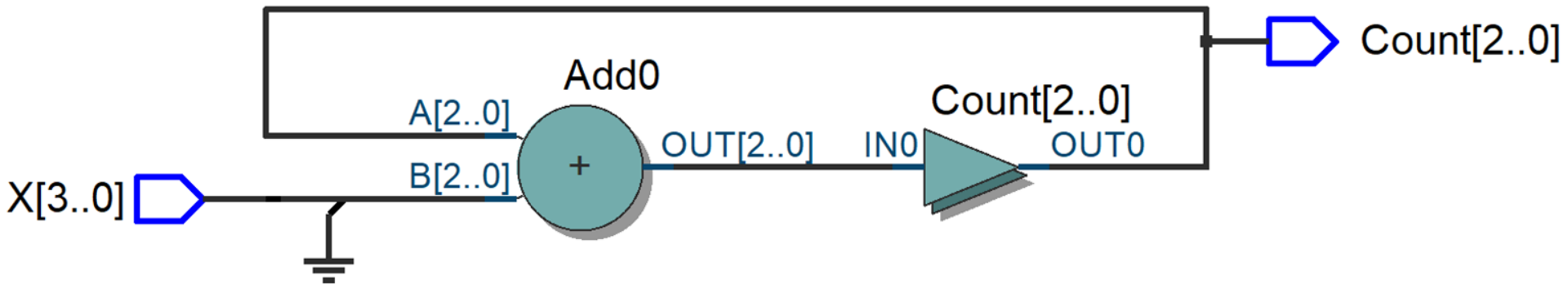
Count <= Count + X[1];

Count <= Count + X[2];

…..

Since non-blocking assignments are involved, each subsequent assignment statement sees the starting value of Count = 0

Count <= 0 + X[2];



Example: Mixing Blocking and Non-Blocking

```
module bit_count (X, Count);
  parameter n = 4;
  parameter logn = 2;
  input [n-1:0] X;
  output reg [logn:0] Count;
  integer k;

  always @(X)
  begin
    Count = 0;
    for (k = 0; k < n; k = k+1)
      Count <= Count + X[k];
  end
endmodule
```

```
12127 Elaborating entity "top" for the top level hierarchy
10110 Verilog HDL error at top.v(5): variable "Count" has mixed blocking and nonblocking Procedural Assignments -- must be all blocking or all nonblocking assignments
12153 Can't elaborate top-level user hierarchy
      Quartus Prime Analysis & Elaboration was unsuccessful. 2 errors, 3 warnings
```

Takeaway: Expressing Combinational and Sequential Ccts in Verilog

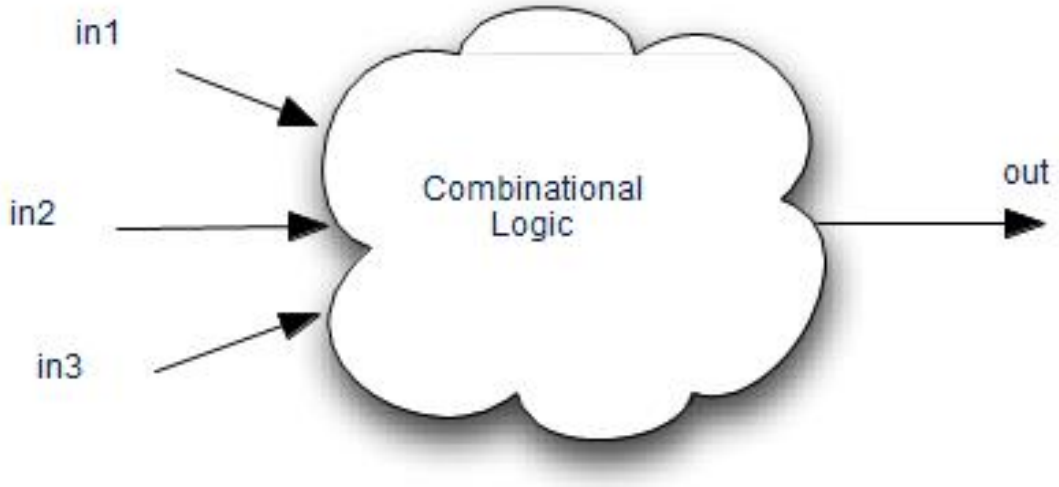
Takeaway

- Expressing Combinational or Sequential circuits using Procedural Statements:
  - Remember THREE patterns (discussed in the next 3-slides)
    - Purely based on observation and experience!
    - Not part of IEEE standard (not at all)!
    - Books might not preach that!

Always Pattern - 1

- **Combinational Logic:**

always@(in1, in2, in3) = always @ (*)

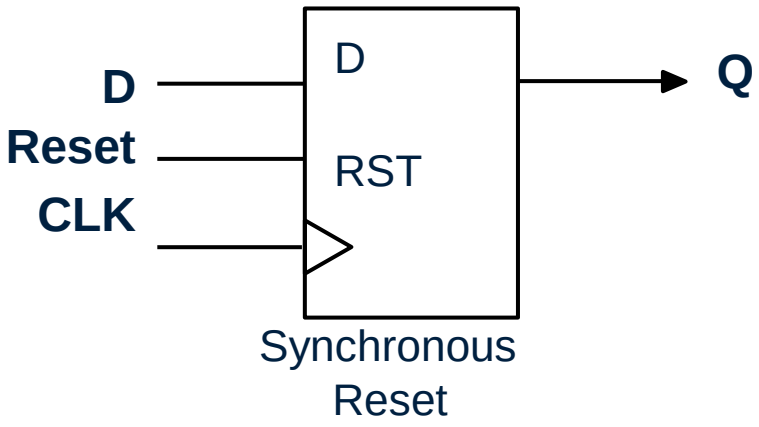


- Simulation – outputs are re-evaluated anytime a signal in the sensitivity list changes
- Synthesis – The synthesized block's output may only change when a change occurs on one of the sensitivity list signals

Always Pattern - 2

- **Sequential Logic and Synchronous Reset:**

```
always@(posedge CLK)
```

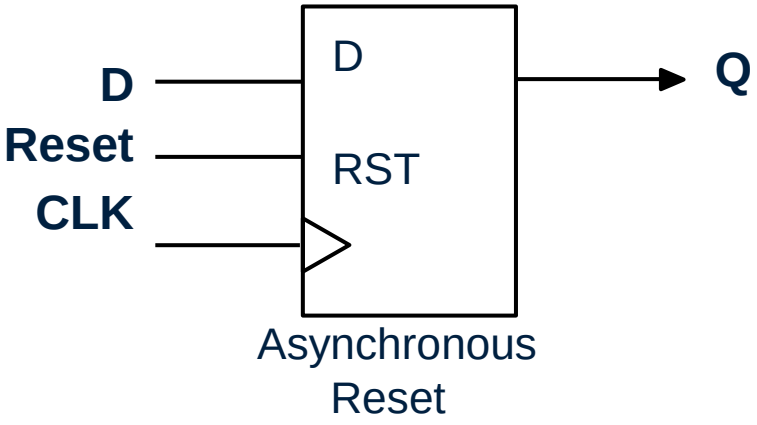


- Only the clock signal goes inside the sensitivity list.

Always Pattern - 3

- **Sequential Logic and Asynchronous Reset:**

```
always@(posedge CLK or negedge RESET)
```



- Only the clock and reset signals go inside the sensitivity list.

THANK YOU