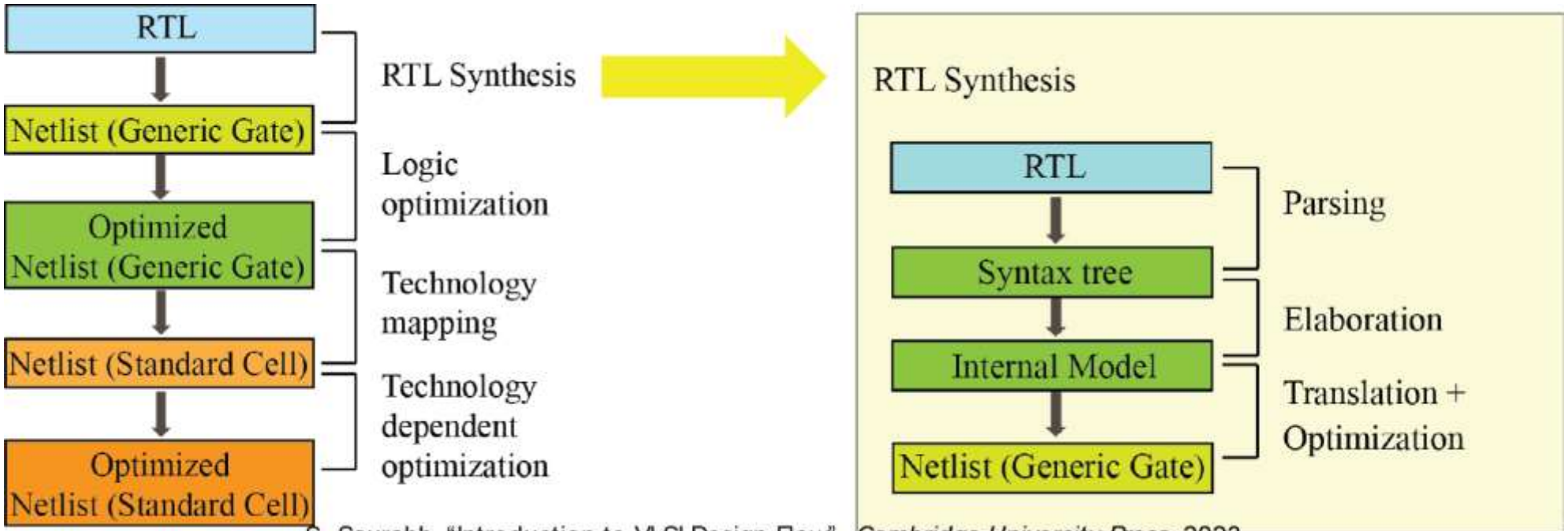
VLSI DESIGN FLOW: RTL TO GDS

Lecture 12
RTL Synthesis- Part I

Sneh Saurabh
Electronics and Communications Engineering
IIIT Delhi

Lecture Plan



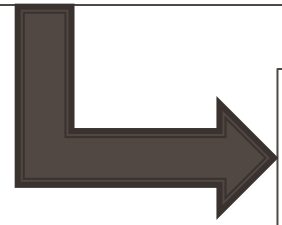
S. Saurabh, "Introduction to VLSI Design Flow". *Cambridge University Press*, 2023.

RTL synthesis:

- Initial part of logic synthesis that translates the Verilog code into netlist of generic logic gates

- Parsing, Elaboration
- Translation of some Verilog Constructs to Circuit

RTL Synthesis



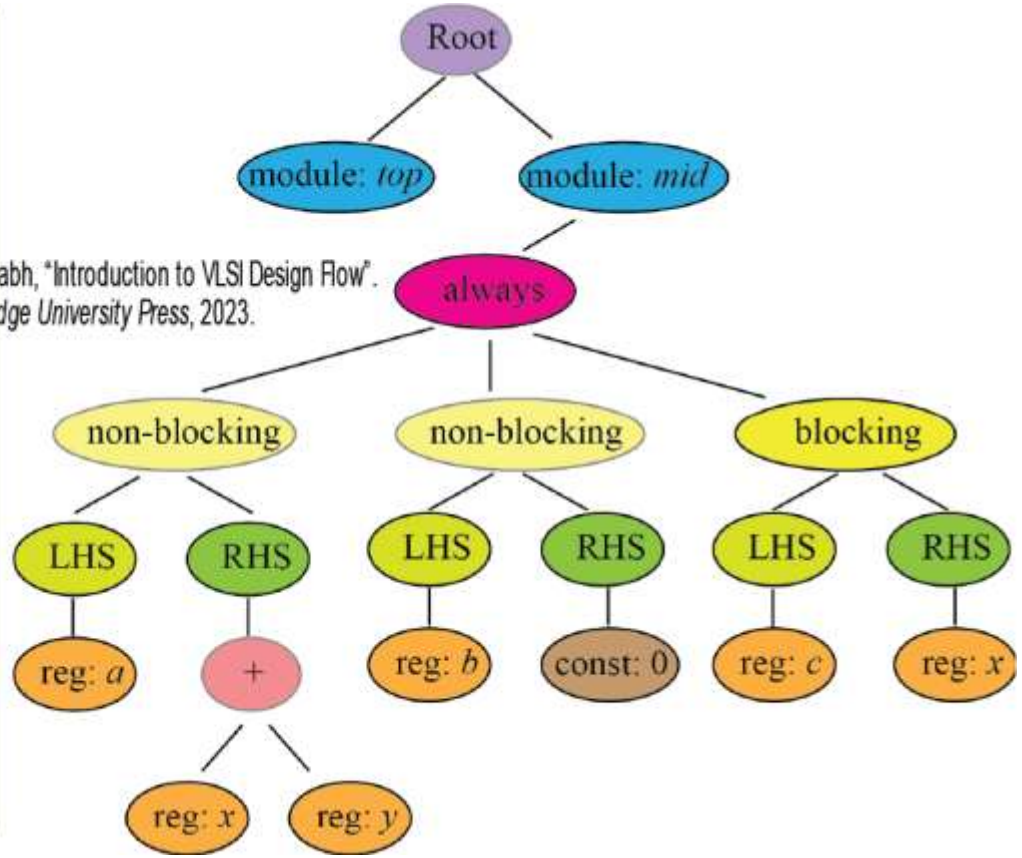
Tasks

RTL Synthesis: Parsing

- Reads the given RTL files and populates a data structure for further processing

- Lexical analysis: keywords, identifiers
- Grammar/syntax checking
- Syntax tree built:
  - If grammar is correct
  - Hierarchical data structure

```
module top ();

endmodule

module mid ();
...
        always @ (*)
        begin
                a <= x+y;
                b <= 0;
                c = x;
        end

endmodule
```



S. Saurabh, "Introduction to VLSI Design Flow". Cambridge University Press, 2023.

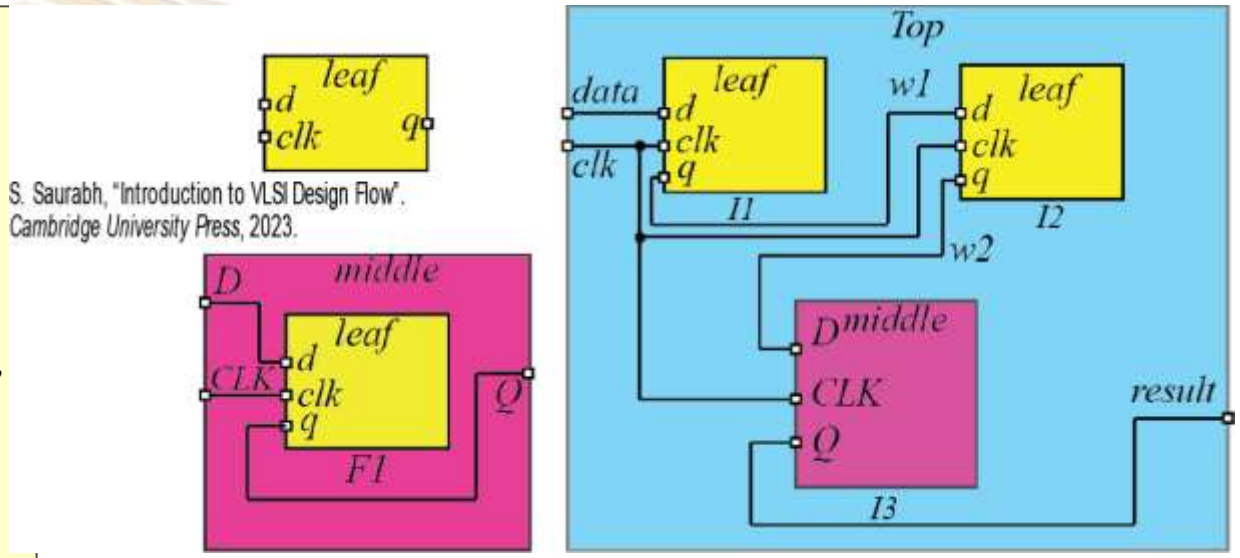
RTL Synthesis: Elaboration (1)

- Checks whether the connections among RTL-specified components are legitimate.
  - ➢ If legitimate, then make connections in the internal model; else report error

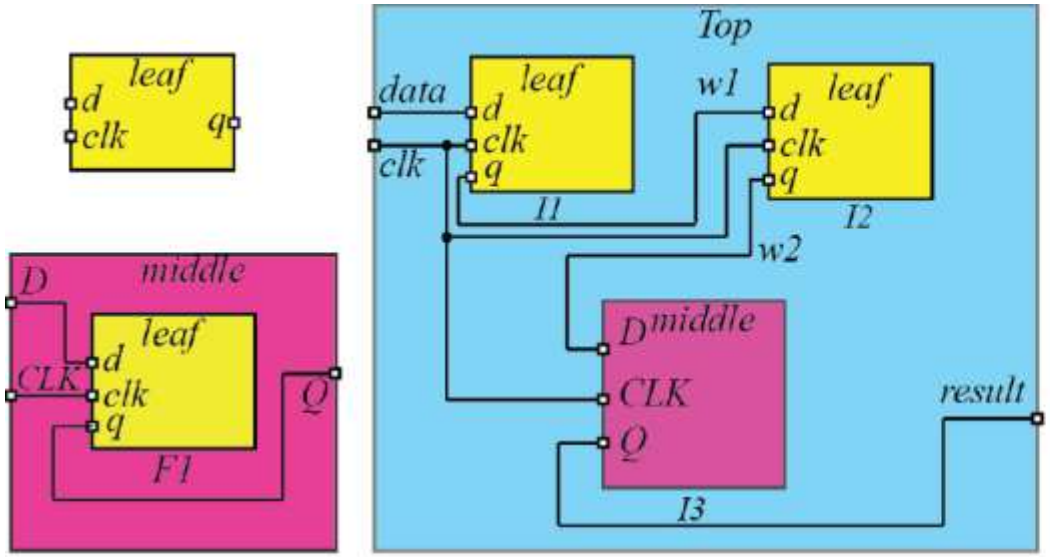
```
module leaf(d, clk, q);
input d, clk;
output q;
...
endmodule
```

```
module middle(D, CLK, Q);
input D, CLK;
output Q;

leaf F1(.d(D), .clk(CLK), .q(Q));

endmodule
```

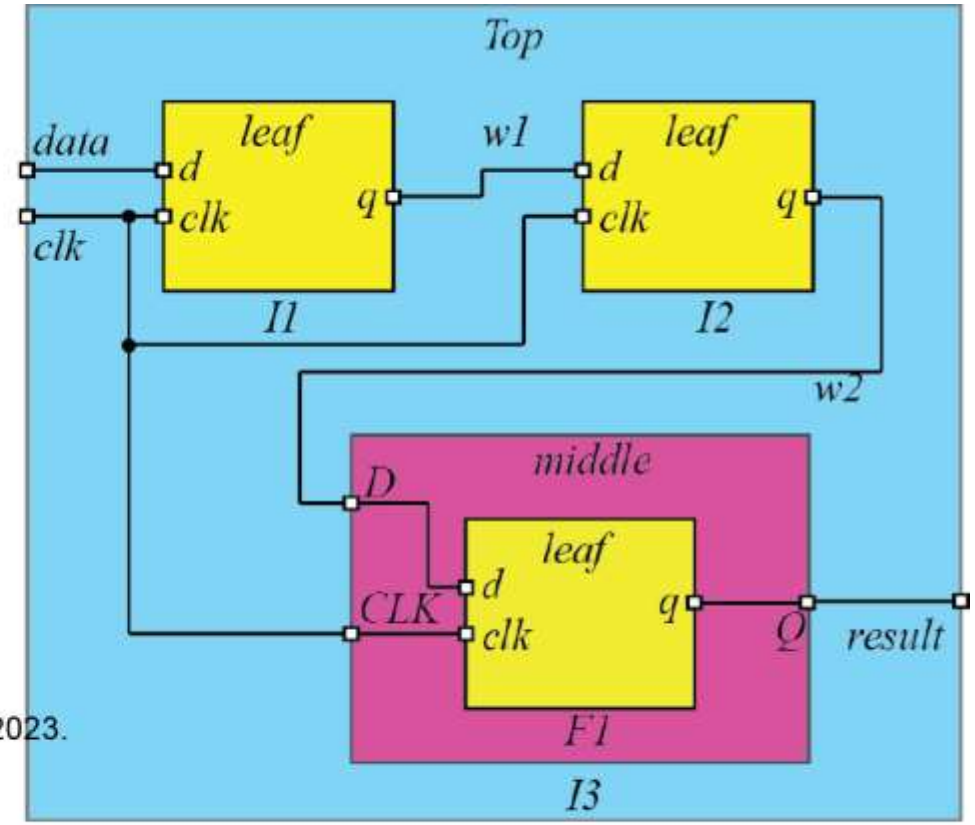
```
module Top(data, clk, result);
input data, clk;
output result;
wire w1, w2;

leaf I1(.d(data), .clk(clk), .q(w1));

leaf I2(.d(w1), .clk(clk), .q(w2));

middle I3(.D(w2), .Q(result), .CLK(clk));

endmodule
```



RTL Synthesis: Elaboration (2)



Before Elaboration



After Elaboration

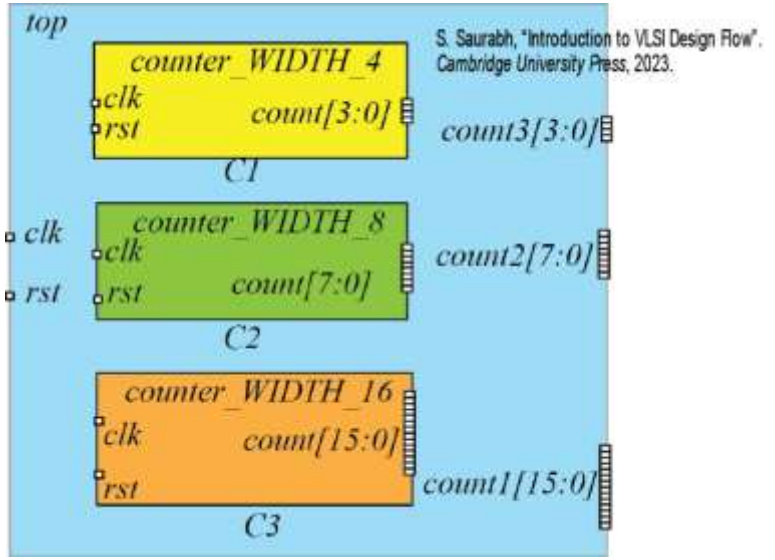
S. Saurabh, "Introduction to VLSI Design Flow". *Cambridge University Press*, 2023.

RTL Synthesis: Elaboration (3)

- Elaboration needs to process parameterized modules
  - ➢ Parameterized modules can have different interfaces for varying parameters.
- Creates separate modules with different interfaces for each distinct set of parameters

```
module counter(clk, rst, count);
        parameter WIDTH=4;
        input clk, rst;
        output [WIDTH-1:0]count;
        reg [WIDTH-1:0]count;
...
endmodule
```

```
module top(clk, rst, count1, count2, count3);
        input clk, rst;
        output [15:0]count1;
        output [7:0]count2;
        output [3:0]count3;

        counter C1(clk, rst, count3);
        counter #(8)  C2(clk, rst, count2);
        counter #(.WIDTH(16)) C3(clk, rst, count1);

endmodule
```



S. Saurabh, "Introduction to VLSI Design Flow". *Cambridge University Press*, 2023.

RTL Synthesis

Verilog Constructs to Circuit

Synthesizable and Non-synthesizable Constructs

- Some Verilog constructs cannot be synthesized into circuit elements
  - ➢ They are helpful in other design tasks such as functional verification

- A given RTL synthesis tool may not support all synthesizable Verilog constructs or modeling styles.
  - ➢ Need to be aware of the Verilog constructs supported by the synthesis tool used in design implementation

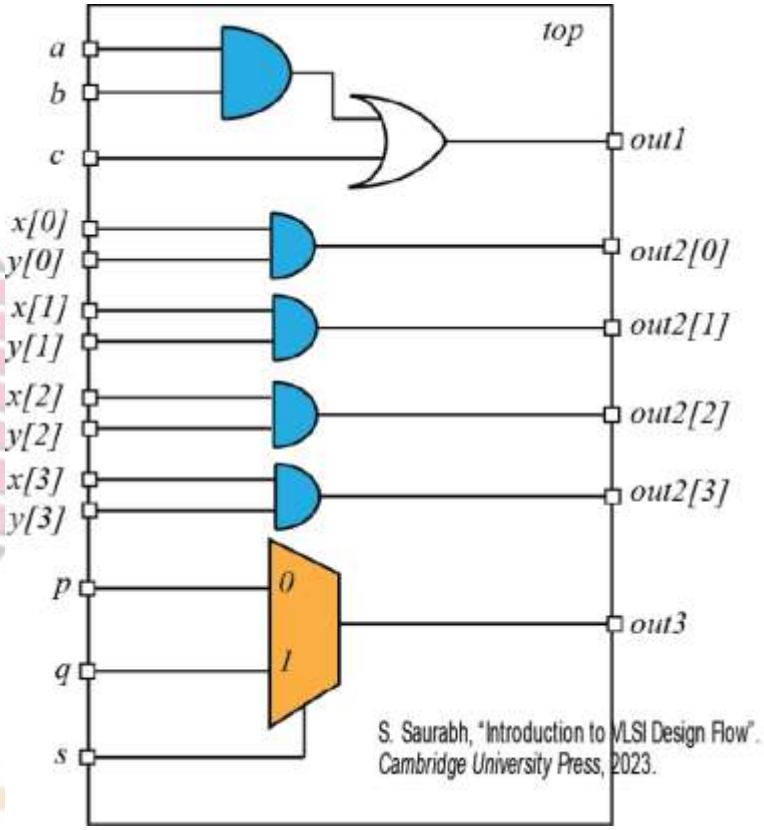
Typically non-synthesizable:

- Delay specification:
  - *out1 <= #12 a;* treated as *out1 <= a;*
- initial block
- fork, join, force, release
- data types real and time
- $display, $monitor, and other system tasks

Assign Statement

- Combinational circuit elements inferred

```
module top(a, b, c, p, q, s, x, y, out1, out2, out3);
      input a, b, c, p, q, s;
      input [3:0]x, y;
      output out1;
      output [3:0]out2;
      output out3;

      assign out1 = (a & b) | c; // logic network
      assign out2 = (x & y); // bitwise logic network
      assign out3 = (s) ? q : p; // multiplexer

endmodule
```



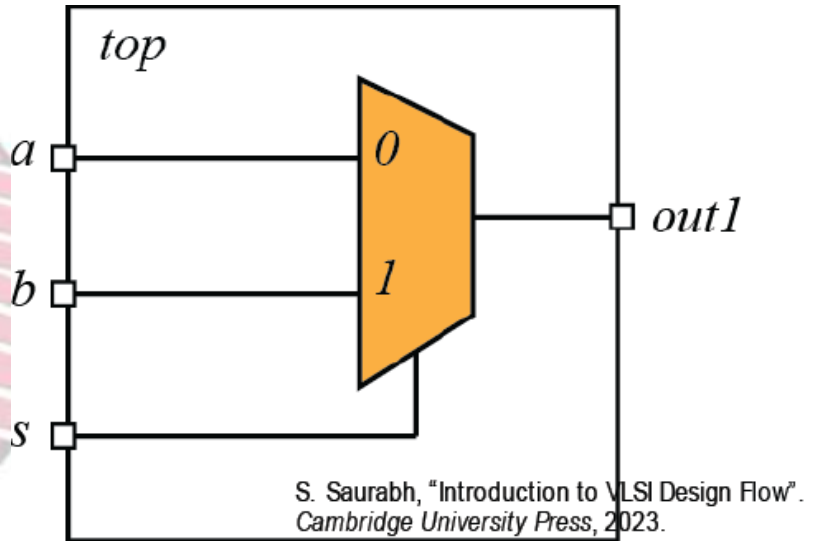
S. Saurabh, "Introduction to VLSI Design Flow". *Cambridge University Press*, 2023.

If-else Statement

- Multiplexer or selecting logic inferred

```
module top(a, b, s, out1);
        input a, b, s;
        output out1;
        reg out1;

        always @(*) begin
                if (s==1'b0)
                   out1 = a;
                else
                   out1 = b;
        end

endmodule
```

S. Saurabh, "Introduction to VLSI Design Flow". *Cambridge University Press*, 2023.

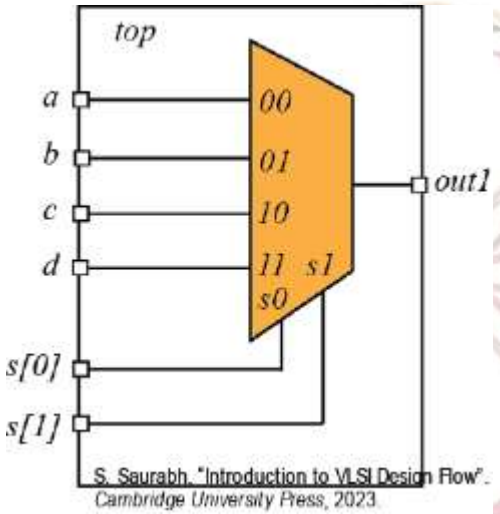


S. Saurabh, "Introduction to VLSI Design Flow". *Cambridge University Press*, 2023.

Case Statement

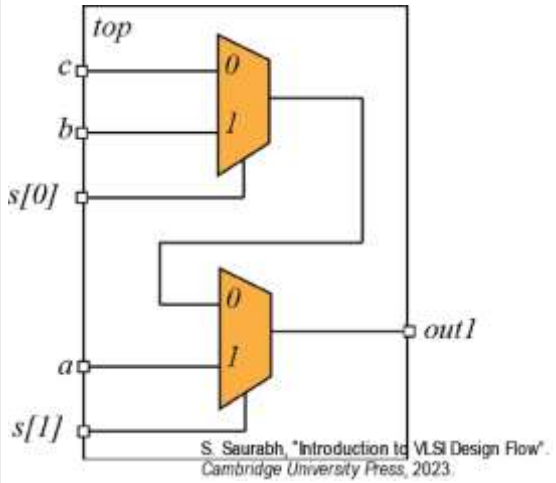
- Multiplexer or selecting logic inferred

```
module top(a, b, c, d, s, out1);
        input a, b, c, d;
        input [1:0]s;
        output out1;
        reg out1;

        always @(*) begin
        case (s)
                2'b00: out1 = a;
                2'b01: out1 = b;
                2'b10: out1 = c;
                2'b11: out1 = d;
                default: out1=a;
        endcase

        end

endmodule
```



S. Saurabh, "Introduction to VLSI Design Flow".*Cambridge University Press*, 2023.

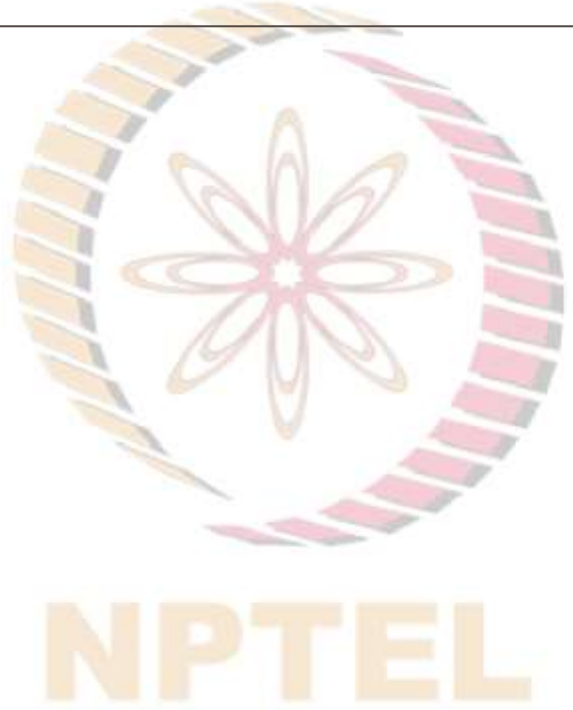
```
module top(a, b, c, s, out1);
        input a, b, c;
        input [1:0]s;
        output out1;
        reg out1;

        always @(*) begin
        case (s)
                2'b1?: out1 = a;
                2'b?1: out1 = b;
                default: out1=c;
        endcase

        end

endmodule
```



If-else-if with priority will be synthesized similarly

Synthesis of Always Block

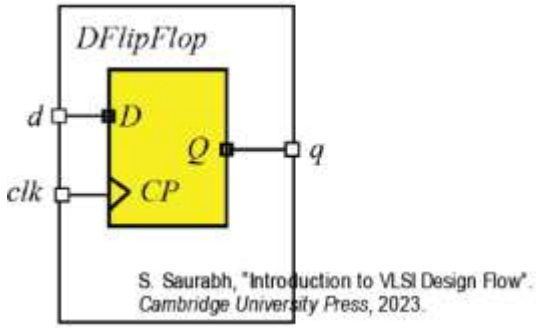
- Always block can be synthesized to combinational logic, latches, or flip-flop
- Depends on the sensitivity list and how are variables assigned with the always block



Always Block: Edge-sensitive

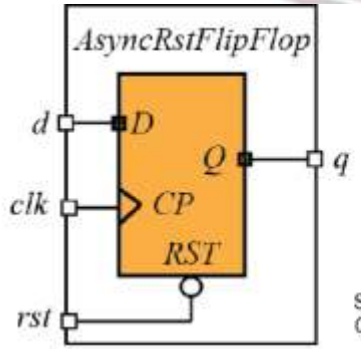
Flip-flops are inferred for edge-sensitive always blocks

```
module DFlipFlop(d, clk, q);
        input d, clk;
        output q;
        reg q;

        always @ ( posedge clk)
                q <= d;

endmodule
```



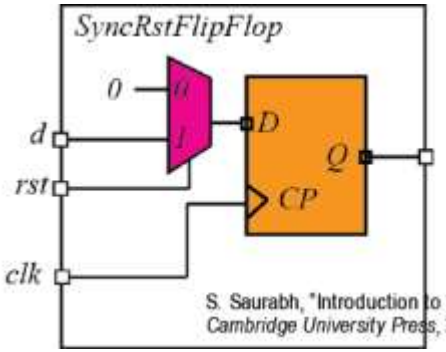
S. Saurabh, "Introduction to VLSI Design Flow". Cambridge University Press, 2023.

```
module AsyncRstFlipFlop(d, clk, q, rst);
        input d, clk, rst;
        output q;
        reg q;

        always @ ( posedge clk or
        negedge rst)
                if (rst == 1'b0) begin
                        q <= 1'b0;
                end else begin
                        q <= d;
                end

endmodule
```



S. Saurabh, "Introduction to VLSI Design Flow". Cambridge University Press, 2023.

```
module SyncRstFlipFlop(d, clk,
q, rst);
        input d, clk, rst;
        output q;
        reg q;

        always @ ( posedge clk)
                if (rst == 1'b0) begin
                        q <= 1'b0;
                end else begin
                        q <= d;
                end

endmodule
```



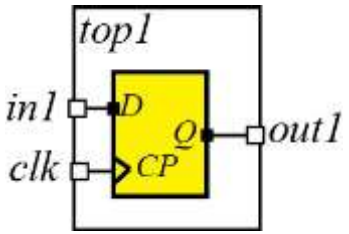
S. Saurabh, "Introduction to VLSI Design Flow". Cambridge University Press, 2023.

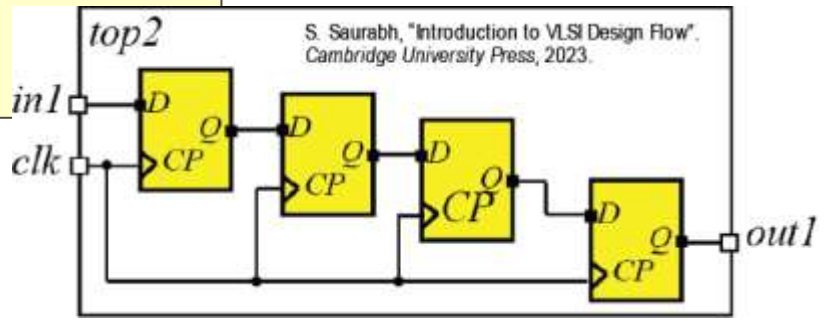
Always Block: Blocking and Non-blocking Assignments

Handling of blocking and non-blocking assignments in always block done differently (follows the semantics of simulation).

```
module top1(in1, clk, out1);
      input in1, clk;
      output out1;
      reg reg1, reg2, reg3, out1;

      always @ ( posedge clk)
      begin
            reg1 = in1;
            reg2 = reg1;
            reg3 = reg2;
            out1 = reg3;
      end
endmodule
```

```
module top2(in1, clk, out1);
      input in1, clk;
      output out1;
      reg reg1, reg2, reg3, out1;

      always @ ( posedge clk)
      begin
            reg1 <= in1;
            reg2 <= reg1;
            reg3 <= reg2;
            out1 <= reg3;
      end
endmodule
```





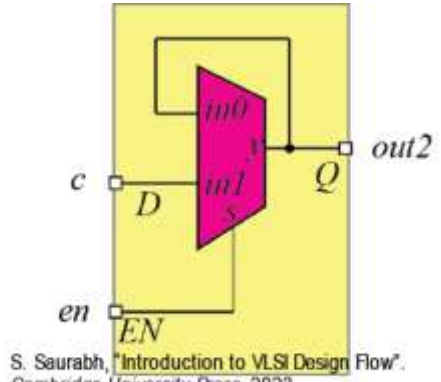
Always Block: Level-sensitive

When always block does NOT contain edges in the sensitivity list, combinational circuit elements or latches are inferred.

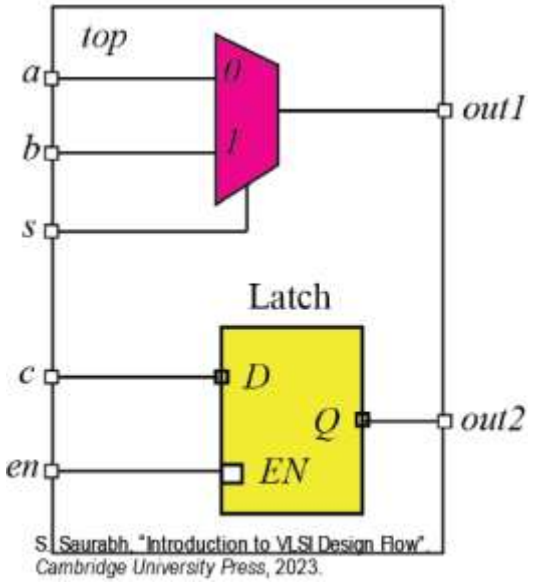
- **Combinational elements:** when the value of a variable is updated (refreshed) in every possible path (conditional branches in the code) within an always block
- **Latch:** if the variable retains its old value in some paths of the always block

```
module top(a, b, c, s, en, out1, out2);
    input a, b, c, s, en;
    output out1, out2;
    reg out1, out2;

    always @(*) begin
        if (s==1'b0)
            out1 = a;
        else
            out1 = b;
    end

    always @(*) begin
        if (en==1'b1)
            out2 = c;
    end
endmodule
```

```
    always @(*) begin
        if (en==1'b1) out2 = c;
        else out2 = out2;
    end
```



S. Saurabh, "Introduction to VLSI Design Flow", Cambridge University Press, 2023.



S. Saurabh, "Introduction to VLSI Design Flow", Cambridge University Press, 2023.

Always Block: Unintentional Latch Inference

- Latches often get inferred due to incorrect modeling of a combinational block
- Example: inadvertently miss updating value in one of the branches of a case statement.

To avoid such problems:

- Use default case to cover all possible paths in a case statement.
- Set the output of a combinational logic to a default value at the beginning of the always block

```
module top(in1, out1);
        input [0:1]in1;
        output out1;
        reg out1;

        always @* begin
                case(in1[0:1])
                        2'b00: out1 = 1'b0;
                        2'b01: out1 = 1'b1;
                        2'b10: out1 = 1'b1;
                endcase
        end
endmodule
```

S. Saurabh, "Introduction to VLSI Design Flow".*Cambridge University Press*, 2023.

References

- S. Saurabh, “Introduction to VLSI Design Flow”. Cambridge: *Cambridge University Press*, 2023.