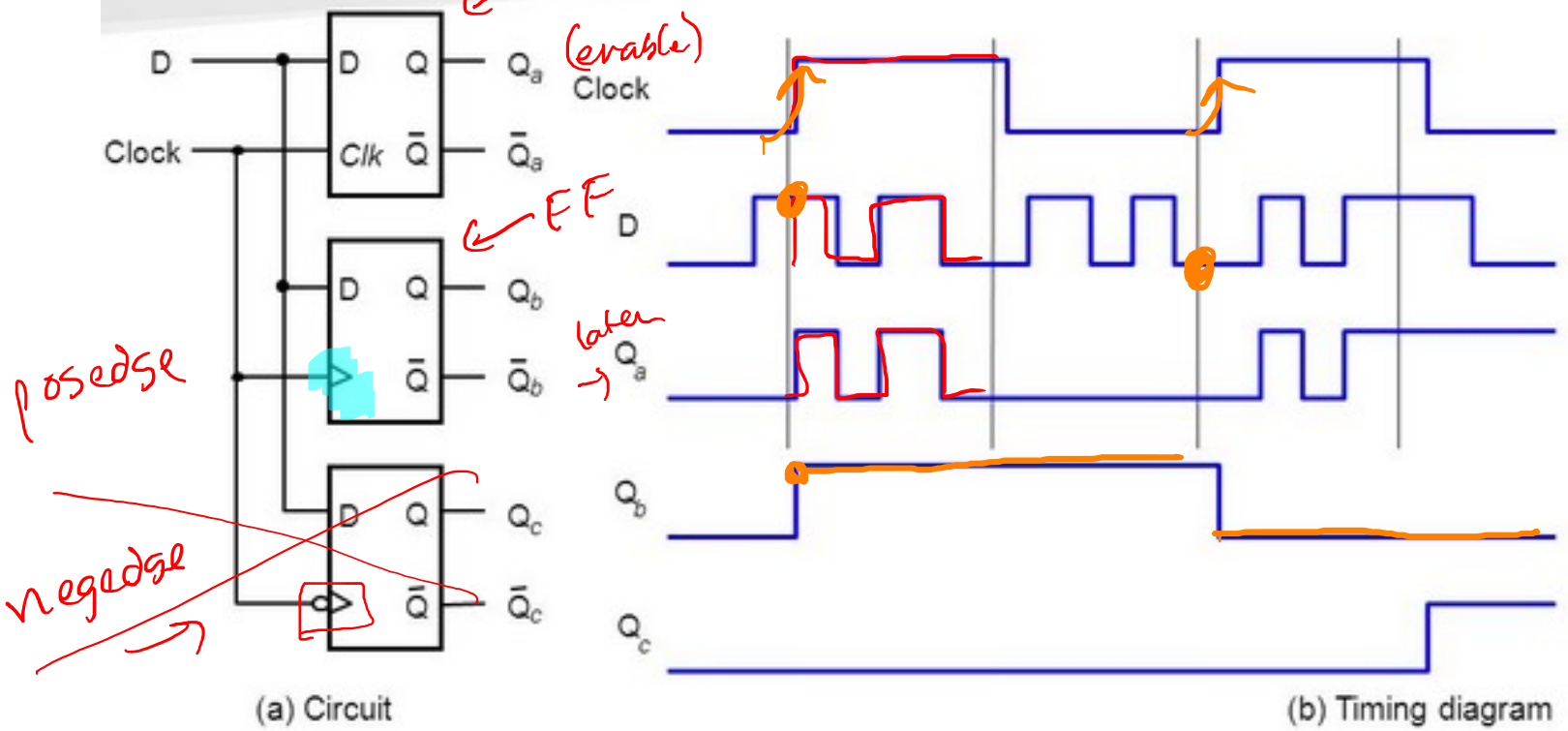
Flip Flops + Sequential Logic

Andrew Lukefahr

D Latch versus D Flip-Flop



(a) Circuit (b) Timing diagram

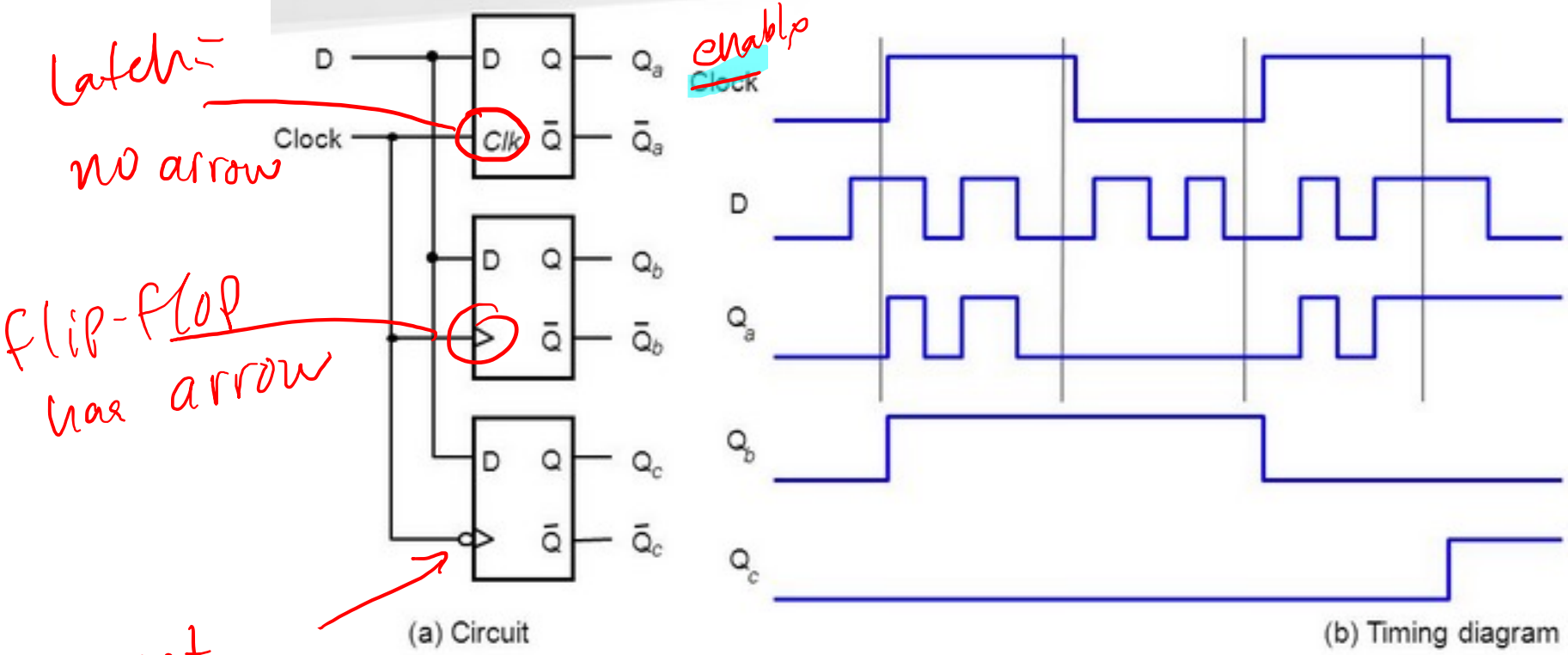
Comparison of level-sensitive and edge-triggered devices

51

Latch → output follows input (D) when enable (clk) is high

FF → output follows input only on rising edge of enable (clk)

D Latch versus D Flip-Flop



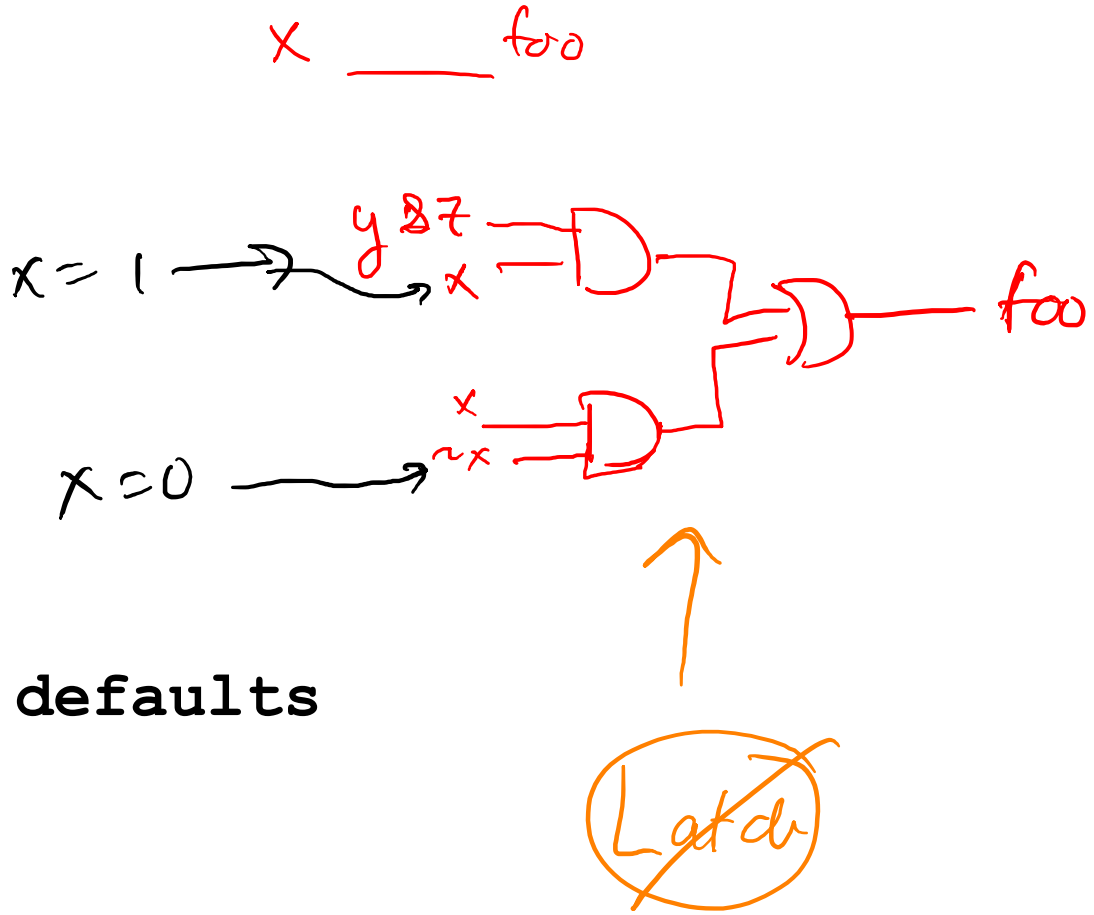
(a) Circuit

(b) Timing diagram

Comparison of level-sensitive and edge-triggered devices

51

Defaults




```
wire x,y,z;
logic foo, bar ;

always_comb begin
    foo = x; bar = x; //good: defaults
     if (x) foo =  y & z; //
     if (x) bar = y | z ; //
end
```

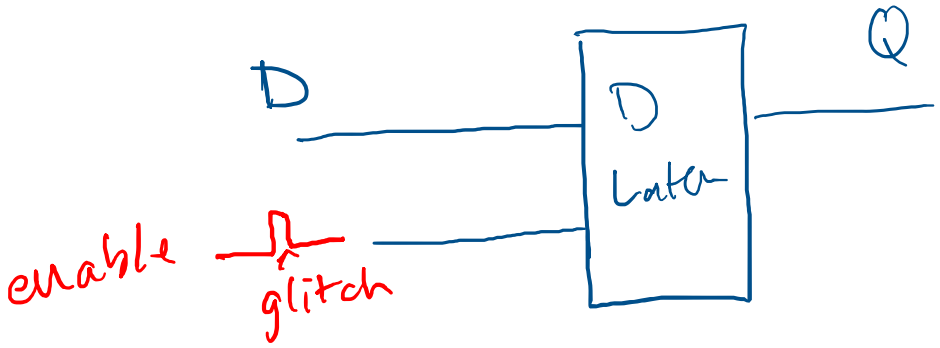
What if `x == 0`? `foo = bar = x`!

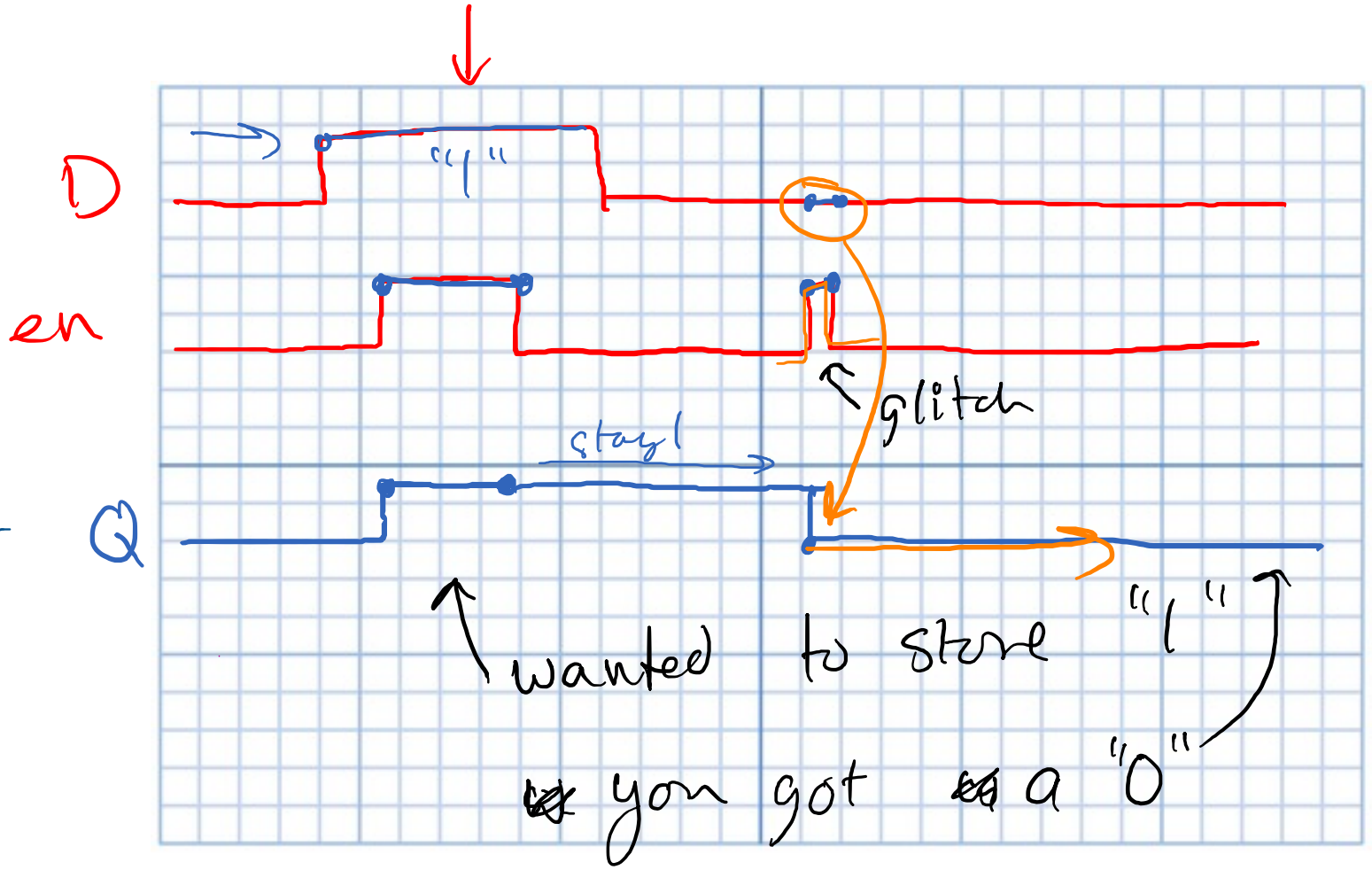
Always specify defaults for **`always_comb`**!

"Warnis: Inferris Later"

Always specify defaults for `always_comb`!

Glitches on D-Latches





Flip-Flop in Verilog

```
module d_ff (
    input          d,   //data
    input          clk, //clock
    output logic q    //output register
);

     always_ff @( posedge clk )
     begin
          q <= d; //non-blocking assign
     end

endmodule
```

BLOCKING (=) FOR `always_comb`

NON-BLOCKING (<=) for `always_ff`

Blocking vs. NonBlocking

```
always_comb
begin
    x = a + 1;
    y = x + 1;
    z = z + 1;
end
```

x = a + 2;

```
always_ff @(posedge clk)
begin
    x <= a + 1;
    y <= x + 1;
    z <= z + 1;
end
```

all happen @ same time

ordering matters

x = a+1
x = a+2 ⇒
y = x+1

x = a+2
x = a+1
y = x+1

ordering does NOT matter

```
x <= a+1;
y <= x+1;
```

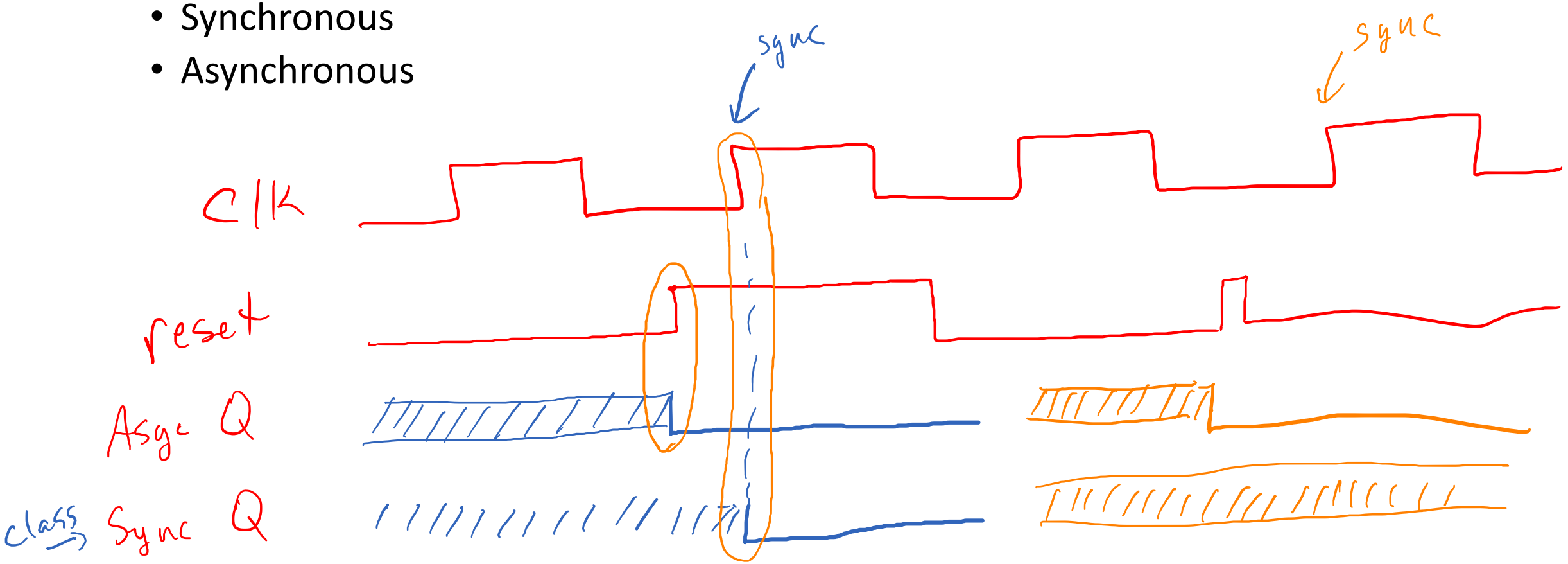
```
y <= x+1
x <= a+1
```

What's the initial value for a DFF?

```
module d_ff (
     input d,              //data
     input clk,            //clock
     output logic q              //output
);

      always_ff @( posedge clk )
      begin
            q <= d; //non-blocking assign
      end

endmodule
```

D-FF's with Reset

- Two different ways to build in a reset
  - Synchronous
  - Asynchronous

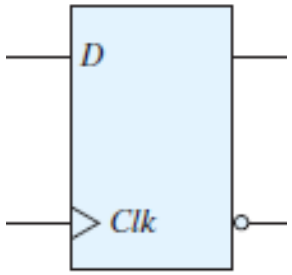


D-FF's with Reset

- Two different ways to build in a reset
  - Synchronous
  - Asynchronous

- We always use synchronous resets for this class!

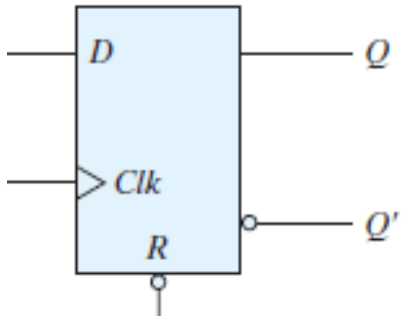
Verilog models of D flip-flop



Edge triggered D flip-flop:

```
logic Q;
always_ff @ (posedge clk)
  Q <= D;
```

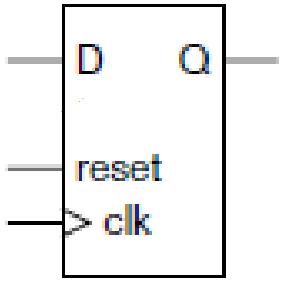
No reset ff



Edge triggered, asynchronous reset D flip-flop:

```
logic Q;
always_ff @ (posedge clk, negedge rst)
  if (~rst) Q <= 1'b0;  //asynch. reset
  else Q <= D;
```

Not used in class



Edge triggered, synchronous reset, clock enable D flip-flop:

```
logic Q;
always_ff @(posedge clk)
  if (reset) Q <= 1'b0; // synch. reset
  else Q <= d;
```

DFF with Synchronous Reset

```
module d_ff (
    input d,                //data
    input clk,              //clock
    input rst,              //reset
    output logic q          //output
);

      always_ff @( posedge clk )
      begin
            if (rst) q <= 'h0; //reset case
            else     q <= d; //non-reset case
      end

endmodule
```

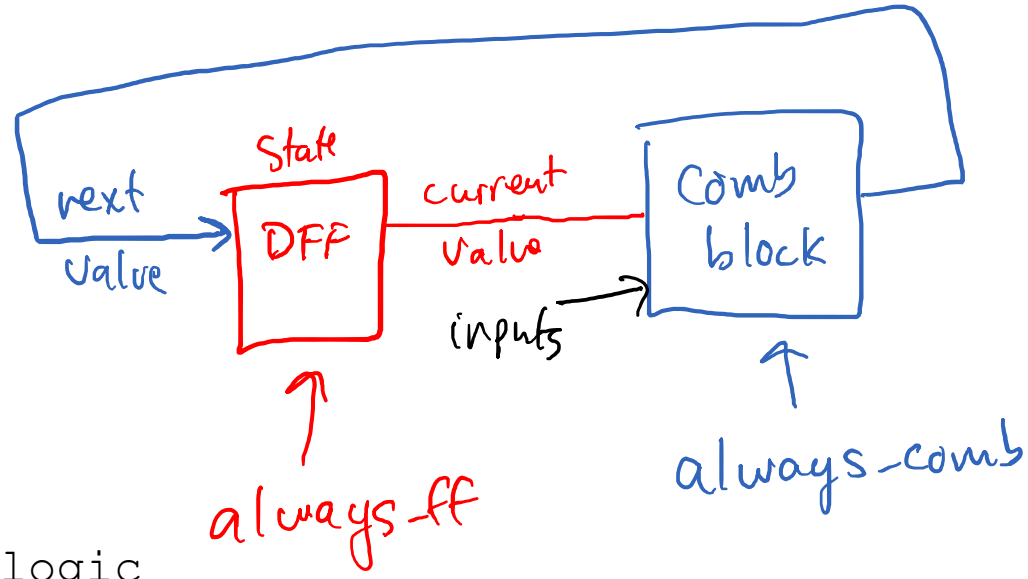
What does this do?

```
module mystery(
    input           clk,   //clock
    input           rst,   //reset
    output logic    out    //output
);
    logic [3:0] Q;
    logic [3:0] sum;

    always_ff @( posedge clk ) // <- sequential logic
    begin
       if (rst) Q <= 4'h0;
       else     Q <= sum;    //non-blocking
    end

    always_comb begin // <- combinational logic
        sum = Q + 4'h1;  //blocking
        out = sum[3];
    end

endmodule
```

What does this do?

```
module counter(
    input            clk,   //clock
    input            rst,   //reset
    output logic     out    //output
);
    logic [3:0] Q;
    logic [3:0] sum;

    always_ff @( posedge clk ) // <- sequential logic
    begin
       if (rst) Q <= 4'h0;
       else     Q <= sum;    //non-blocking
    end

    always_comb begin // <- combinational logic
        sum = Q + 4'h1;  //blocking
        out = sum[3];
    end

endmodule
```

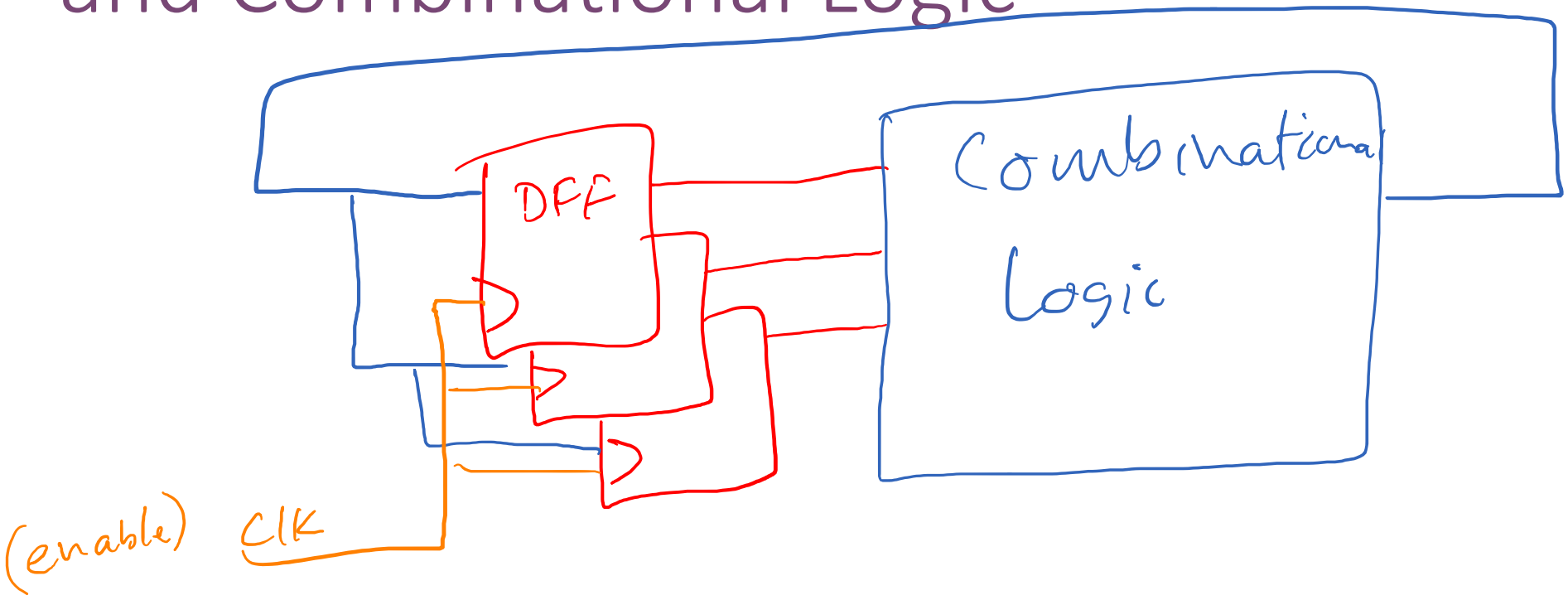
```
module counter(
    input clk,                  //clock
    input rst,                  //reset
    output logic out//output
);
    logic [3:0] D;
    wire [4:0] sum;

    always_ff @( posedge clk ) // <- sequential logic
    begin
       if (rst) D <= 4'h0;
       else     D <= sum;     //non-blocking
    end

    always_comb // <- combinational logic
        {out,sum} = {0,D} + 5'h1;  //blocking

endmodule
```



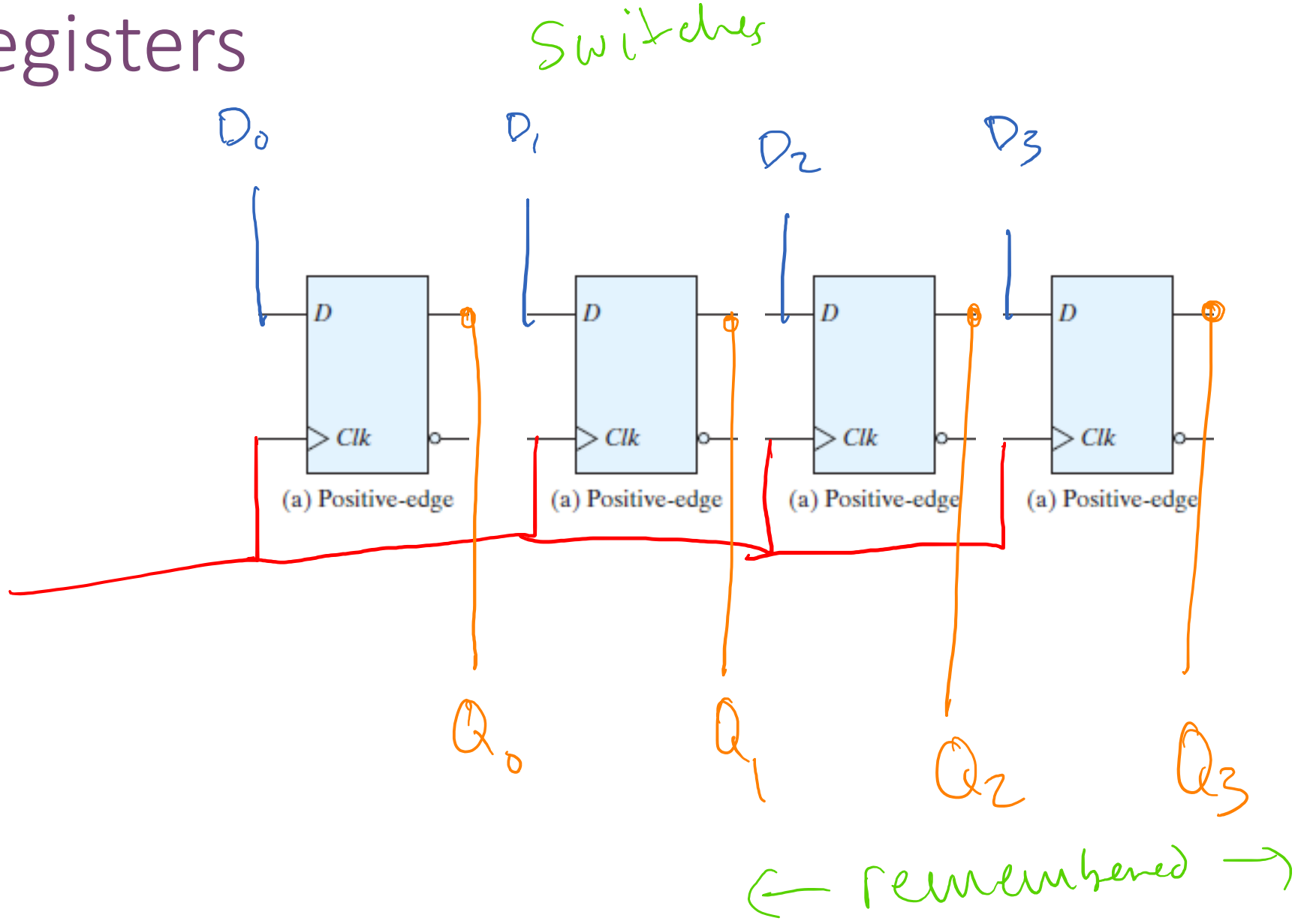
uses FF's

uses AND, OR, NOT Gates

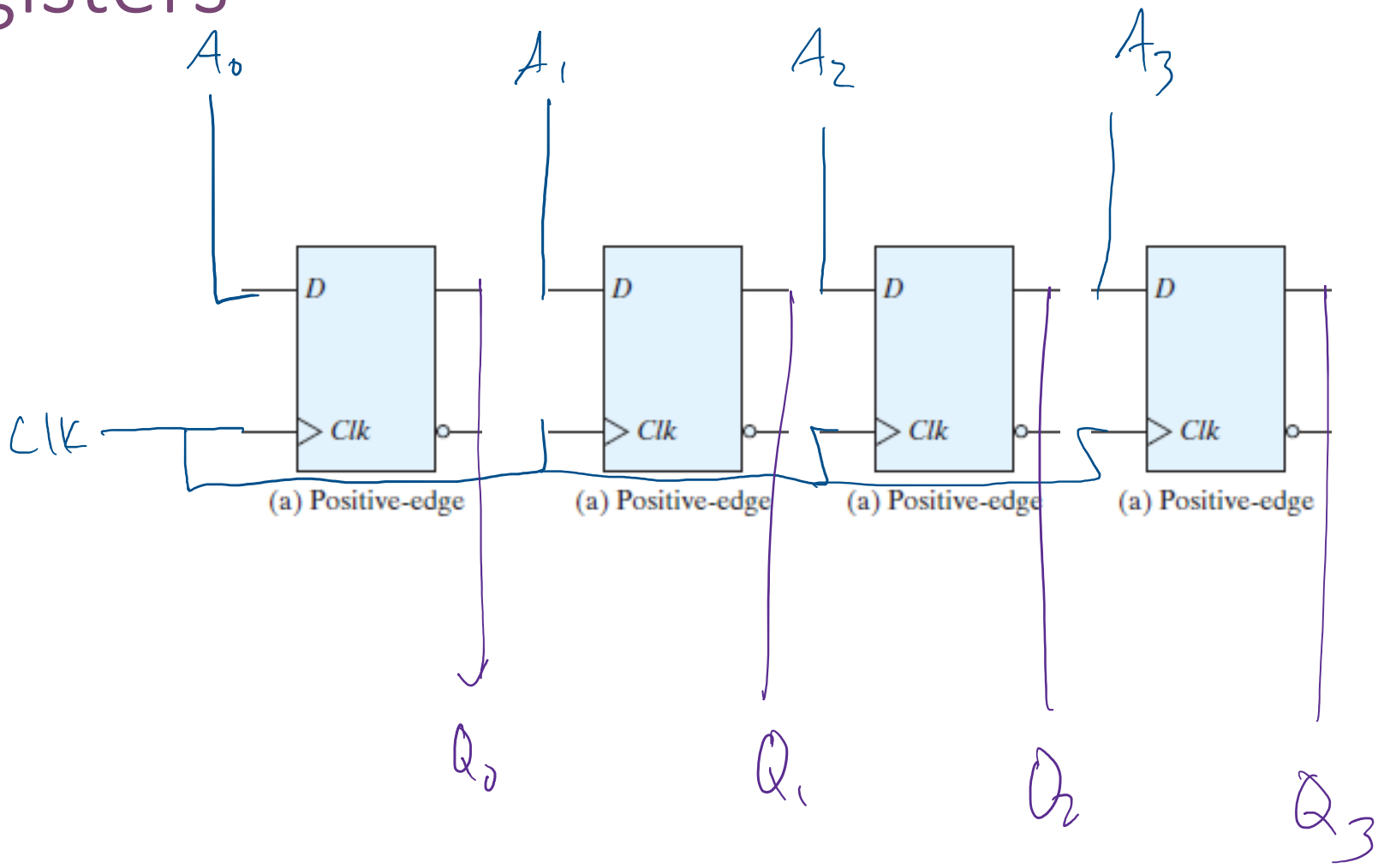
Sequential Logic uses both Flip-Flops and Combinational Logic



Registers


Switches
D_0
D_1
D_2
D_3
D
Clk
(a) Positive-edge
Q_0
Q_1
Q_2
Q_3
← remembered →

Registers



registers in a CPU?

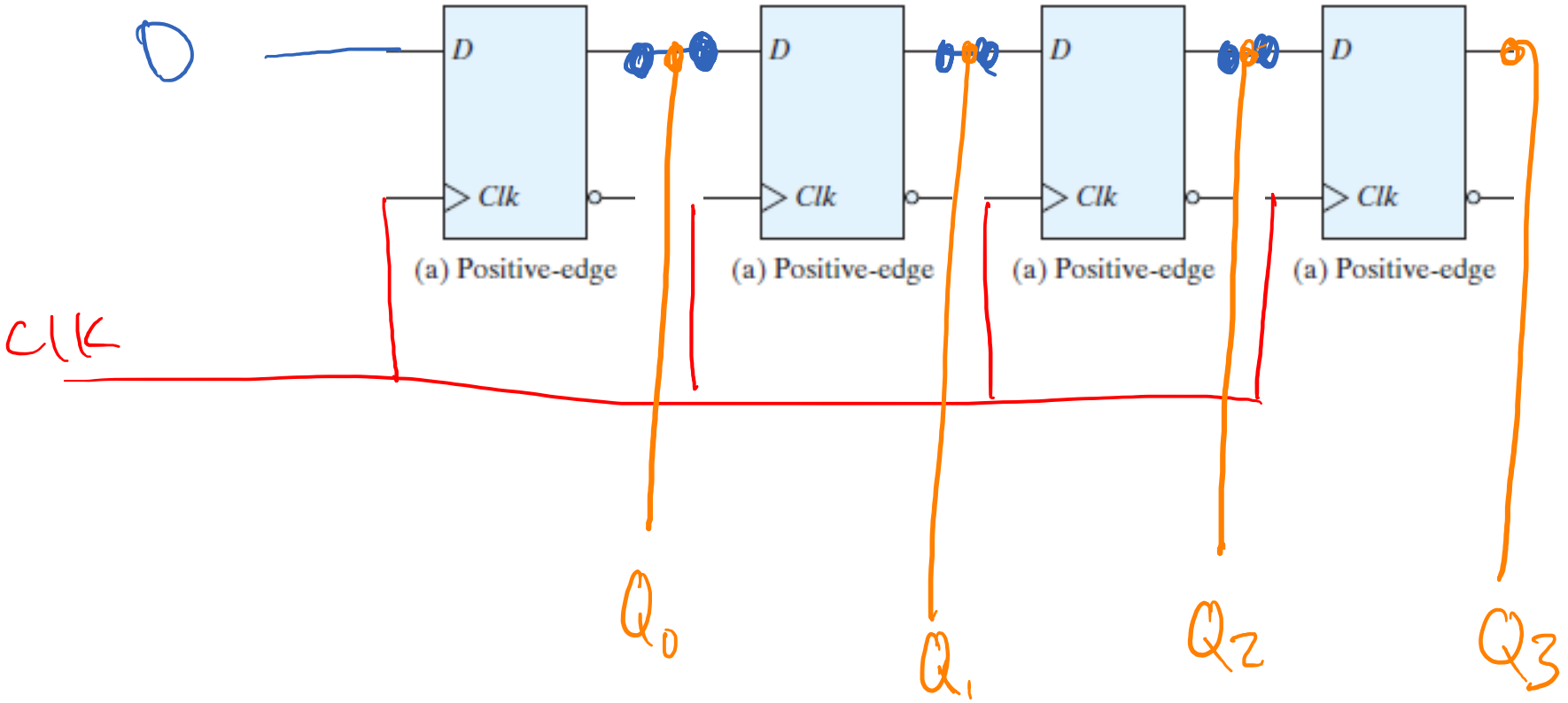
4-bit Register in Verilog

```
module d_ff (
    input               d,  //data
    input               clk,//clock
    input               rst,//reset
    output logic        q   //output
);

      always_ff @( posedge clk )
      begin
            if (rst) q <= 'h0; //reset case
            else     q <= d; //non-reset case
      end

endmodule
```

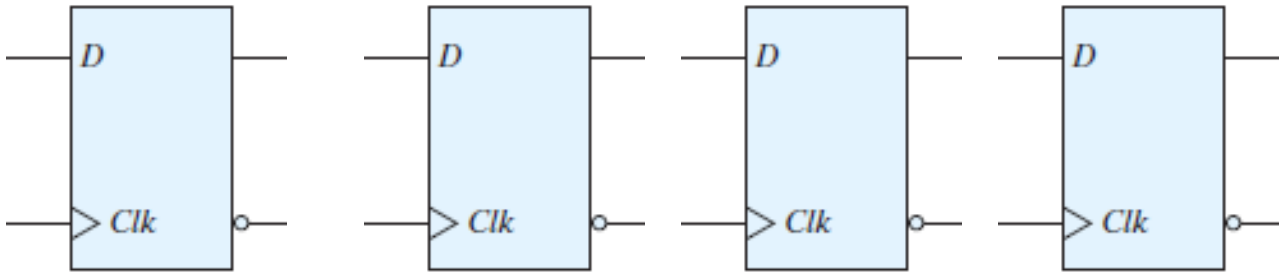
4-bit Register in Verilog

```
module d_ff (
    input        [3:0] d,  //data
    input              clk,//clock
    input              rst,//reset
    output logic [3:0] q   //output
);

      always_ff @( posedge clk )
      begin
            if (rst) q <= 'h0; //reset case
            else     q <= d; //non-reset case
      end

endmodule
```

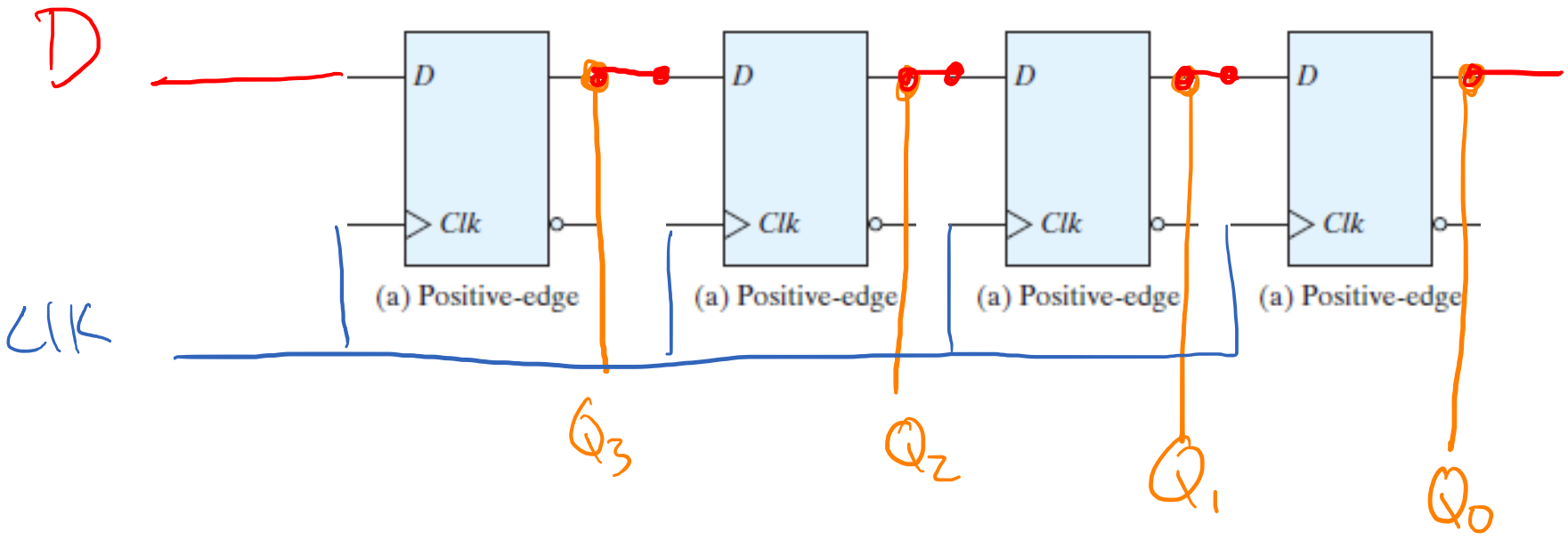
D Flip-Flops as Shift Registers



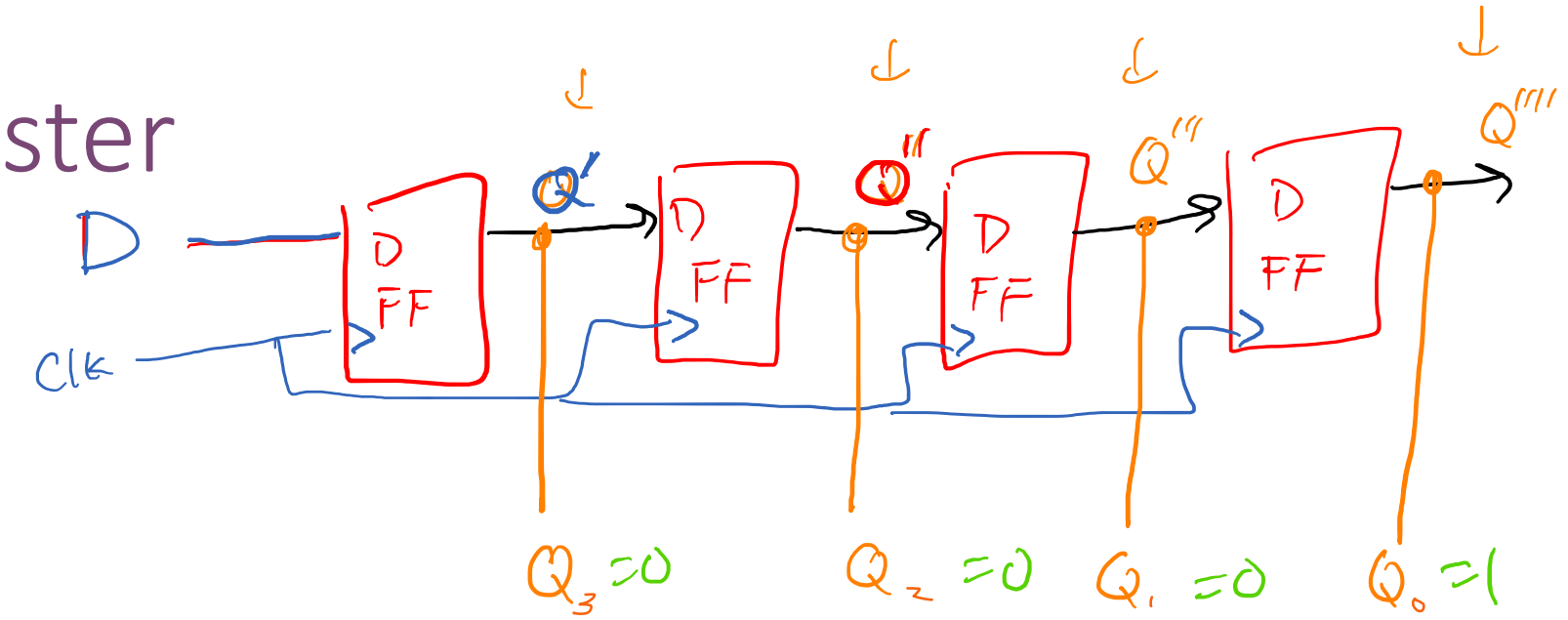
D Flip-Flops as Shift Registers

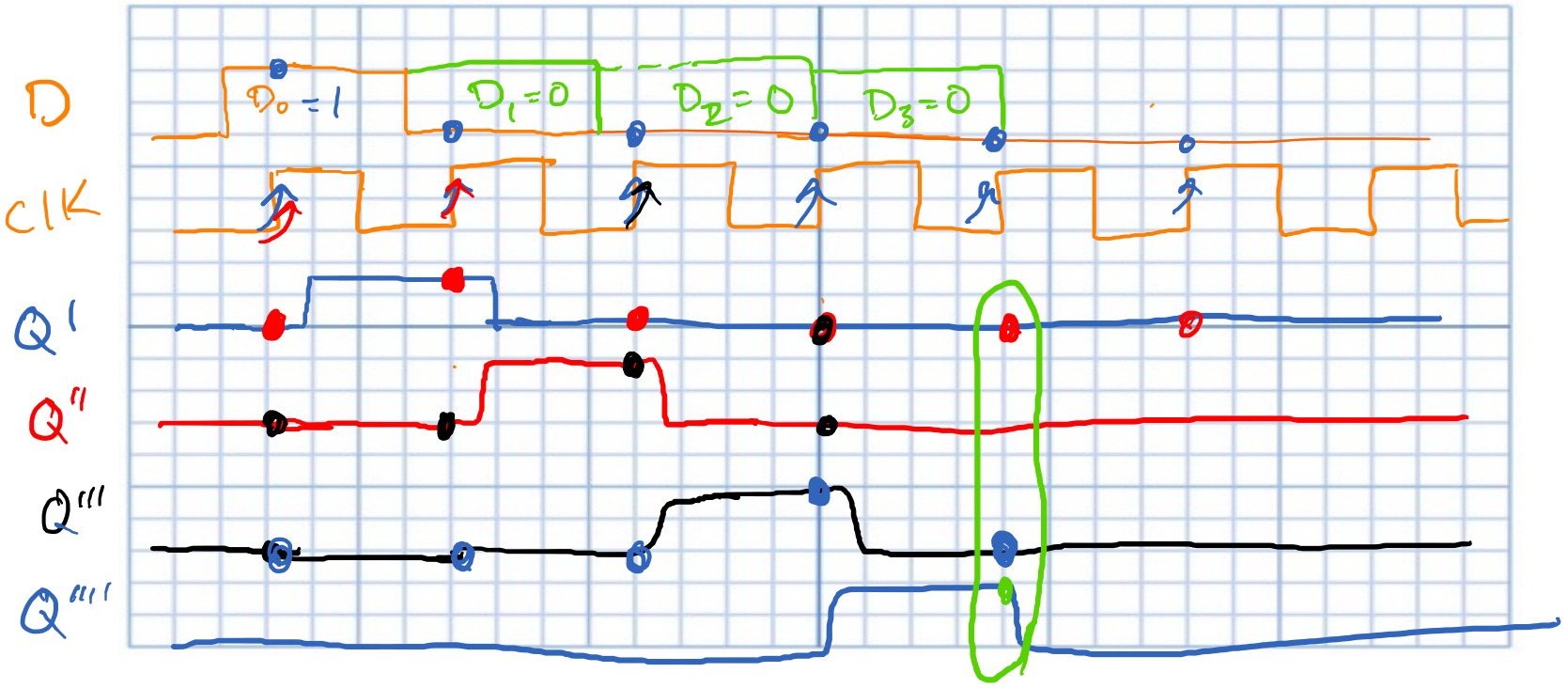


D Flip-Flops as Shift Registers

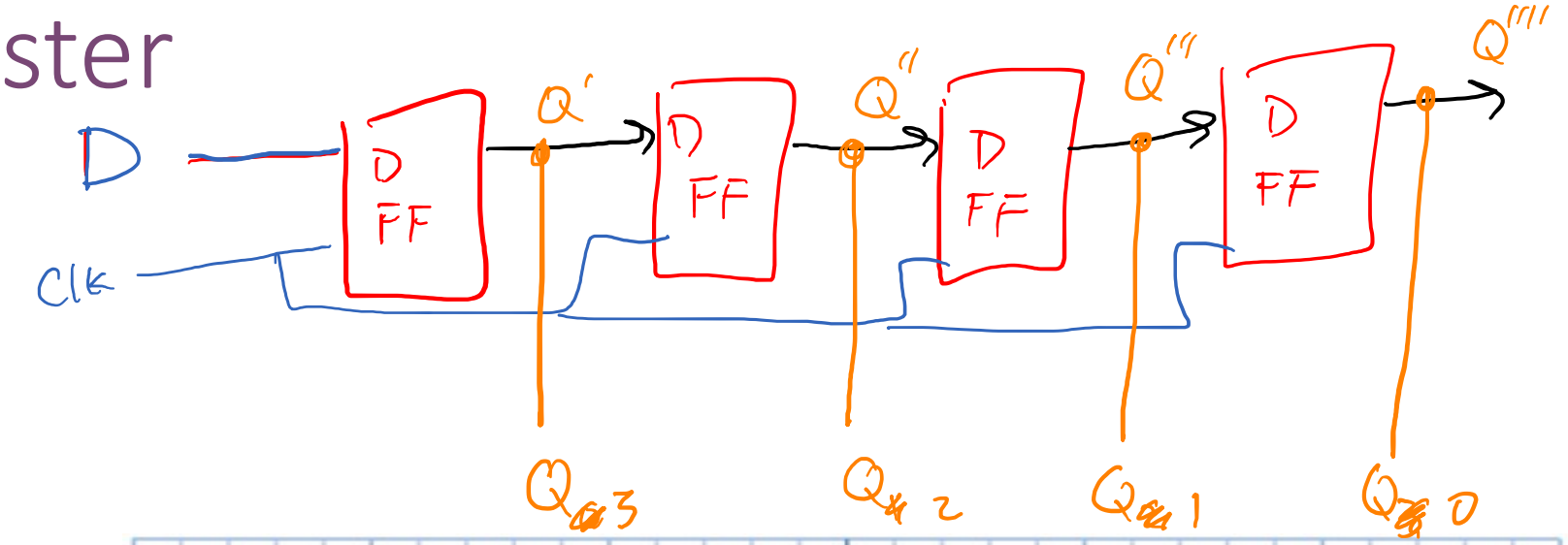


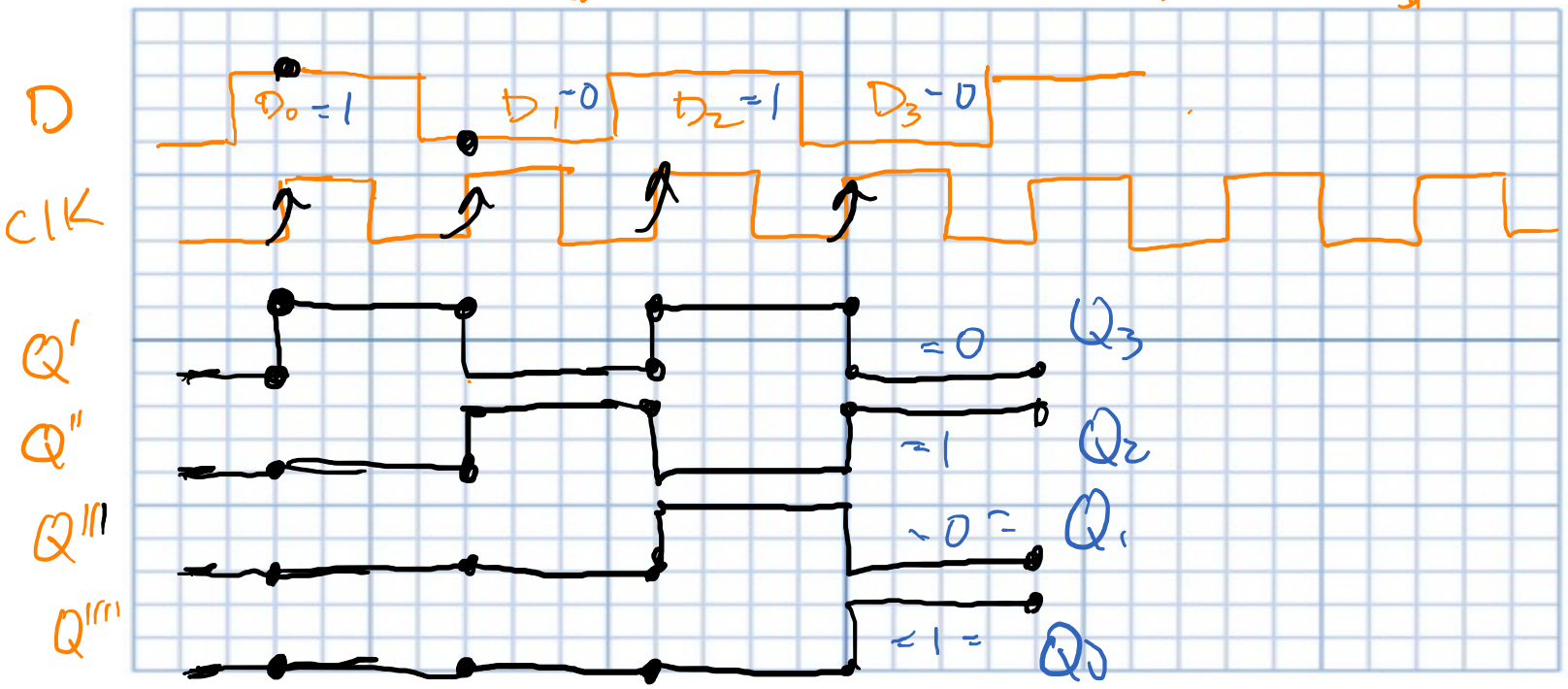
Shift Register





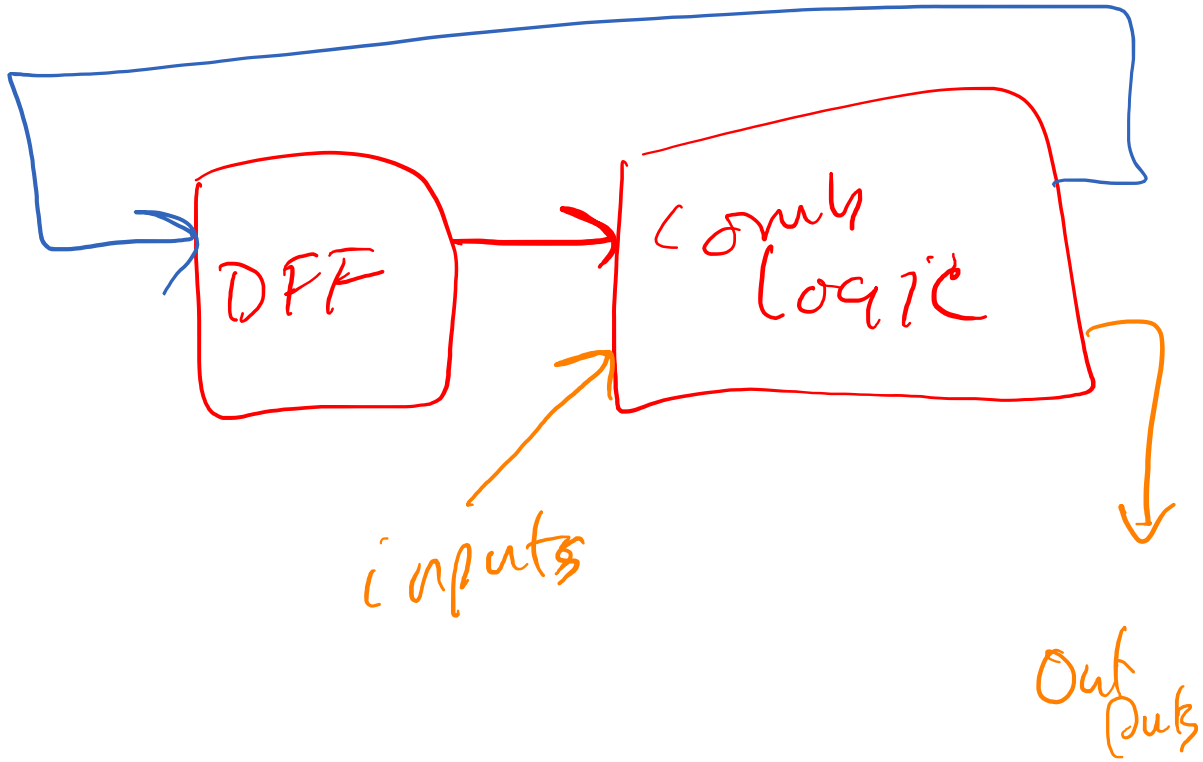
Shift Register





Shift-Register in Verilog

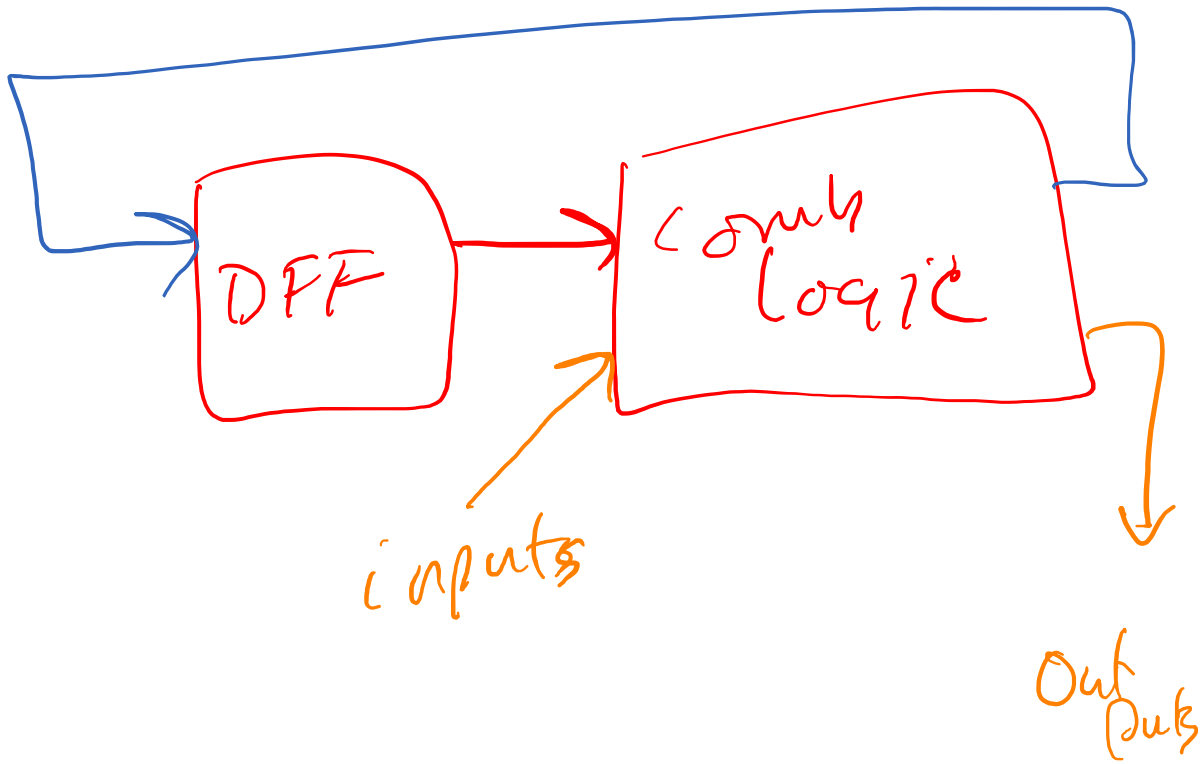
```
module shift_register (
    input clk, rst, D,
    output [3:0] Q );
```




```
endmodule
```

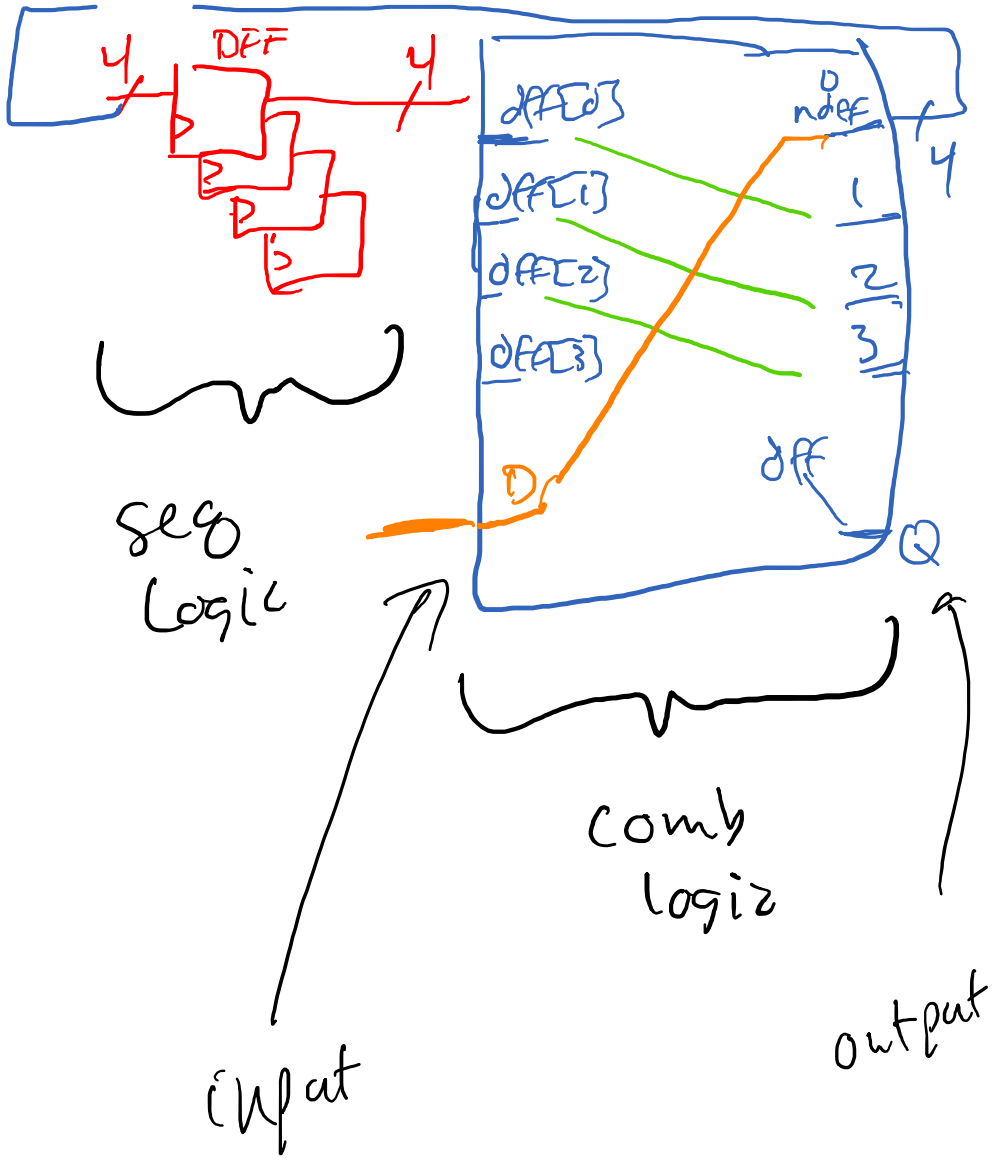
Shift-Register in Verilog

```
module shift_register (
    input clk, rst, D,
    output [3:0] Q );

    logic [3:0] dff;
    logic [3:0] next_dff;

    always_ff(@posedge clk) begin
       if (rst) dff <= 4'h0;
       else     dff <= next_dff;
    end

    always_comb
       next_dff = { dff[2:0], D};

    assign Q = dff;

endmodule
```



Shift-Register in Verilog

```
module shift_register (
    input clk, rst, D,
    output [3:0] Q );

    logic [3:0] dff;
    logic [3:0] next_dff;

    always_ff(@posedge clk) begin
       if (rst) dff <= 4'h0;
       else     dff <= next_dff;
    end

    always_comb
       next_dff = { dff[2:0], D};

    assign Q = dff;

endmodule
```



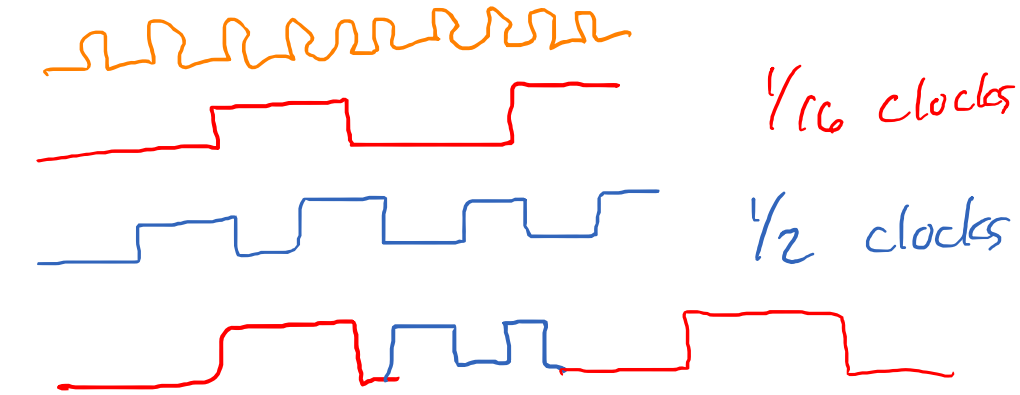
Inputs can affect output or state

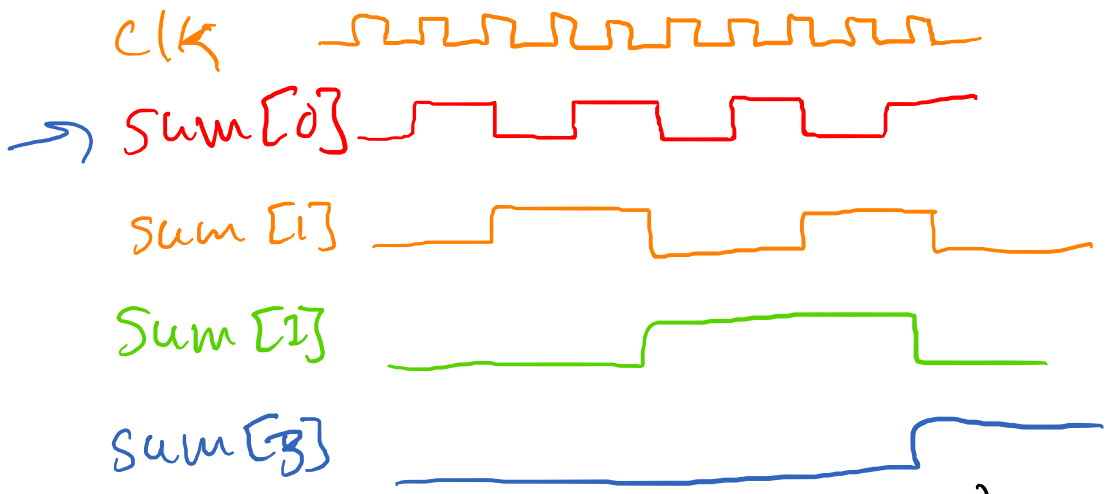
```
module counter(
    input clk, rst
    input          out_fast, //faster output
    output logic   out    //output
);
    logic [3:0] Q;
    logic [3:0] sum;

    always_ff @( posedge clk ) begin
      if (rst) Q <= 4'h0;
      else     Q <= sum;
    end

    always_comb begin
        sum = Q + 4'h1;
        out = sum[3];
    end

endmodule
```

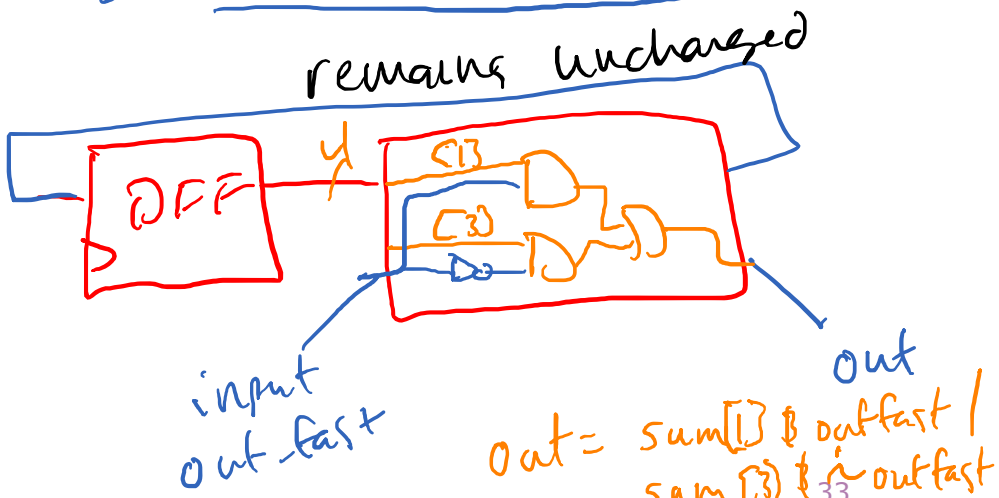
```
module counter(
    input clk, rst
    input          out_fast, //faster output
    output logic   out    //output
);
    logic [3:0] Q;
    logic [3:0] sum;

    always_ff @( posedge clk ) begin
      if (rst) Q <= 4'h0;
      else     Q <= sum;
    end

    always_comb begin
        sum = Q + 4'h1;
      ✓ out = sum[3]; // default
      ✓ if ( out_fast) out = sum[1];
    end

endmodule
```

out_fast=0

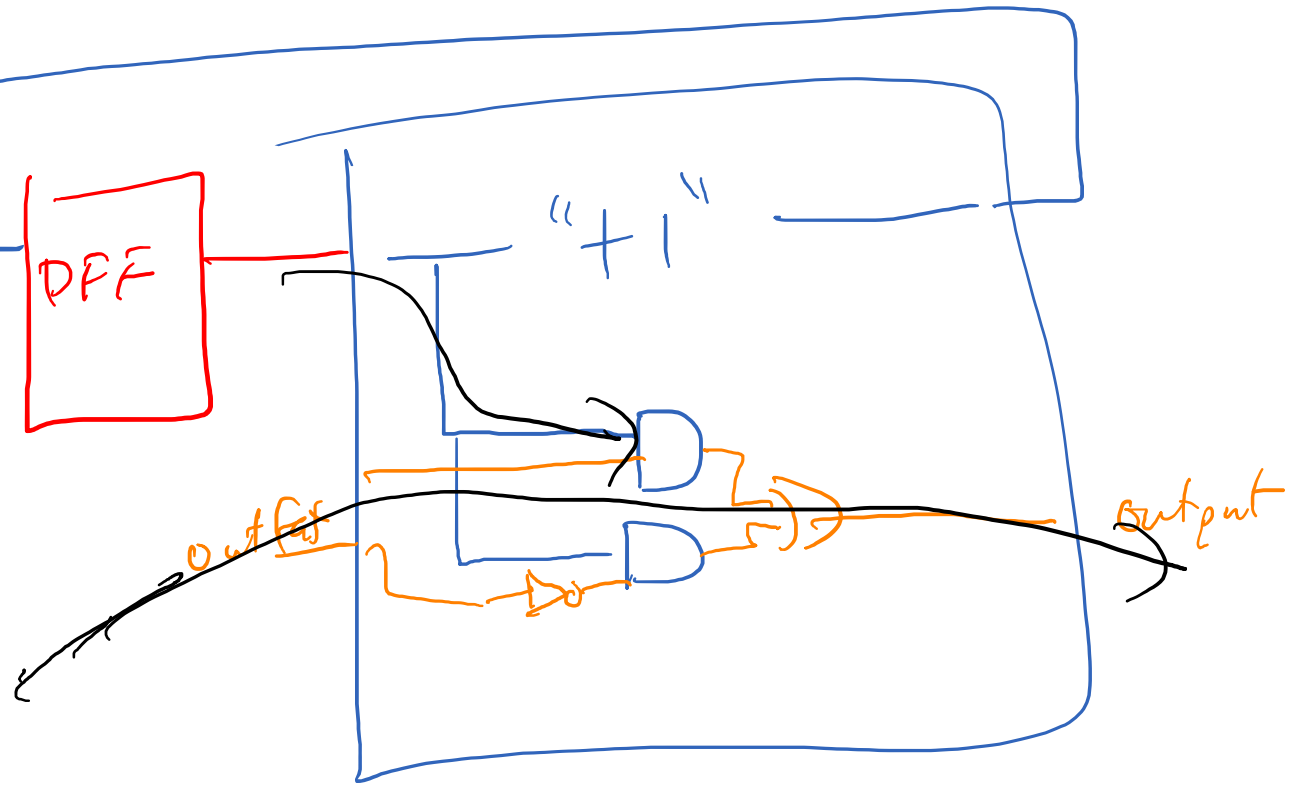
out_fast=1



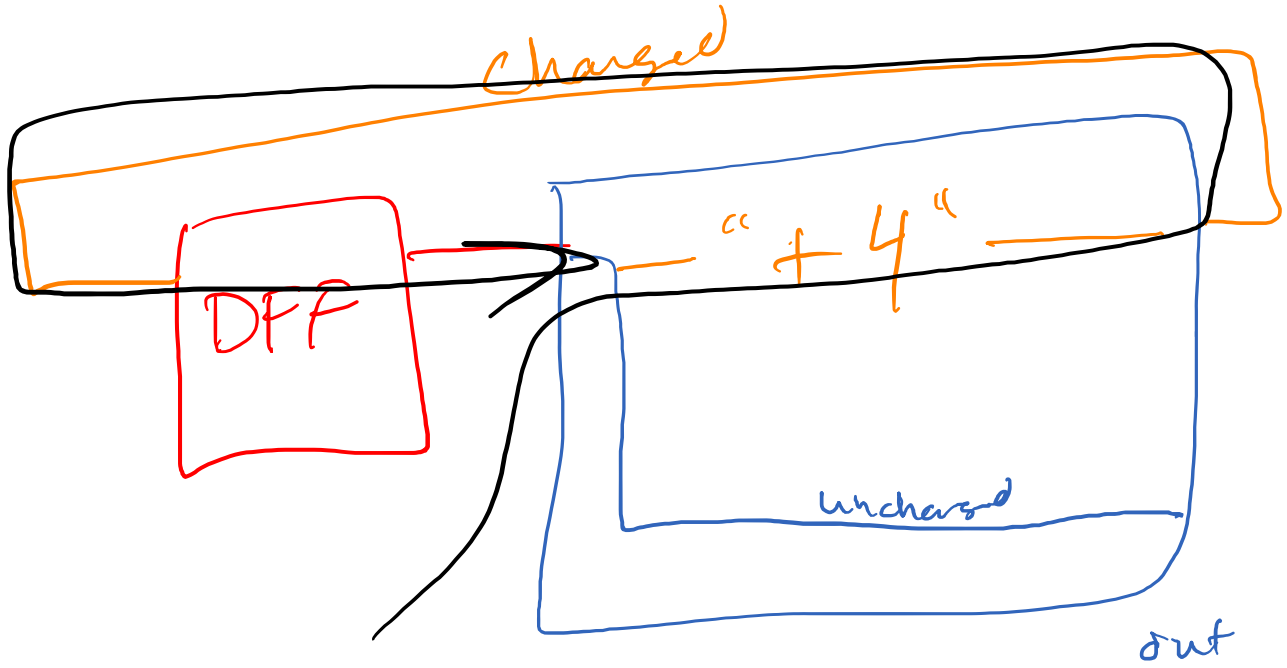




```
module counter(
    input clk, rst
    input           out_fast, //faster output
    output logic    out    //output
);
    logic [3:0] Q;
    logic [4:0] sum;

    always_ff @( posedge clk ) begin
       if (rst) Q <= 4'h0;
       else     Q <= sum;
    end

    always_comb begin
       sum = Q + 4'h1;
        out = sum[3];
       if (out_fast) out = sum[2];
    end

endmodule
```



```
module counter(
    input clk, rst
    input           out_fast, //faster output
    output logic    out    //output
);
    logic [3:0] Q;
    logic [4:0] sum;

    always_ff @( posedge clk ) begin
       if (rst) Q <= 4'h0;
       else     Q <= sum;
    end

    always_comb begin
        sum = Q + 4'h1;
        out = sum[3];
        if (out_fast) sum = {0,Q} + 5'h4;
    end

endmodule
```



Next Time

- Finite State Machines (FSMs)