CSE 260M / ESE 260
Intro. To Digital Logic & Computer Design

Bill Siever
&
Jim Feher

This week

- Thursday:
    Studio — Here / Seigle 301

- Hw#5 posted by late Friday

    - Verilog!

Chapter 4

Studio / Environment: nand2 / add2 (Simulation)

`always`: Based on *Events*

- Concept of "event" is related to simulation and "event driven programming"

- JLS uses events: An OR gate "reacts" to events and schedules an update
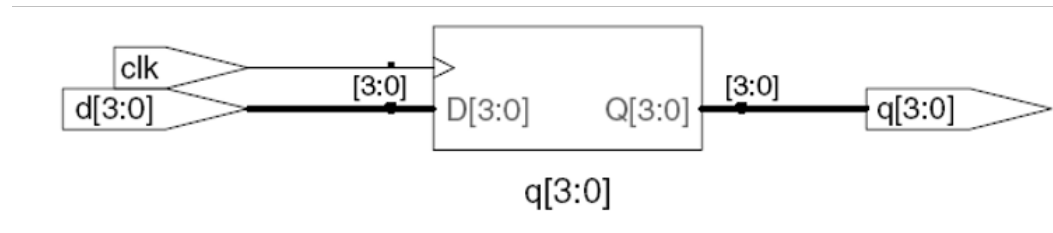 See here

`always` **Statement**

- **Form:**
  ```
  always @(sensitivity list)
      statement;
  ```
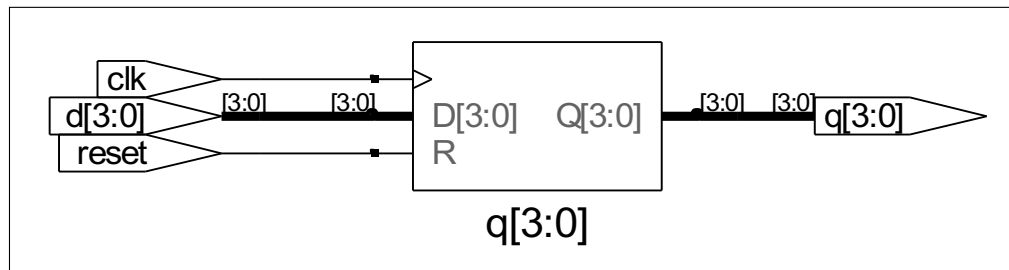
- **When event in sensitivity list occurs, statement is executed**

Verilog: D Flip-Flop


```
module flop(input  logic        clk,
            input  logic [3:0] d,
            output logic [3:0] q);
   always_ff @(posedge clk)
     q <= d;                // pronounced "q gets d"
endmodule
```
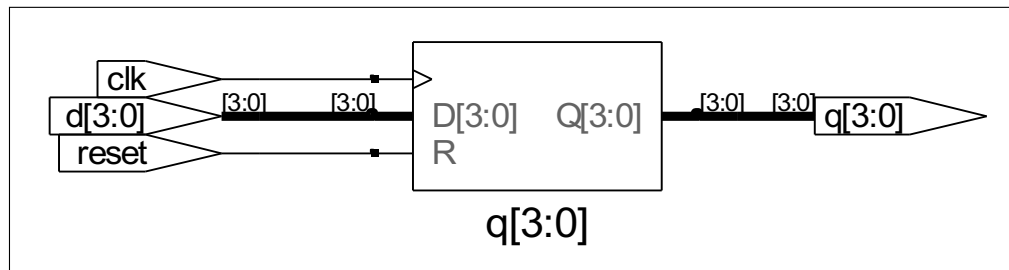
Resettable D-Flip-Flop 1


```
module flopr(input  logic       clk,
             input  logic       reset,
             input  logic [3:0] d,
             output logic [3:0] q);

  always_ff @(posedge clk)
    if (reset) q <= 4'b0;
    else       q <= d;
endmodule
```
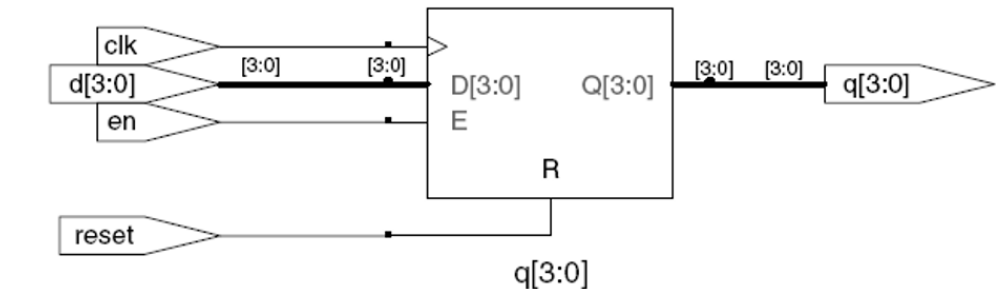
Resettable D-Flip-Flop 2


```
module flopr(input  logic        clk,
             input  logic        reset,
             input  logic [3:0] d,
             output logic [3:0] q);

  always_ff @(posedge clk, posedge reset)
    if (reset) q <= 4'b0;
    else       q <= d;
endmodule
```

Resettable D-Flip-Flop 3


```
module flopr(input  logic        clk,
             input  logic        reset,
             input  logic        en,
             input  logic [3:0]  d,
             output logic [3:0]  q);

  always_ff @(posedge clk, posedge reset)
    if (reset)    q <= 4'b0;
    else if (en)  q <= d;
endmodule
```

Blocking vs. Non-Blocking Assignment
(in `always` blocks)

- <= is nonblocking assignment
  Occurs simultaneously with others

- = is blocking assignment
  Occurs in order it appears in file

Rules for Assignments

- **Synchronous sequential logic**
  use `always_ff @(posedge clk)` **and nonblocking assignments (**`<=`**)**
  ```
  always_ff @(posedge clk)
        q <= d; // nonblocking
  ```

- *Simple* **combinational logic**
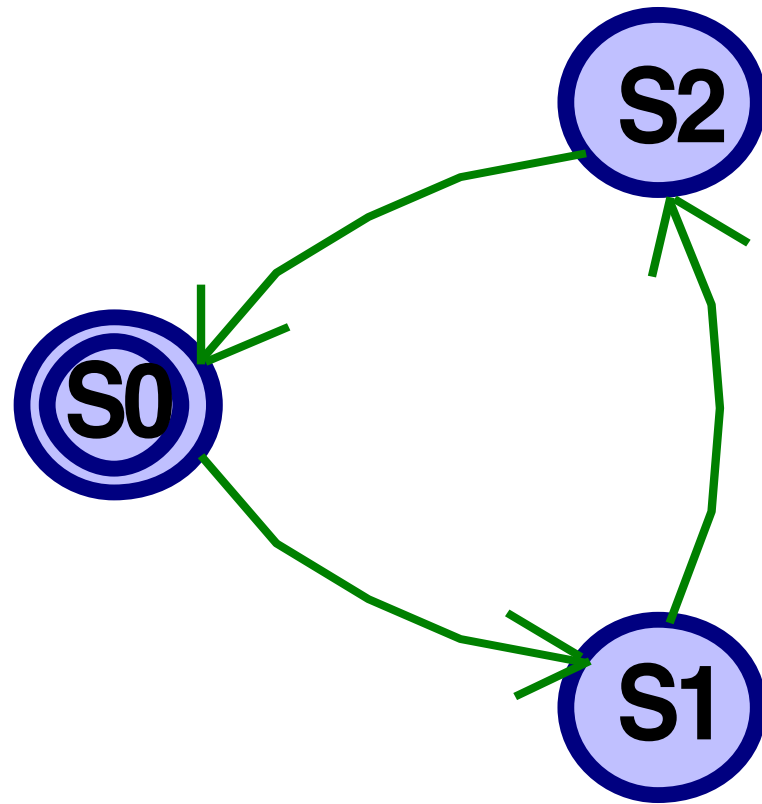 **use continuous assignments (**`assign`**)**
  ```
  assign y = a & b;
  ```

- Complex Combinational `Logic`
  use `always_comb` **and blocking assignments (**`=`**)**

- **Assign signals in only one** `always` **or** `assign` **statement!**

Verilog FSMs

- **Three parts**

  - **Next state logic**
    (arrows / next state table)

  - **State register (active bubble)**

  - **Output logic (output equations)**

Moore FSM

Mealy FSM
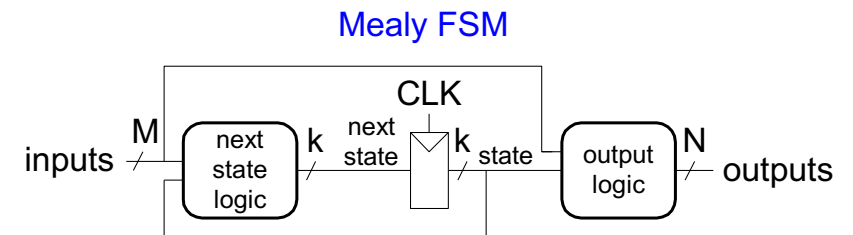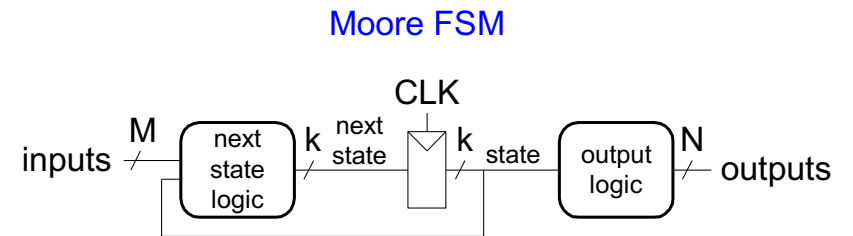
Divide by 3 Counter

Verilog

```verilog
module divideby3FSM(input  logic clk,
                    input  logic reset,
                    output logic q);

    typedef enum logic [1:0] {S0, S1, S2} statetype;
    statetype state, nextstate;

    // state register
    always_ff @(posedge clk, posedge reset)
        if (reset) state <= S0;
        else       state <= nextstate;

    // next state logic
    always_comb
        case (state)
            S0:      nextstate = S1;
            S1:      nextstate = S2;
            S2:      nextstate = S0;
            default: nextstate = S0;
        endcase

    // output logic
    assign q = (state == S0);
endmodule
```
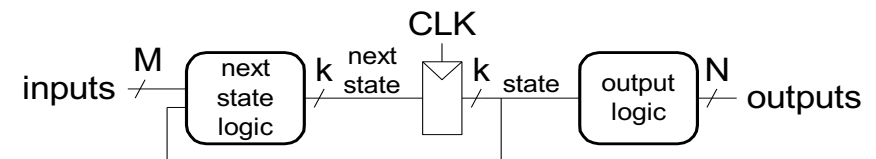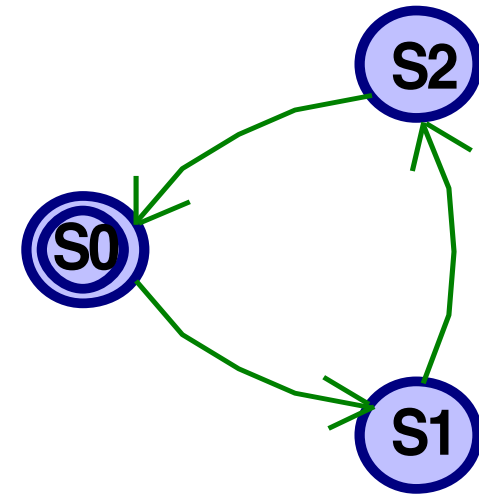
Parameterized Modules: Declaration

- Way to specify additional details for an *instance* of a generic part

  - Commonly the "width" of the part

```
module mux2
  #(parameter width = 8)  // name and default value
   (input  logic [width-1:0] d0, d1,
    input  logic             s,
    output logic [width-1:0] y);
  assign y = s ? d1 : d0;
endmodule
```

Parameterized Modules: Use

- Default or specify parameter for instance:

```
mux2 myMux(d0, d1, s, out);

mux2 #(12) lowmux(d0, d1, s, out);
```

Ports: Positional vs. Named

• Default or specify parameter for instance:

```
logic a, b, see, y
mux2 myMux(a, b, sel, y);
```
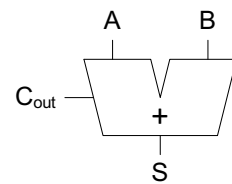vs.
```
mux2 myMux(.d0(a), .d1(b),
           .s(sel), .out(y));
```

```
module mux2
  #(parameter width = 8)
   (input  logic [width-1:0] d0
    input  logic             s,
    output logic [width-1:0] y)
  assign y = s ? d1 : d0;
endmodule
```

Test Bench: Overview & Concept
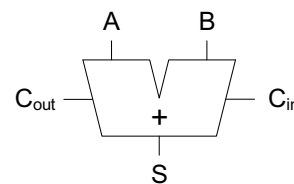# (Simple w/ Asserts)

Adders

Half Adder

A   B

C_{out} +

S

A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S \quad = A \oplus B$$
$$C_{out} \quad = AB$$

Full Adder

A   B

C_{out} + C_{in}

S

C_{in}	A	B	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S \quad = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in}$$

Ripple Carry Adder: Propagation Delay
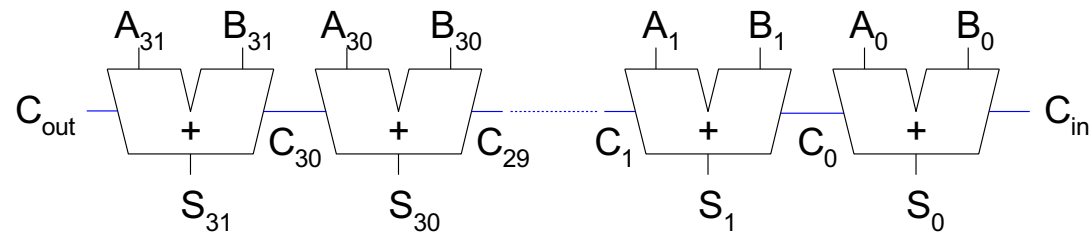
Questions

Next Time

- Studio