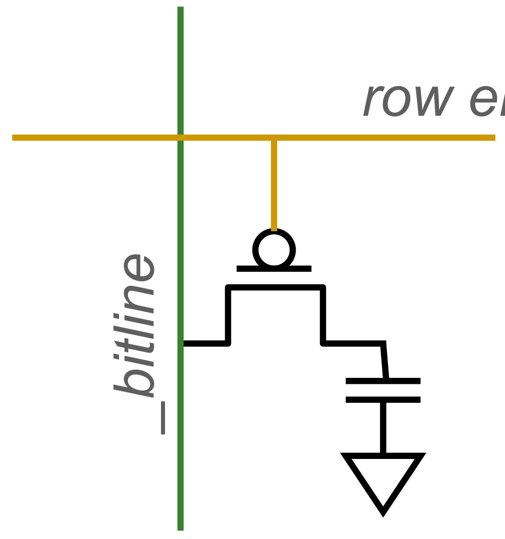
# EECE 2322: Fundamentals of Digital Design and Computer Organization

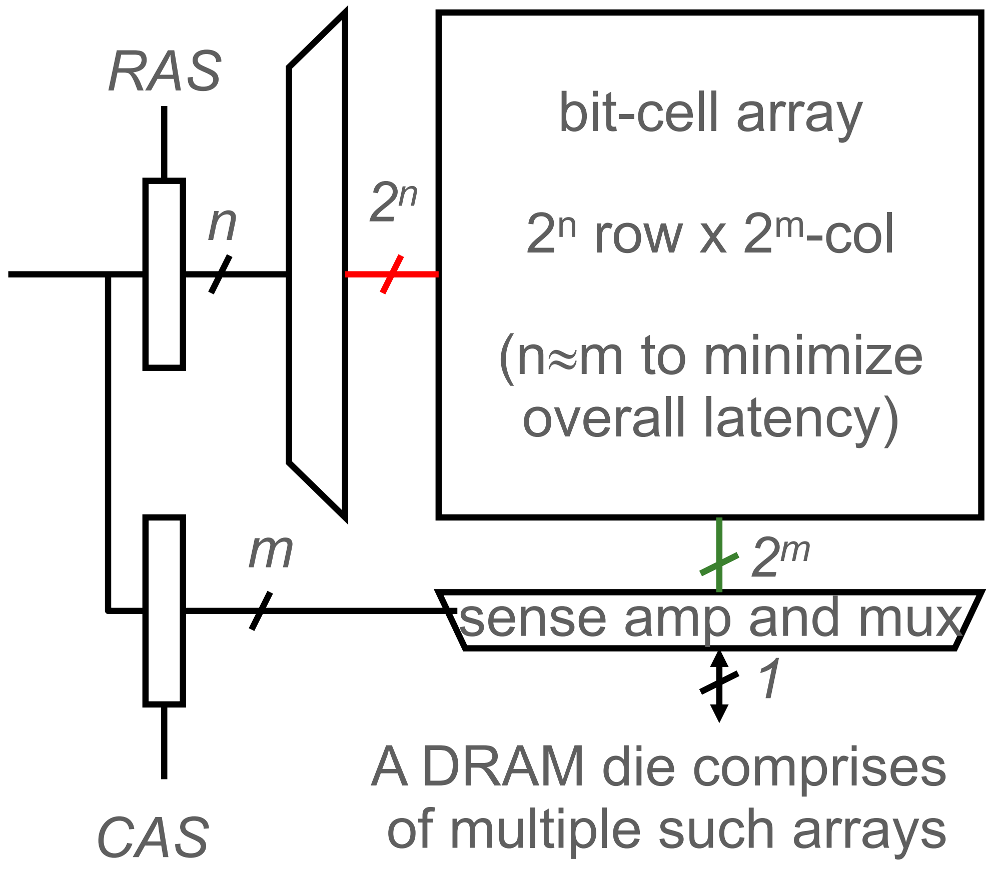
## Lecture 4_2: Sequential Circuit and ALU

Xiaolin Xu
Department of ECE
Northeastern University

# DRAM (Dynamic Random Access Memory)

- Bits stored as charges on node capacitance (non-restorative)
  - bit cell loses charge when read
  - bit cell loses charge over time
- Read Sequence
  - 1~3 same as SRAM
  - 4. a "flip-flopping" sense amp amplifies and regenerates the bitline, data bit is mux'ed out
  - 5. precharge all bitlines
- **Refresh**: A DRAM controller must periodically read each row within the allowed refresh time (10s of ms) such that charge is restored

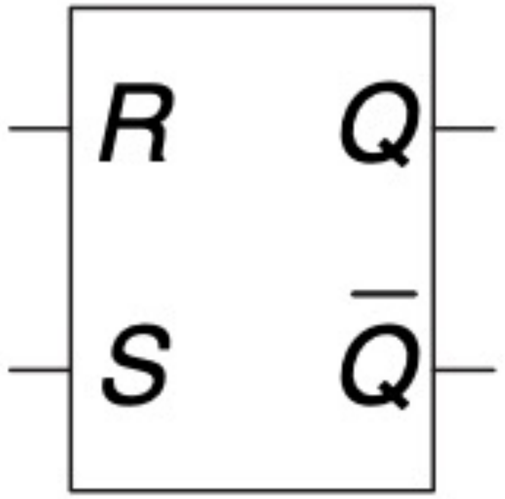




A DRAM die comprises of multiple such arrays

# Sequential Logic (2): SR Latch

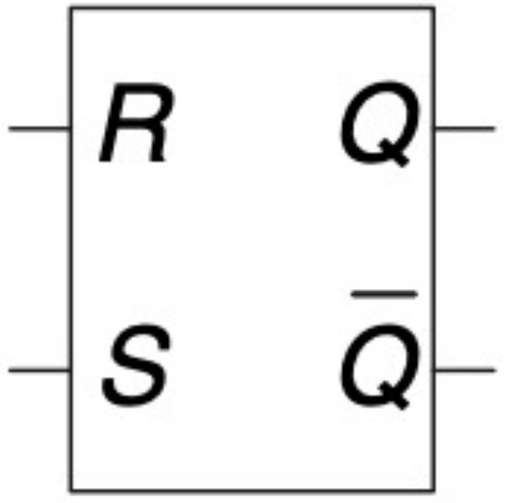
- SR latch
  - Cross-coupled NOR gates
  - S: Set
  - R: Reset



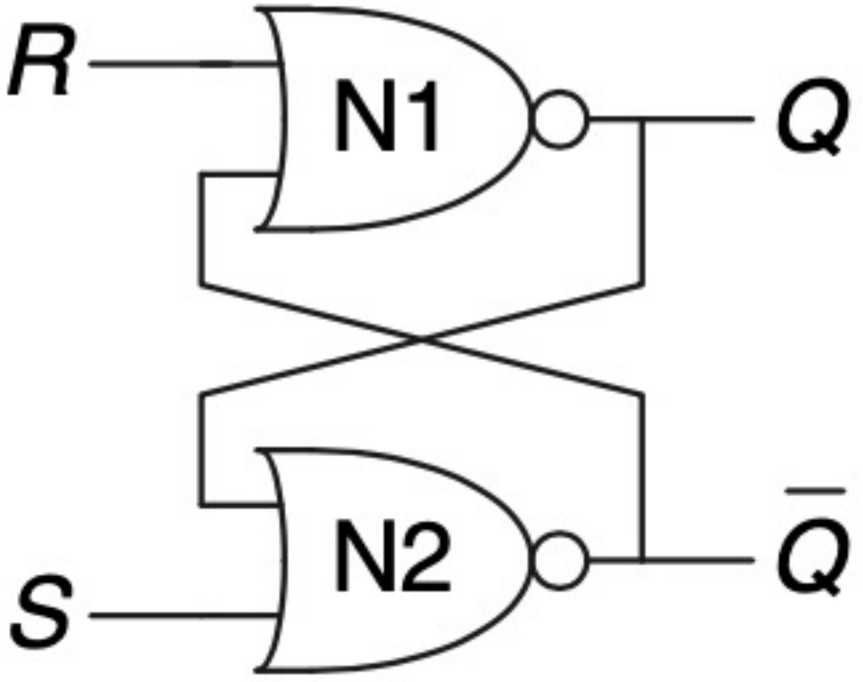
SR latch symbol

## Sequential Logic (2): SR Latch

- SR latch
  - Cross-coupled NOR gates
  - S: Set
  - R: Reset



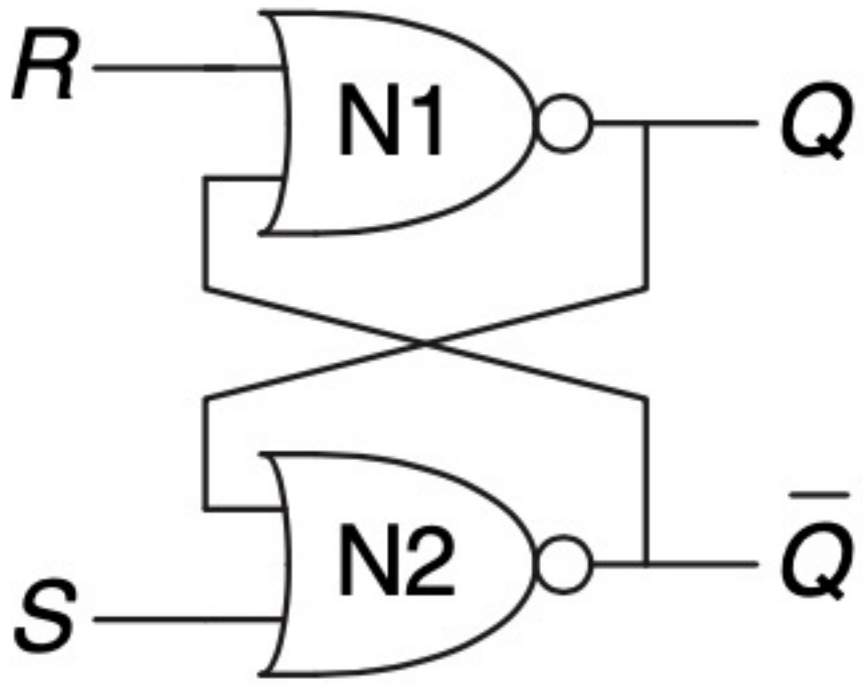
SR latch symbol



# Sequential Logic (2): SR Latch

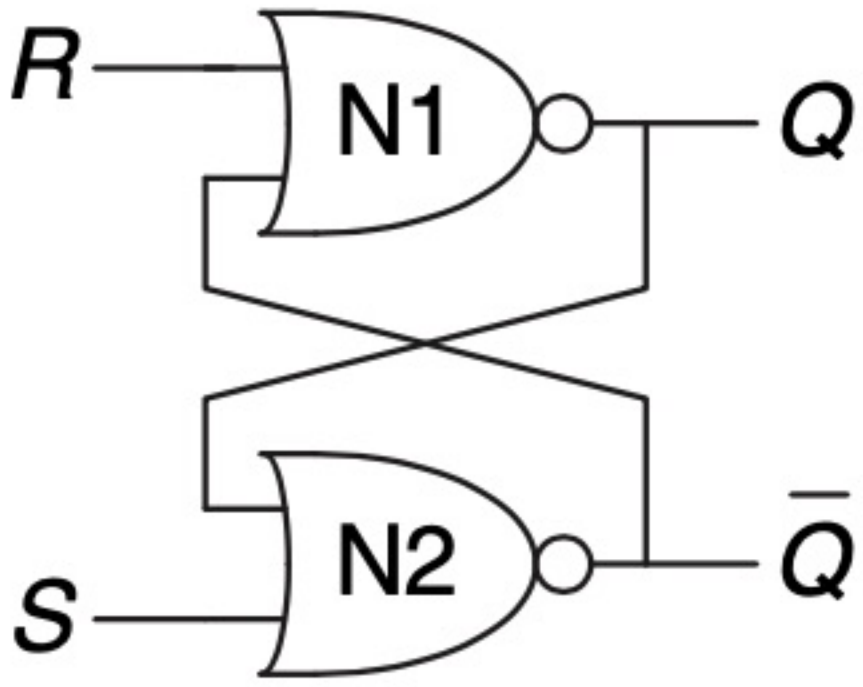
- SR latch
  - S: Set
  - R: Reset
- Truth table of SR latch

| R | S | Q | ~Q |
|---|---|---|---|
| 1 | 0 | | |
| 0 | 1 | | |
| 1 | 1 | | |
| 0 | 0 | | |
| | | | |



# Sequential Logic (2): SR Latch

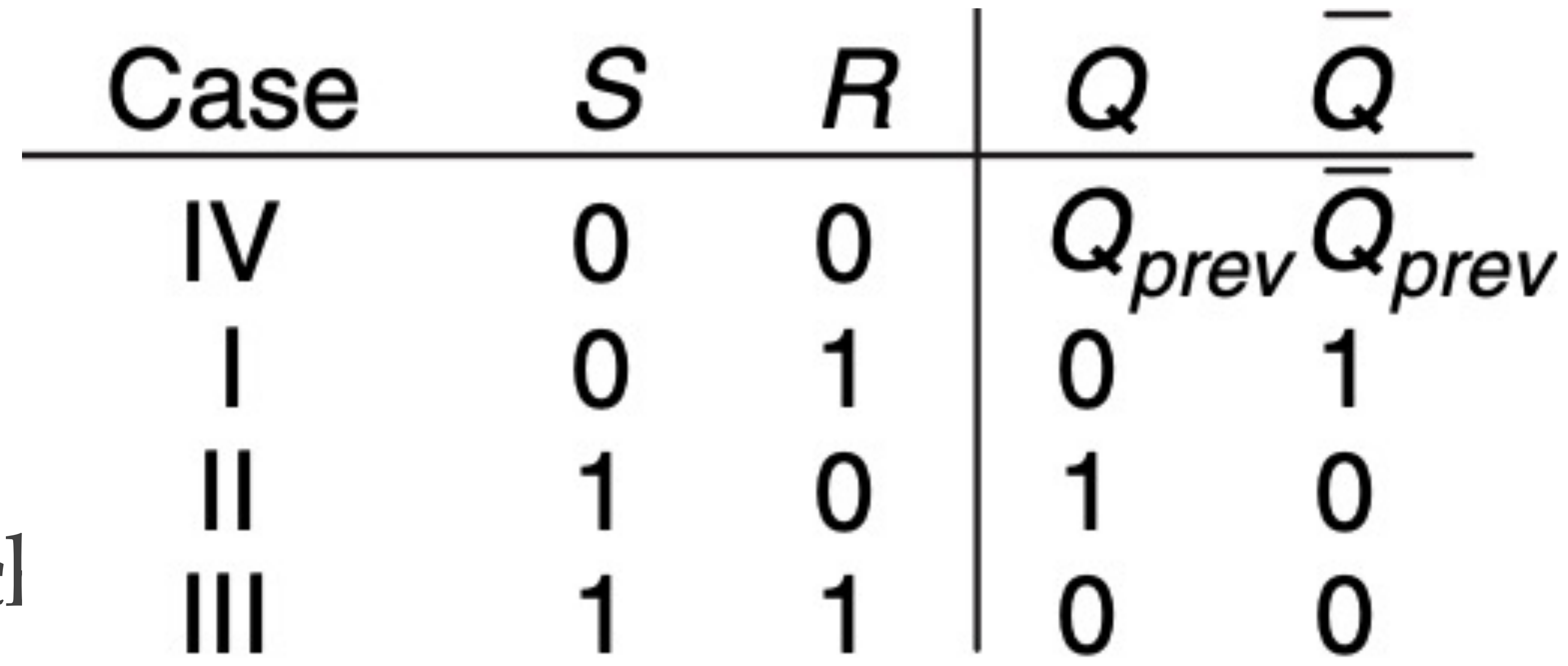
- SR latch
    - S: Set
    - R: Reset
- Truth table of SR latch



| R | S | Q | ~Q |
|---|---|---|---|
| 1 | 0 | | |
| 0 | 1 | | |
| 1 | 1 | | |
| 0 | 0 | | |
| | | | |

# Sequential Logic (2): SR Latch

- SR latch
  - S: Set
  - R: Reset
- Truth table of SR latch

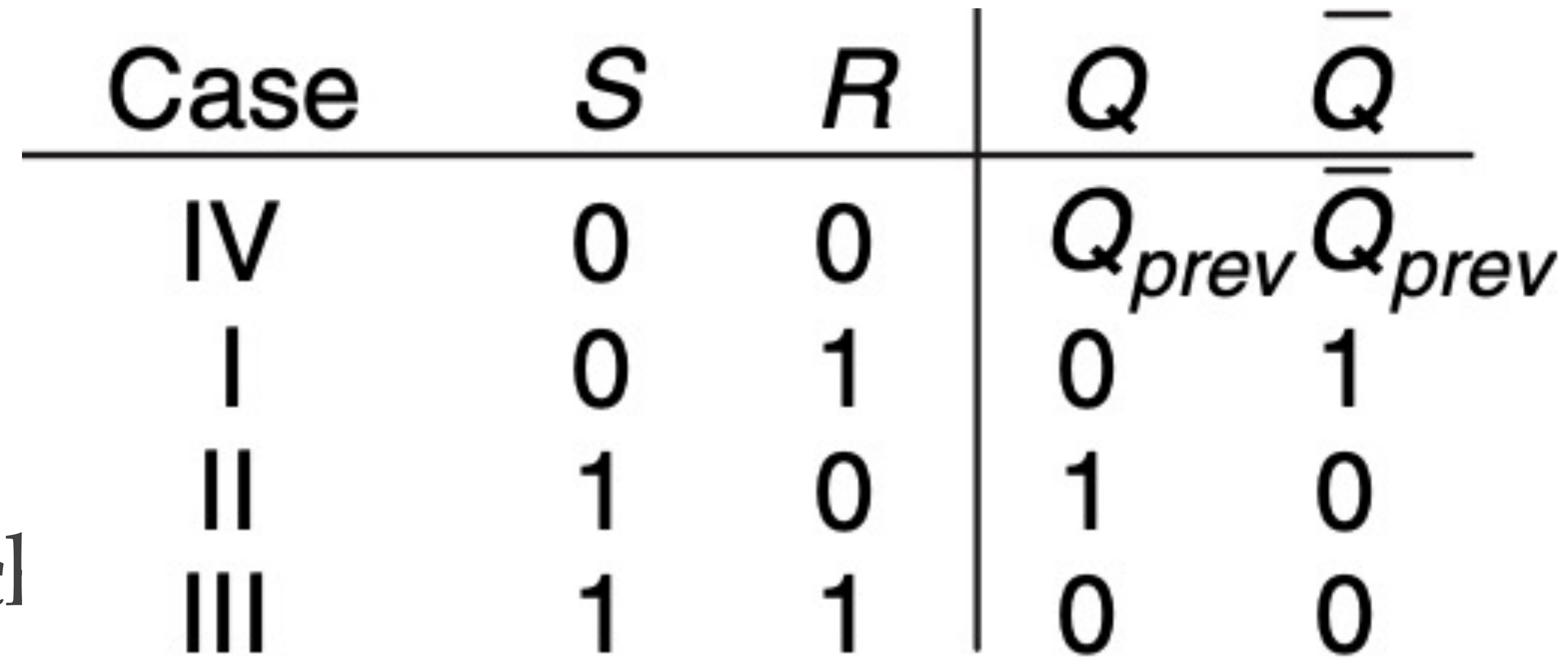


| Case | $S$ | $R$ | $Q$ | $\overline{Q}$ |
|---|---|---|---|---|
| IV | 0 | 0 | $Q_{prev}$ | $\overline{Q}_{prev}$ |
| I | 0 | 1 | 0 | 1 |
| II | 1 | 0 | 1 | 0 |
| III | 1 | 1 | 0 | 0 |

| R | S | Q | ~Q |
|---|---|---|---|
| 1 | 0 | | |
| 0 | 1 | | |
| 1 | 1 | | |
| 0 | 0 | | |
| | | | |

# Sequential Logic (2): SR Latch

- SR latch
  - S: Set
  - R: Reset
- Truth table of SR latcl



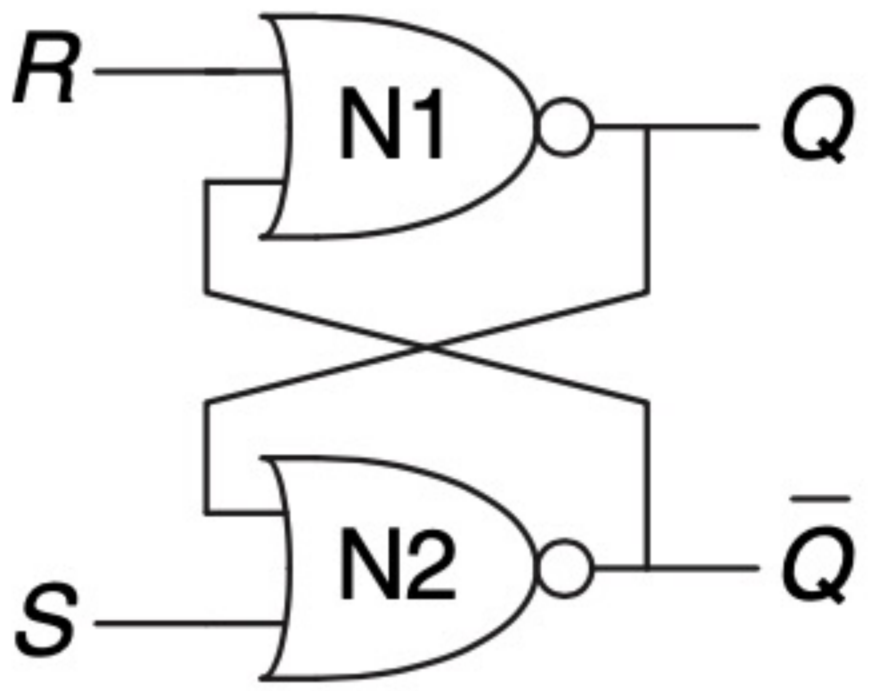
| Case | $S$ | $R$ | $Q$ | $\overline{Q}$ |
|---|---|---|---|---|
| IV | 0 | 0 | $Q_{prev}$ | $\overline{Q}_{prev}$ |
| I | 0 | 1 | 0 | 1 |
| II | 1 | 0 | 1 | 0 |
| III | 1 | 1 | 0 | 0 |

- **Asserting both S and R simultaneously doesn't make sense, i.e., the latch should be set and reset at the same time, which is impossible**

| R | S | Q | ~Q |
|---|---|---|---|
| 1 | 0 | | |
| 0 | 1 | | |
| 1 | 1 | | |
| 0 | 0 | | |
| | | | |

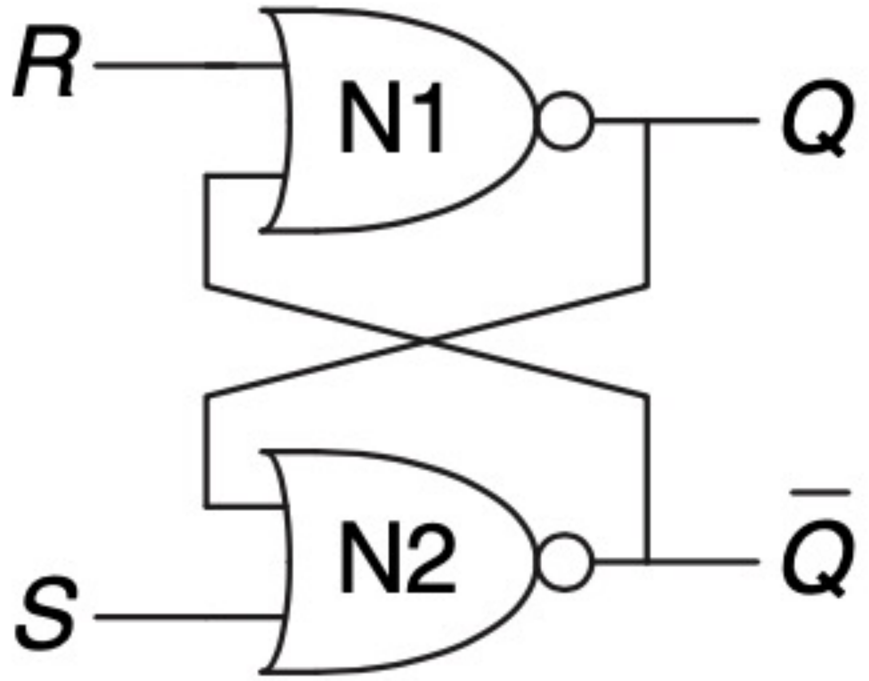
# Problem with SR Latch

- ***SR latch behaves strangely when both S and R are simultaneously asserted***
- How to solve this issue?
  - Two complementary inputs forever!
  - How?
  - Using an inverter



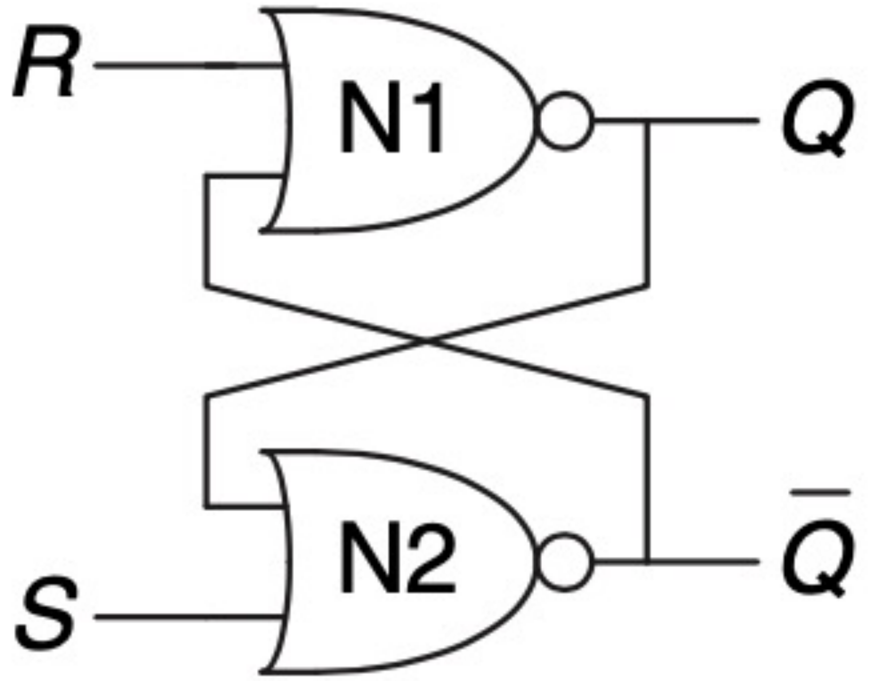
# Problem with SR Latch

- ***SR latch behaves strangely when both S and R are simultaneously asserted***
- How to solve this issue?
  - Two complementary inputs forever!
  - How?
  - Using an inverter
- Truth table



# Problem with SR Latch

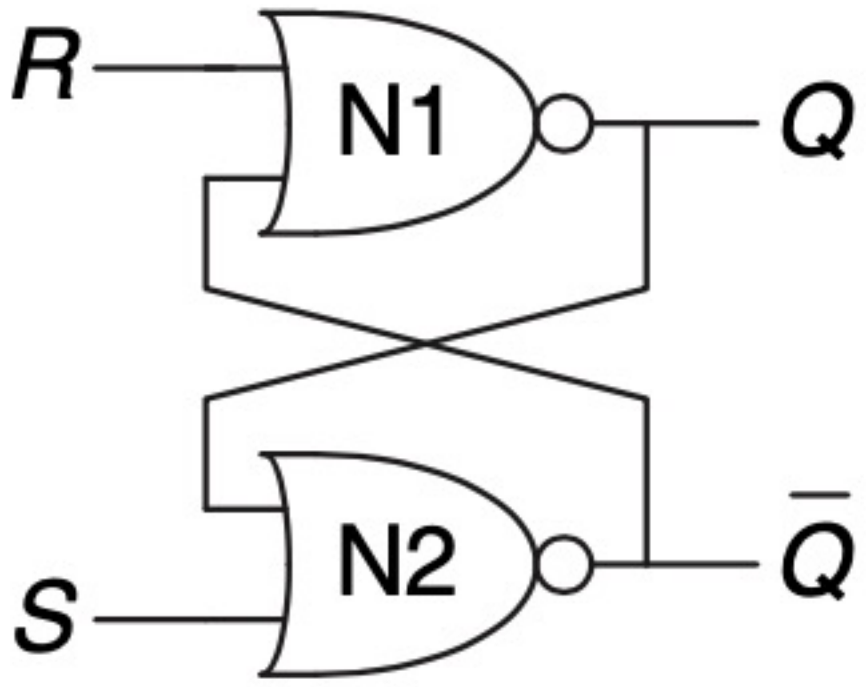
- ***SR latch behaves strangely when both S and R are simultaneously asserted***
- How to solve this issue?
  - Two complementary inputs forever!
  - How?
  - Using an inverter



- Truth table

| D | Q | ~Q |
|---|---|---|
| 0 | | |
| 1 | | |

## SR Latch in Verilog

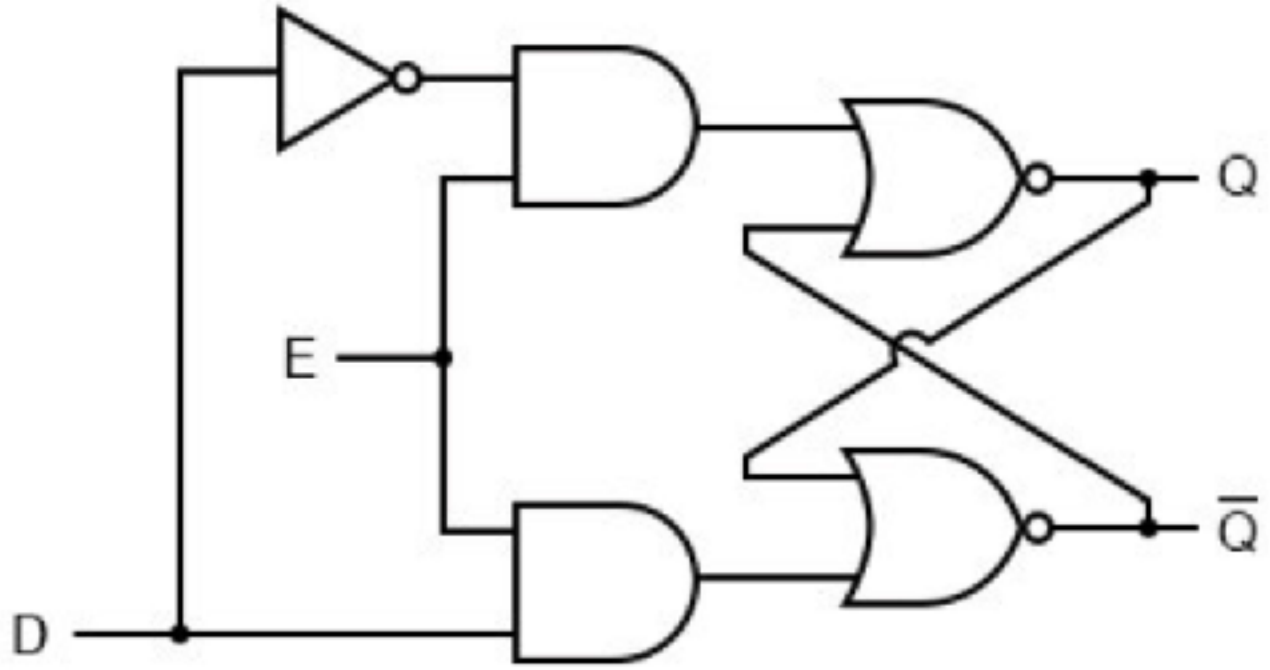


```
module sr_latch(
    input wire S, R,
    output wire Q, Q_not);

    assign Q     = ~(R | Q_not);
    assign Q_not = ~(S | Q);
endmodule
```

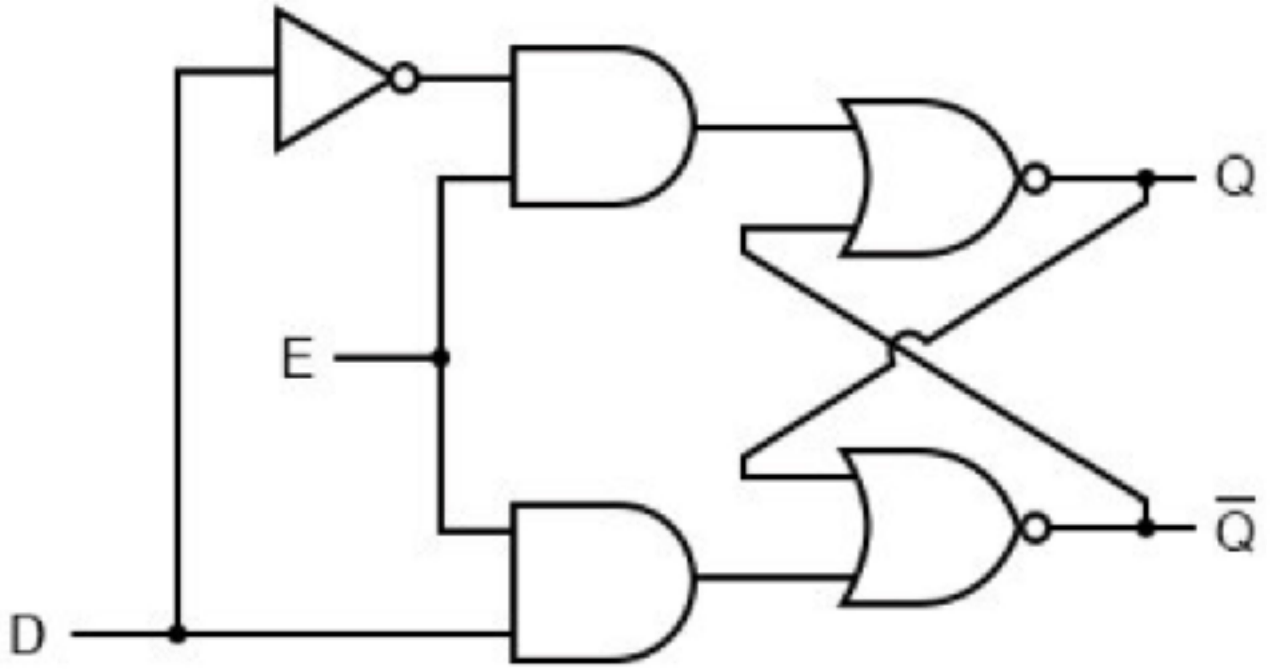
# Sequential Logic (2): D Latch

- ***The current design CANNOT latch the D value***
- How to solve this issue?
  - Adding an ***Enable***



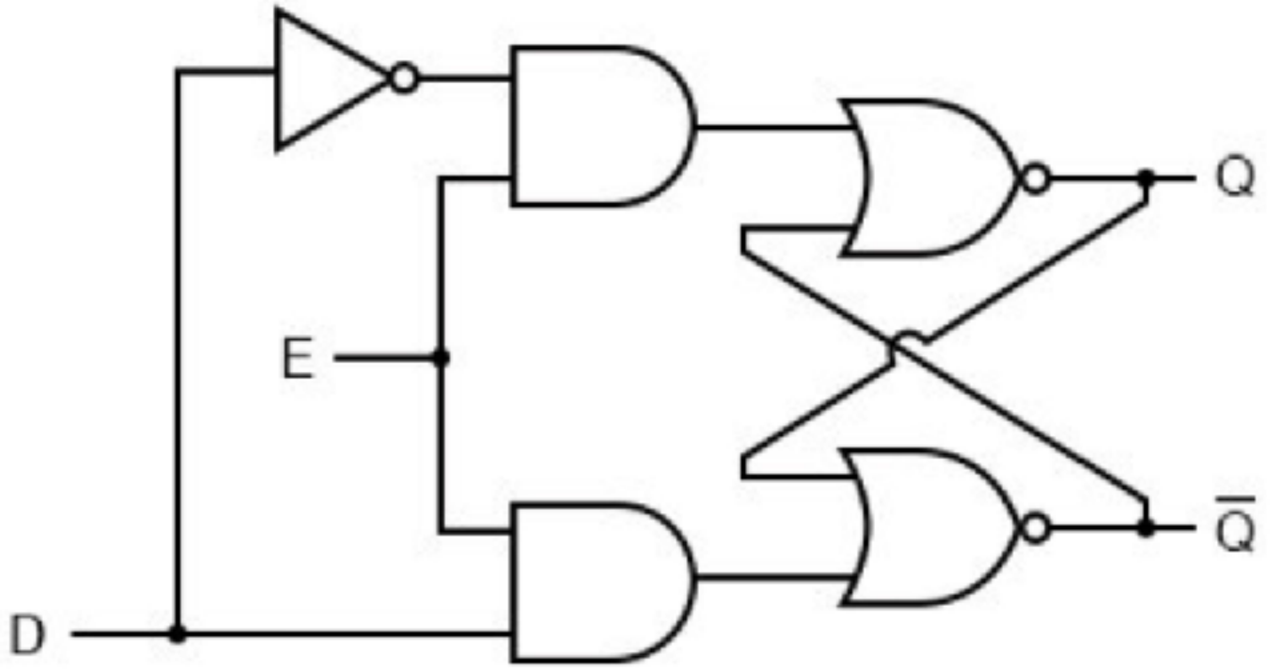
## Sequential Logic (2): D Latch

- ***The current design CANNOT latch the D value***
- How to solve this issue?
  - Adding an ***Enable***
- Truth table



# Sequential Logic (2): D Latch

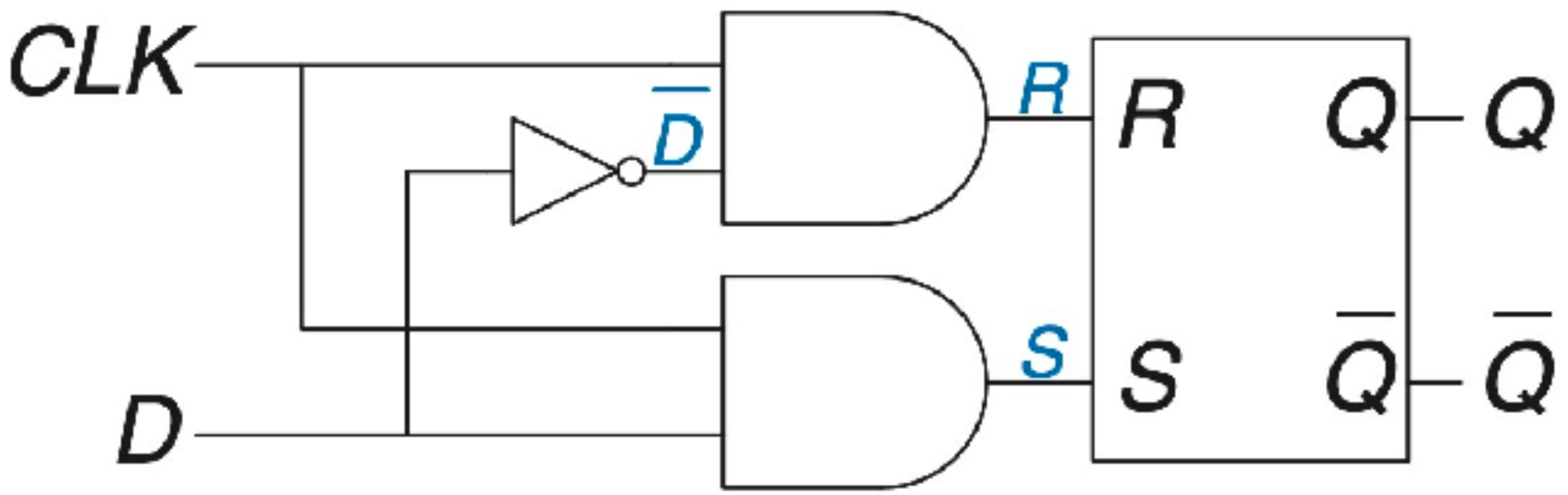
- ***The current design CANNOT latch the D value***
- How to solve this issue?
  - Adding an ***Enable***
- Truth table

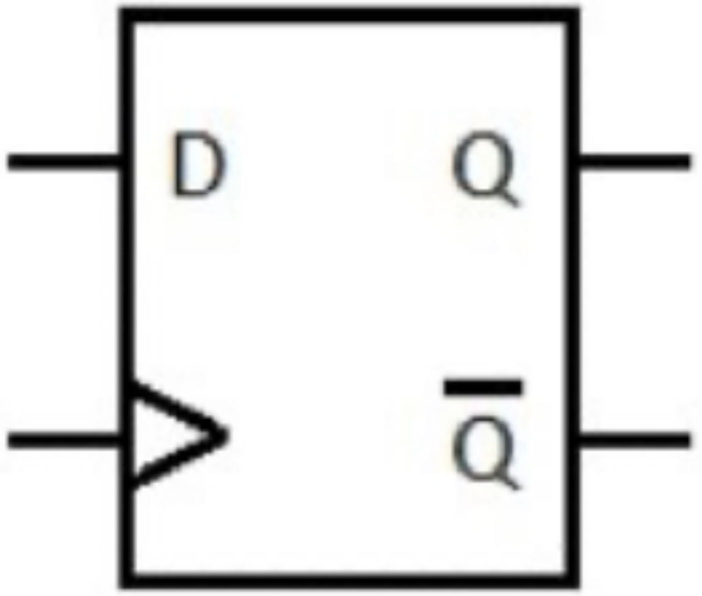


| E | D | Q | ~Q |
|---|---|---|---|
| 1 | 0 | | |
| 0 | 1 | | |
| 0 | 1 | | |
| 0 | 0 | | |

## Sequential Logic (2): D Latch

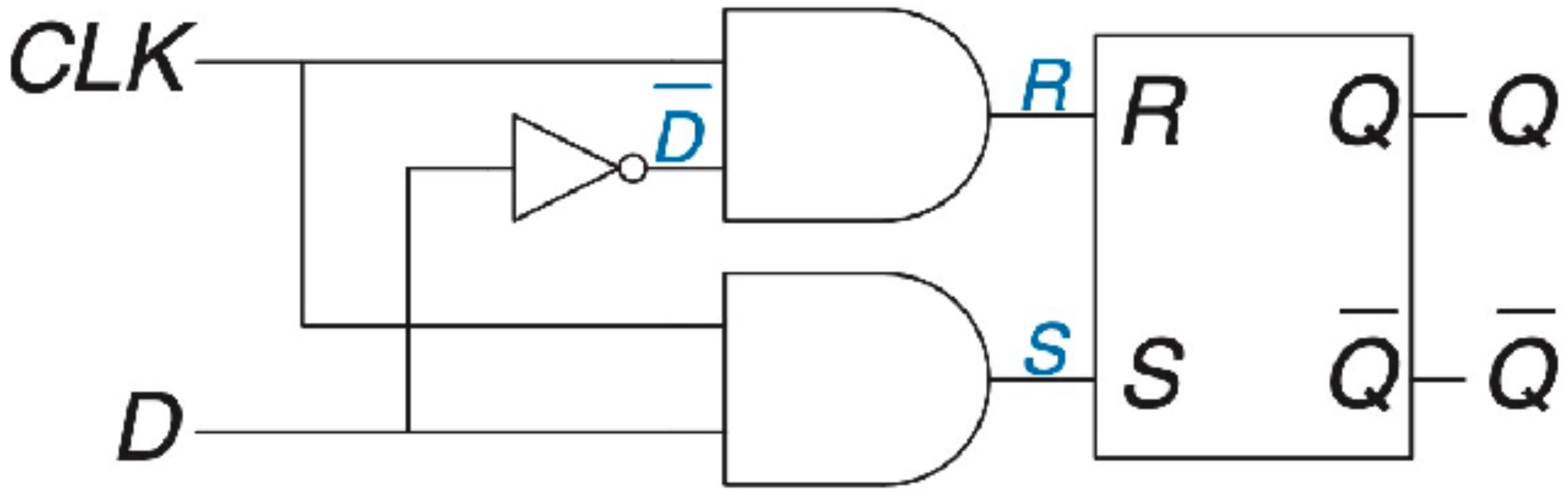
- Replacing **Enable** with **CLK** (clock)

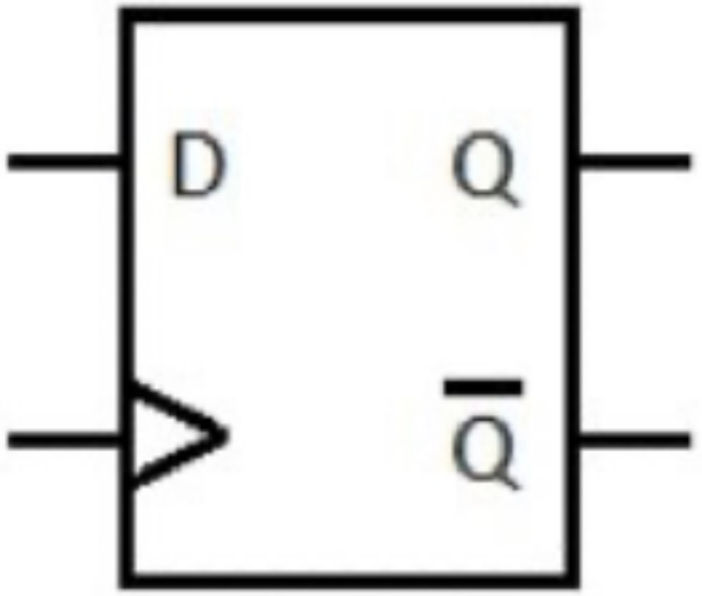




## Sequential Logic (2): D Latch

- Replacing **Enable** with **CLK** (clock)





| CLK | D | $\overline{D}$ | S | R | Q | $\overline{Q}$ |
|---|---|---|---|---|---|---|
| 0 | X | $\overline{X}$ | 0 | 0 | $Q_{prev}$ | $\overline{Q}_{prev}$ |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 |

## Level-Sensitive and Edge-Sensitive

- Left: will block until there is a change in the value of a or b
- Right: will block until clk transitions from 0 to 1.

```
always  @ (a or b or sel)
begin
   y = 0;
   if (sel == 0) begin
      y = a;
   end else begin
      y = b;
   end
end
```

```
always  @ (posedge clk )
if (reset == 0) begin
   y <= 0;
end else if (sel == 0) begin
   y <= a;
end else begin
   y <= b;
end
```

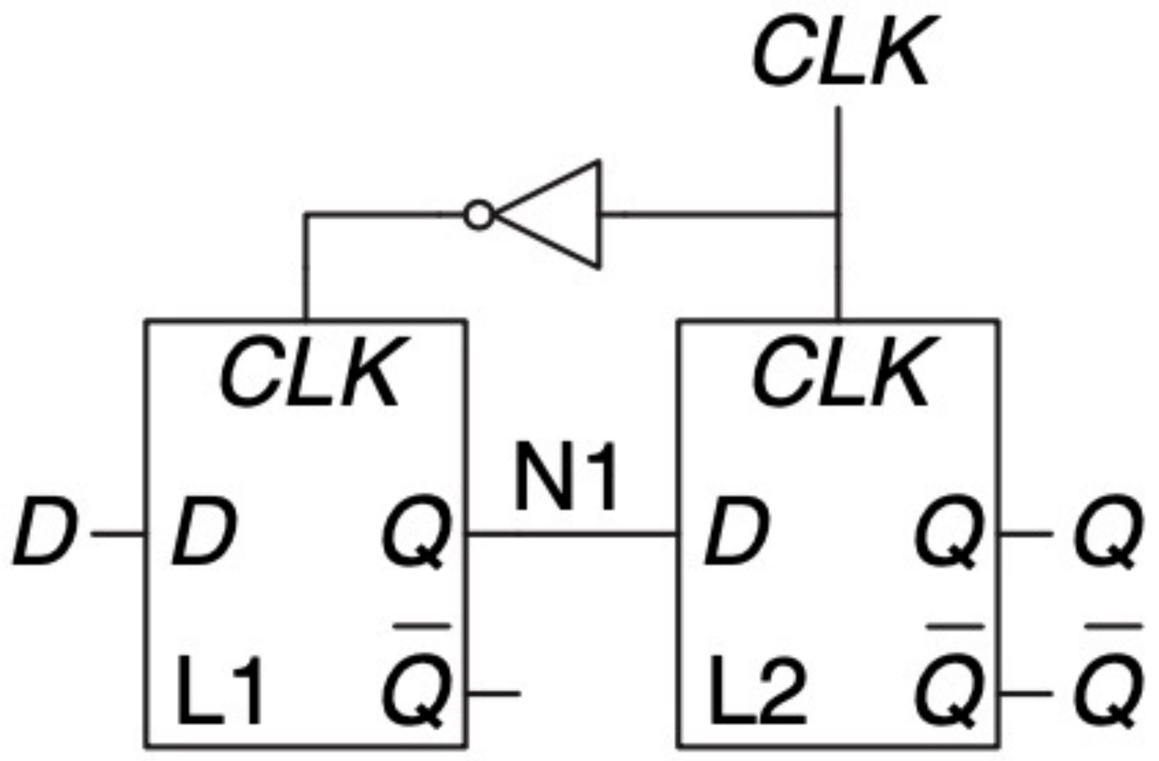
## D Latch in Verilog

- module dlatchmod(e, d, q);
    - input e;
    - input d;
    - output q;
    - reg q;
    - always @(e or d)
        - begin
        - if (e)
        - q<=d;
        - end
- endmodule

# Sequential Logic (3): D Flip-flop

- Sequential circuit is edge-sensitive!
- Truth table
  - CLK = 0, first transparent
  - CLK = 1, second transparent

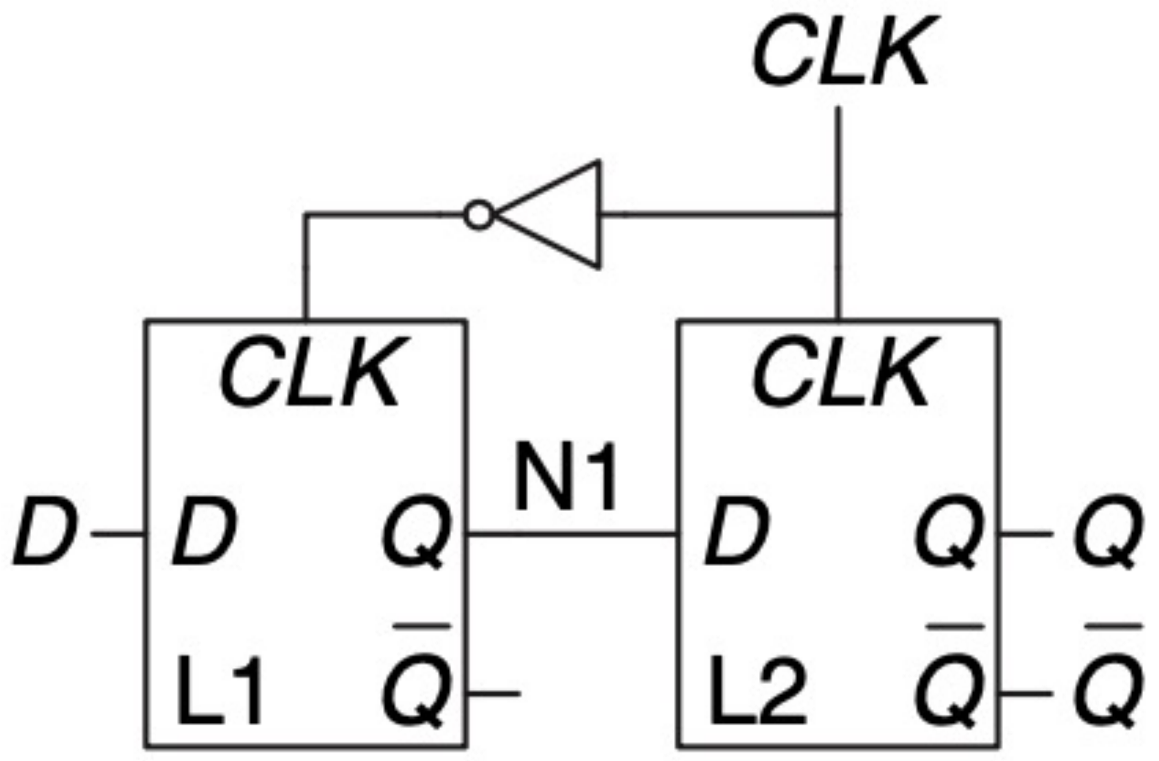
| CLK | D | N1 | Q | ~Q |
|---|---|---|---|---|
| 0 | | | | |
| 1 | | | | |
| 0 | | | | |
| 1 | | | | |



# Sequential Logic (3): D Flip-flop

- Sequential circuit is edge-sensitive!
- Truth table
  - CLK = 0, first transparent
  - CLK = 1, second transparent

| CLK | D | N1 | Q | ~Q |
|---|---|---|---|---|
| 0 | | | | |
| 1 | | | | |
| 0 | | | | |
| 1 | | | | |



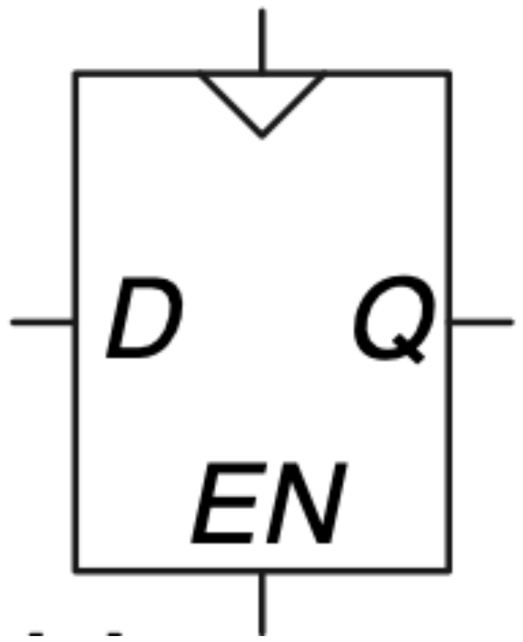
**A D flip-flop copies D to Q on the rising edge of the clock, and remembers its state at all other times**

## D Flip-flop in Verilog

```
module RisingEdge_DFlipFlop(D,clk,Q);
input D; // Data input
input clk; // clock input
output Q; // output Q
always @(posedge clk)
begin
 Q <= D;
end
endmodule
```

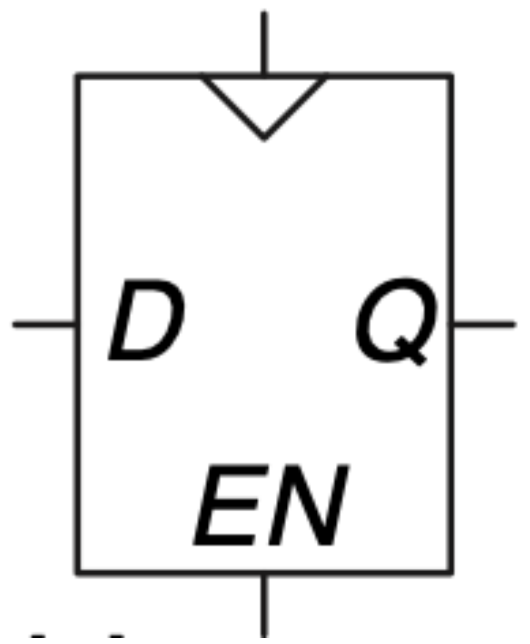
# Sequential Logic (4): Enabled Flip-flop

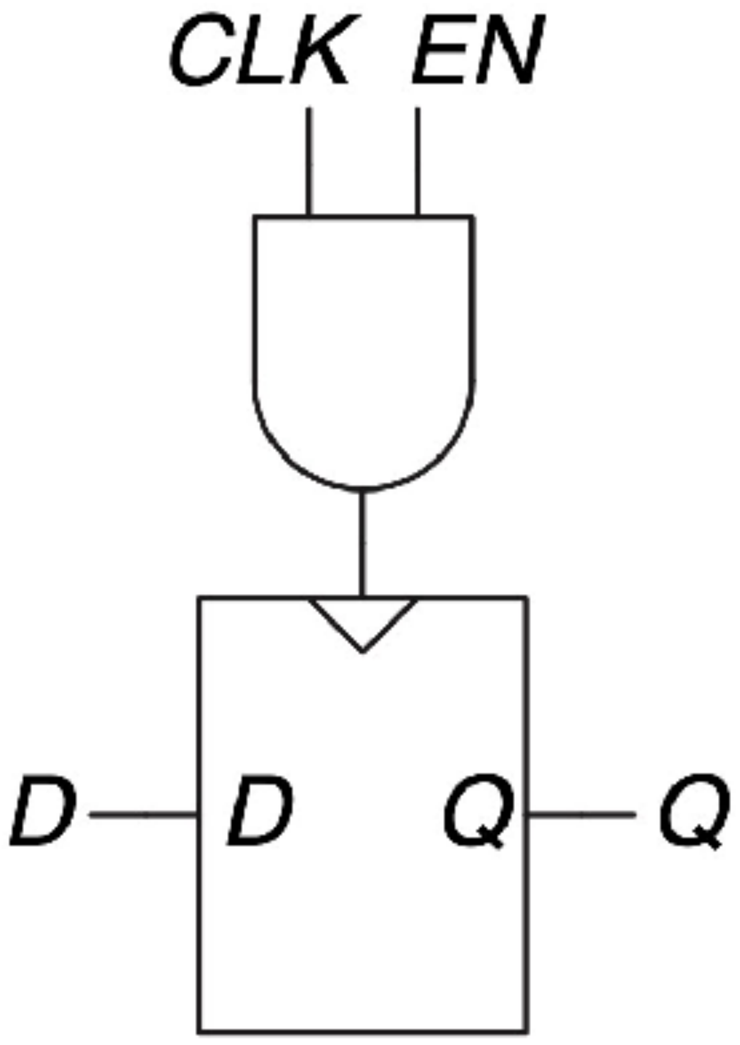
- How to implement?
- Lets analyze what we want
  - EN is true, ordinary D Flip-flop
  - EN is false, value retained



# Sequential Logic (4): Enabled Flip-flop

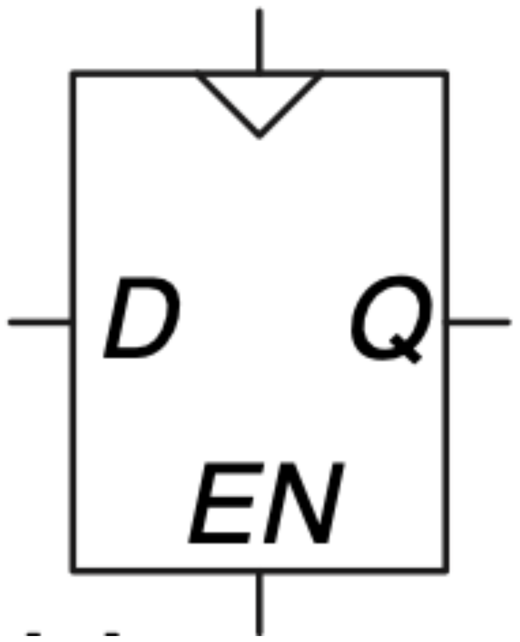
- How to implement?
- Lets analyze what we want
  - EN is true, ordinary D Flip-flop
  - EN is false, value retained

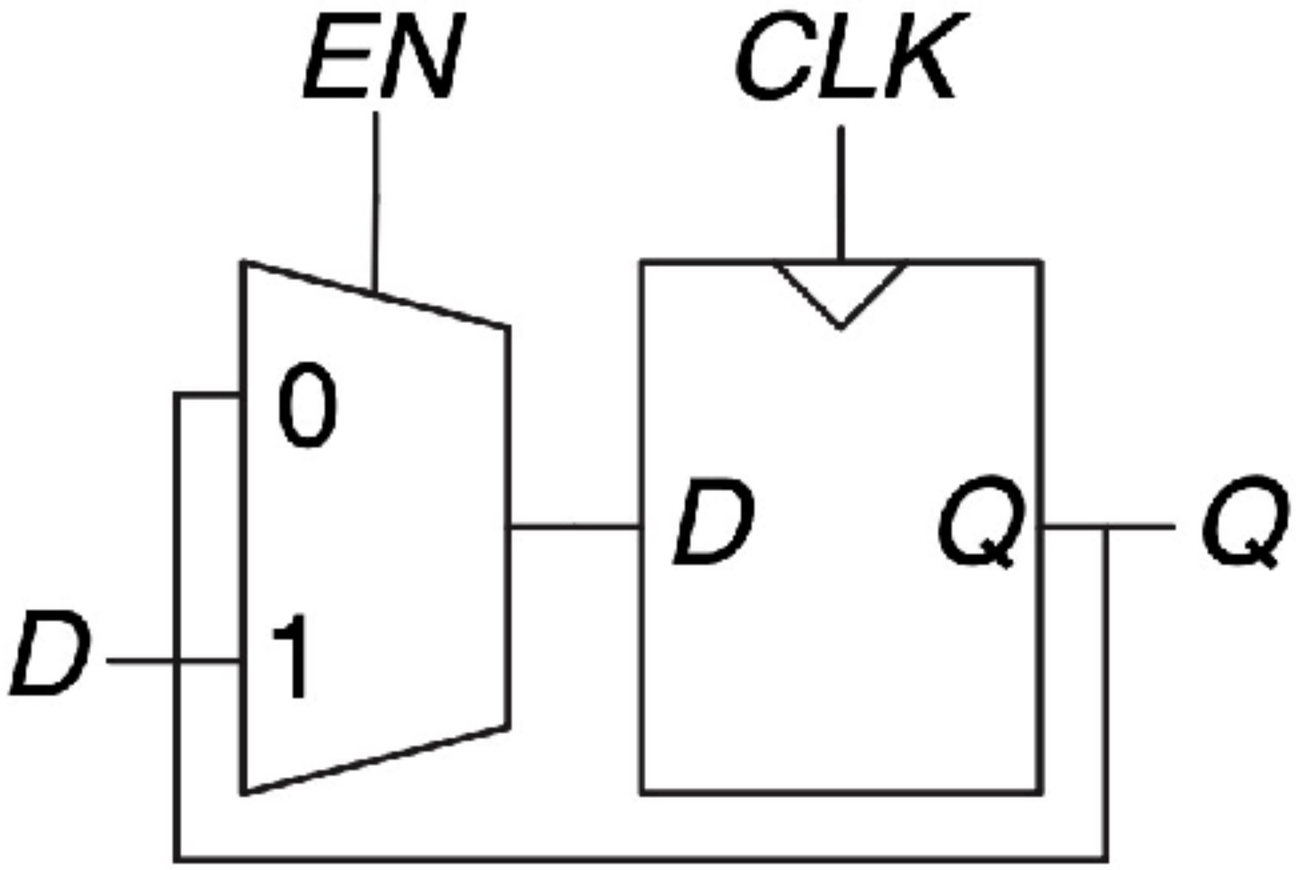




# Sequential Logic (4): Enabled Flip-flop

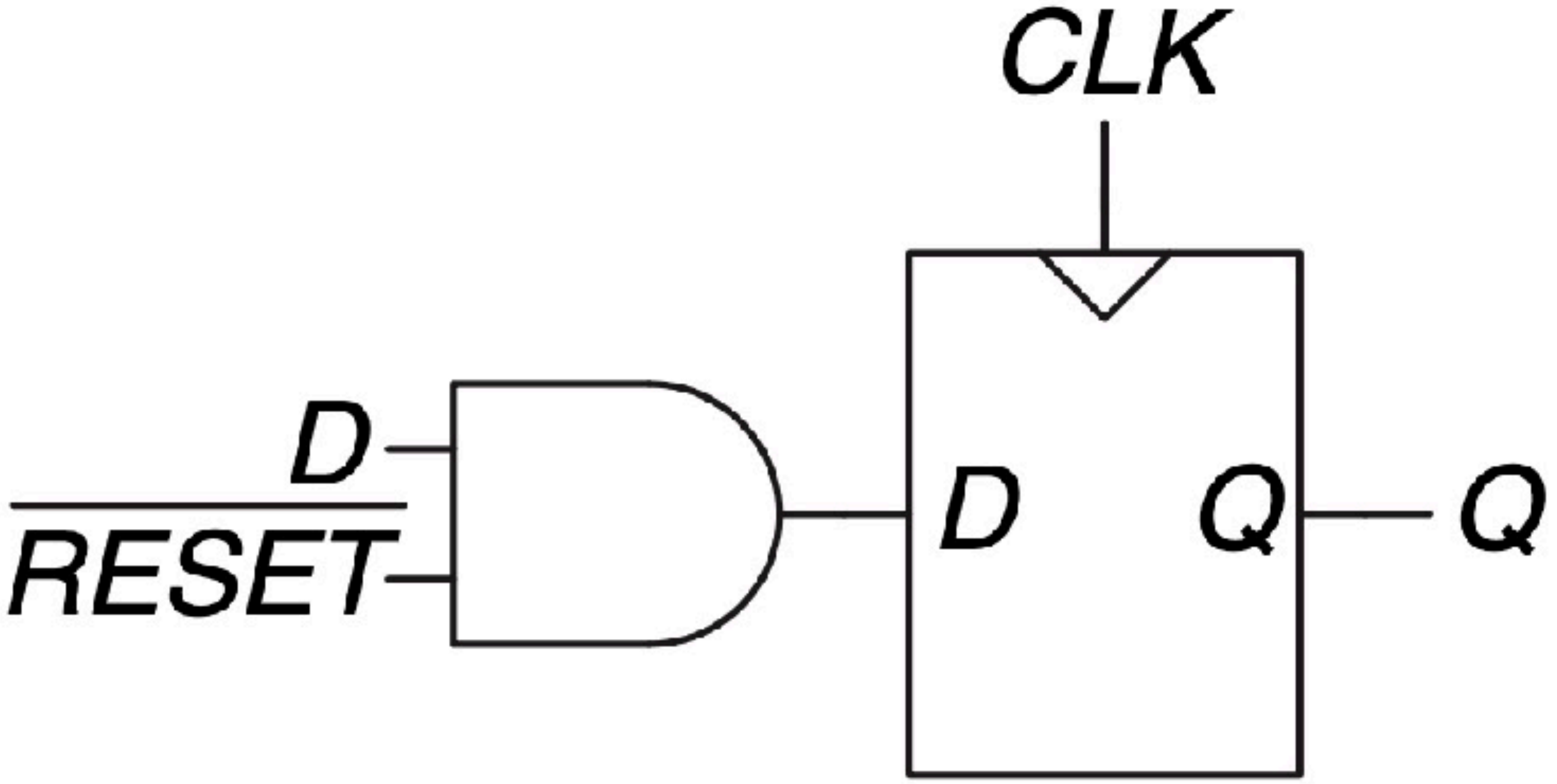
- How to implement?
- Lets analyze what we want
  - EN is true, ordinary D Flip-flop
  - EN is false, value retained





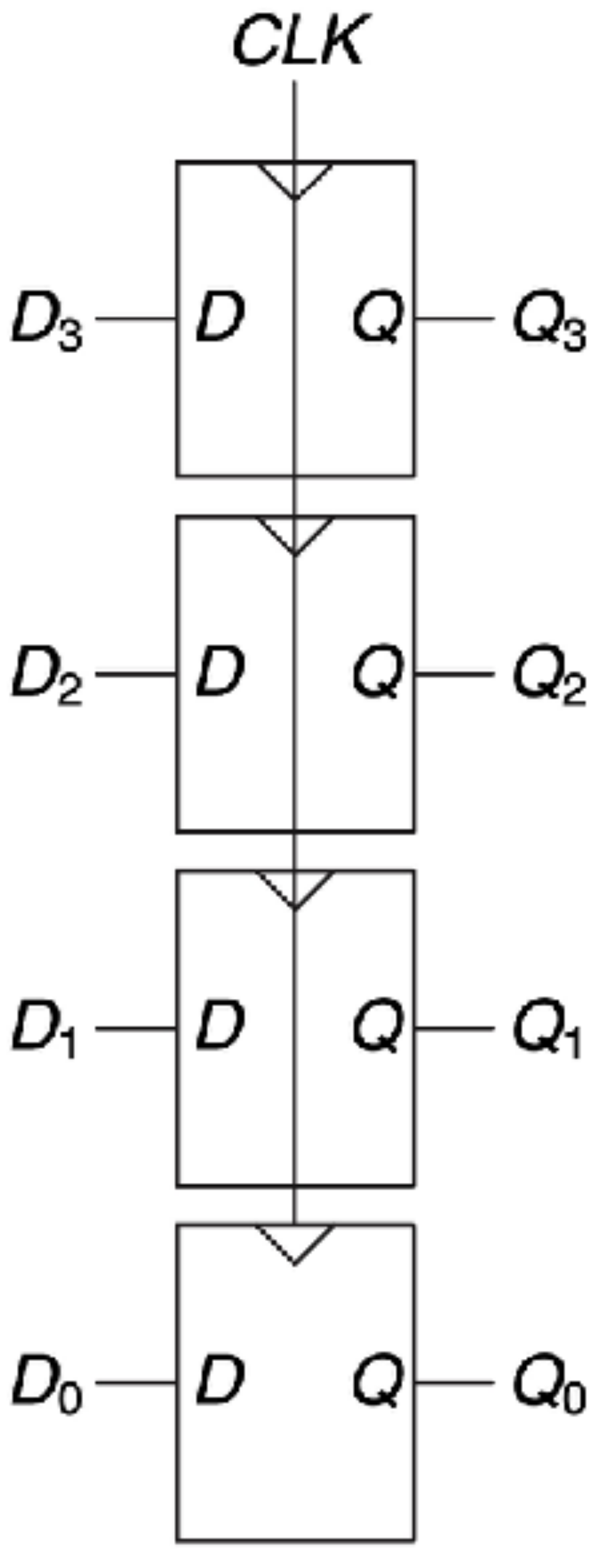
## Sequential Logic (5): Resettable Flip-flop

- Active-low reset signal



# Sequential Logic (6): Register

- Consists of N flip-flops
- Share a common CLK
    - All inputs updated at the same time
- 4-bit example



# Register in Verilog

- *// positive edge-triggered 4-bit register*
- module reg4 (CLK,Q,D,RST);
- input [3:0] D;
- input CLK, RST;
- output [3:0] Q;
- reg [3:0] Q;
- always @ (RST, posedge CLK)
- if (RST) Q <= 0; else Q <= D;
- endmodule / / reg4