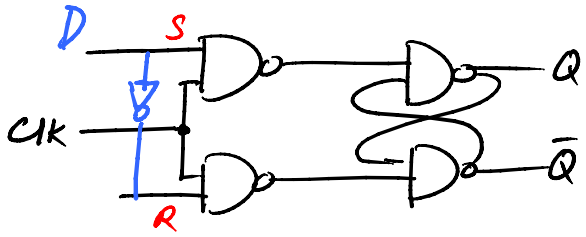
5.3-4 Gated D-latch



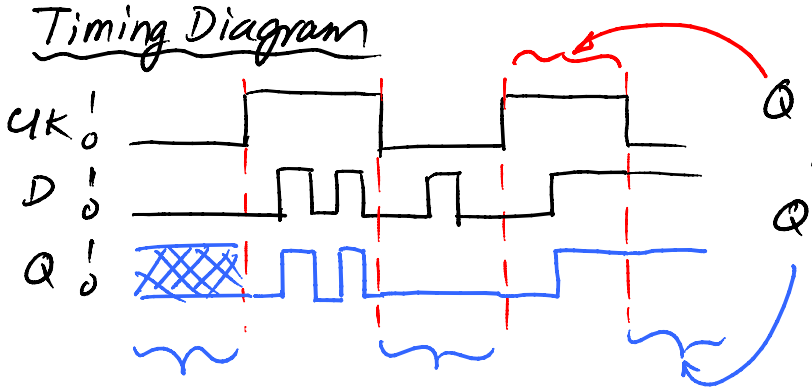
Gated RS latch

Clk	S	R	Q(t+1)
0	x	x	Q(t)
1	0	0	Q(t)
1	0	1	1
1	1	0	0
1	1	1	don't use

Gated D-latch

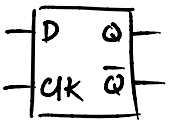
Clk	D	Q(t+1)
0	x	Q(t)
1	0	0
1	1	1

Timing Diagram

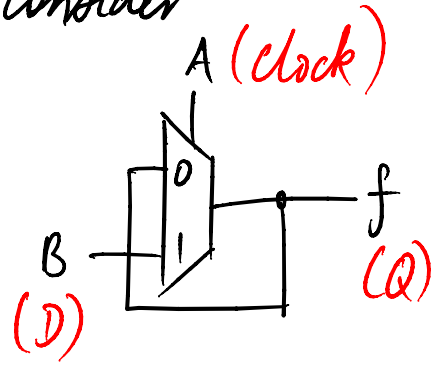


Q follows D when Clk=1

Q holds the old value when Clk=0



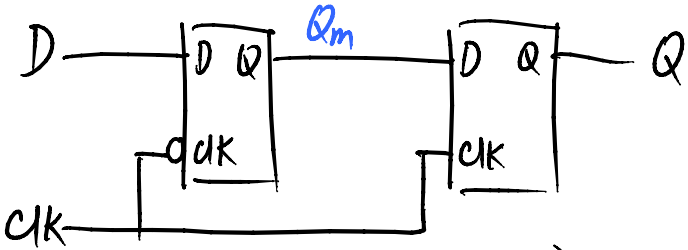
Consider



A	B	f	
0	x	0/1	hold old value (storage)
1	0	0	f follows B
1	1	1	f follows B

Gated D-latch

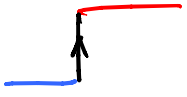
Cascading 2 D-latches



master-slave D-ff

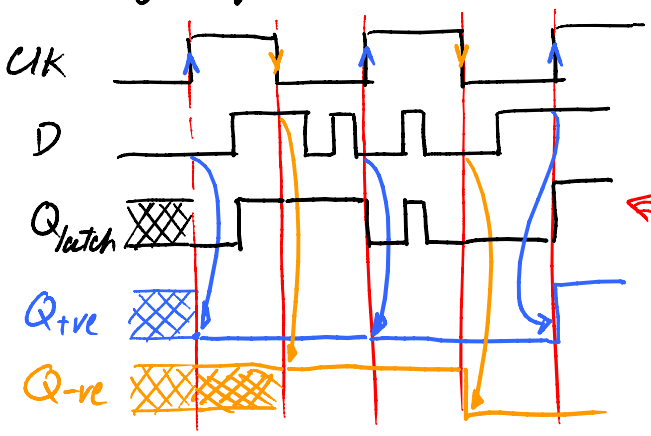
Case 1: Clk=0, Q_m=D, Q=Q_{old}

Case 2: Clk=1, Q_m can't change, Q=Q_m



change takes place only at the +ve edge of the clk

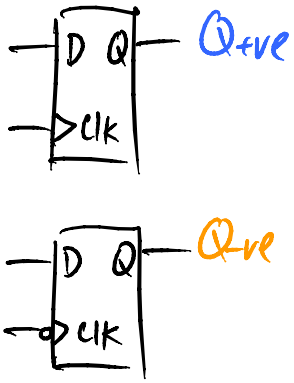
Timing Diagram



latch is level sensitive

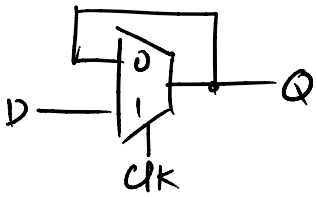
+ve edge triggered D-ff

edge sensitive



-ve edge triggered D-ff

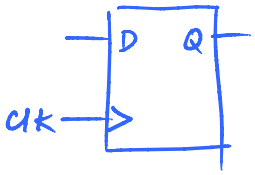
Verilog code for storage elements



Gated D-latch

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

it implies if clk=0, don't make any change



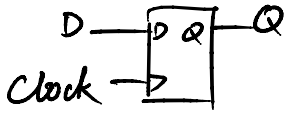
+ve edge triggered D-flip flop

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

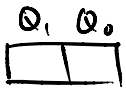
<= non-block statement,

What can you do with a D-ff ?

→ use it as a storage (register)

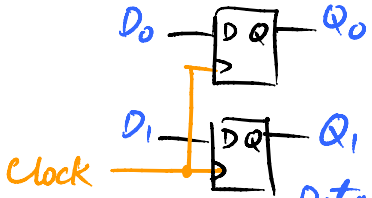


1 bit register



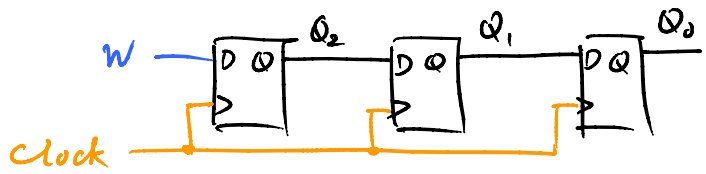
2-bit register

Data to be store



Data stored in Q when Clock

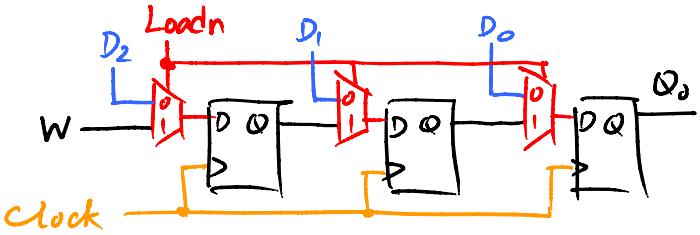
→ use them as a shift register



eg. W = 1 1 0 1

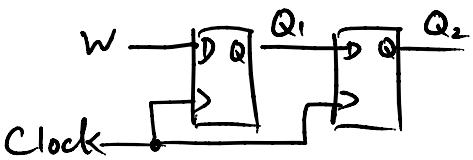
Clock	Q_2	Q_1	Q_0
t_1	1 ①		
t_2	1 ②	1 ①	
t_3	0 ③	1 ②	1 ①
t_4	1 ④	0 ③	1 ②
t_5		1 ④	0 ③
t_6			1 ④

→ shift register w/ parallel load

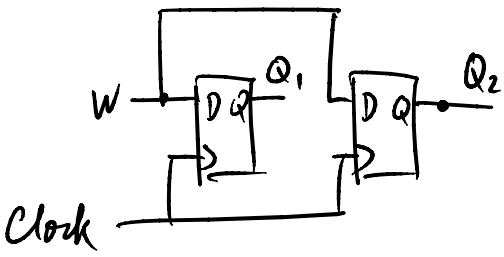


Blocking (=) / non-block (<=) statements

```
module nonBlocking(input W, clock, output Q1, Q2);   (2-bit shift register)
    reg Q1, Q2;
    always@(posedge clock)
      begin
        Q1 <= W;
        Q2 <= Q1;
      end
endmodule
```

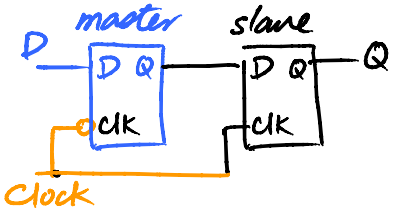


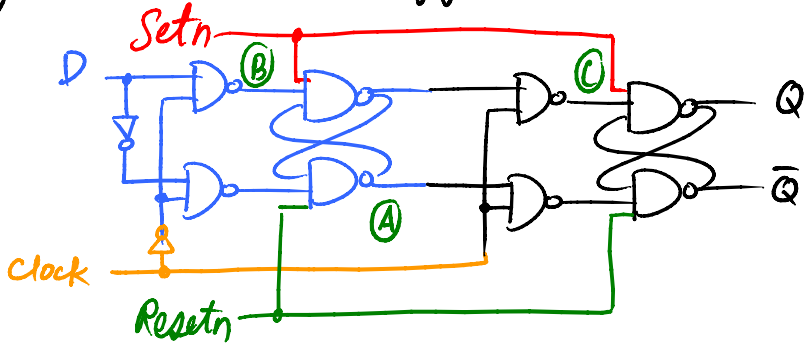

```
module Blocking(input W, clock, output Q1, Q2);
    reg Q1, Q2;
    always@(posedge clock)
      begin
        Q1 = W;
        Q2 = Q1;   (Q2 = W)
      end
endmodule
```



* use non-blocking statements in f-f.
  use blocking statements for combinational logic.

extending the master-slave D-ff





Resetn → active low make output (Q=0)

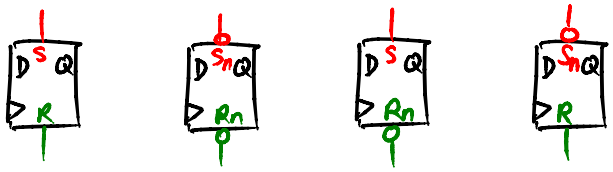
Resetn=0, if clock=0, C=1, Q=0 (reset)

if clock=1, B=1, A=1, C=1, Q=0 (reset)

This is called active low asynchronous reset !

Setn → active low asynchronous set (or preset) ⇒ Q=1

There are different variations




```
module Dff (input D, clock, Resetn, output Q);
   reg Q;
   always@(negedge Resetn , posedge clock)
      if (Resetn == 0)
         Q<= 0;
      else
         Q<= D;
endmodule
```

Resetn inside a sensitivity list means this is an asynchronous reset !

What about Synchronous Resetn?

```
module Dff (input D, clock, Resetn, output Q);
   reg Q;
   always@(posedge clock)
      if ( !Resetn )
         Q<= 0;
      else
         Q<= D;
endmodule
```

to reset Q, first set resetn=0 the change (Q=0) will take place at the next clock

a possible circuit to implement a Synchronous reset