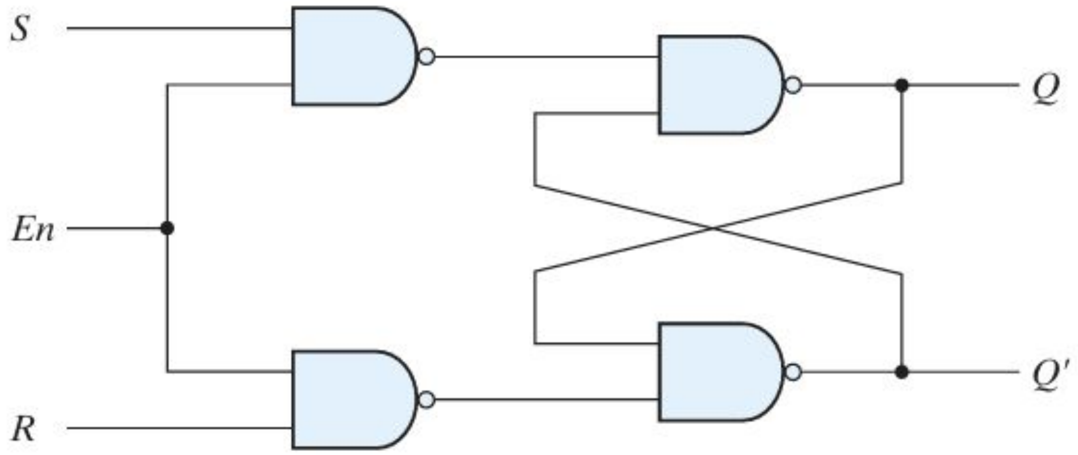
Latches + Behavioural Modelling Tutorial

Manan Nandha
Hrishant Bheda

SR Latch

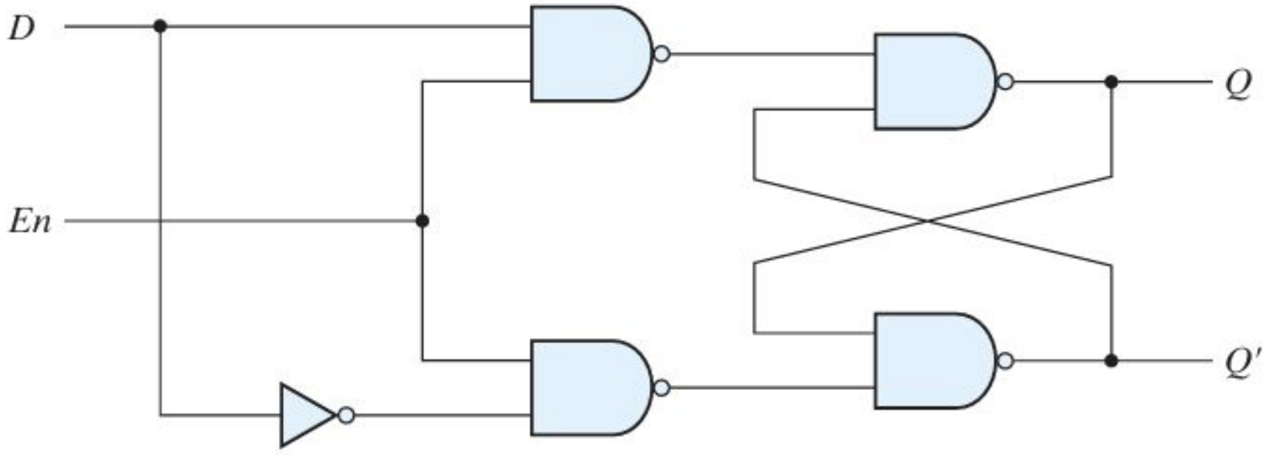


(a) Logic diagram

En	S	R	Next state of Q
0	X	X	No change
1	0	0	No change
1	0	1	Q = 0; reset state
1	1	0	Q = 1; set state
1	1	1	Indeterminate

(b) Function table

D Latch



(a) Logic diagram

En	*D*	Next state of *Q*
0	X	No change
1	0	$Q = 0$; reset state
1	1	$Q = 1$; set state

(b) Function table

Verilog Modeling Styles

- **Gate-level modeling:** Describes circuits with logic gates (`and`, `or`, `not`).
- **Dataflow modeling:** Uses continuous assignments and operators (`assign out = a & b;`).
- **Behavioral modeling:** Describes *what* the circuit should do, not *how* it’s built.
- Behavioral = closest to writing pseudo-code → compiler synthesizes hardware.

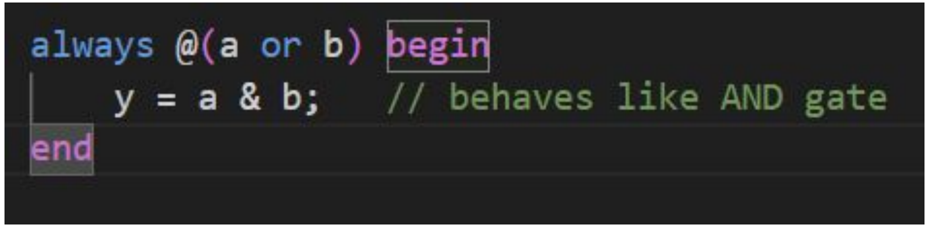
What is behavioural modelling?

This approach focuses on describing the behavior of the system rather than the specific structure.It allows you to specify the logic of the system using higher-level constructs like if, case, and loops. Structural Modeling describes the hardware by specifying how different components (modules, gates, etc.) are connected. It mirrors the actual physical circuit.

Structural - WHAT

Behavioural - HOW

Always Block



```
always @(a or b) begin
    y = a & b;   // behaves like AND gate
end
```

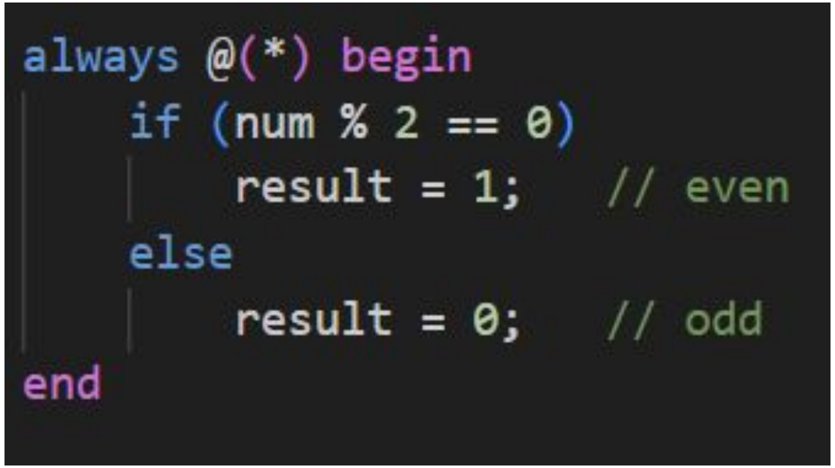
Defined using `always @(sensitivity_list)`

Executed whenever signals in sensitivity list change.

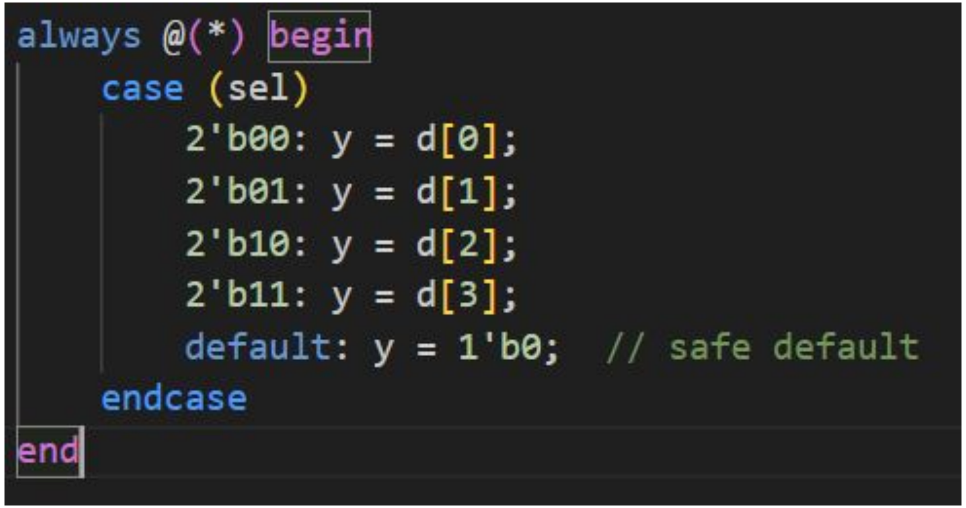
`begin` and `end` are like opening and closing parenthesis in a code.

In the given example y is updated to the value of a&b whenever input a or input b changes.

Procedural Statements



```
always @(*) begin
    if (num % 2 == 0)
        result = 1;    // even
    else
        result = 0;    // odd
end
```

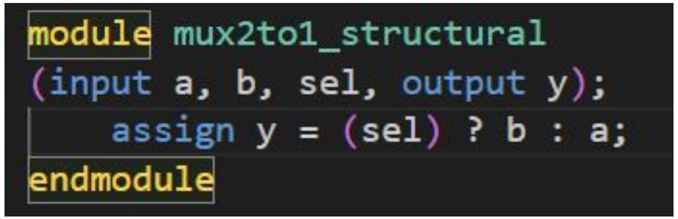


```
always @(*) begin
    case (sel)
        2'b00: y = d[0];
        2'b01: y = d[1];
        2'b10: y = d[2];
        2'b11: y = d[3];
        default: y = 1'b0;  // safe default
    endcase
end
```

`wire` in Structural vs `reg` in Behavioral

Structural Modeling

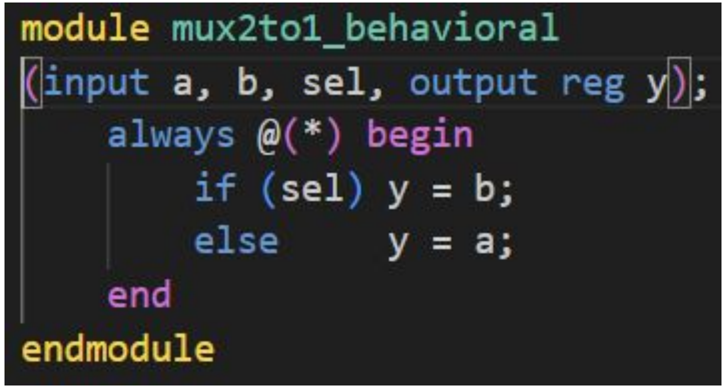
- Outputs are driven by **continuous assignments** (`assign`) or gate primitives.
- Hence, outputs are usually `wire` type.



```
module mux2to1_structural
(input a, b, sel, output y);
    assign y = (sel) ? b : a;
endmodule
```

Behavioral Modeling

- Outputs are assigned **inside an `always` block**.
- `always` requires storage → must be declared `reg`.



```
module mux2to1_behavioral
(input a, b, sel, output reg y);
    always @(*) begin
        if (sel) y = b;
        else     y = a;
    end
endmodule
```

Please refer to the shared verilog code to see the behavioral implementation of SR latch and D latch. For the next lab please prepare mux, decoder, encoder and other modules covered in class.