

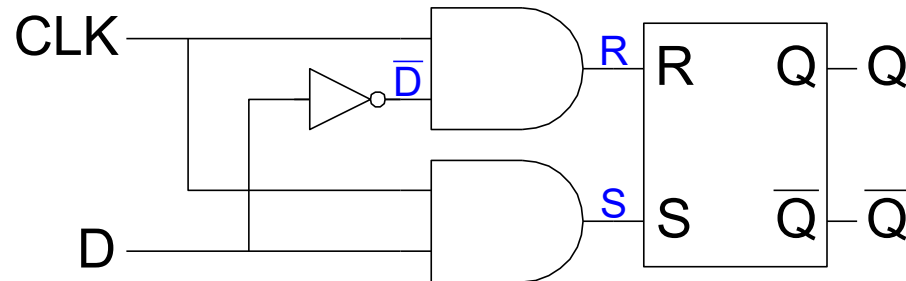


CSE140L: Components and Design Techniques for Digital Systems

Registers, shift registers, counters

SOURCE: <http://www.pitt.edu/~kmram/132/lectures/registers+counters.pdf>

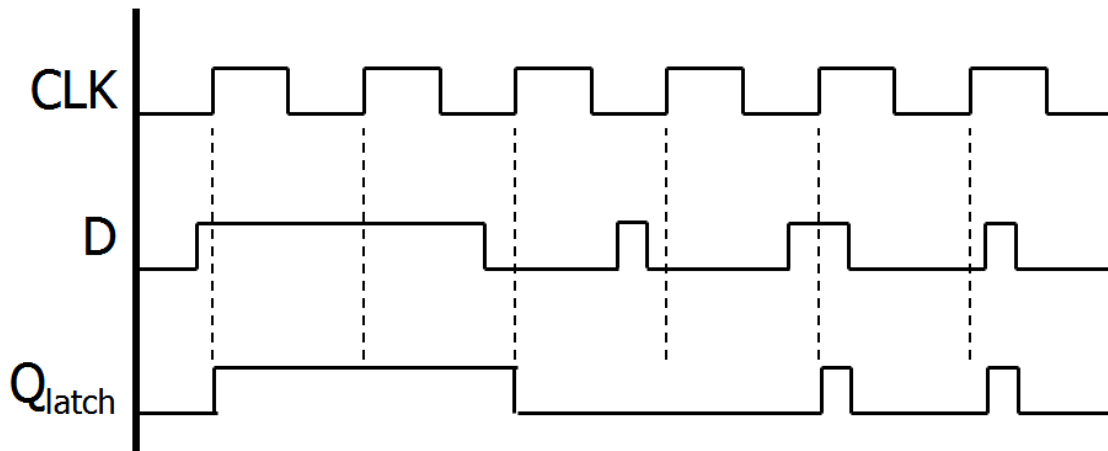
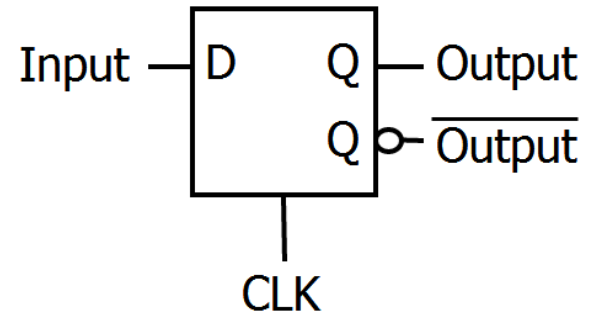
D Latch Truth Table



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

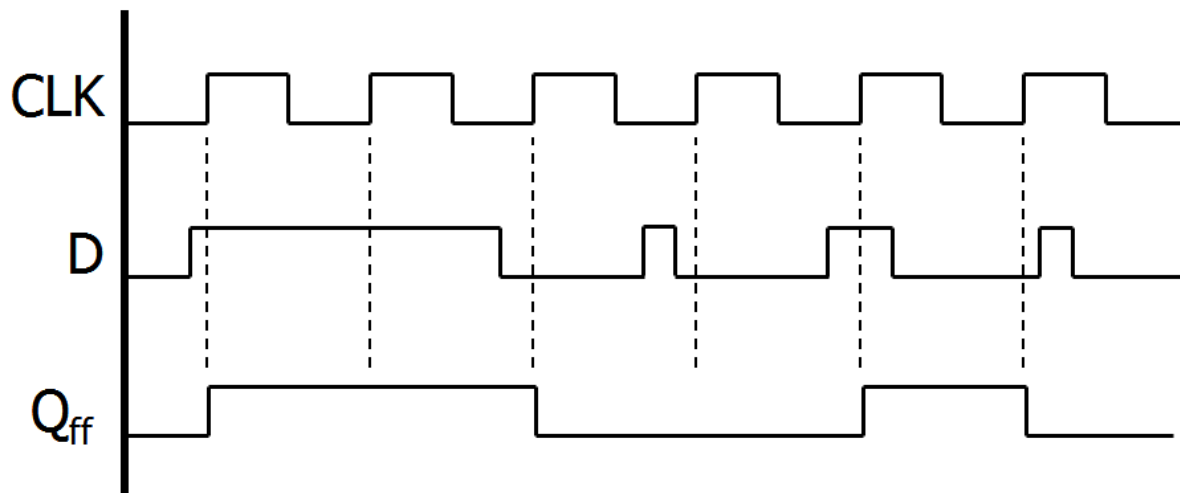
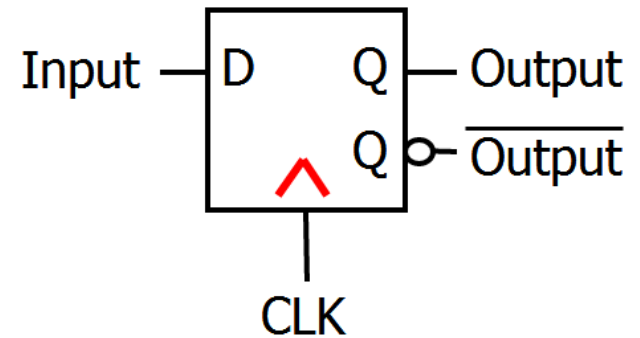
D-Latch

- ◆ Output depends on clock
 - Clock high: Input passes to output
 - Clock low: Latch holds its output
- ◆ Latches are level sensitive and transparent

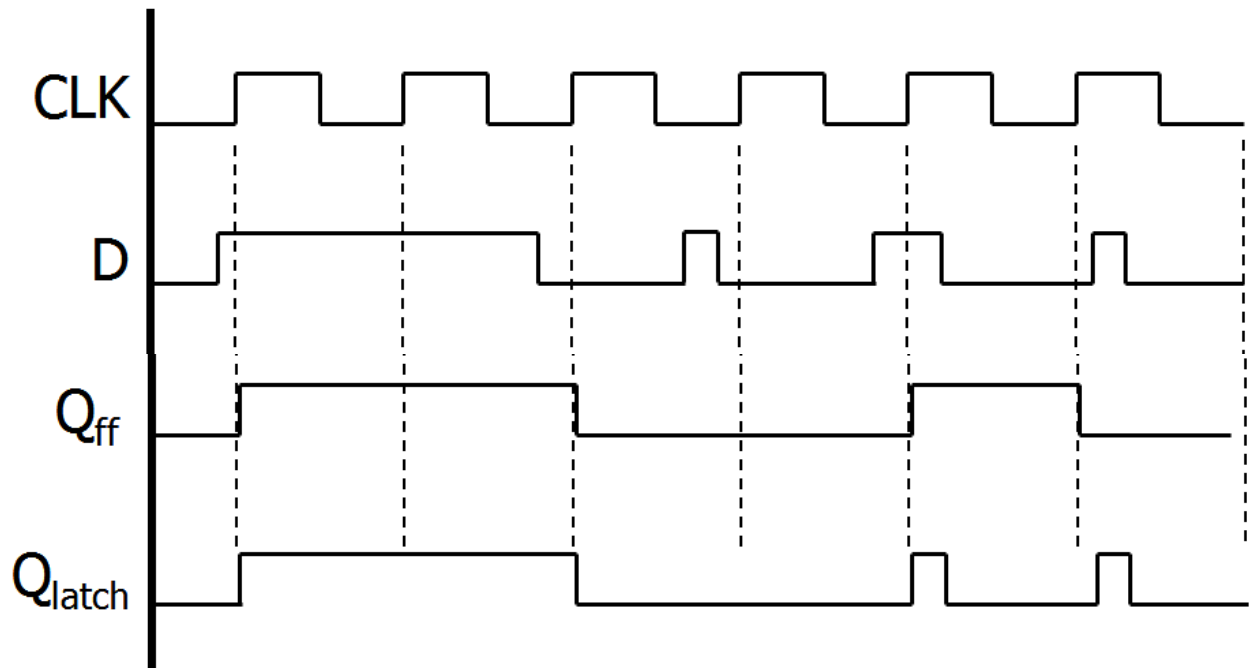
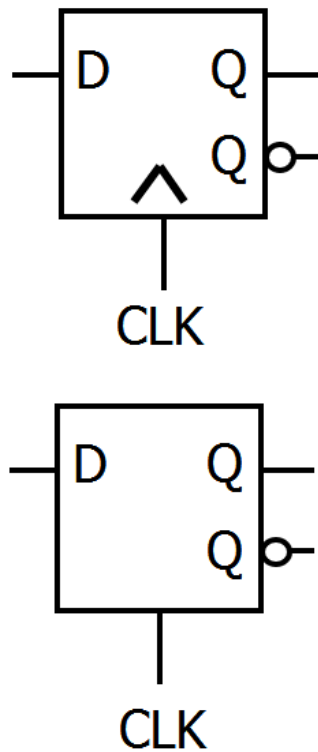


The D flip-flop

- ◆ Input sampled at clock edge
 - Rising edge: Input passes to output
 - Otherwise: Flip-flop holds its output
- ◆ Flip-flops are rising-edge triggered, falling-edge triggered, or master-slave

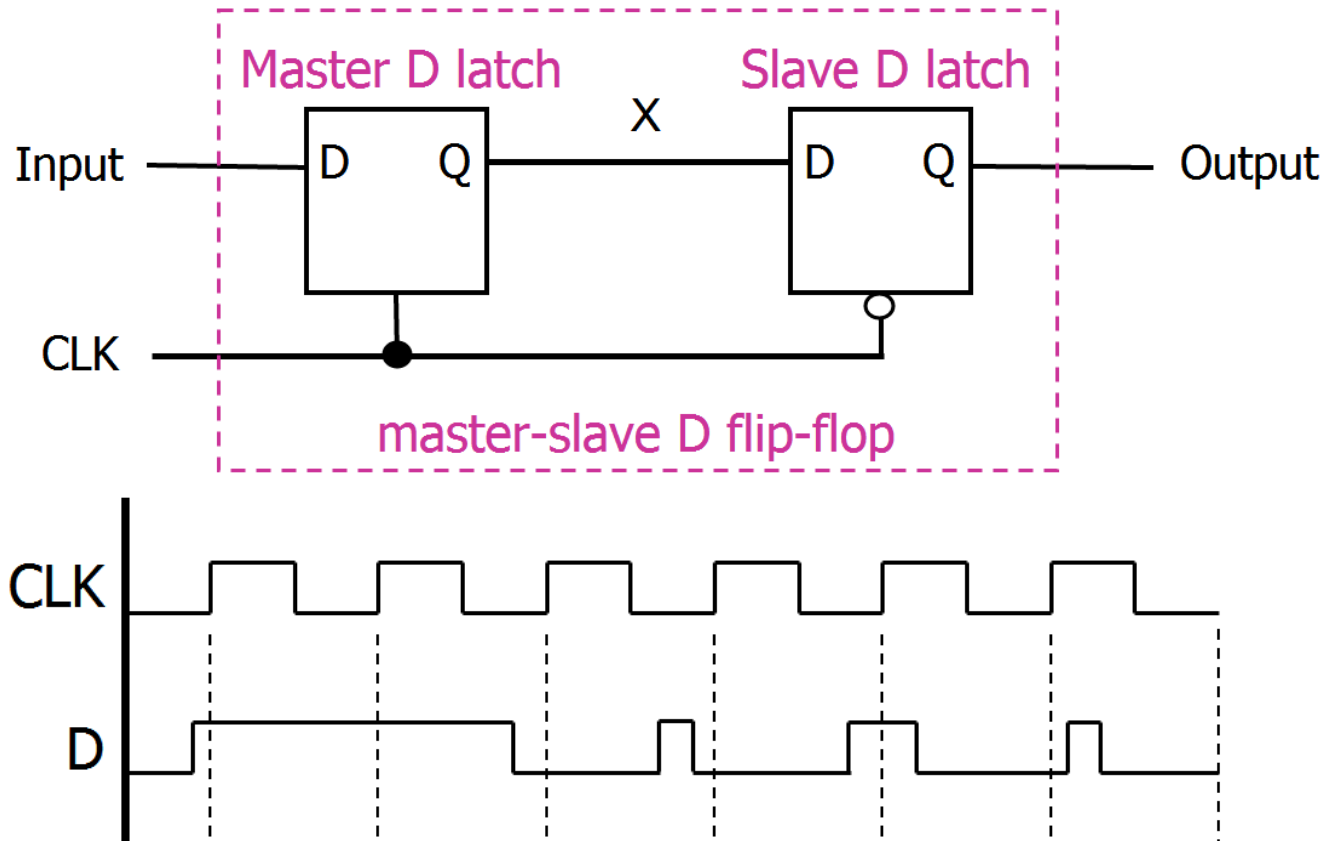


Latches versus flip-flops



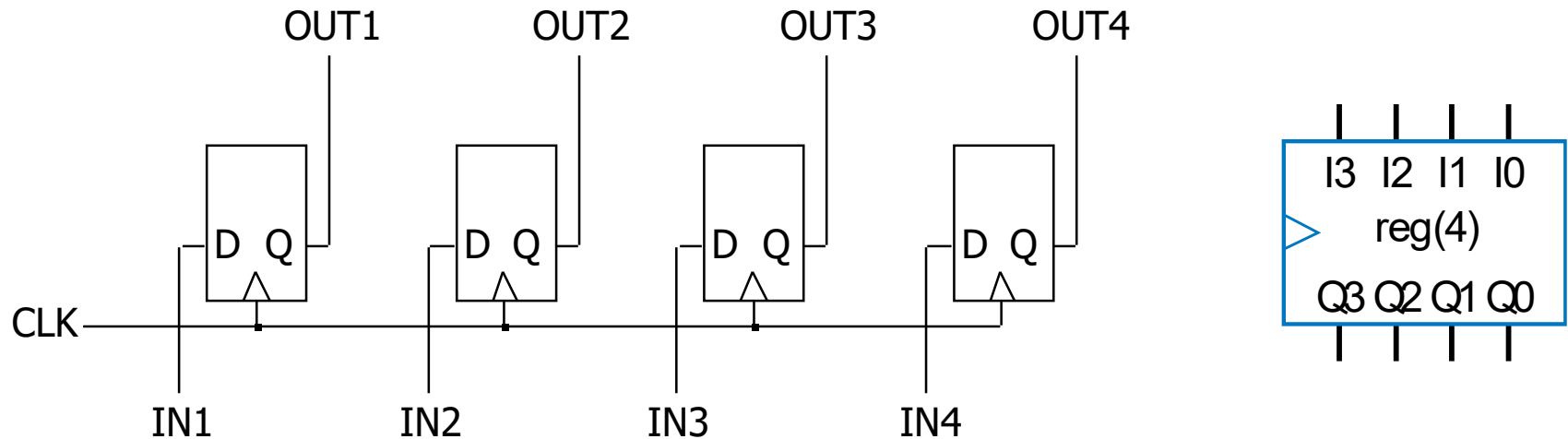
behavior is the same **unless** input changes while the clock is high

The master-slave D



Building blocks with FFs: Basic Register

- Register: a sequential component that can store multiple bits
- A basic register can be built simply by using multiple D-FFs



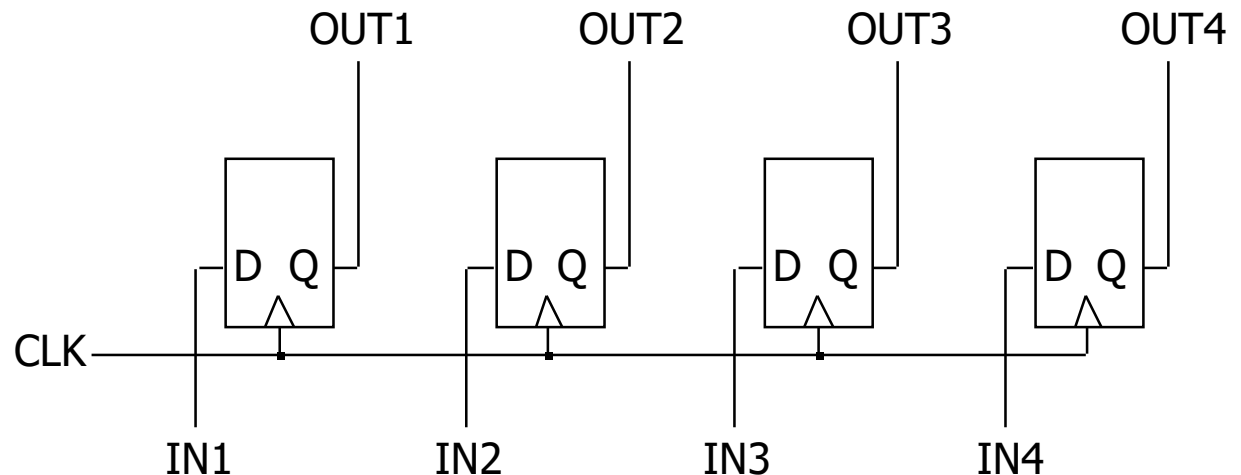
Register: basic Verilog

```
module Register (D, Clk, Q);  
input [3:0] D;  
input Clk;  
output reg [3:0] Q;
```

```
    always @(posedge Clk)
```

```
        Q <= D;
```

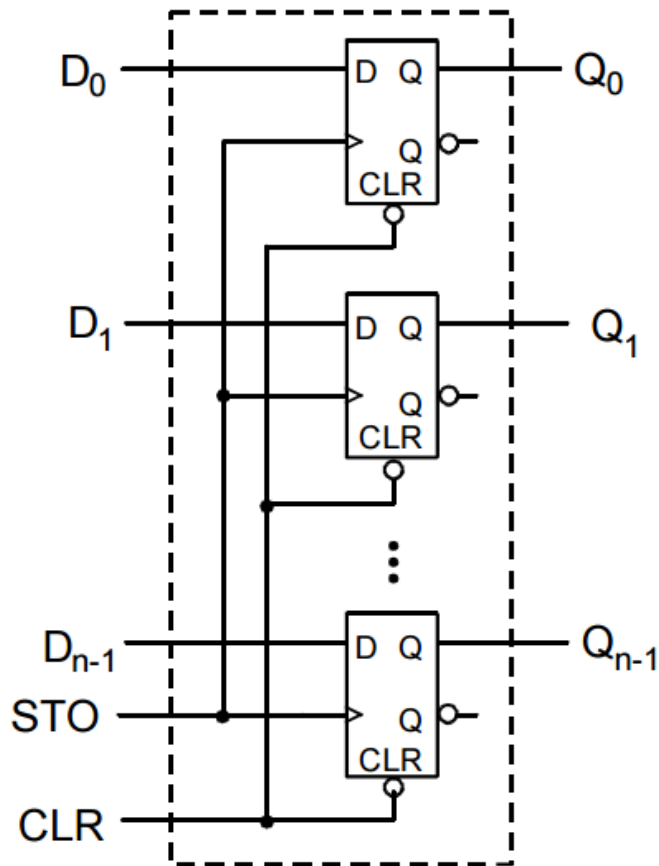
```
endmodule
```



Register

- A register is a memory device that can be used to store more than one bit of information.
- A register is usually realized as several flip-flops with **common control signals** that control the movement of data to and from the register
 - Common refers to the property that the control signals apply to all flip-flops in the same way
 - **Load** or **Store**: put new data into the register
 - **Read**: retrieve the data stored in the register (usually without changing the stored data)
 - **Clear**: writes a default value into the register (usually all zeros)
 - **(Read/Write)-Enable**: Enables respectively reading from and writing to the register

Example: register with clear



```
module reg1 (STO, CLR, D, Q);  
  parameter n = 16;  
  input STO, CLR;  
  input [n-1:0] D;  
  output [n-1:0] Q;  
  reg [n-1:0] Q;  
  
  always @(posedge STO or negedge CLR)  
    if (CLR == 0) Q <= 0;  
    else Q <= D;  
  
endmodule
```

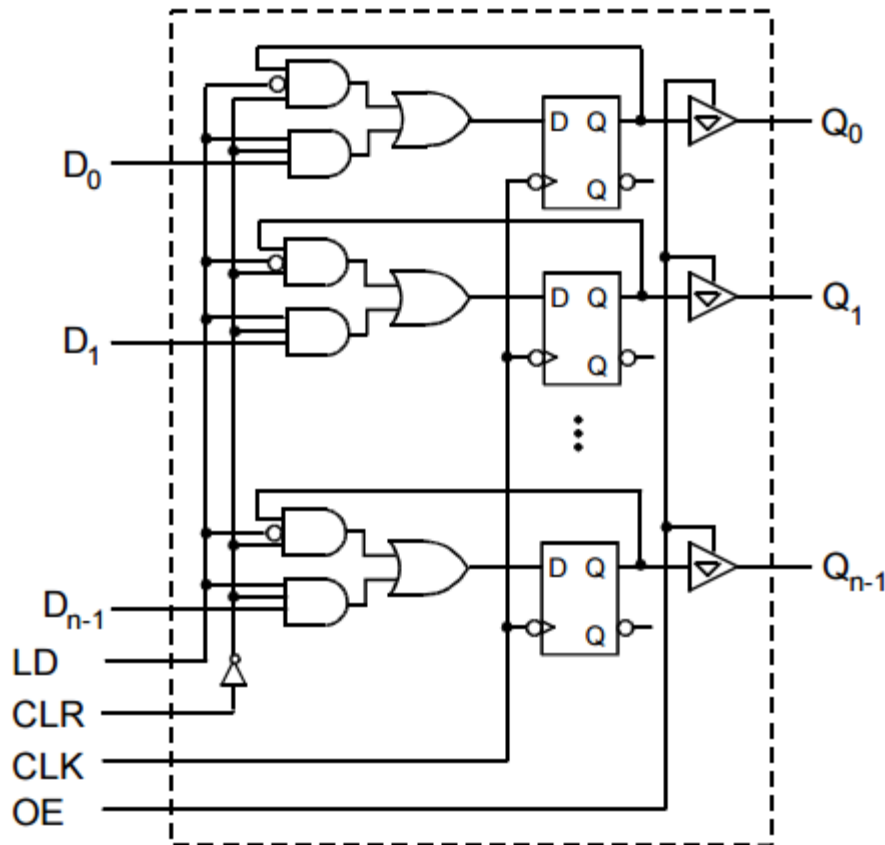
Variable type “reg” is not a register,
but it is used to build registers

- What does STO doing in this code?
- A. It is the enable of DFF blocks
 - B. It's clock
 - C. It loads a value
 - D. It's an input which loads to register
 - E. None of the above

Synch/Asynch control signals

- Control Signals
 - When they are asserted, they initiate an action in the register
 - **Asynchronous Control Signals** cause the action to take place immediately
 - **Synchronous Control Signals** must be asserted during a clock assertion to have an effect
- “Traditional” control signals:
 - Load/Store/Write
 - Read
 - Clear
 - Enable
- Depending on the functionalities of your register, you might want to design other control signals
 - Dummy example: a control signal that writes a 0 on all even positions

Example 2: register with load, clear and output enable



```

module reg2 (CLK, CLR, LD, OE, D, Q);
    parameter n = 4;
    input CLK, CLR, LD, OE;
    input [n-1:0] D;
    output [n-1:0] Q;
    reg [n-1:0] IQ, Q;
    integer k;

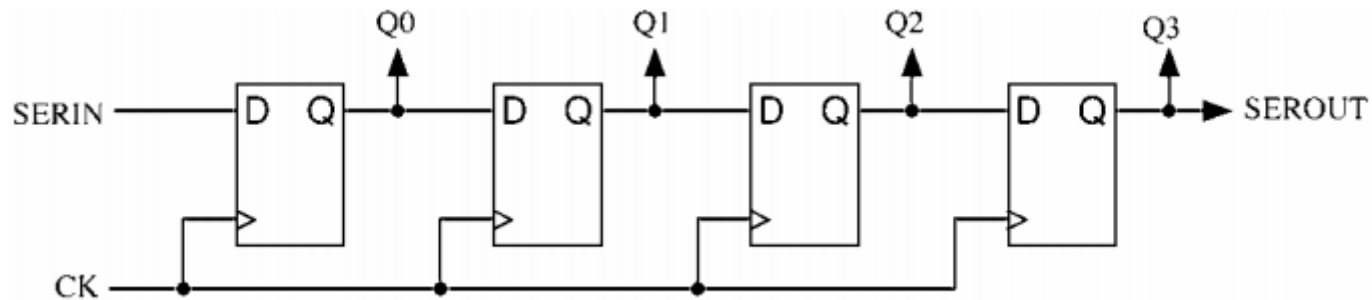
    always @(posedge CLK)
        if (CLR) IQ <= 0;
        else if (LD) IQ <= D;

    always @(OE)
        if (OE) Q = IQ;
        else Q = 'bz;

endmodule
    
```


Shift registers

- A shift register is a register capable of shifting its bits from one FF to the next one
- NOTE: do not confuse the shift register with the logic/arithmetic shifter



The shift register drawn above is:

- A. Left shifter
- B. Right shifter
- C. Rotational
- D. None of the above

Example: Verilog shift register with load

```
module shift4 (D, LD, LI, Ck, Q);
```

```
  input [3:0] D;
```

```
  input LD, LI, Ck;
```

```
  output [3:0] Q;
```

```
  reg [3:0] Q;
```

```
  always @(posedge Ck)
```

```
    if (LD)
```

```
      Q <= D;
```

```
    else
```

```
      begin
```

```
        Q[0] <= Q[1];
```

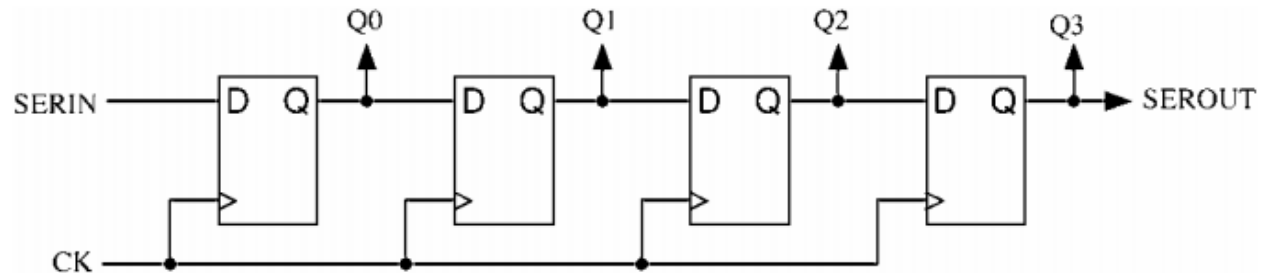
```
        Q[1] <= Q[2];
```

```
        Q[2] <= Q[3];
```

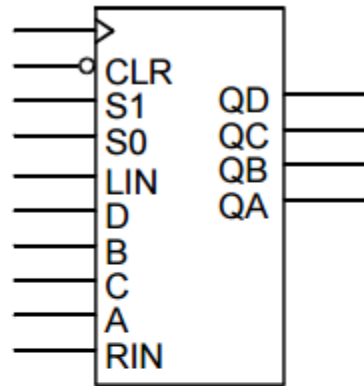
```
        Q[3] <= LI;
```

```
      end
```

```
endmodule
```



Example 2: A multifunction shift register with control signals



S1	S0	Action	QA*	QB*	QC*	QD*
0	0	hold	QA	QB	QC	QD
0	1	shift right	RIN	QA	QB	QC
1	0	shift left	QB	QC	QD	LIN
1	1	load	A	B	C	D

- Shift left is from A to D
- Shift right is from D to A
- CLR is asynchronous



(5PCS) SN74LS194AN IC BI-DIR UNIV SHIFT REG 16-DIP LS194 74LS194
by SICSTOCK

Currently unavailable

Counters

- A counter is a register capable of incrementing and/or decrementing its contents
- More general definition: a register capable of changing its content between a set of possible predefined sequences (this definition accounts for more “fancy” counters)
- What other possible control signals you can apply to counters:
 - Counting up/down
 - Modulus
 - ...

Example: counter

```
module counter (C, CLR, Q);  
input C, CLR;  
output [3:0] Q;  
reg [3:0] tmp;
```

```
always @(posedge C or posedge CLR)  
begin  
    if (CLR)  
        tmp = 4'b0000;  
    else  
        tmp = tmp + 1'b1;  
    end  
assign Q = tmp;
```

What is this counter doing?

The set (S) signal is synchronous or asynchronous?

A. Yes

B. No

C. I don't know

Example 2: counter

```
module counter (C, S, Q);  
input C, S;  
output [3:0] Q;  
reg [3:0] tmp;  
  
always @(posedge C)  
begin  
    if (S)  
        tmp = 4'b1111;  
    else  
        tmp = tmp - 1'b1;  
    end  
    assign Q = tmp;  
endmodule
```

What is this counter doing?

The set (S) signal is synchronous or asynchronous?

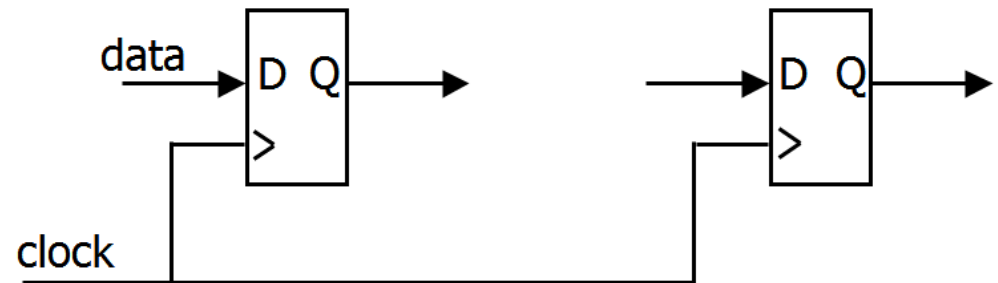
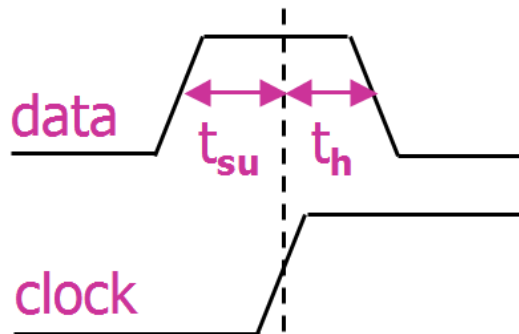
A. Yes

B. No

C. I don't know

Flip-flop timing

- **Setup time t_{su}** : Amount of time the input must be stable before the clock transitions high (or low for negative-edge triggered FF)
- **Hold time t_h** : Amount of time the input must be stable after the clock transitions high (or low for negative-edge triggered FF)



There is a timing "window" around the clock edge during which the input **must** remain stable

