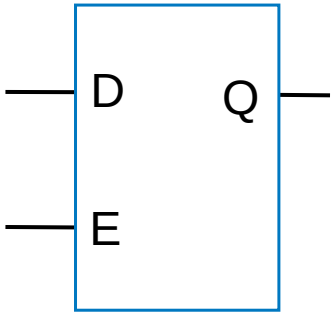
Latches and Flip-Flops

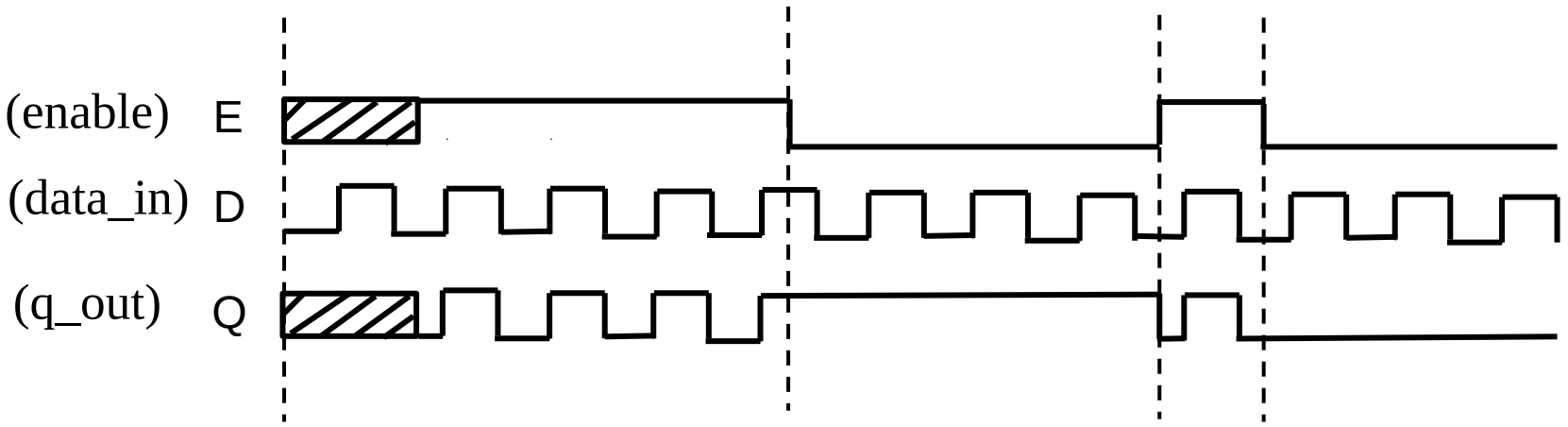
- Level Sensitive Circuits
- Edge Triggered Behaviour
- Resets

Transparent Latch




```
module Latch_CA (output Q, input D, E);
   assign Q = E ? D : Q;
endmodule
```

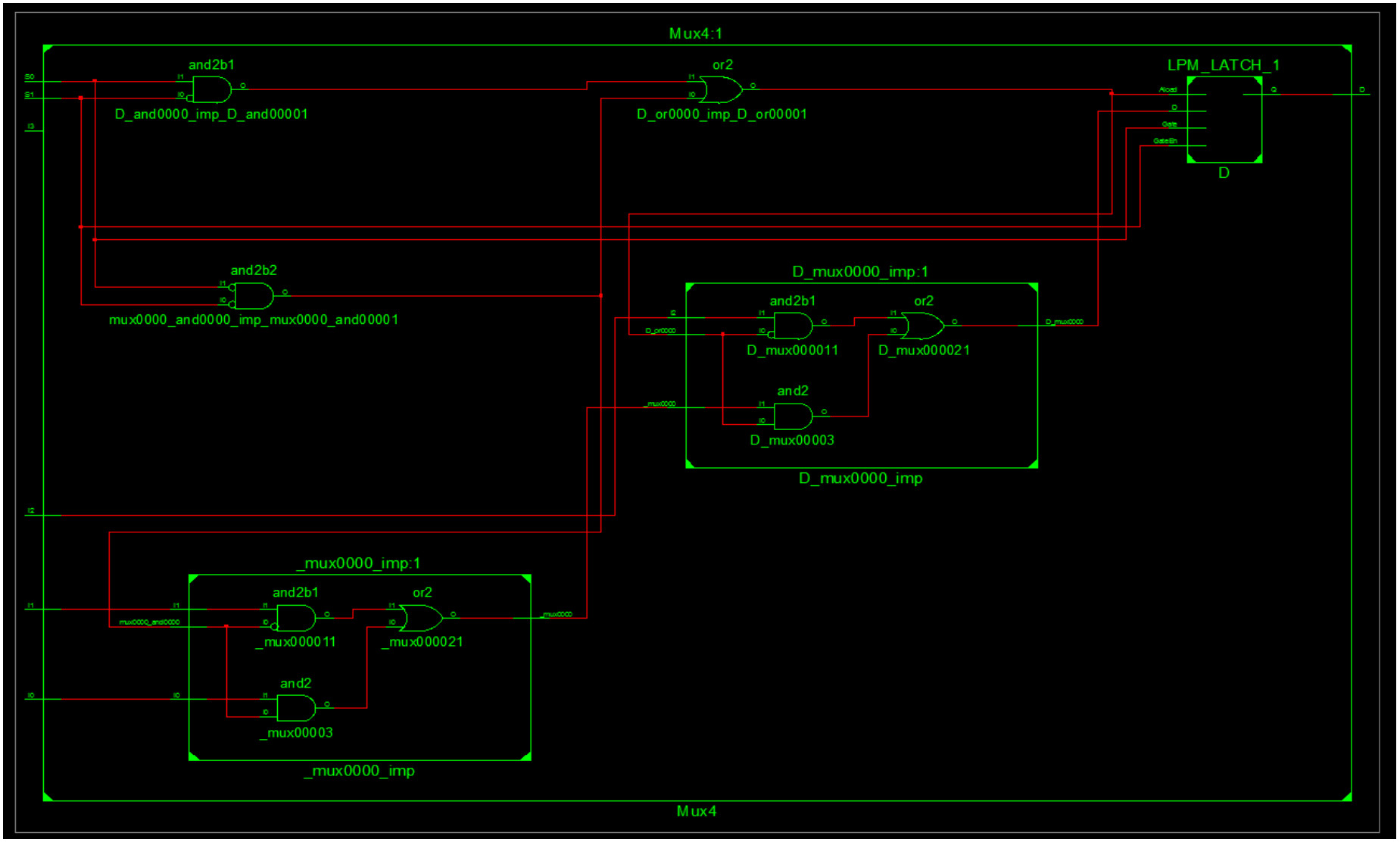
Continuous assignment with feedback used to model a latch.



What is the problem with this model ?

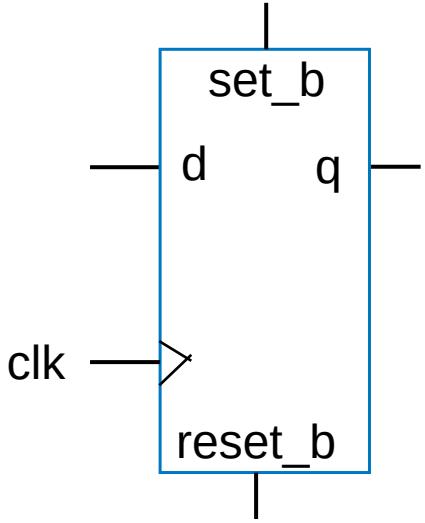
(see pitfalls in lecture 2)

```
module Mux4(I3, I2, I1, I0, S1, S0, D);

  input I3, I2, I1, I0;
  input S1, S0;
  output D;
  reg D;

  always @(S1, S0)
  begin
    if (S1==0 && S0==0)
      D <= I0;
    else if (S1==0 && S0==1)
      D <= I1;
    else if (S1==1 && S0==0)
      D <= I2;
end
endmodule
```



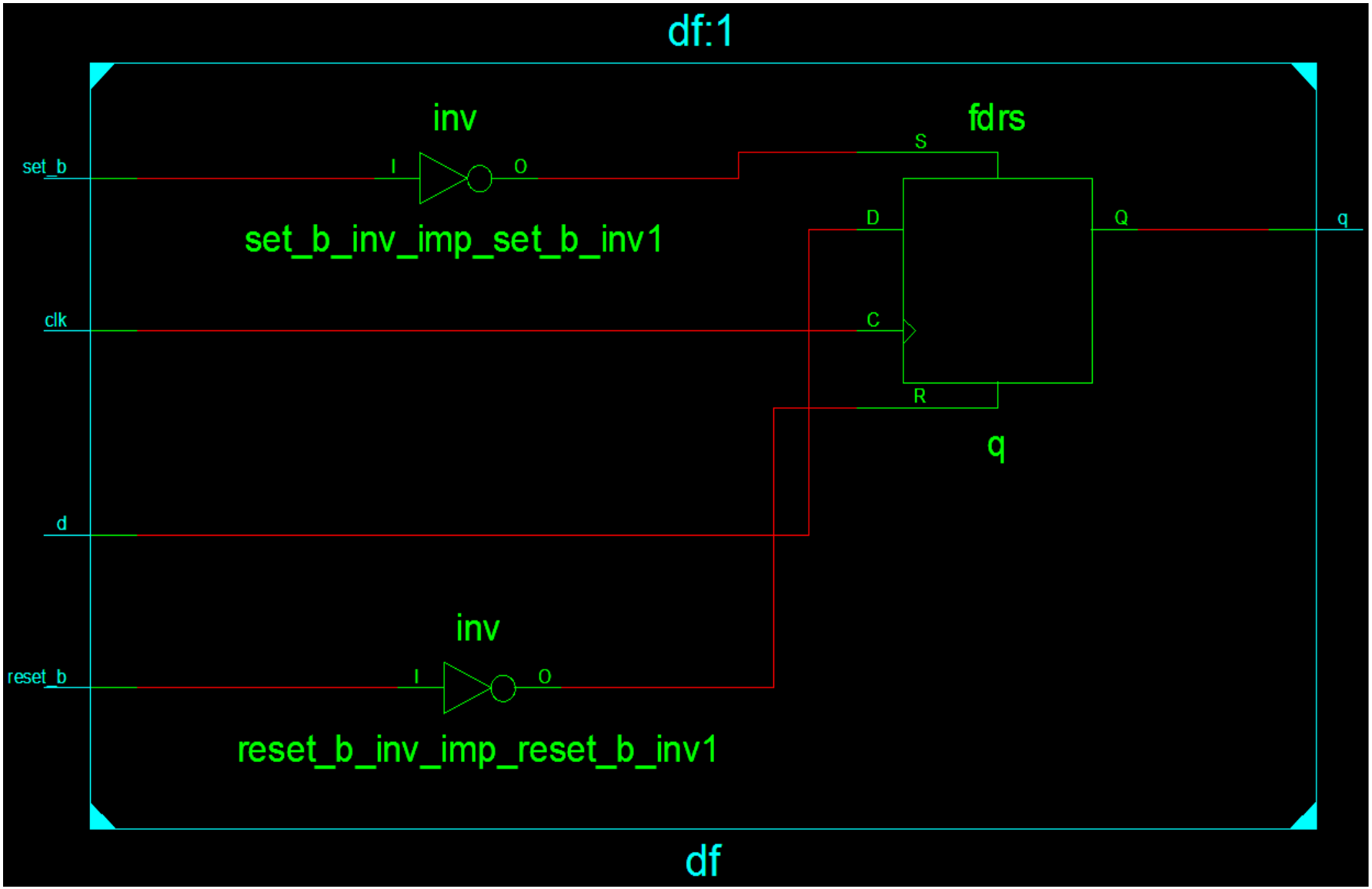
The synthesis result shows that a latch has been inferred for what should be a purely combinational circuit.

- Continuous assignments are limited to modeling combinational logic and transparent latches.
- For synchronous behaviour (flip-flop)we require edge-sensitive functionality.

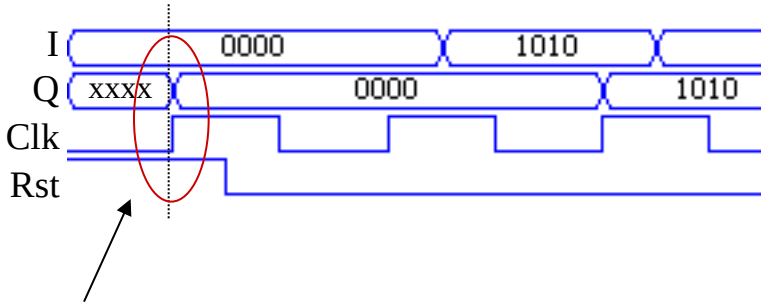



```
module df (output reg q, input d, set_b, reset_b, clk);
  always @(posedge clk)
    if (reset_b == 1'b0) q <= 0;
      else if (set_b == 1'b0) q <= 1;
    else q <= d;
endmodule
```

*Active clock edges specified by **posedge** and **negedge** keywords in Verilog.*

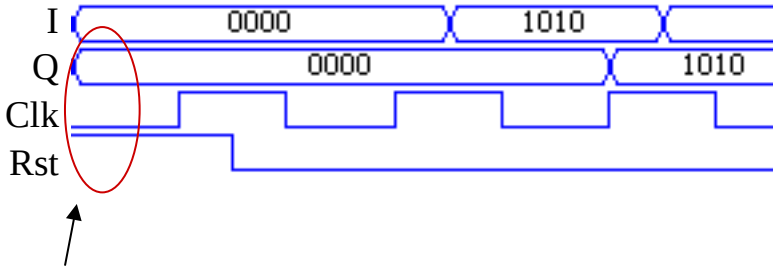

df:1
inv
fdrs
S
set_b
I
O
D
Q
q
set_b_inv_imp_set_b_inv1
clk
C
R
q
d
inv
reset_b
I
O
reset_b_inv_imp_reset_b_inv1
df

- synchronous resets
  - Rst input only considered during rising clock



Rst=1 has no effect until rising clock

- asynchronous reset
  - Rst input considered independently from clock
  - Add "posedge Rst" to sensitivity list



Rst=1 has almost immediate effect

Which is better – synchronous or asynchronous reset?

Hotly debated in design community

Each has pros and cons

e.g., asynchronous can still reset even if clock is not functioning, synchronous avoids timing analysis problems sometimes accompanying asynchronous designs

Blocking and Non Blocking Assignments

- Blocking assignment statement
  - Uses "="
  - Variable is updated before execution proceeds
  - Like variable update in C language
- Non-blocking assignment statement
  - Uses "<="
  - Update is scheduled but doesn't occur until later in simulation cycle

- Guideline for **always** block
  - Use blocking assignment when computing intermediate values
  - Use blocking assignment to generate combinational logic
  - Use nonblocking assignment to generate sequential logic