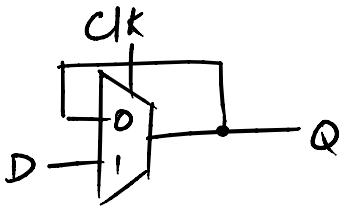
Verilog code for storage elements



Gated D-latch

```
Module D-latch (input D, Clk, output Q);
  reg Q;
  always@(D, Clk)
    if (Clk==1)
      Q=D;
  endmodule
```

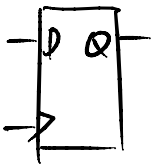
blocking statement

* in the absence of "else" it means stay unchanged. → implies storage

```
    else
      Q=Q;
```

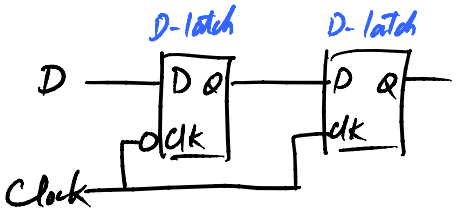
D-ff

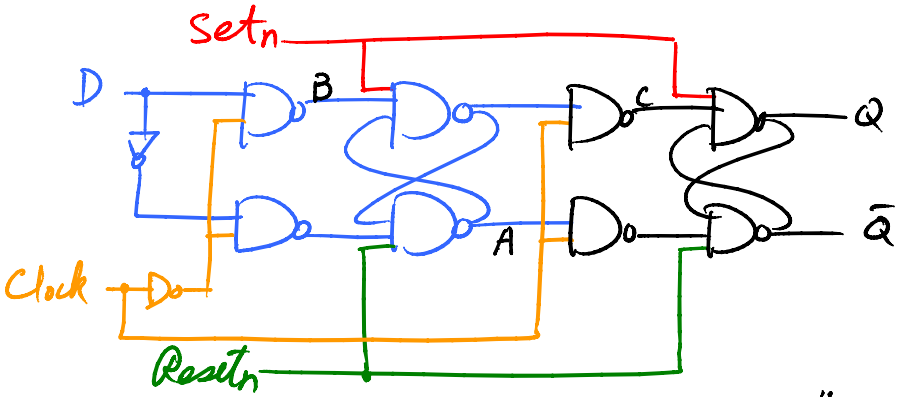
```
Module D-ff (input D, clock, output Q)
  reg Q;
  always@(posedge clock)
    Q<=D;
  endmodule
```



⊛ non-blocking statement
– used in ff. discuss more later!

Go back to master-slave D-ff





Setn / Resetn

active low → means activate when the signal is set to "0"

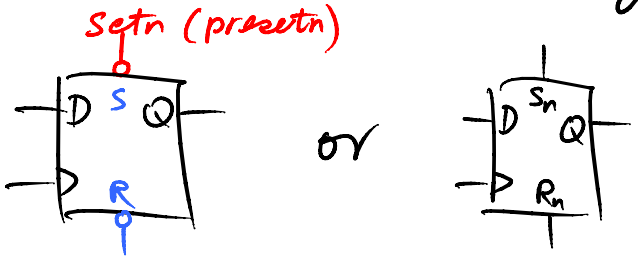
when Resetn=0, then $\bar{Q}$=1, if Clock=0, then C=1, Q=0

if Clock=1, B=1 and also A=1, C=1 and Q=0

Asynchronous reset!

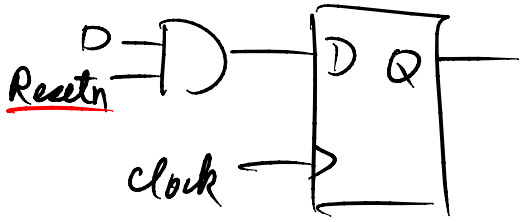
synchronous → means synch. to the clock.

Setn ("0") will set Q=1 regardless of the clock.



Module D-ff (input D, clock, Resetn, output Q);
reg Q;
always@(negedge Resetn, posedge clock)
if (Resetn == 0)
Q <= 0;
else
Q <= D;
endmodule

Synchronous Reset



when Resetn is inside the sensitivity list, it means it will affect the Q. Hence this describe asynchronous Resetn.

Reset will take place right at the ↓ of the Resetn signal

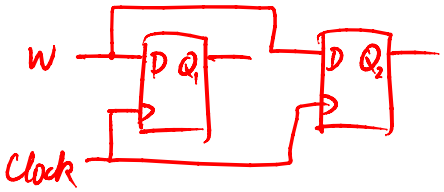
Module D-ff(...);
reg Q;
always@(posedge clock)
if (!Resetn)
Q <= 0;
else
Q <= D;
endmodule

only the clock will control the changes to the output

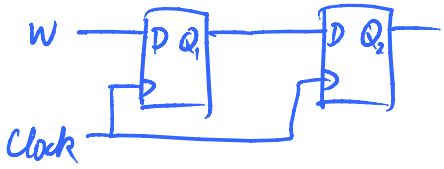
Blocking / non-blocking statement

= <=

example1
```
module blah (input w, clock, output Q1, Q2);
    reg Q1, Q2;
    always@(posedge clock)
    begin
        Q1 = w;
        Q2 = Q1;   // Q2 = w
    end
endmodule
```



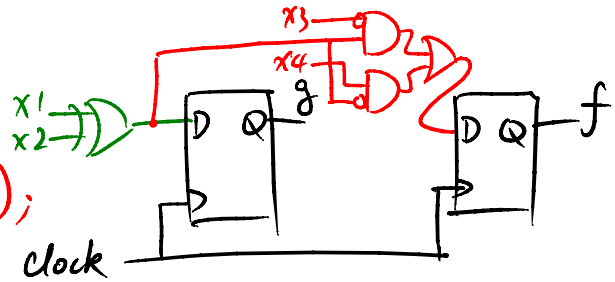
```
module blah (input w, clock, output Q1, Q2);
  reg Q1, Q2;
  always@(posedge clock)
  begin
    Q1 <= w;      Q1 copies w
    Q2 <= Q1;     Q2 .. Q1
  end
endmodule
```



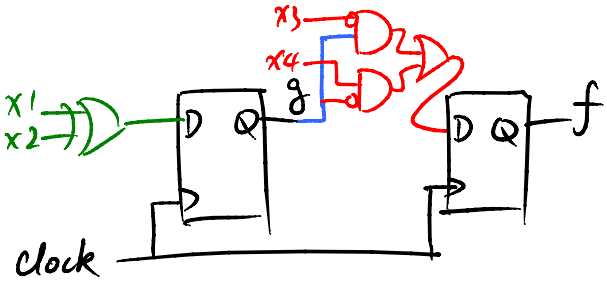
use "=" blocking statements for combinational logic.
" "<=" non-blocking statements for f/f.

example 2

```
module gosh (input x1, x2, x3, x4, clock, output reg f, g);
  always@(posedge clock)
  begin
    g = x1 ^ x2;
    f = (g & ~x3) | (~g & x4);
  end
endmodule
```



if we change "=" into "<=", then the cct. looks like:



Propagation Delays and Timing Issues

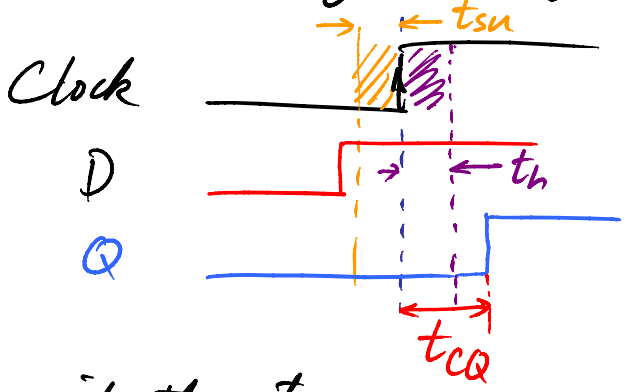
flip-flop timing parameters: t_{su}, t_h and t_{CQ}

t_{su} = set up time – is the window of time before the clock edge

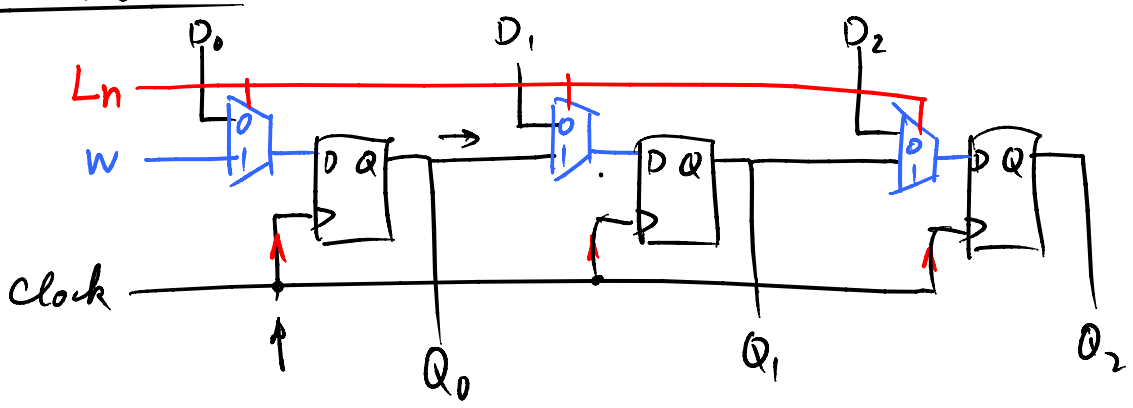
in which D should not be changed.

t_h : hold time – is the window of time after the clock edge in which D should remain unchanged.

t_{CQ} : Clock-to-Q time is the propagation delay required for Q to change value after a clock edge.



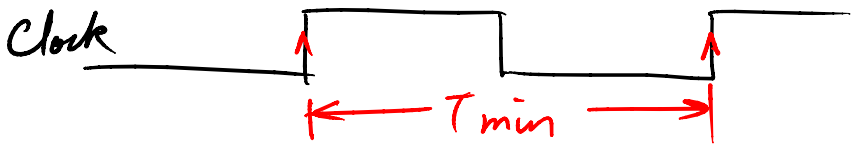
Consider the cct:



if Ln=0, then the 3 ffs will be loaded with D_0, D_1 and D_2 (parallel load)

if Ln=1, then w&Q_0 Q_1 are shifted from left to right, so the cct. is called a (shift register)

Frequency of the clock: f_{max}. Period $T_{min} = \frac{1}{f_{max}}$

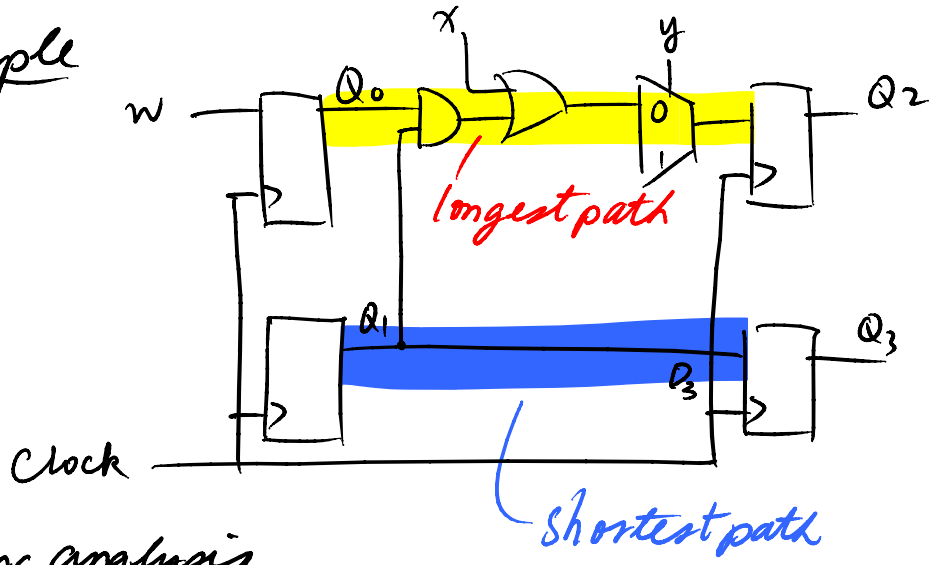


$T_{min} = t_{cQ} + t_{mux} + t_{su}$, max. freq $f_{max} = \frac{1}{T_{min}}$

eg. $t_{mux} = 2ns$, $t_{su} = 1ns$, $t_{cQ} = 1.5ns$

$T_{min} = 1.5 + 2 + 1 = 4.5ns$, $f_{max} = \frac{1}{4.5ns} \doteq 200\,MHz$.

example



Critical path: the longest delay path between any ffs.

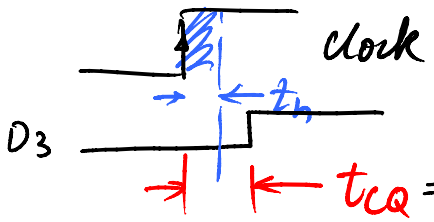
Timing analysis

$T_{min} = t_{cQ} + t_{AND} + t_{OR} + t_{mux} + t_{su}$

Hold time analysis

We want to check that D value at a ff does not change too quickly. so we need to look for the shortest path between any ffs.

This case $Q_1 \rightarrow Q_3$.



if $t_{cQ} \geq t_h$ there is no problem (no hold time violation)

if $t_{cQ} < t_h$ there is a hold-time violation (meaning the ckt. may not be reliable)

$\Longrightarrow$ Note: we assume here that all ff's "see" the clock edge at exactly the same time. More complicated analysis later!