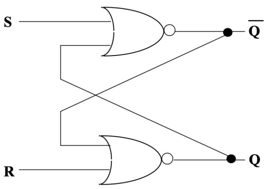
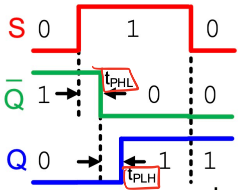
Tutorial 5 Questions (Part 2)

sequential logic: memory. (Q).
↓
previous state



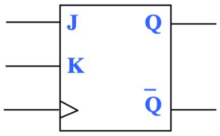
S	R	Output	Q+
0	0	Hold	Q
0	1	Clear	0
1	0	Set	1
1	1	Invalid	Invalid
0 0 is the rest state			

Assume R=0 :



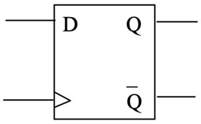
propagation delays
(passing through logic gates)

Asynchronous: unclocked,
Q+ changes whenever
S/R change



CLK	J	K	Q+	
↑	0	0	Q	Hold
↑	0	1	0	Clear
↑	1	0	1	Set
↑	1	1	$\overline{Q}$	Toggle

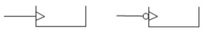
condensed characteristic table

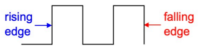


D Flip-Flop

CLK	D	Q+
↑	0	0
↑	1	1

i.e., LOW → HIGH, HIGH → LOW transitions





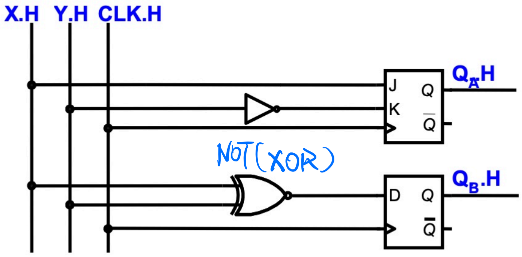
clock: important in complex
circuits

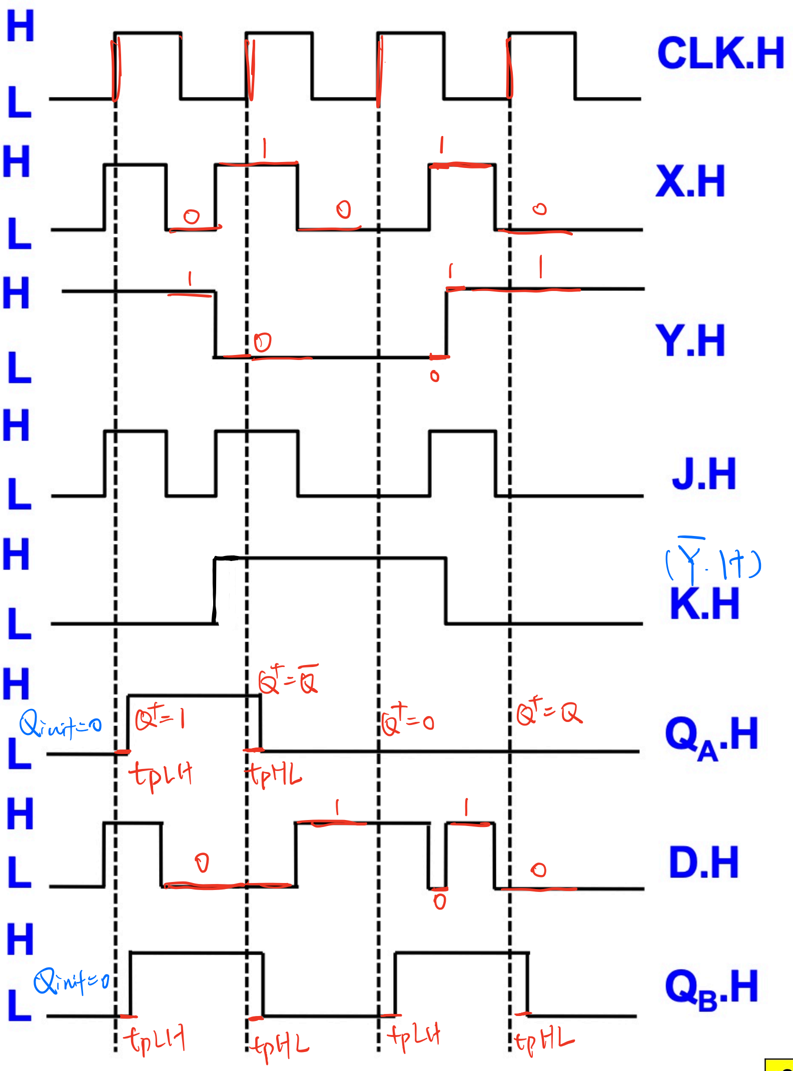
many components need to
change states in a
coordinated manner

Synchronous: clock input
Q+ changes
when:
① J/K change
AND
② CLK ↑/↓

Flip Flops and Verilog Modeling

1. Given the circuit diagram below, complete the timing diagram below by filling in Q_A and Q_B. Include propagation delays tp_{HL} and tp_{LH}.





CLK	J	K	Q+
↑	0	0	Q
↑	0	1	0
↑	1	0	1
↑	1	1	Q̄

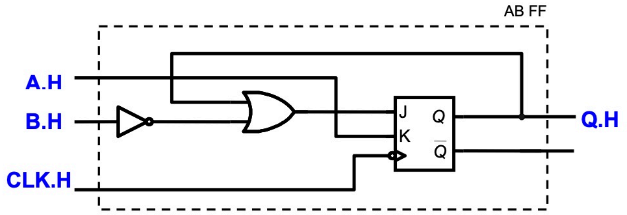
XOR (add)

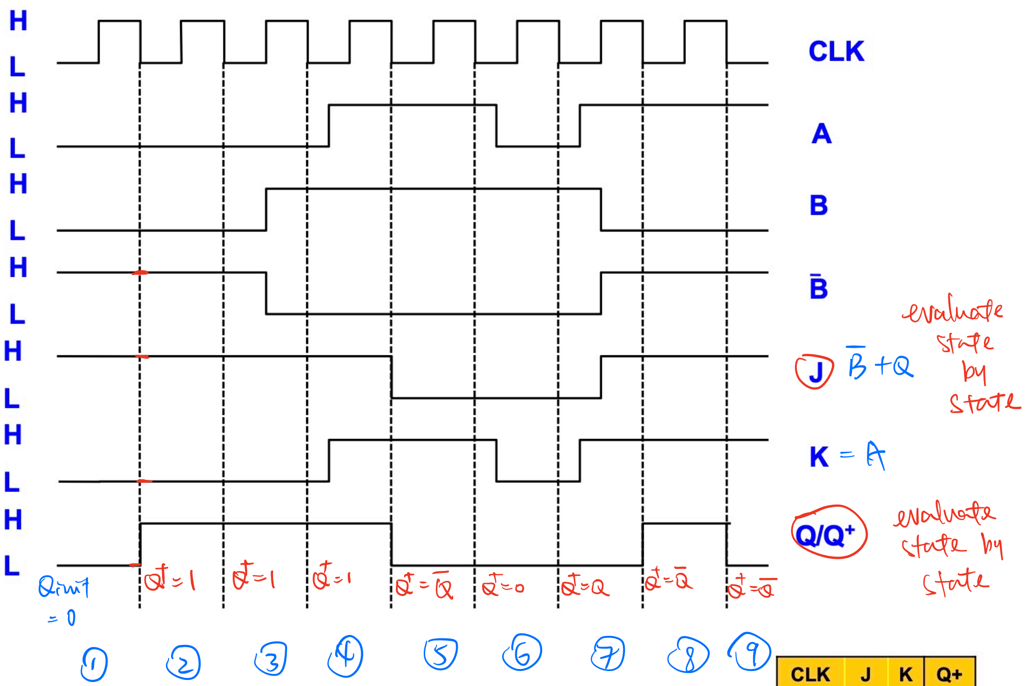
A	B	Y
0	0	0
1	0	1
0	1	1
1	1	0

CLK	D	Q+
↑	0	0
↑	1	1

2. (a) Given the circuit diagram below, complete the timing diagram below by filling in J, K and Q. Neglect all propagation delays in this question.

(b) If the circuit below represents a type of AB flip flop, fill in its characteristic table and condensed characteristic table below.





CLK	J	K	Q+
↑	0	0	Q
↑	0	1	0
↑	1	0	1
↑	1	1	$\overline{Q}$

	A	B	Q	Q^+
① → ②	0	0	0	**1**
② → ③	0	0	1	**1**
⑥ → ⑦	0	1	0	**0**
③ → ④	0	1	1	**1**
⑦ → ⑧	1	0	0	**1**
⑧ → ⑨	1	0	1	**0**
⑤ → ⑥	1	1	0	**0**
④ → ⑤	1	1	1	**0**

Characteristic Table

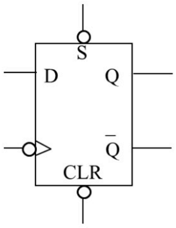
A	B	Q^+
0	0	**1**
0	1	**Q**
1	0	$\overline{\mathbf{Q}}$
1	1	**0**

Condensed Characteristic Table

3. The 74'74B is an integrated circuit containing negative-edge triggered D flip-flops with synchronous set and asynchronous reset inputs. The D flip-flop receives four 1-bit input signals, **D**, **CLK**, **S**, **CLR** and produces two 1-bit output signals, **Q** and **QB**. Write a Verilog program that implements the D flip-flop according to the characteristic table shown below.

negedge

CLK	CLR	S	D	Q^+
X	0	X	X	0
↓	1	0	X	1
↓	1	1	0	0
↓	1	1	1	1



```
module dff ( input D, CLK, CLR, S, output reg Q, output QB);

always @ ( negedge CLK, negedge CLR ) //Refer * Below for explanation

begin
if ( CLR == 0 )
    Q <= 0;
else
begin
    if ( S == 0 )
    Q <= 1;
   else
    Q <= D;
end
end

    assign QB = ~Q;

endmodule
```

asynchronous reset (ignore clk)

sequential

non-blocking

combinational

* In order to achieve an ***asynchronous reset***, the CLR signal must be included in the sensitivity list. Otherwise, the value of CLR would only be checked when there is a falling edge in the clock signal. This would implement a synchronous reset instead of asynchronous reset.

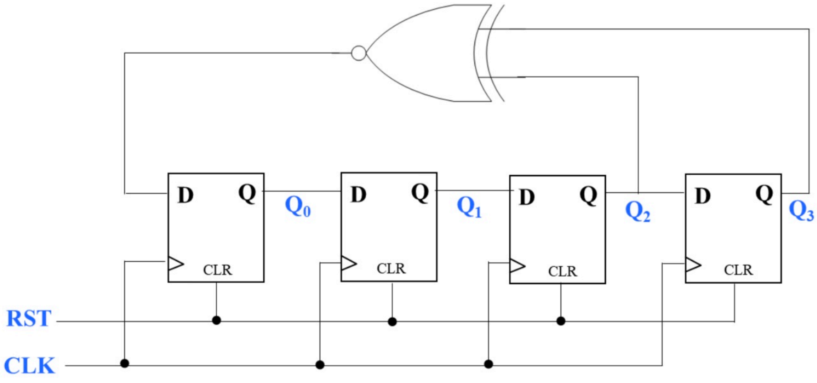
* When using posedge / negedge in the sensitivity lsit (eg. posedge clk), all signals need to specified with either posedge or negedge. This is to ensure that the design can be synthesized (mapped to an available device) and implemented on the FPGA. As such, "always@ (negedge CLK, CLR)" is not allowed.

4. The circuit as shown below is a linear feedback shift register (LFSR), commonly used as a Random Number Generator in hardware implementations.

The input bit to the chain of FFs is a function of its previous output. LFSRs are able to generate pseudo-random bit sequences which also have applications in games, cryptography, bit-error-rate measurements to wireless communication systems.

The LFSR below receives two 1-bit inputs, CLK and RST and generates a 4-bit output Q. The RST signal sets the output of the LFSR to 4'b000 asynchronously when enabled. RST= 1, clear

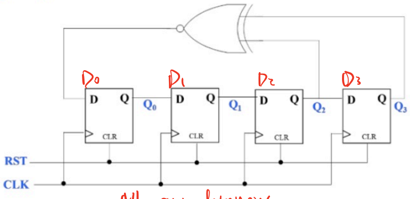
The D flip-flops used are positive-edge triggered with asynchronous active high clear.



CLK	D	Q+
↑	0	0
↑	1	1

Write a Verilog program that implements the LFSR according to the circuit above. Simulate / work out the operation of the circuit to work out the bit sequence of Q.

```
module lfsr ( input CLK, RST, output reg [3:0] Q);

reg [3:0] Q = 4'b0000;

always @ ( posedge CLK, posedge RST )

begin
    if ( RST )
        Q <= 0;
    else
    begin
        Q <= { Q[2:0], ~(Q[2]^Q[3]) } ; //Refer below for alternatives!
    end

end

endmodule
```



All synchronous

XOR

= concat operator

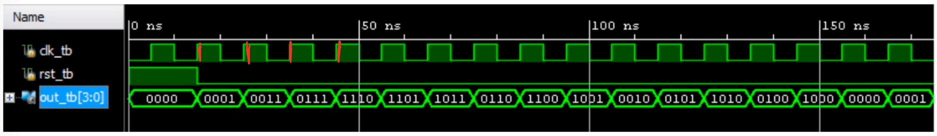
$D_3=Q_2$ $D_2=Q_1$ $D_1=Q_0$ $D_0=\overline{Q_2\oplus Q_3}$

current state. Q_3 Q_2 Q_1 Q_0

next state; Q_2 Q_1 Q_0 $\overline{Q_2\oplus Q_3}$

```
// Alternative 1 :
        Q <= { Q[2], Q[1], Q[0], ~(Q[2]^Q[3]) } ;

// Alternative 2 :
        Q[3] <= Q[2] ;
        Q[2] <= Q[1] ;
        Q[1] <= Q[0] ;
        Q[0] <= ~(Q[2]^Q[3]) ;

// Alternative 3 :
        Q[3:1] <= Q[2:0];
        Q[0] <= ~(Q[2]^Q[3]) ;
```

The bit sequence can be derived / simulated as follows :



```
Name          |0 ns                 |50 ns                      |100 ns                      |150 ns
clk_tb
rst_tb
out_tb[3:0]    0000  0001  0011  0111  1110  1101  1011  0110  1100  1001  0010  0101  1010  0100  1000  0000  0001
```

By increasing the number of flip-flops used (eg. 12), the sequence that is generated is sufficiently random for many applications.

	Q0	Q1	Q2	Q3
Q2^Q3	0	0	0	0
1	1	0	0	0
1	1	1	0	0
1	1	1	1	0
0	0	1	1	1
1	1	0	1	1
1	1	1	0	1
0	0	1	1	0