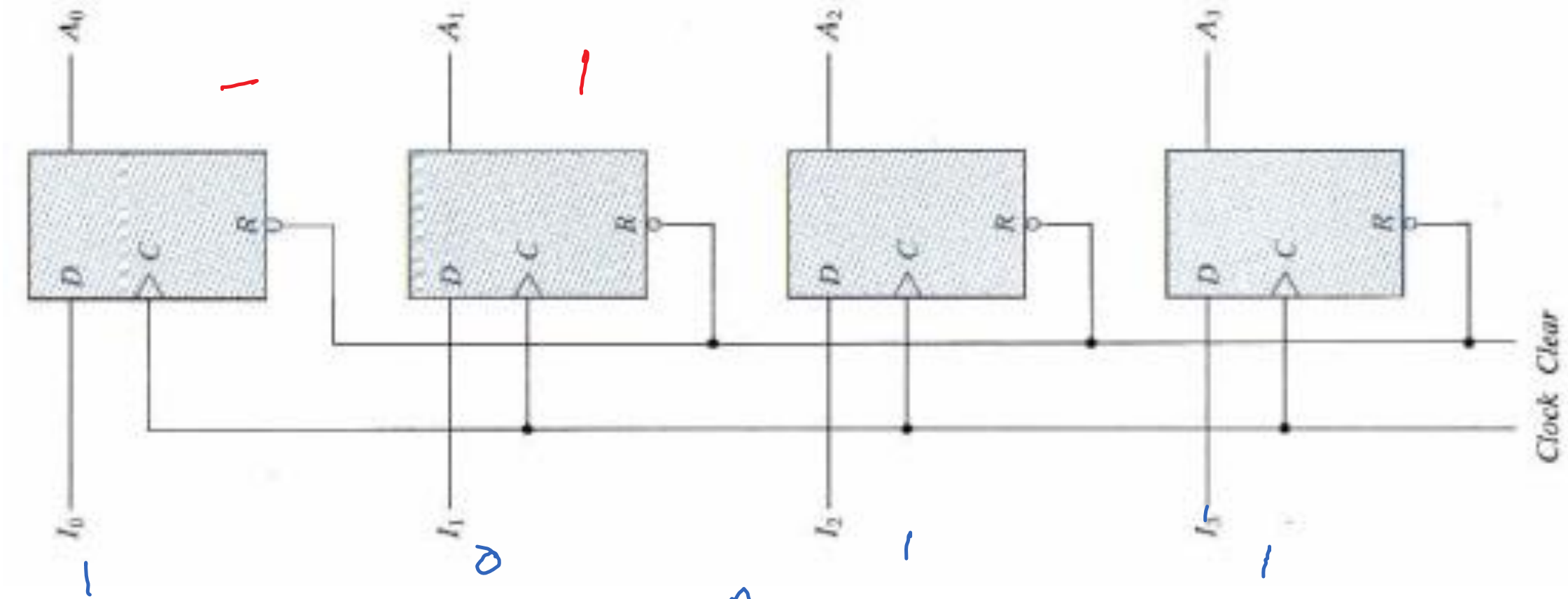
Registers

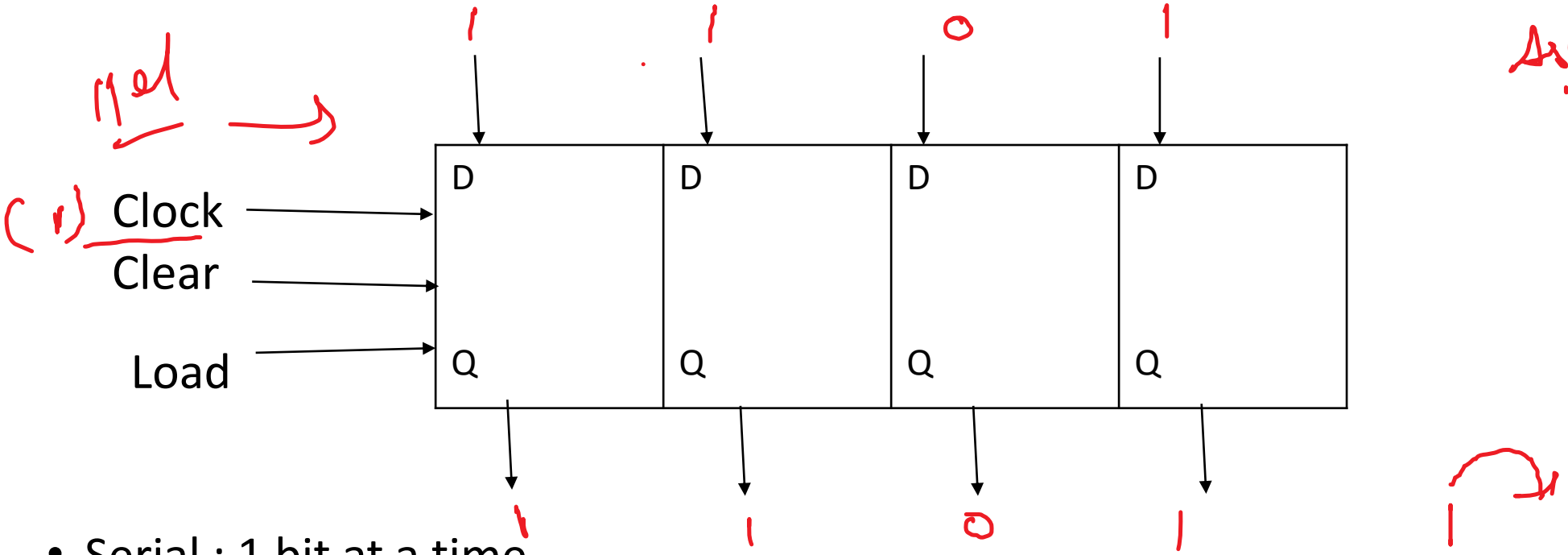
Registers

- A Flip flop is 1-bit memory cell.
- To increase the storage capacity, flip flops are grouped together known as REGISTERS.
- A register is a group of flip flops, each one of which shares a common clock and is capable of storing one bit of information.
- An n-bit register consists of a group of n flip flops capable of storing n bits of binary information.
- A register consists of a group of flip flops together with gates that affect their operation.
- The flip flops hold the binary information, and the gates determine how the information is transferred into the register.

Four Bit Register



- Data storage can be done by independent control called as load input.
- Types:
- Synchronous load – Clock and load are high
- Asynchronous load – Only load is high.
- Data can be entered in serial (temporal code) or parallel (special code)form.



- Serial : 1 bit at a time
- Parallel: all bits at the same time

Classification of Registers

- Depending on Input and output:

1. SISO – Serial In and Serial Out
2. SIPO- Serial In and Parallel Out
3. PISO- Parallel In and Serial Out

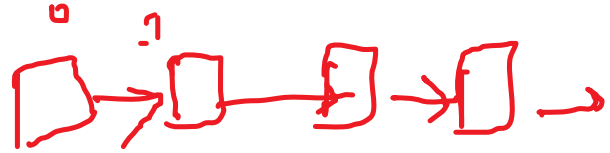
Shift Register

4. PIPO- Parallel In and Parallel Out → **Storage Register**

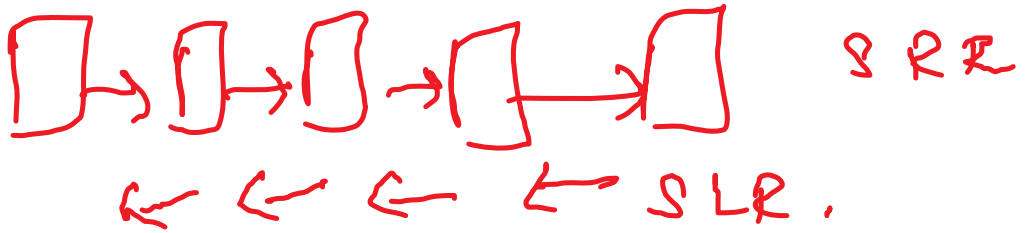
- Depending on Application:

1. Shift Register
2. Storage Register (PIPO)

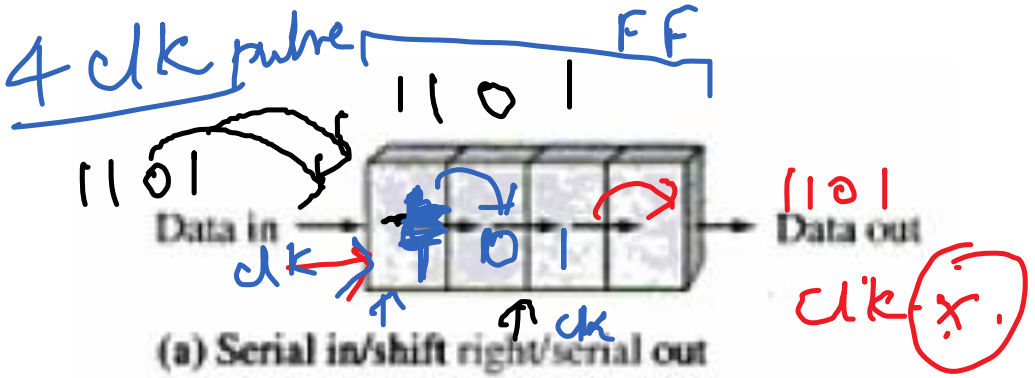
Shift Register



- A register capable of shifting the binary information held in each cell to its neighbouring cell, in a selected direction is called a shift register.
- It consists of chain of flip flops in cascade, with the output of one flip flop connected to the input of the next flip flop.
- All flip flops receive common clock pulses, which activate the shift of data from one stage to the next.
- The bits stored in such registers can be made to move within the registers and in/out of the registers by applying clock pulses.
- The registers which will shift the bits to left are called “Shift left registers”
- The registers which will shift the bits to right are called “Shift right registers”.

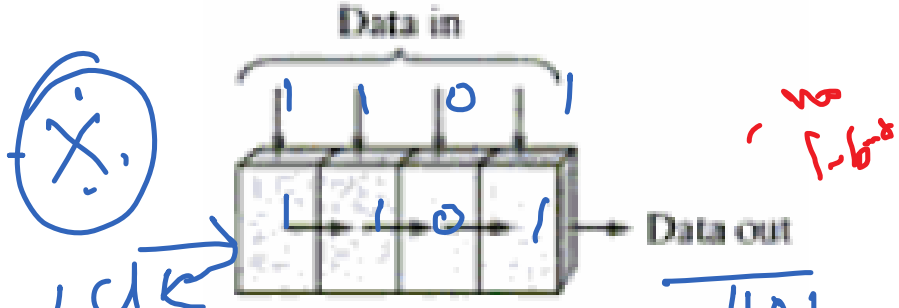


Basic Data Movement in Shift Registers

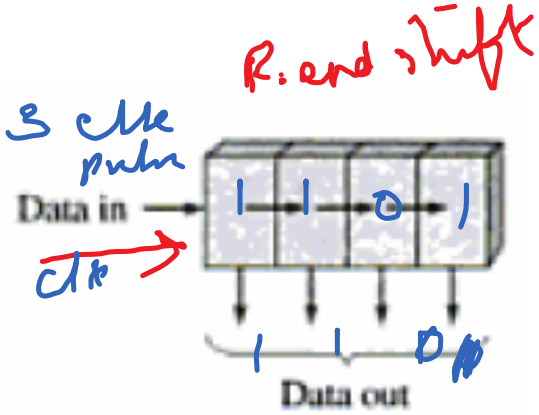


(a) Serial in/shift right/serial out

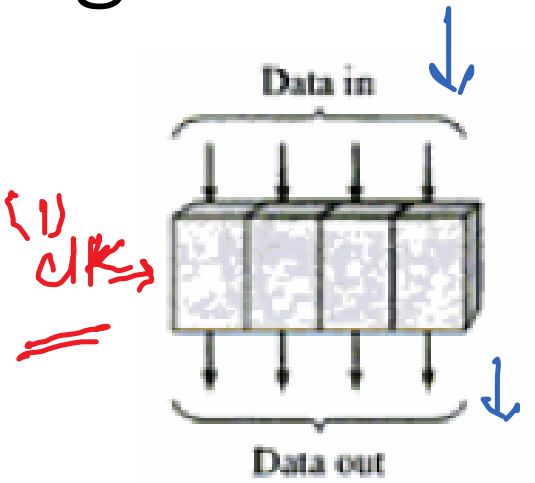
(b) Serial in/shift left/serial out



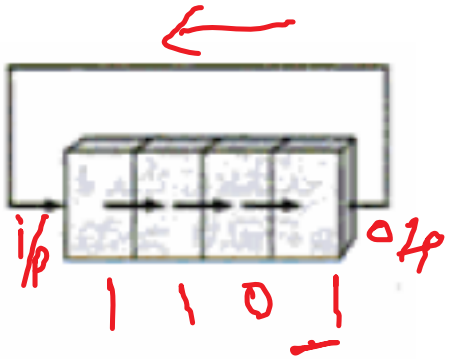
(c) Parallel in/serial out



(d) Serial in/parallel out



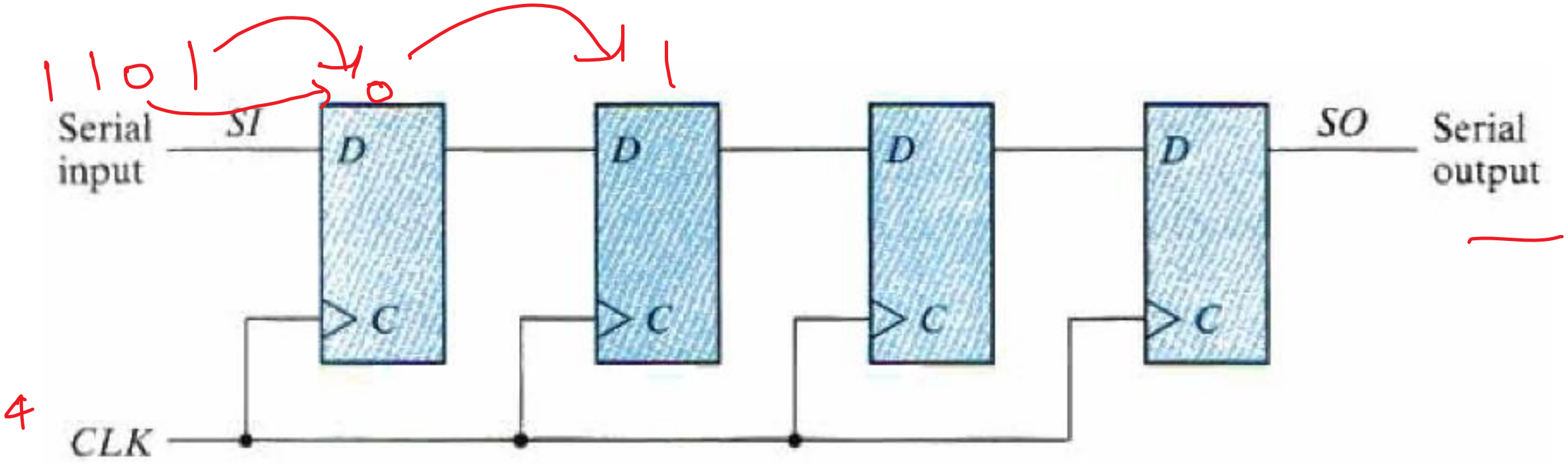
(e) Parallel in/parallel out



(f) Rotate right

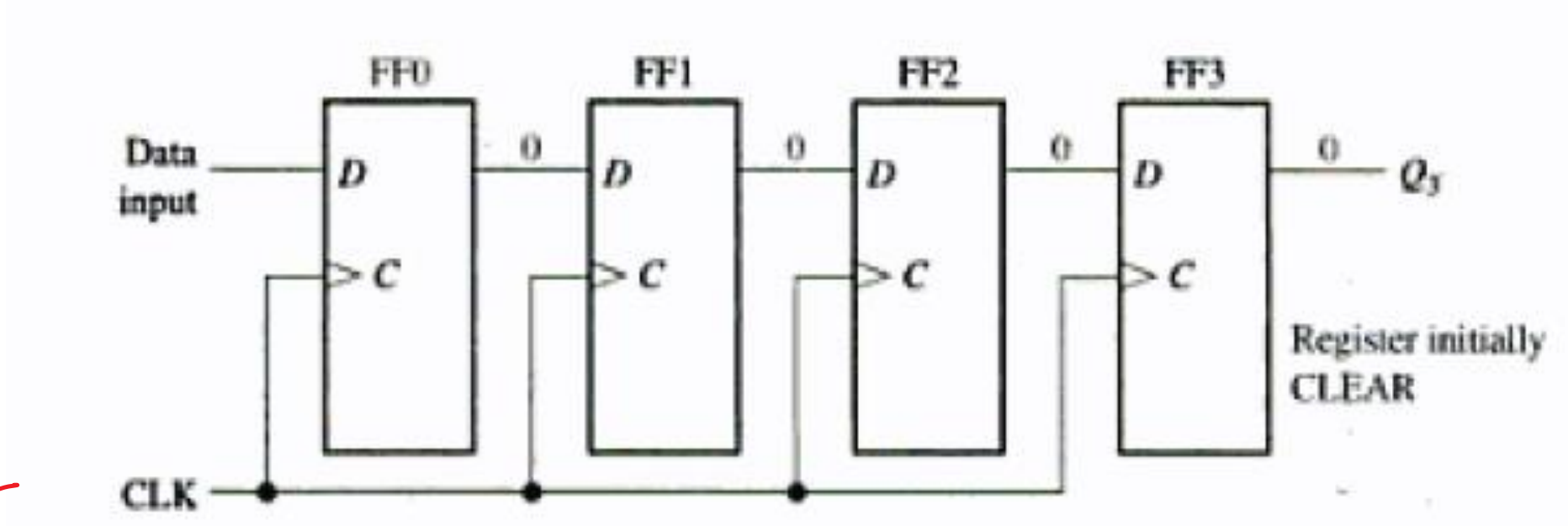
(g) Rotate left

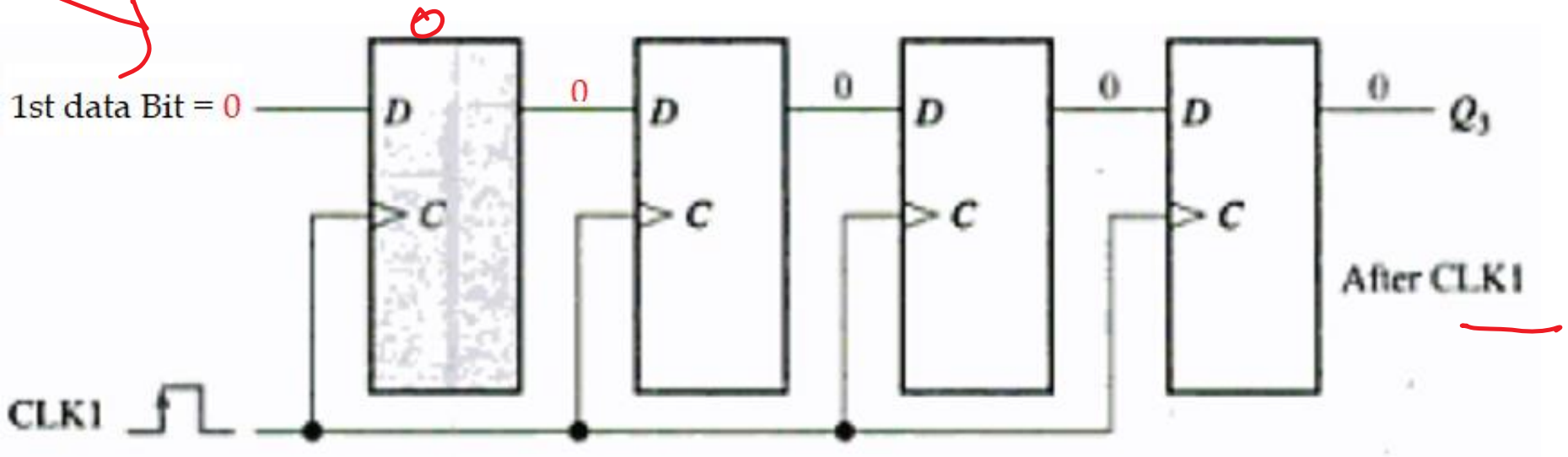
Serial In and Serial Out Shift Register
SISO

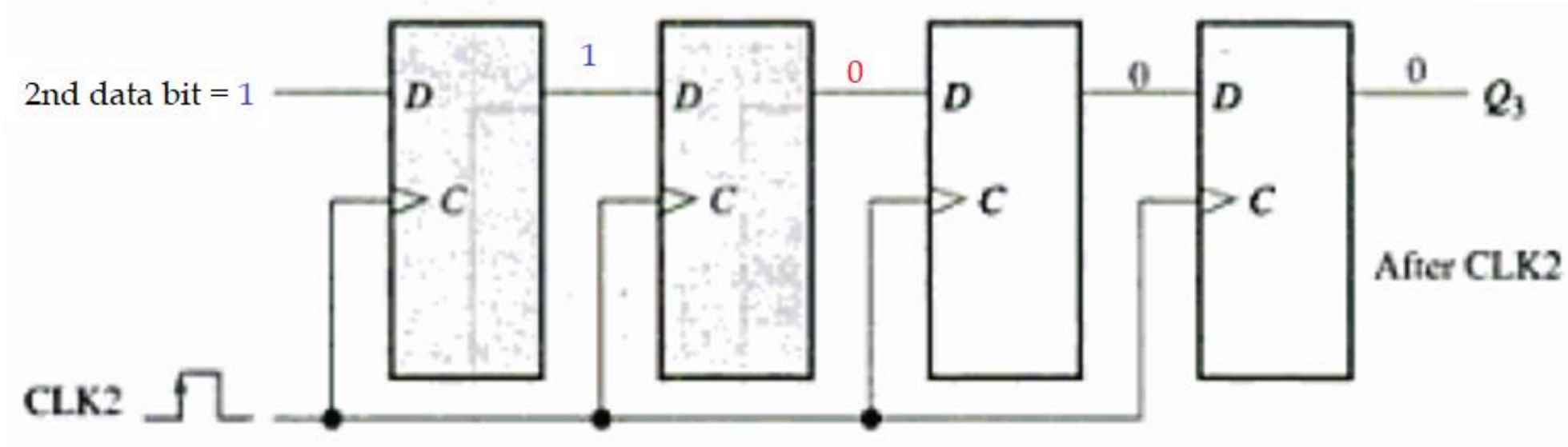


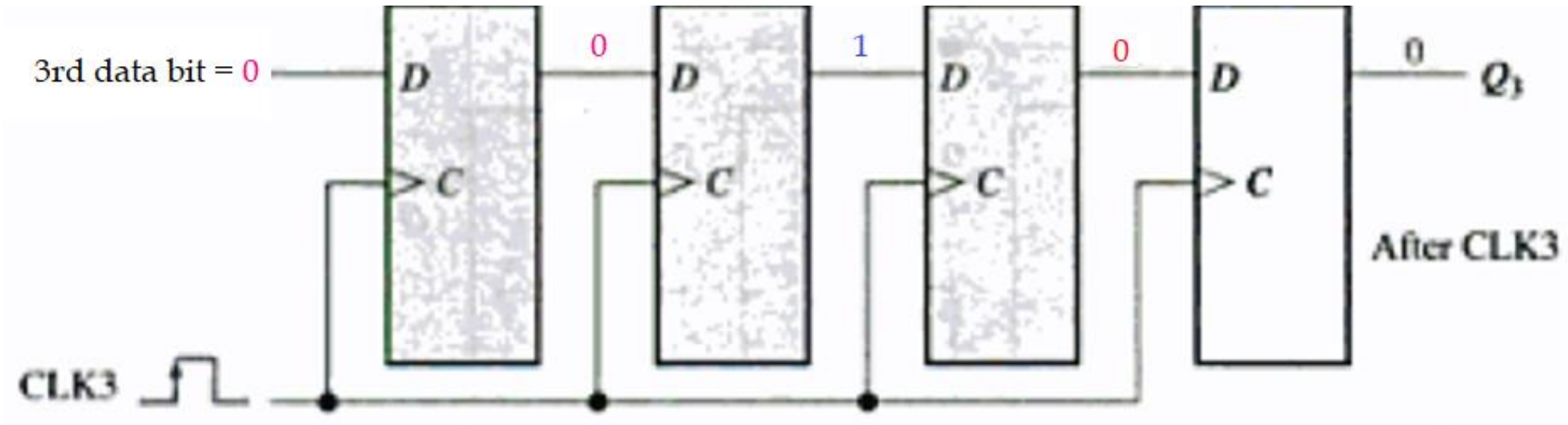
Assume the data input to be stored is 1010

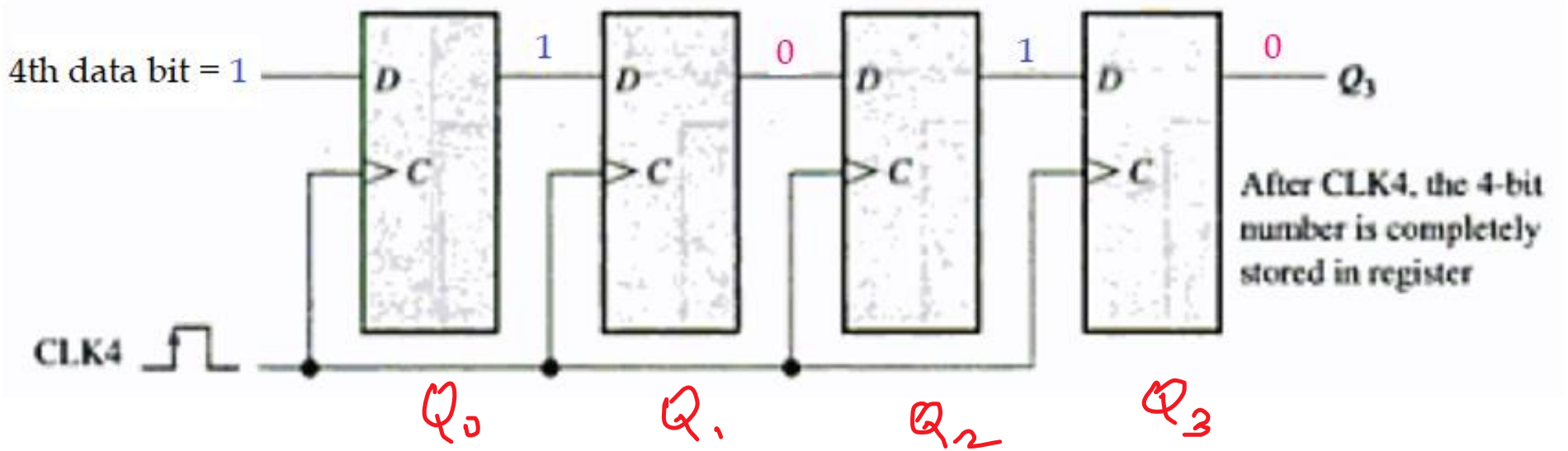
1010


FF0
FF1
FF2
FF3
Data input
D
C
0
Q_3
Register initially CLEAR
CLK


0
1st data Bit = 0
D
C
0
Q_3
After CLK1
CLK1


2nd data bit = 1
D
C
1
0
0
0
Q_3
After CLK2
CLK2


3rd data bit = 0
D
C
0
1
0
0
Q_3
After CLK3
CLK3

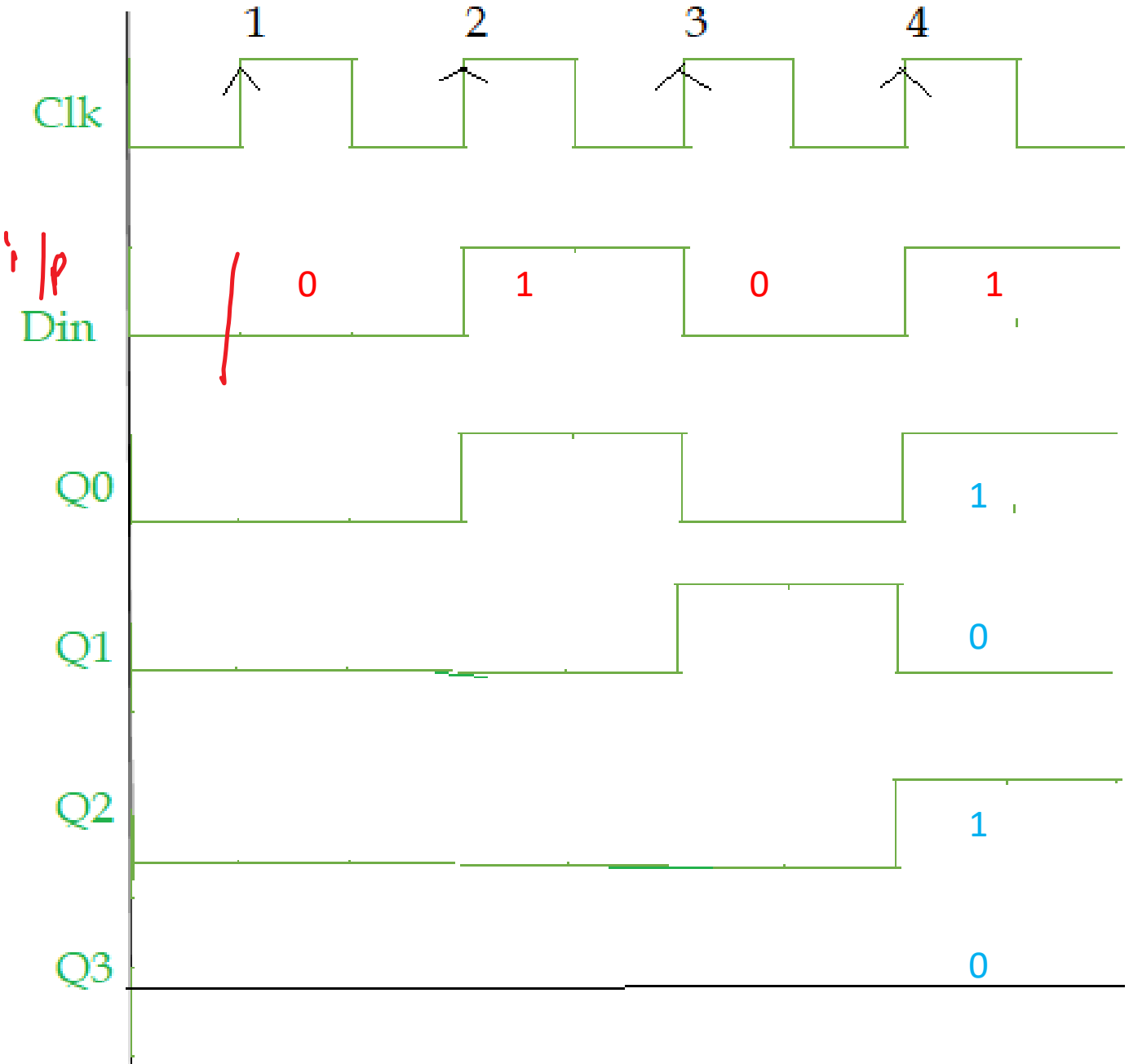


Data input = 1 0 1 0

Load / storing the data

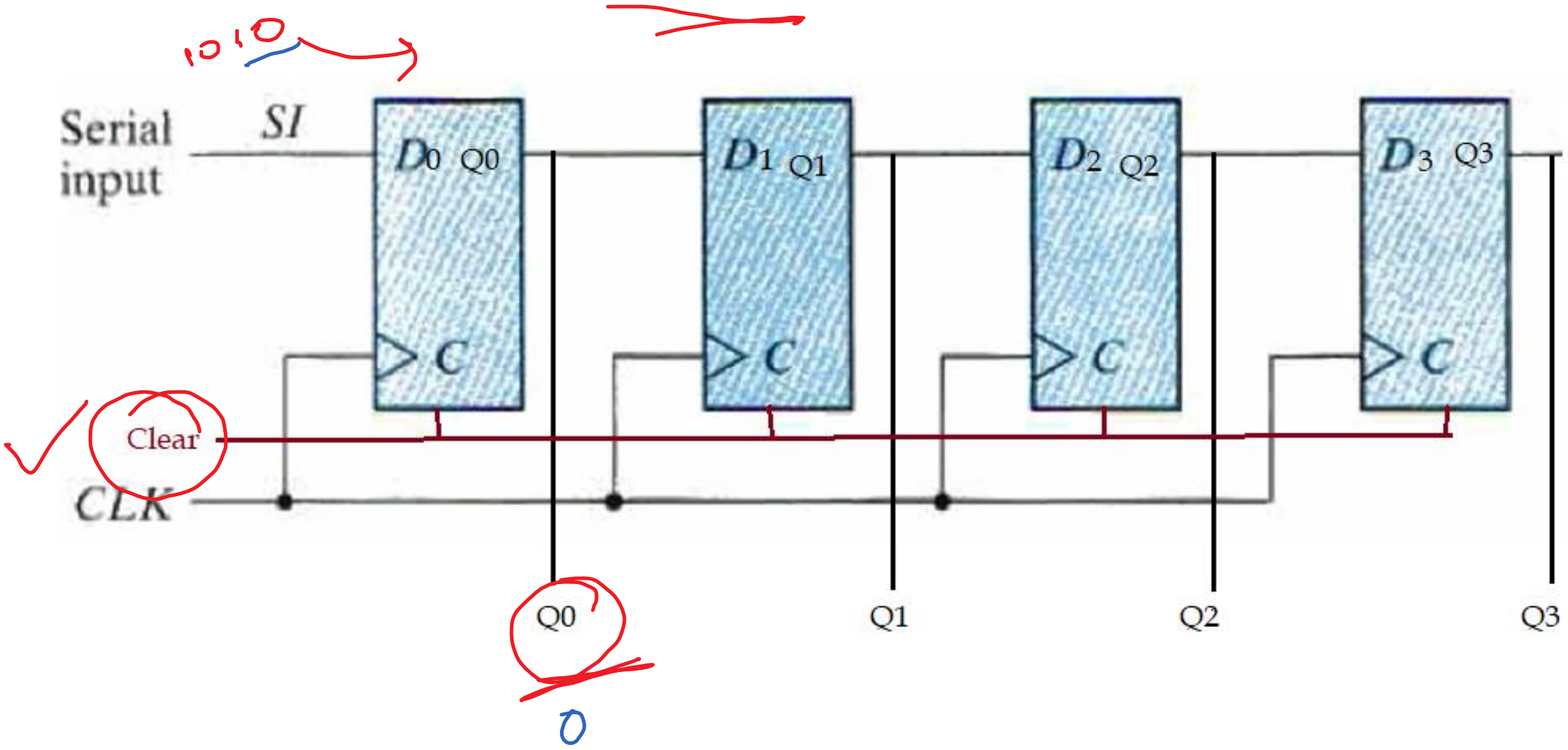
CLK	Q0	Q1	Q2	Q3
0 Initial	0	0	0	0
1	0	0	0	0
2	1	0	0	0
3	0	1	0	0
4	1	0	1	0

Timing Diagram.



- To store/load the data in 4 bit SISO shift register, 4 clk pulses are required.
- To retrieve/read the data (4+3) = 7 clk pulses are required.

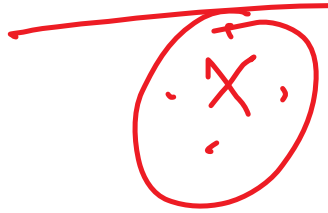
Serial In Parallel Out (SIPO) Shift register



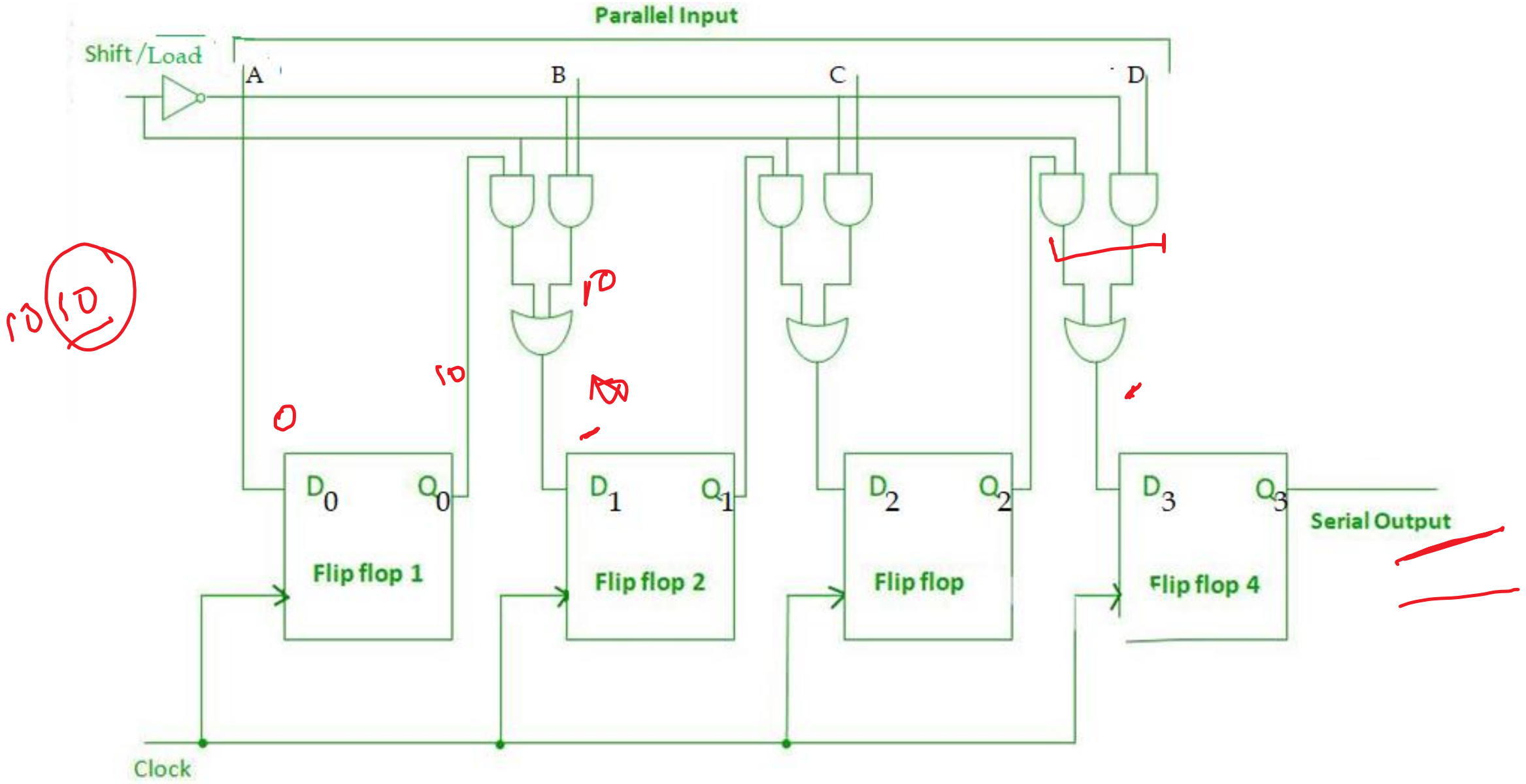
- SIPO:
- Assume the data to be 1101.

CLK	Q0	Q1	Q2	Q3
Initial	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	0	1	0
4	1	1	0	1

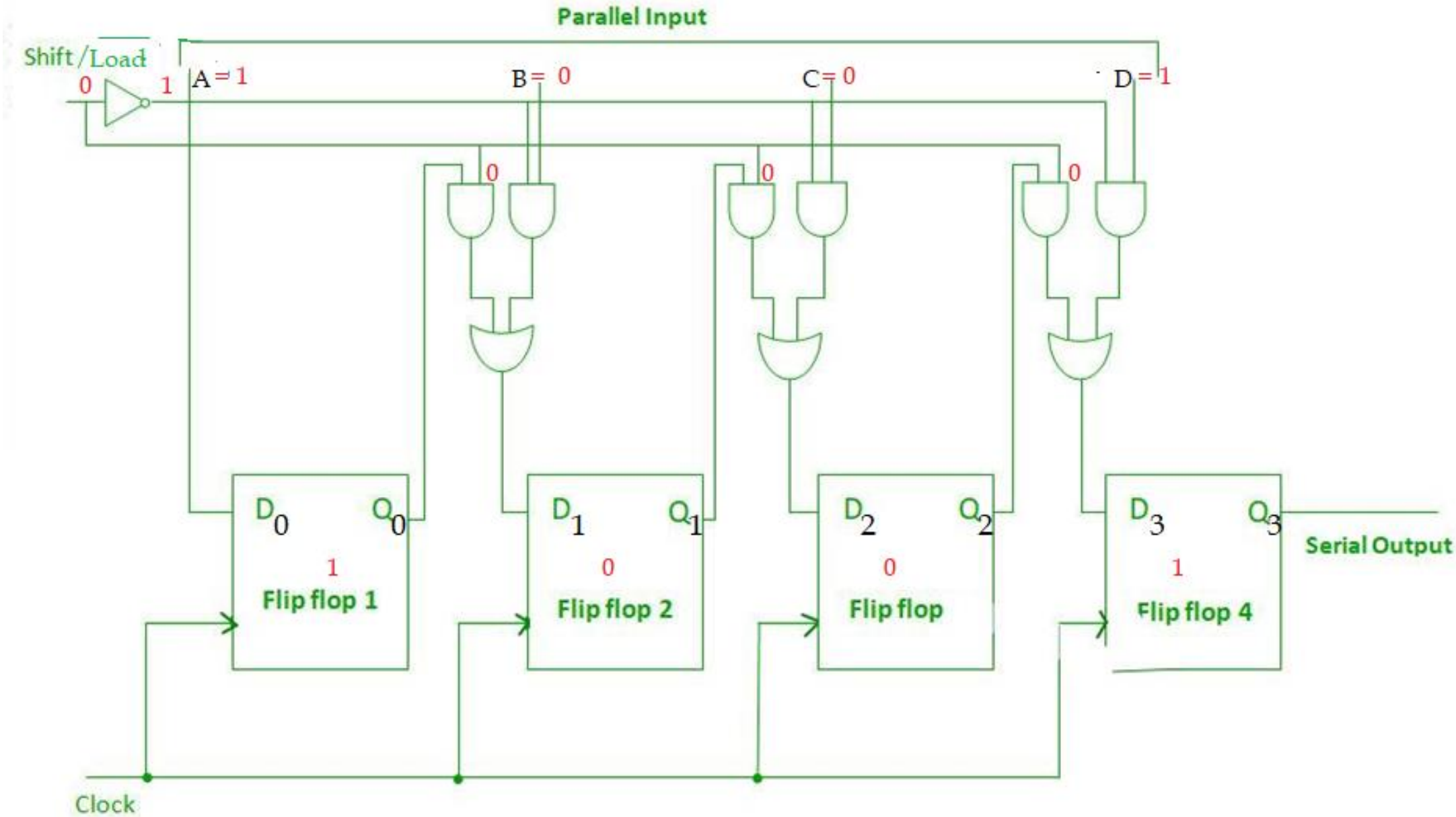
- 4 clk pulses are required to load the data.
- To retrieve/read the data, 1 clk pulse is required. i.e. In the 4th Clk pulse itself we get the data.



Parallel In Serial Out (PISO) Shift register



To Load/write the data 1001.



- PISO:
- Write: To Load the data 1001, 1 clk pulse is required.

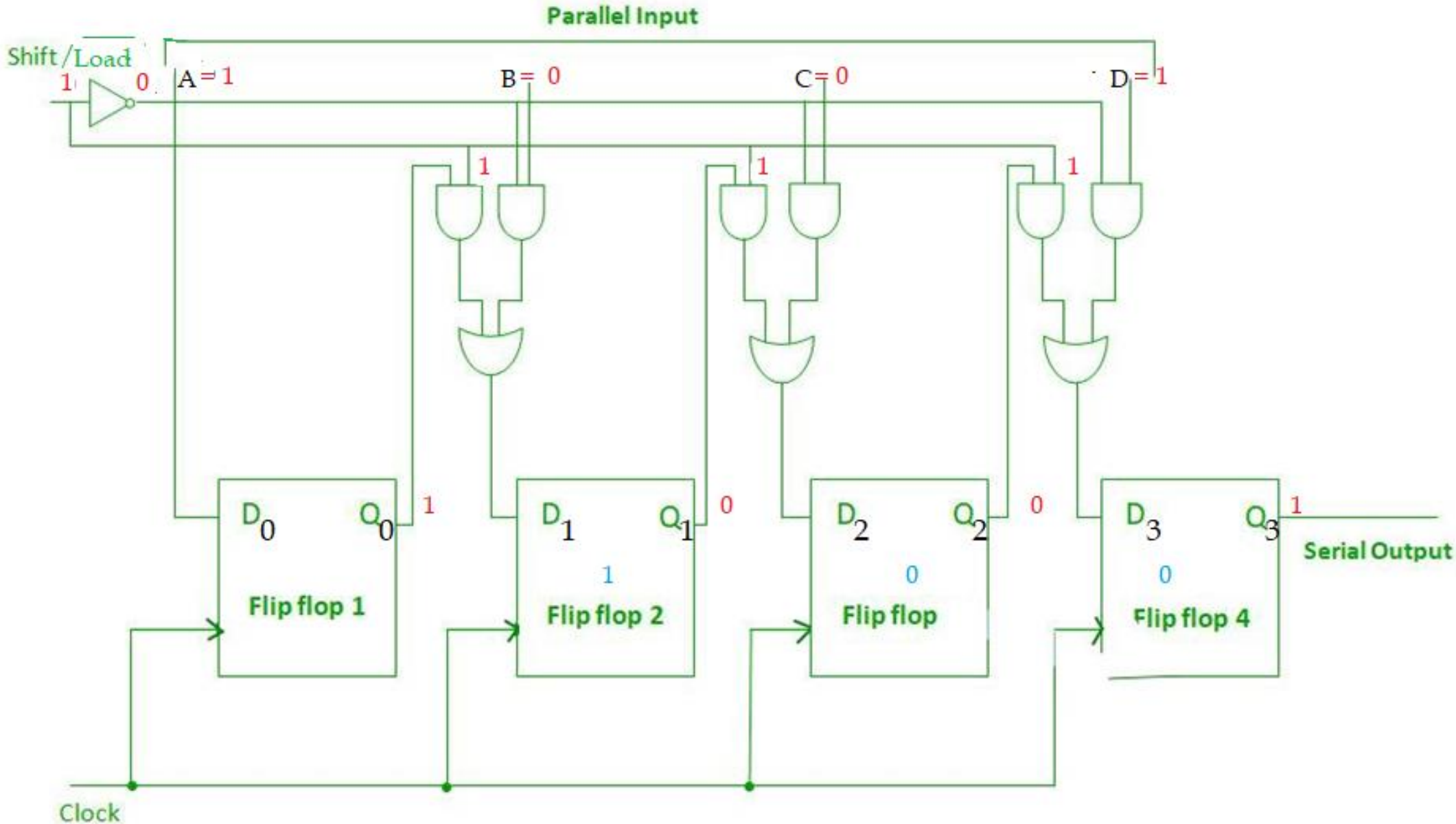
1001

CLK	Q0	Q1	Q2	Q3	Shift/Load’
Initial	0	0	0	0	-
1	1	0	0	1	0

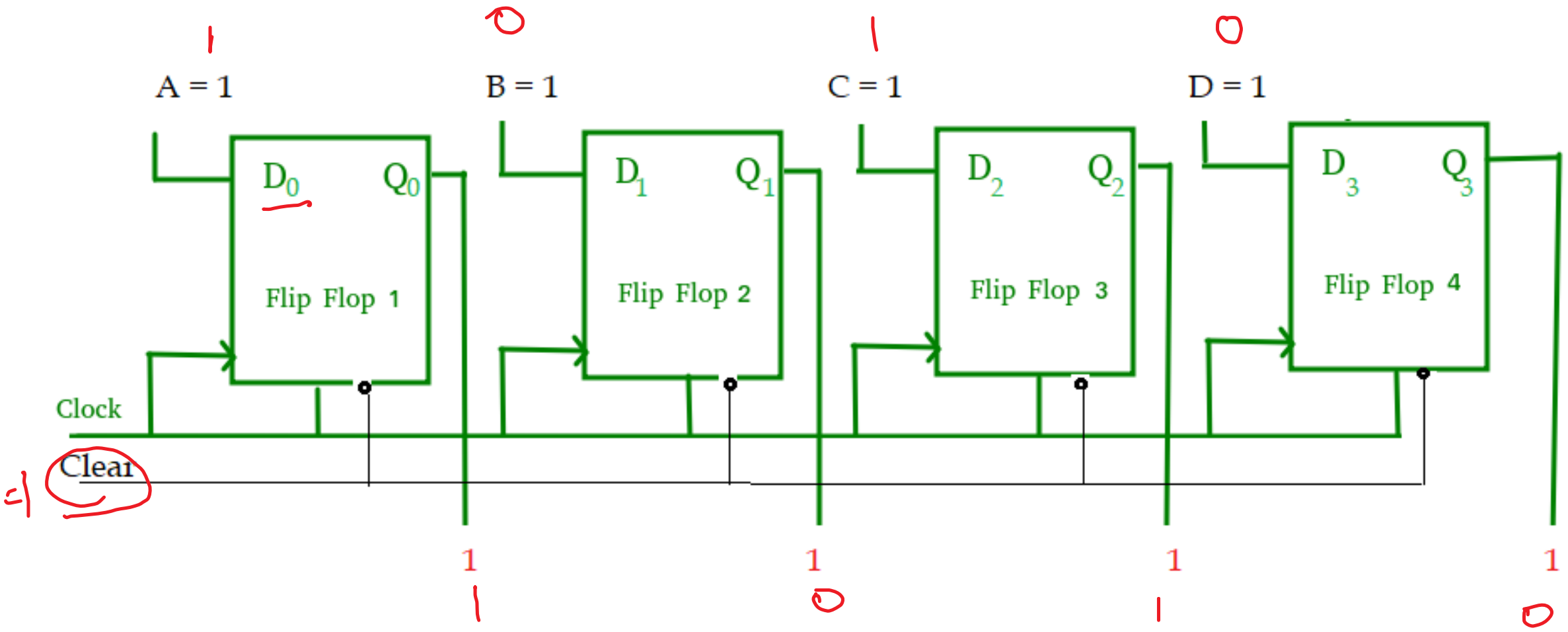
- Read: (n-1) clk pulses are required. i.e. 3 clk pulse. Q3 data is already out at the end of 1st clk pulse.

CLK	Serial Output	Shift/Load’
1	1	0
2	0	1
3	0	1
4	1	1

Read/Retrieve the data:



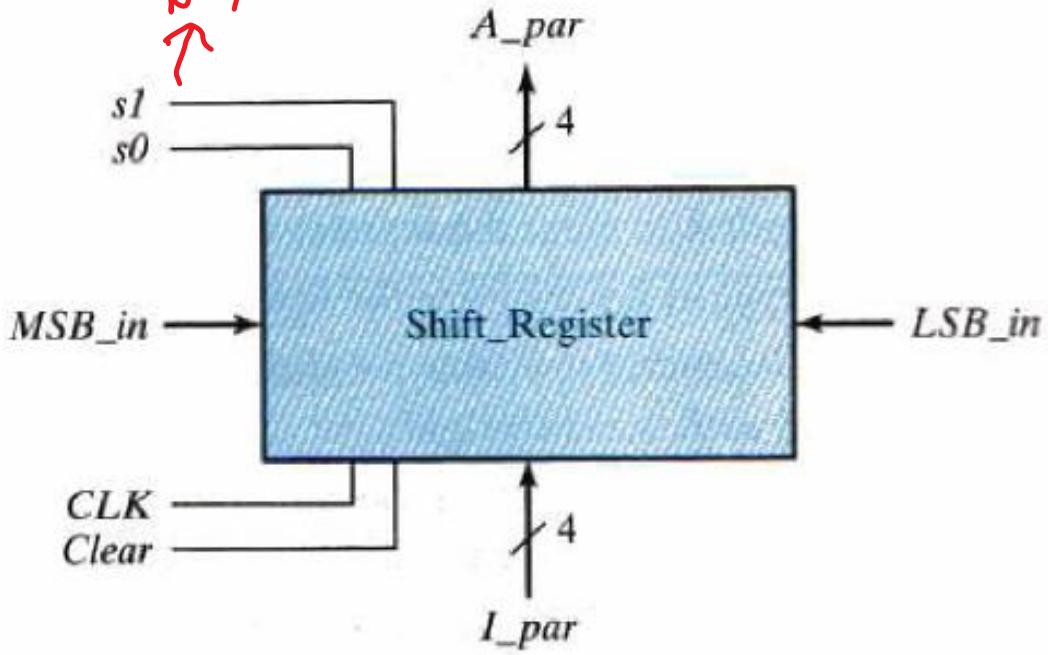
Parallel In and Parallel Out (PIPO) –Storage Register

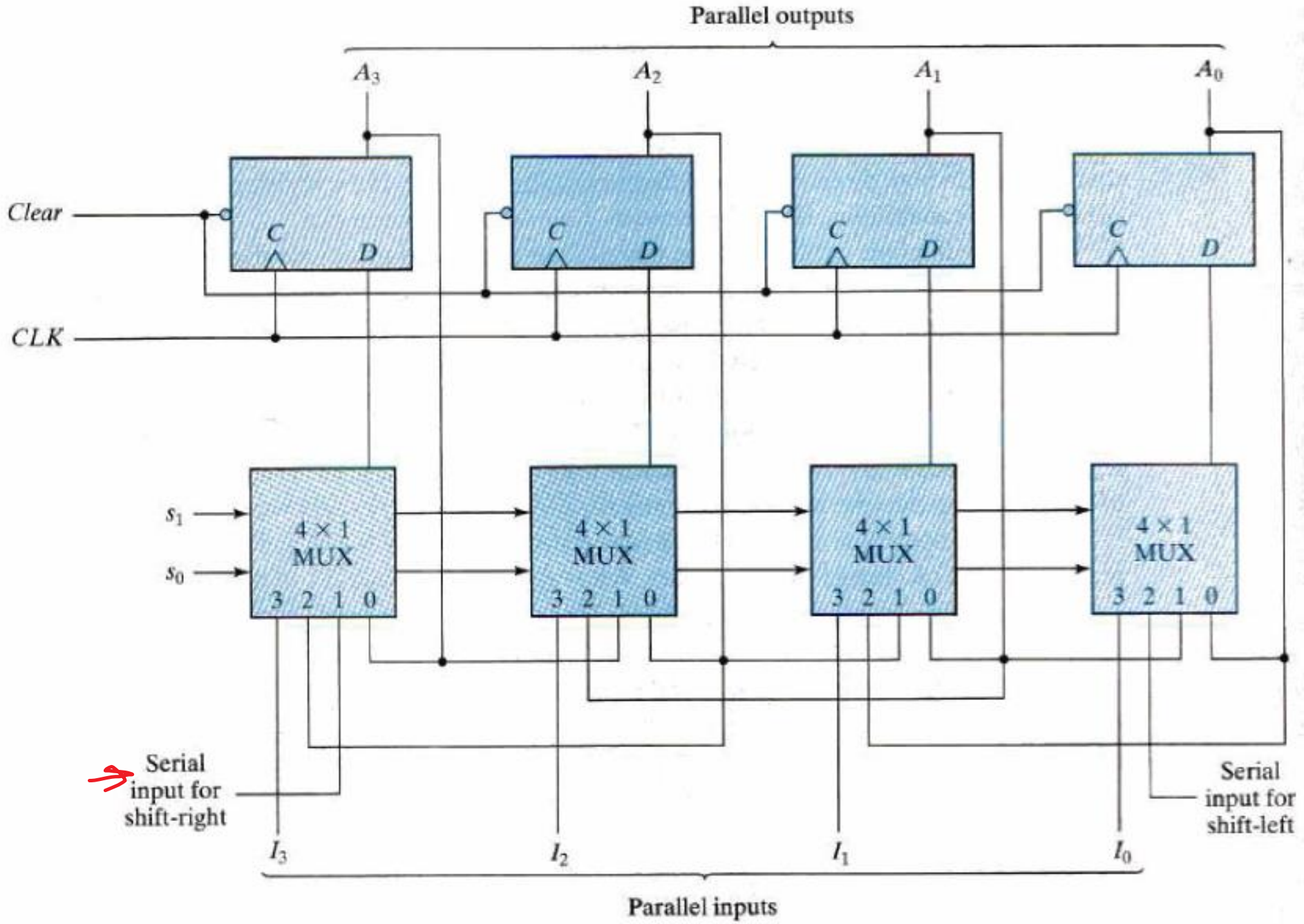


In a single clock pulse both load (read) and write (retrieve) of data is done.

Universal Shift Register

- A register that can store the data and /shifts the data towards the right and left along with the parallel load capability is known as a universal shift register.
- It can be used to perform input/output operations in both serial and parallel modes.
- They are used as memory elements in computers.
- A Unidirectional shift register is capable of shifting in only one direction.
- A bidirectional shift register is capable of shifting in both the directions.
- The Universal shift register is a combinational design of **bidirectional** shift register and a **unidirectional** shift register with parallel load provision.




Parallel outputs
A_3
A_2
A_1
A_0
Clear
CLK
C
D
s_1
s_0
4 × 1
MUX
3 2 1 0
Serial
input for
shift-right
Serial
input for
shift-left
I_3
I_2
I_1
I_0
Parallel inputs

Function Table for the Universal Shift Register:

Mode Control		
s_1	s_0	Register Operation
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

Counters

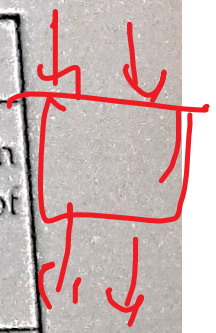
A counter is a register capable of counting the number of clock pulses arriving at its clock input

8.12.3 Synchronous Vs Asynchronous Counters

The Table 8.15 shows the comparison between synchronous and asynchronous counters.

Serial / Ripple / Asynchronous Counters	Parallel / Synchronous Counters
1) In this type of counter flip-flops are connected in such a way that output of first flip-flop drives the clock for the next flip-flop.	1) In this type there is no connection between output of first flip-flop and clock input of the next flip-flop.
2) All the flip-flops are not clocked simultaneously.	2) All the flip-flops are clocked simultaneously.
3) Logic circuit is very simple even for more number of states.	3) Design involves complex logic circuit as number of states increases.
4) Main drawback of these counters is their low speed as the clock is propogated through number of flip-flops before it reaches last flip-flop.	4) As clock is simultaneously given to all flip-flops there is no problem of propagation delay. Hence they are preferred when number of flip-flops increases in the given design.

Table 8.15 Synchronous Vs Asynchronous counters



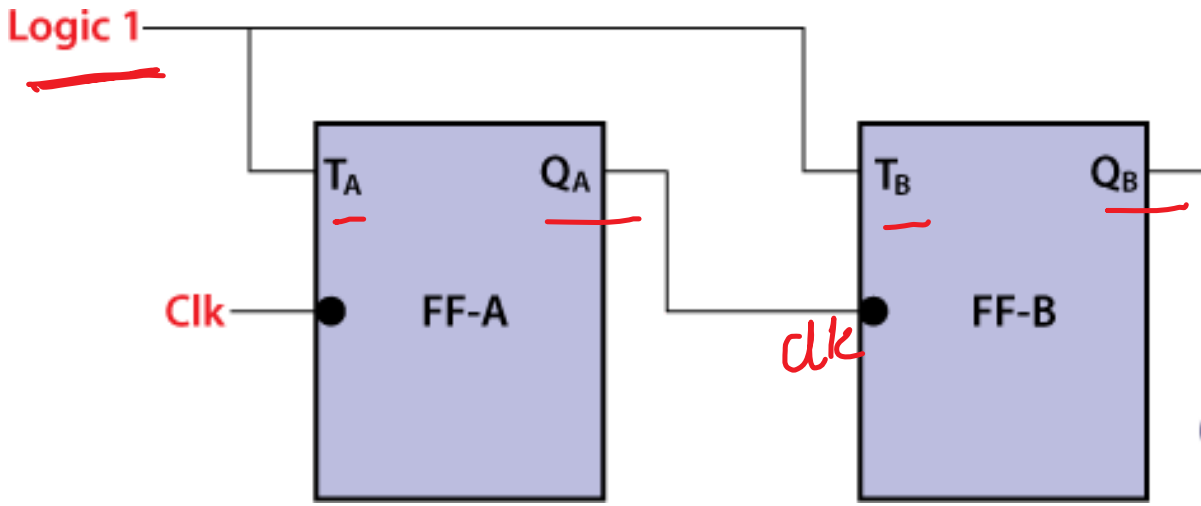
Introduction to Counters

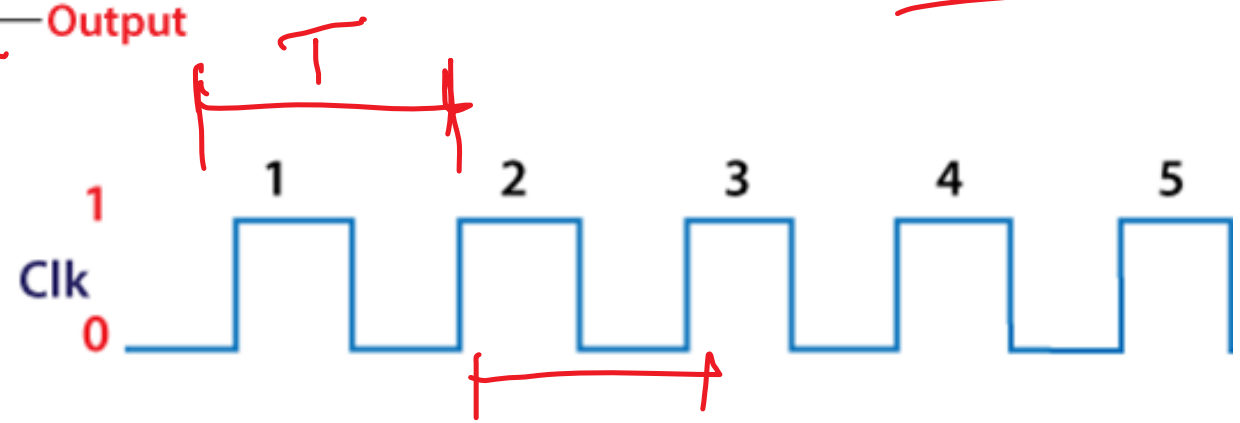
- A register that goes through a prescribed sequence of states upon the application of input pulses is called a counter.
- The input pulses may be clock pulses, or they may originate from some external source and may occur at a fixed interval of time or at random.
- The sequence of states may follow the binary sequence or any other sequence of states.
- A counter that follows the binary number sequence is called a binary counter.
- An n-bit binary counter consists of n flip flops and can count in binary from 0 through 2^n-1.
- Types:
- Asynchronous Counters
- Synchronous Counters

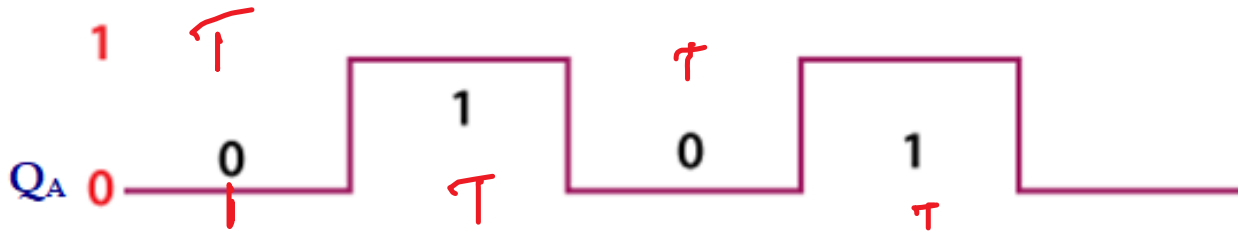
Asynchronous and Synchronous counters

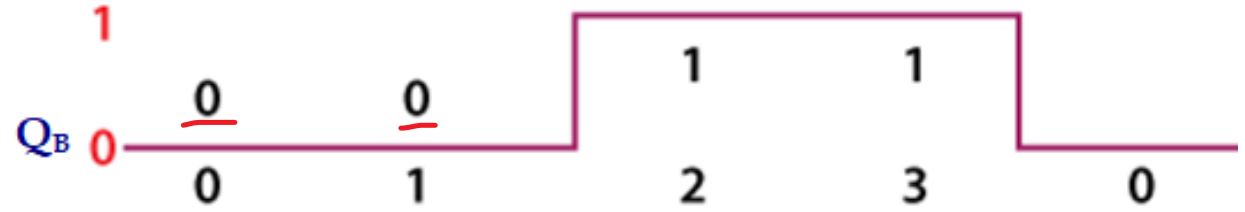
- The **Asynchronous counter** is also known as the **ripple counter**.
- In asynchronous counter we don't use universal clock, only first flip flop is driven by main clock and the clock input of rest of the following flip flop is driven by output of previous flip flops.
- The drawback of this system is that it creates the **counting delay, and the propagation delay** also occurs during the counting stage.
- **The Synchronous counter** has one **global clock connected to clock inputs** of all flip flop so output changes in parallel.
- The advantage of synchronous counter over asynchronous counter is that, it can operate on higher frequency than asynchronous counter as it does not have cumulative delay because of same clock is given to each flip flop.

Asynchronous or Binary Ripple counter (2 bit)



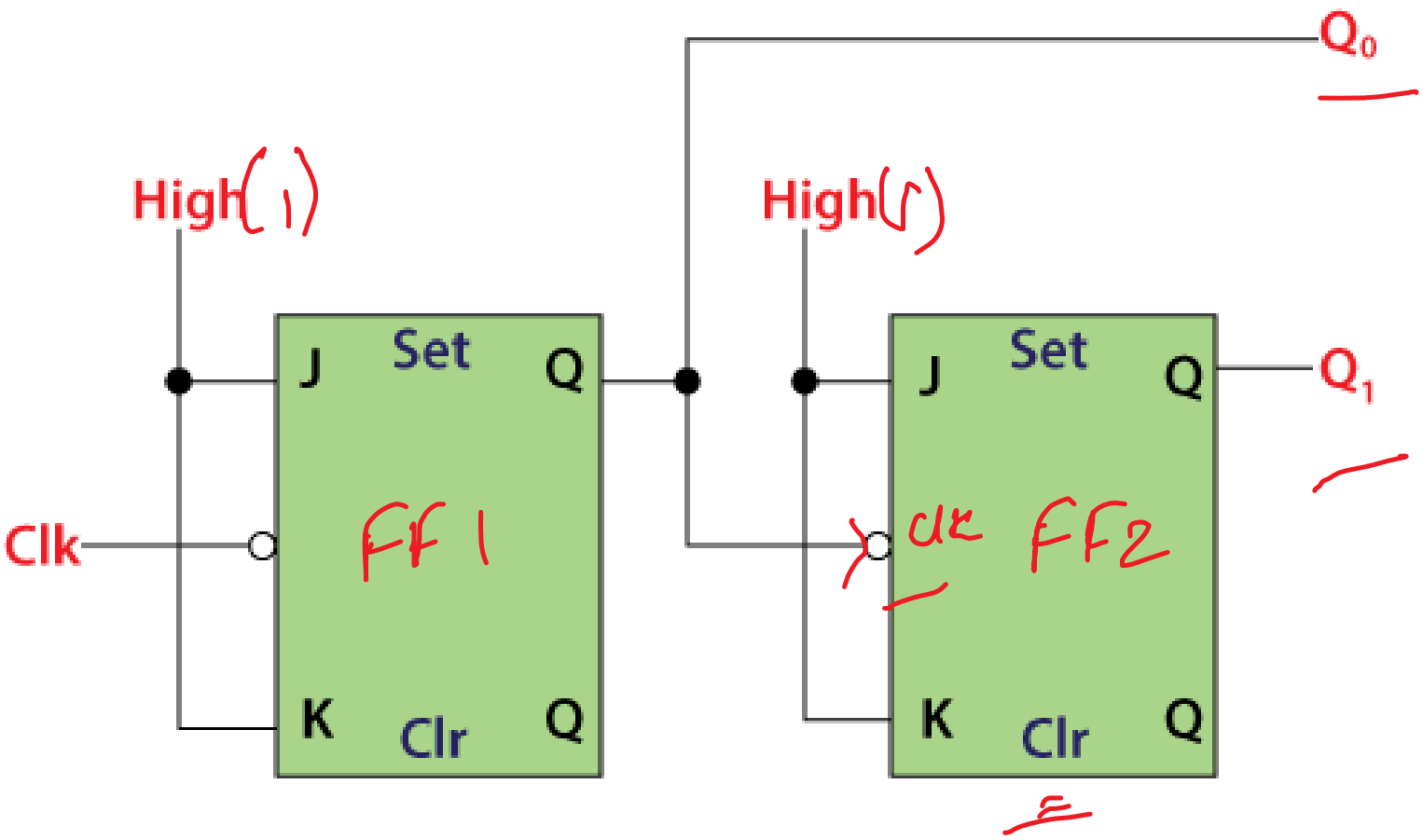




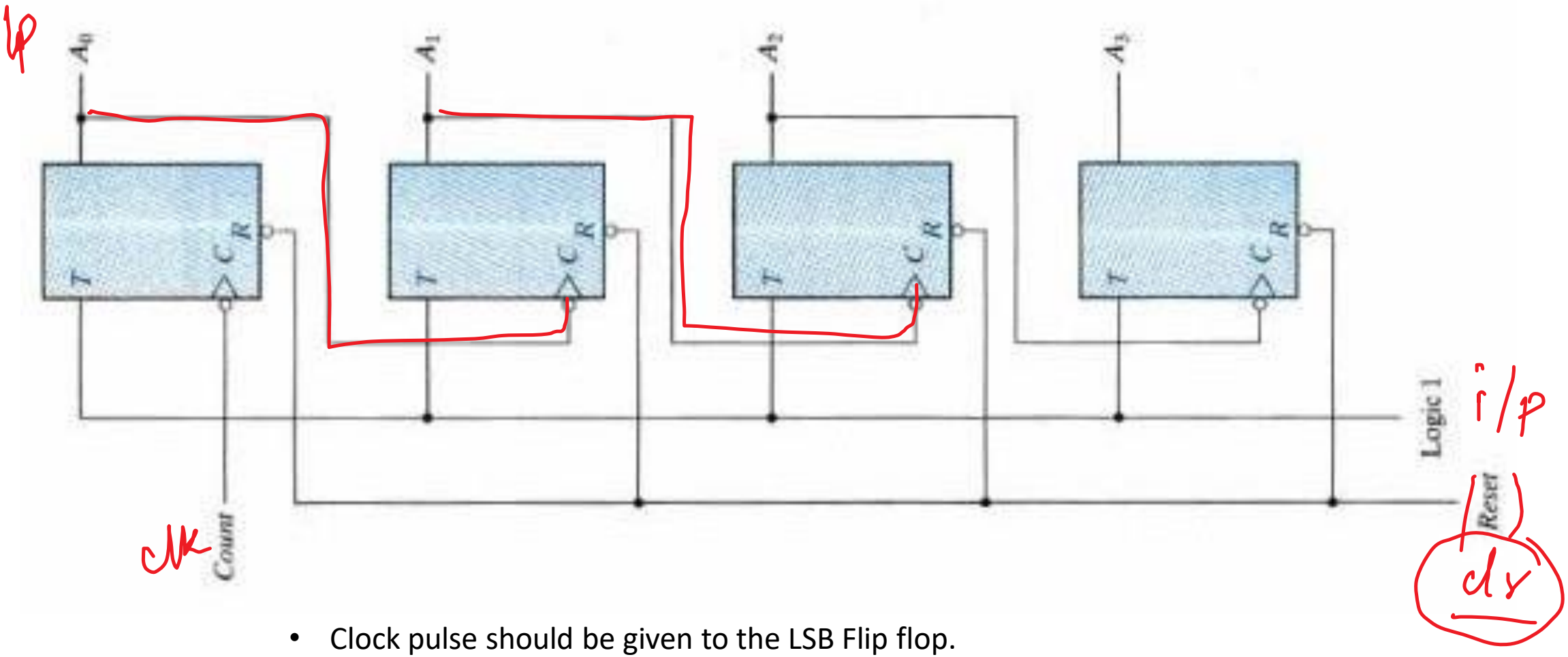


Clk	Q_B	Q_A
0	0	0
1	0	1
2	1	0
3	1	1
4	0	0
5	0	1

Binary Ripple counter using JK flip flop



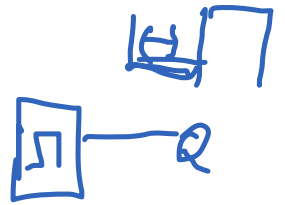
Binary Ripple counter (4 bit) Using T Flip flop



- Clock pulse should be given to the LSB Flip flop.
- Q of one stage will act as clock input for the successive stage.
- JK FF can also be used for ripple counter, connect J and K together and apply high input.

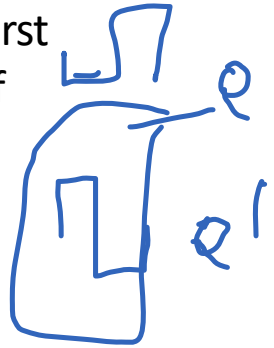
Clk	A3 (MSB)	A2	A1	A0 (LSB)
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Binary Count Sequence:



Binary up counter:

- **For negative edge clock pulse, Q** of first flop should be given as clock input of next FF.
- **Positive edge triggering clk pulse, Q'** acts as clk pulse for successive stage

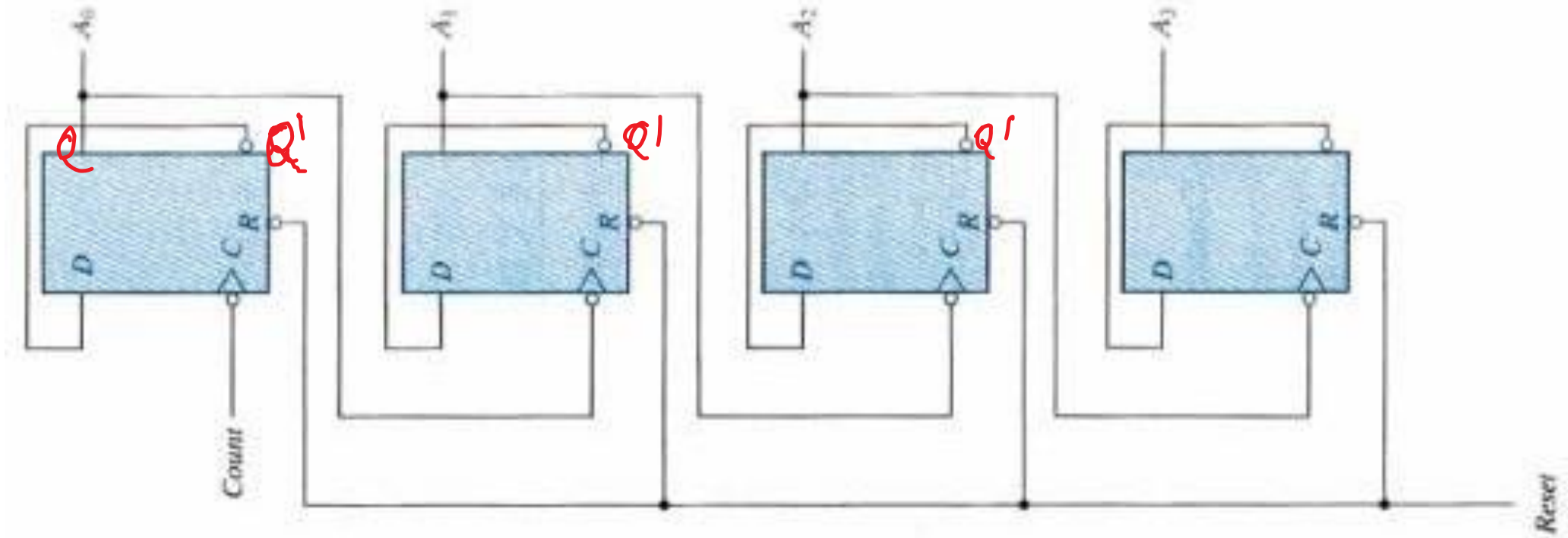


Binary Down counter:

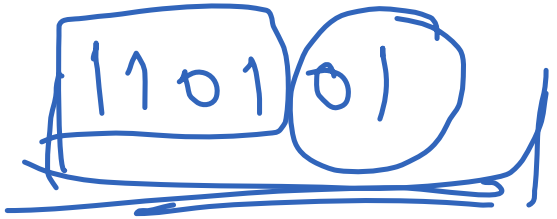
- **For negative edge clock pulse , Q'** of first flop should be given as clock input of next FF.
- **Positive edge triggering clk pulse, Q** acts as clk pulse for successive stage.

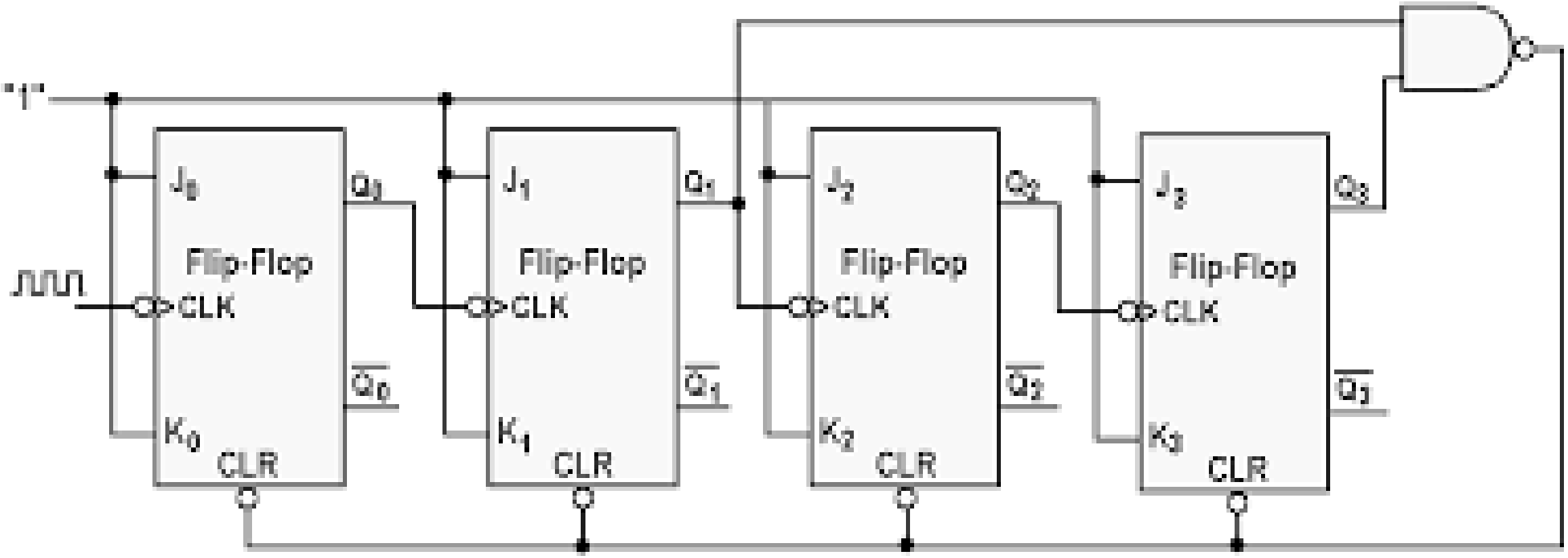
Clk	Up counter	Down counter
-ve edge	Q --- clk pulse	Q'---Clk pulse
+ve edge	Q'---Clk pulse	Q --- clk pulse

4 bit Binary Ripple counter using D flip flop



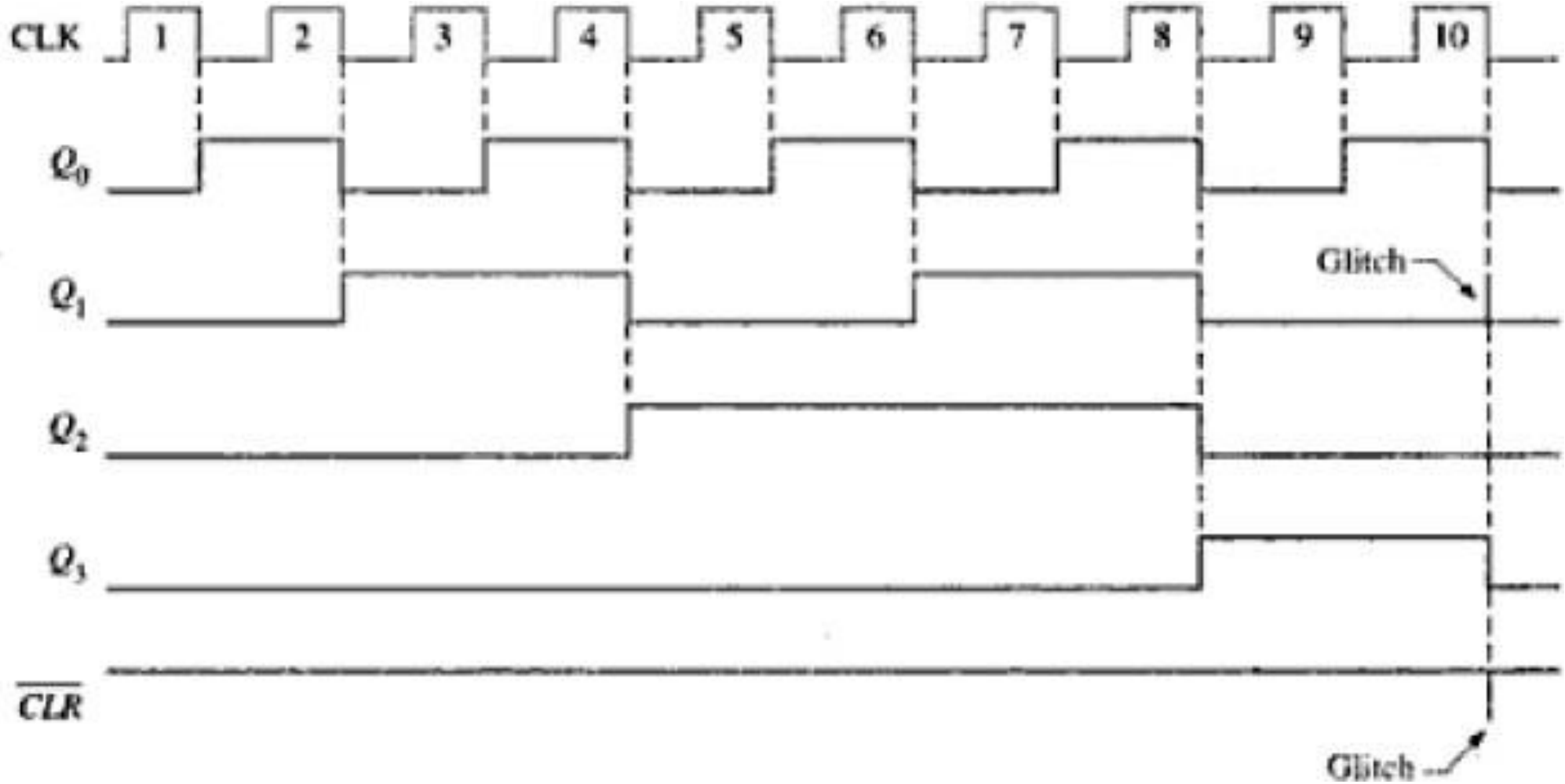
Decade Ripple Counter



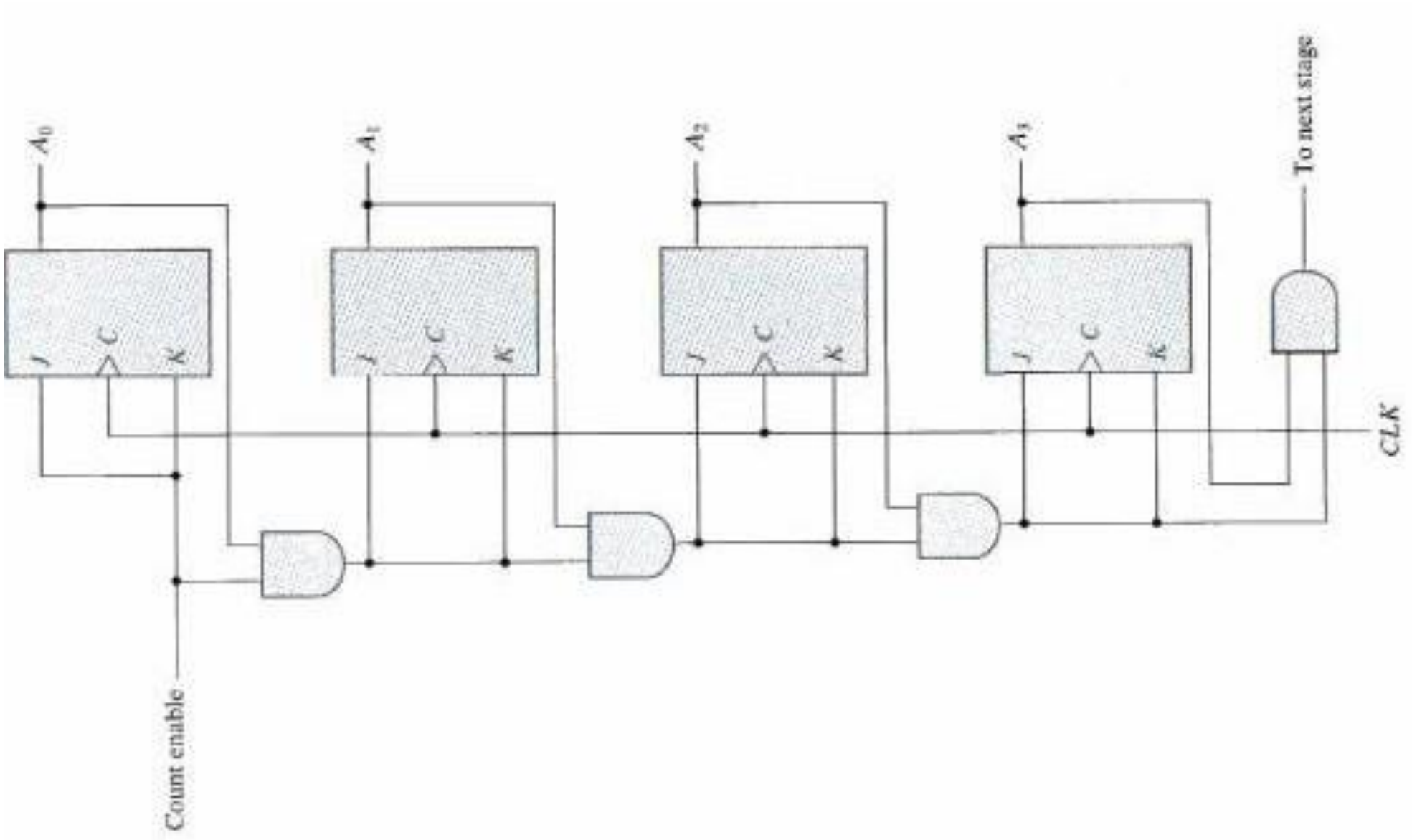


Clock Pulse	Q3	Q2	Q1	Q0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

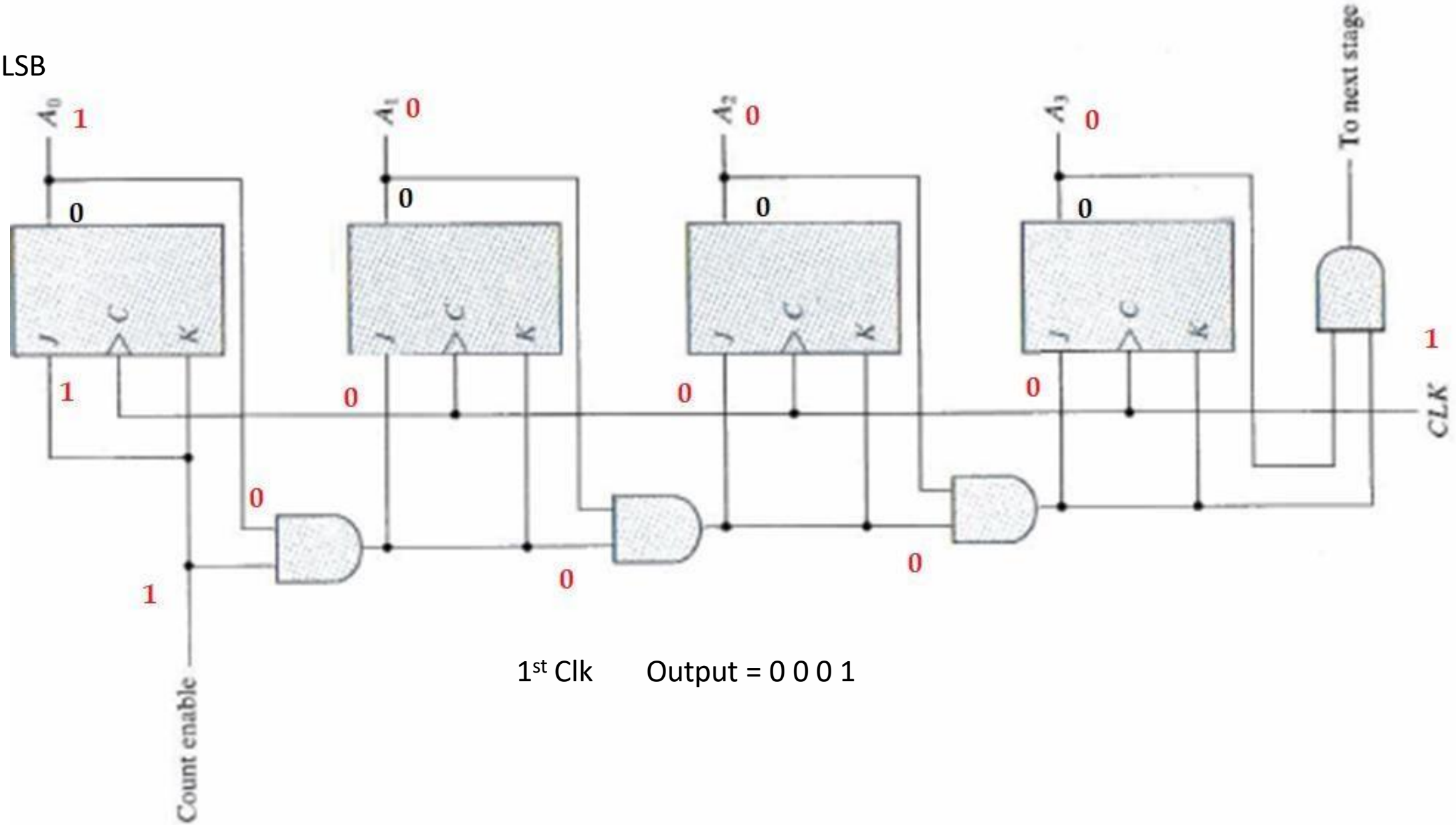
10 1 0 1 0


CLK
1
2
3
4
5
6
7
8
9
10
Q_0
Q_1
Q_2
Q_3
$\overline{CLR}$
Glitch
Glitch

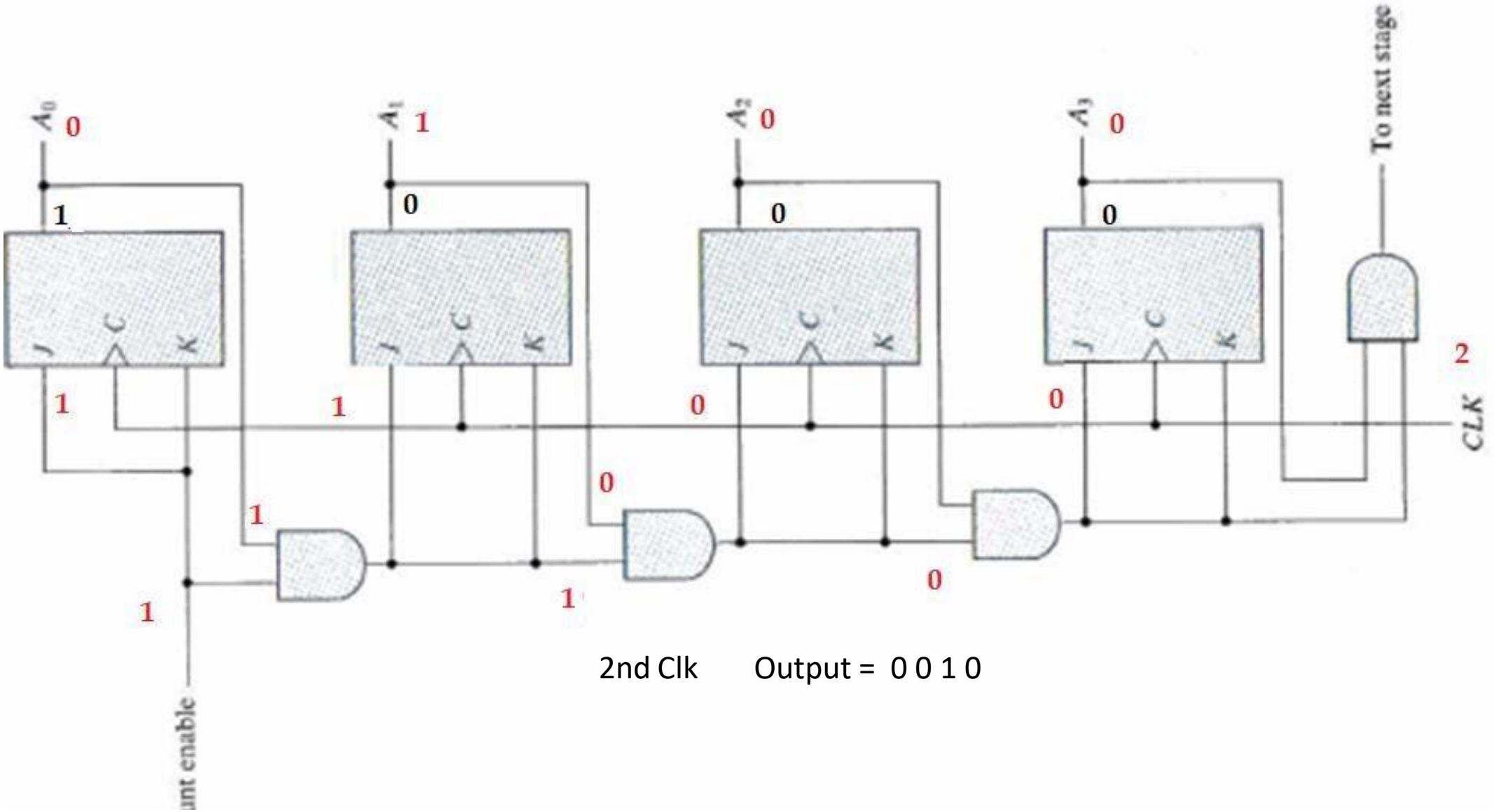
Synchronous Binary Up Counter



LSB



1st Clk Output = 0 0 0 1



2nd Clk Output = 0 0 1 0

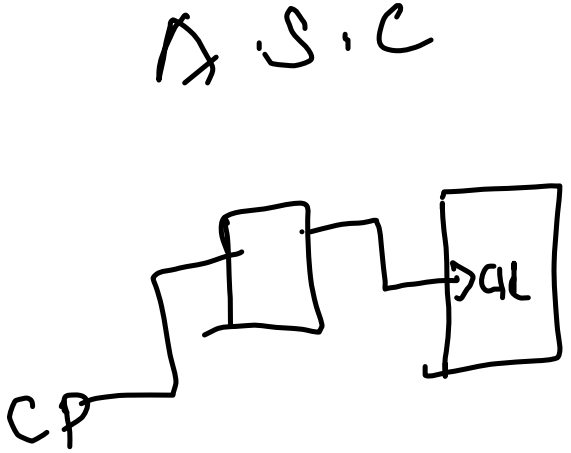
Clk	A3 (MSB)	A2	A1	A0 (LSB)
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Synchronous binary down counter can be constructed from the same circuit of upcounter with a small change.

Input to the AND gates should be taken from Q complement.

Design of Synchronous Counters

- Step 1:Decide the number of flip flops and type of FF
- Step 2: Excitation table of FF chosen.
- Step 3: State diagram and
- Step 4: Circuit excitation table. (State table)
- Step 4: Obtain simplified equation using K map.
- Step 5: Draw the logic diagram.



Design of 2- bit synchronous up counter or MOD-4 counters

- Step 1:

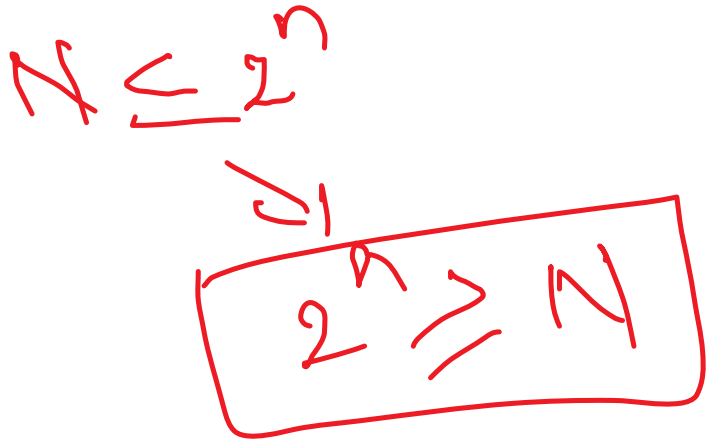
  n = 2 bit, number of FF = 2

  Type of FF = JK

  Range = $2^2 - 1 = 3$; (0,1,2,3);

  Number of states = 4

- Step 2:

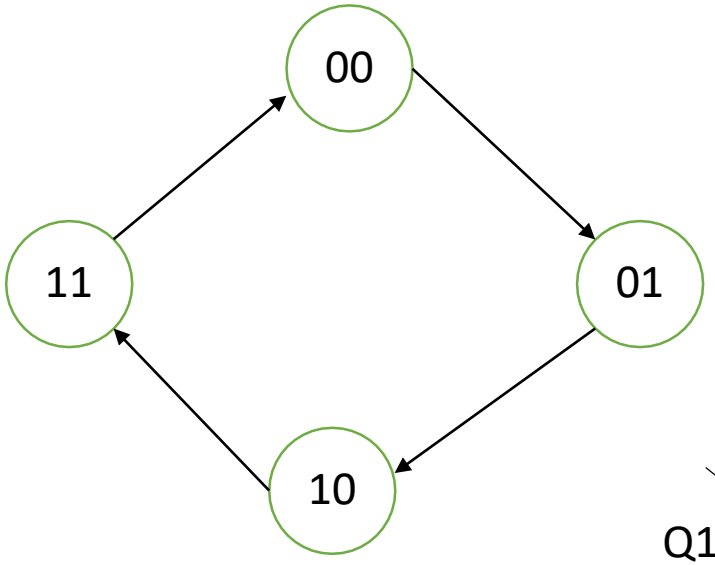


Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Excitation table.

- Step 3:

State diagram



Step 4: Circuit Excitation table

Q_1	Q_2	Q_1^+	Q_2^+	J1	K1	J2	K2
0	0	0	1	0	X	1	X
0	1	1	0	1	X	X	1
1	0	1	1	X	0	1	X
1	1	0	0	X	1	X	1

Step 5: K map

Q1 \ Q2	0	1
0	0	1
1	X	X

J1 = Q2

Q1 \ Q2	0	1
0	X	X
1	0	1

K1 = Q2

Q1 \ Q2	0	1
0	1	X
1	1	X

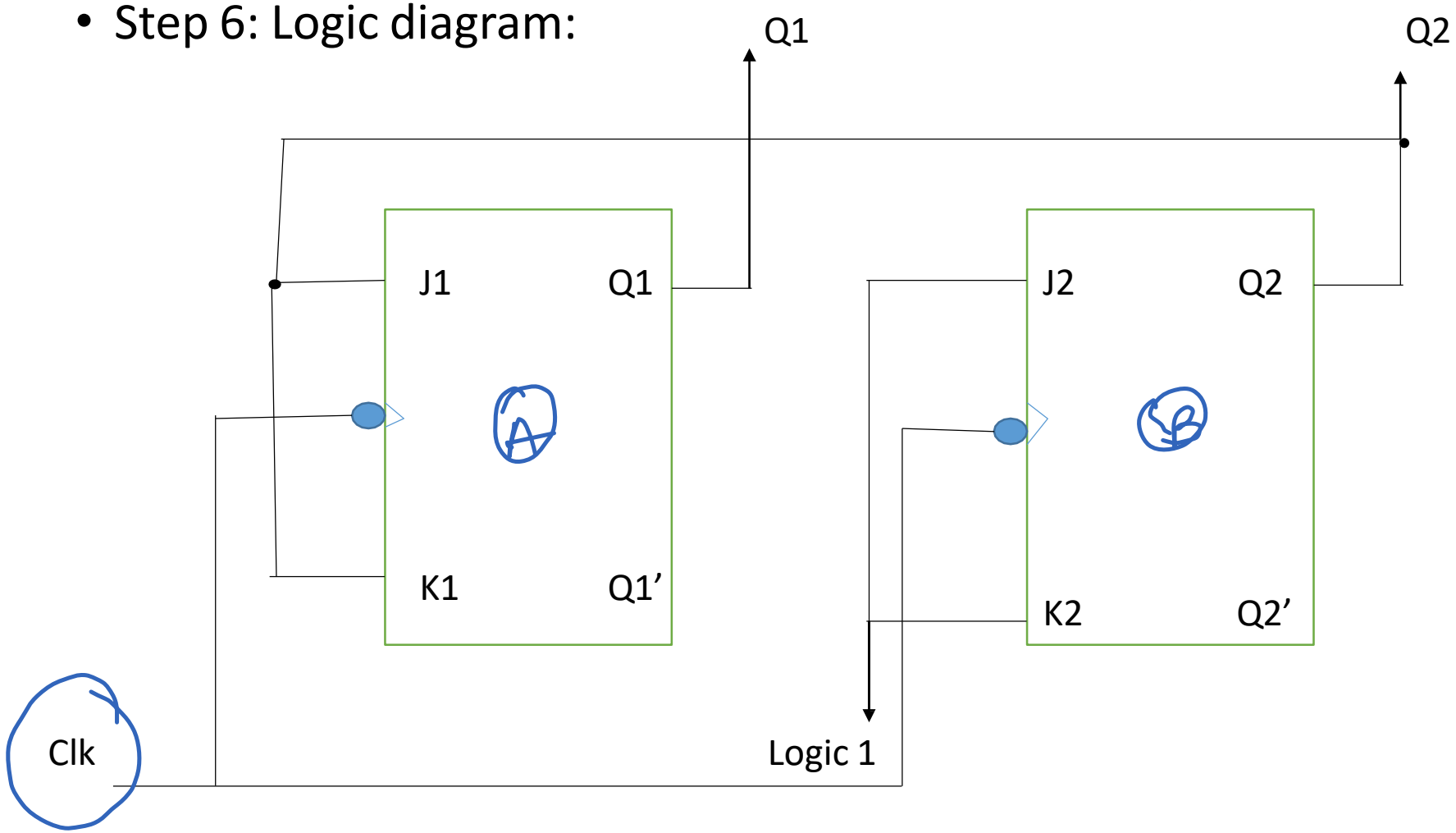
J2 = 1

Q1 \ Q2	0	1
0	X	1
1	X	1

K2 = 1

Sync. Counter

- Step 6: Logic diagram:



Design of 3- bit synchronous up counter or MOD- 8 Counters

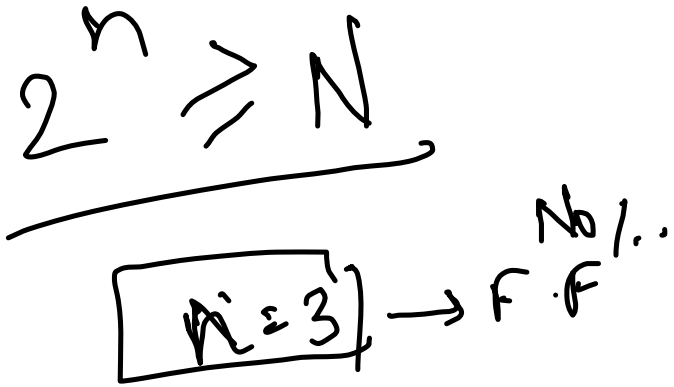
- Step 1:

  n = 3 bit, number of FF = 3

  Type of FF = T

  Range = $2^3 - 1 = 7$; (0,1,2,3,4,5,6,7);

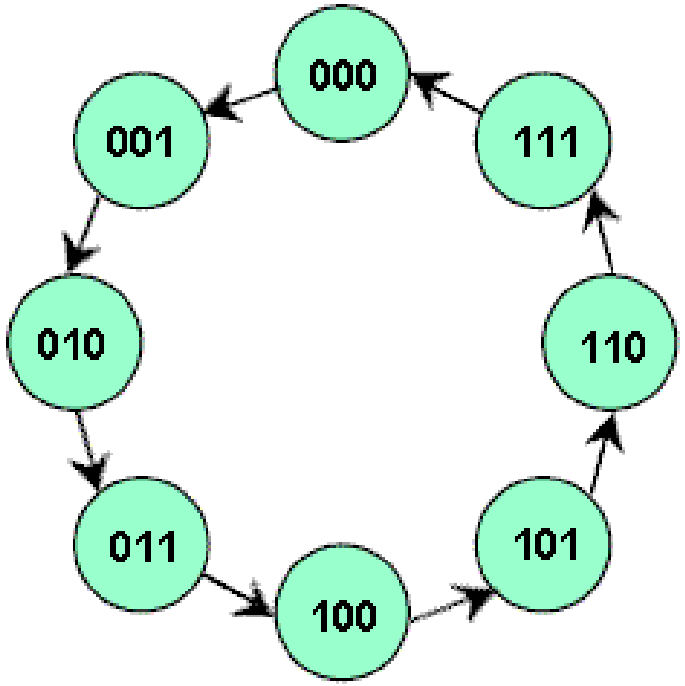
  Number of states = 8



- Step 2:

Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

Step 3: State Diagram



Step 4: State table

Present state			Next State			Input		
Q_C	Q_B	Q_A	Q^+_C	Q^+_B	Q^+_A	T_C	T_B	T_A
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

Step 5: K map Reduction

K map for T_C

Q_C \ $Q_B Q_A$	00	01	11	10
0	0	0	1	0
1	0	0	1	0

$T_C = Q_A Q_B$

K map for T_A

Q_C \ $Q_B Q_A$	00	01	11	10
0	1	1	1	1
1	1	1	1	1

$T_A = 1$

K map for T_B

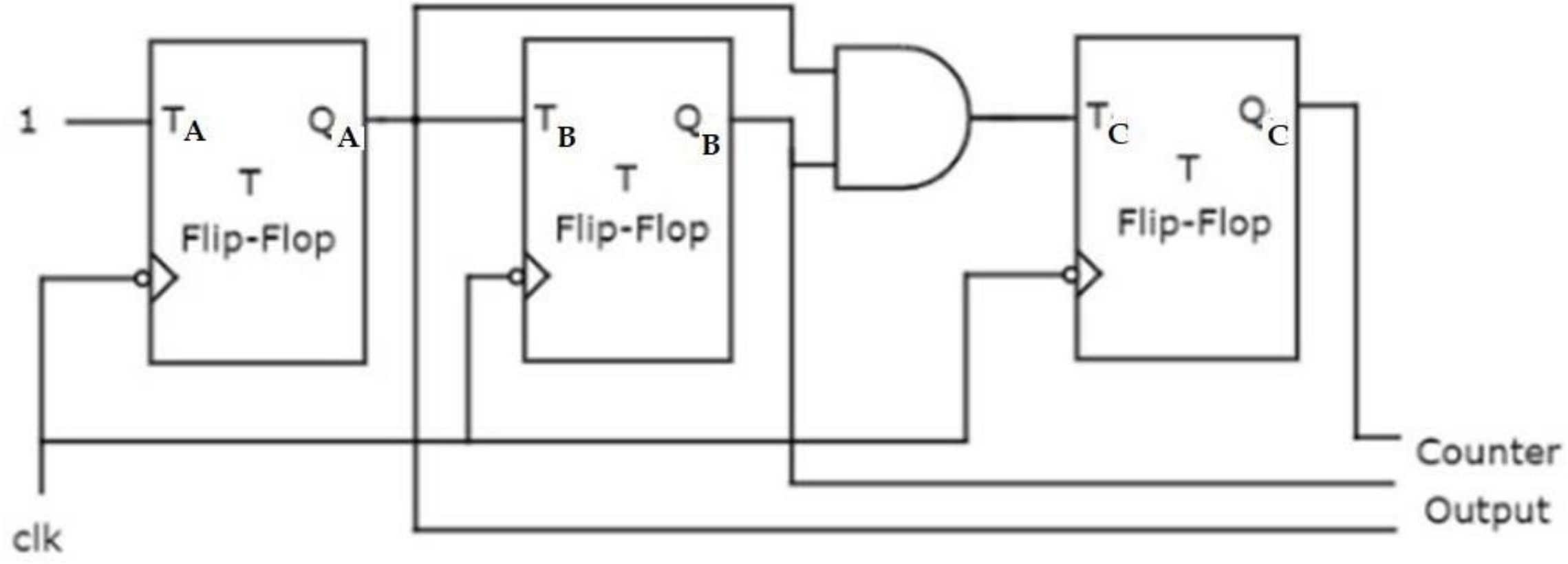
Q_C \ $Q_B Q_A$	00	01	11	10
0	0	1	1	0
1	0	1	1	0

$T_B = Q_A$

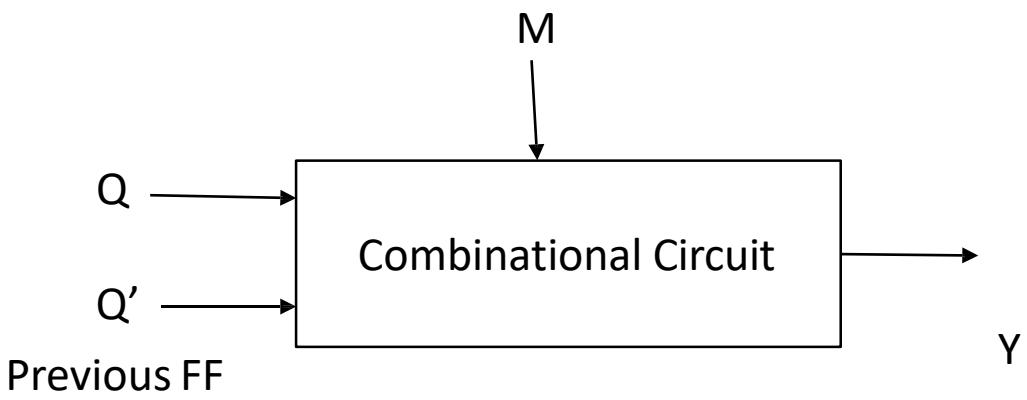
1 clk to all the F.F

Design of 3- bit synchronous up counter

Step 6: Logic diagram



3 bit Up/Down Ripple counter



M = Mode control input

M = 1 ….. Up counting (Q acts as clk)

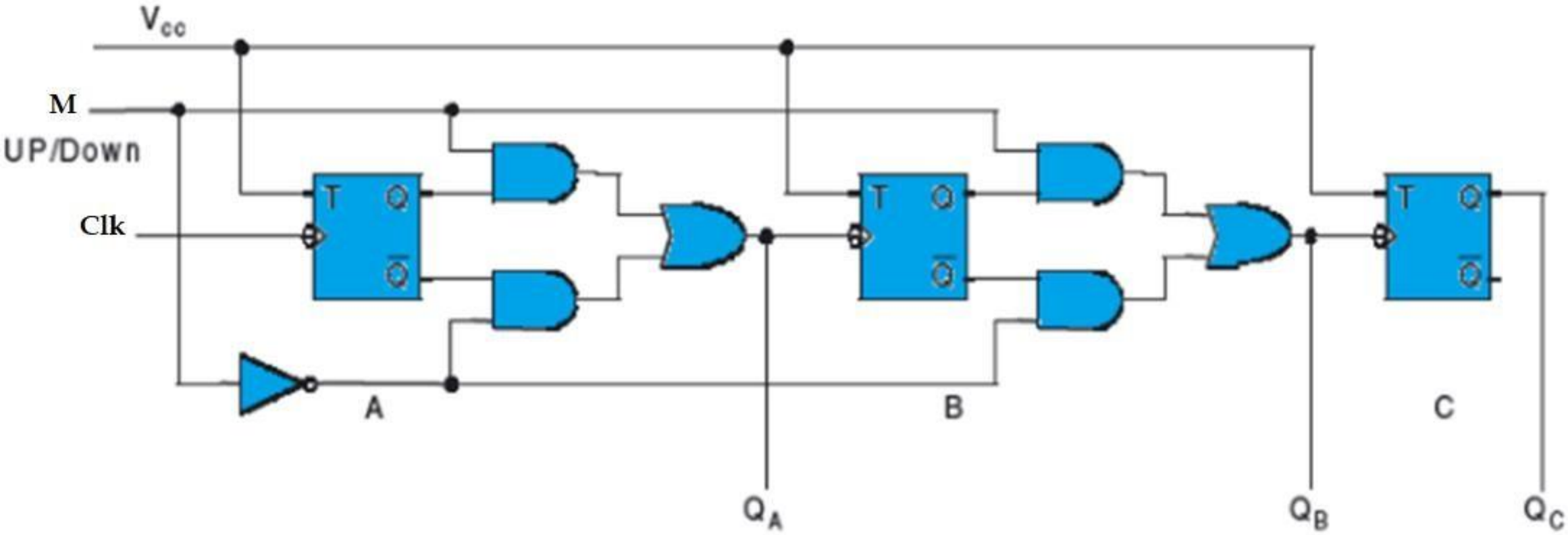
M = 0 …… Down counting (Q' acts as clk)

M	Q'	Q	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

M \ Q'Q	00	01	11	10
0	0	0	1	1
1	0	1	1	0

Y = MQ + M'Q'

3 bit Up/Down Ripple counter



M = 1, It will perform Up counting
M = 0, It will perform down counting

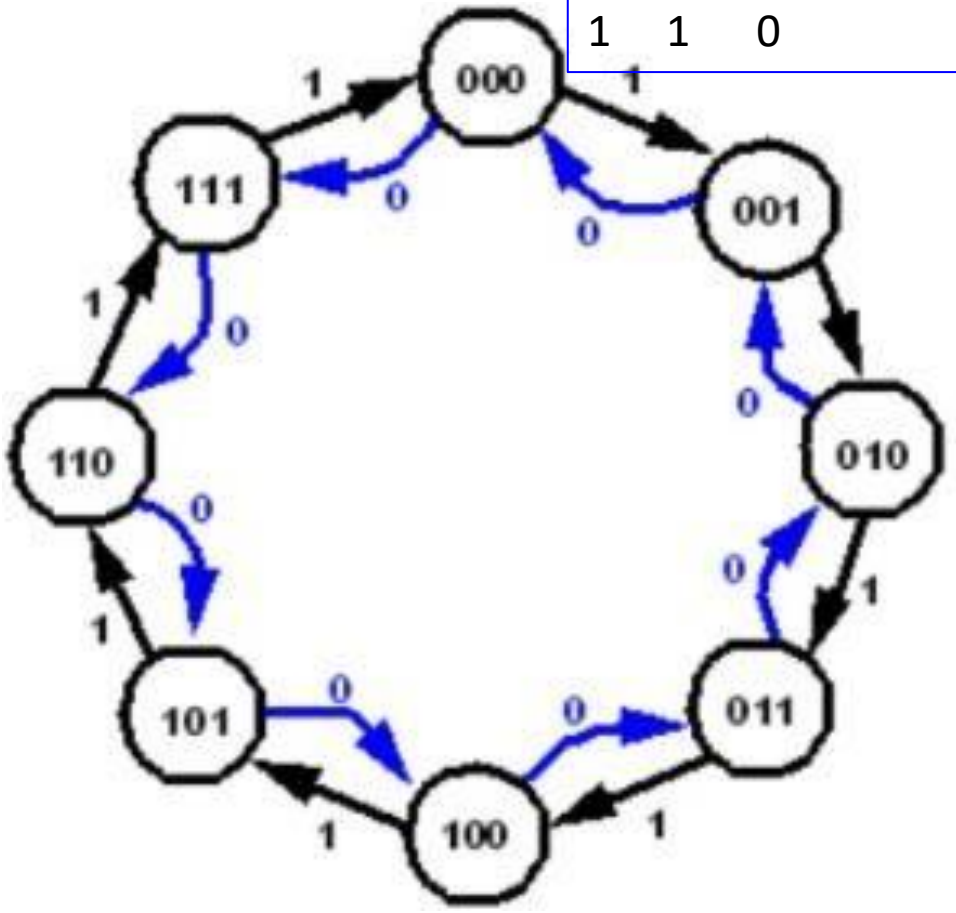
MOD 8 or 3 bit Up/Down Synchronous counter

M = 1 ---→ Up counter

M = 0 ---→ Down counter

excitation table T FF

P	N	T
0	0	0
0	1	1
1	0	1
1	1	0



Control Input M	Present state Q_C	Q_B	Q_A	Next State Q^+_C	Q^+_B	Q^+_A	Input T_C	T_B	T_A
0	0	0	0	1	1	1	1	1	1
0	0	0	1	0	0	0	0	0	1
0	0	1	0	0	0	1	0	1	1
0	0	1	1	0	1	0	0	0	1
0	1	0	0	0	1	1	1	1	1
0	1	0	1	1	0	0	0	0	1
0	1	1	0	1	0	1	0	1	1
0	1	1	1	1	1	0	0	0	1
1	0	0	0	0	0	1	0	0	1
1	0	0	1	0	1	0	0	1	1
1	0	1	0	0	1	1	0	0	1
1	0	1	1	1	0	0	1	1	1
1	1	0	0	1	0	1	0	0	1
1	1	0	1	1	1	0	0	1	1
1	1	1	0	1	1	1	0	0	1
1	1	1	1	0	0	0	1	1	1

$M\ Q_C$ \ $Q_B\ Q_A$	0 0	0 1	1 1	1 0
0 0	1	0	0	0
0 1	1	0	0	0
1 1	0	0	1	0
1 0	0	0	1	0

$T_C = M\ Q_B\ Q_A + M'\ Q'_B\ Q'_A$

$M\ Q_C$ \ $Q_B\ Q_A$	0 0	0 1	1 1	1 0
0 0	1	0	0	1
0 1	1	0	0	1
1 1	0	1	1	0
1 0	0	1	1	0

$T_B = M\ Q_A + M'Q'_A$

$M\ Q_C$ \ $Q_B\ Q_A$	0 0	0 1	1 1	1 0
0 0	1	1	1	1
0 1	1	1	1	1
1 1	1	1	1	1
1 0	1	1	1	1

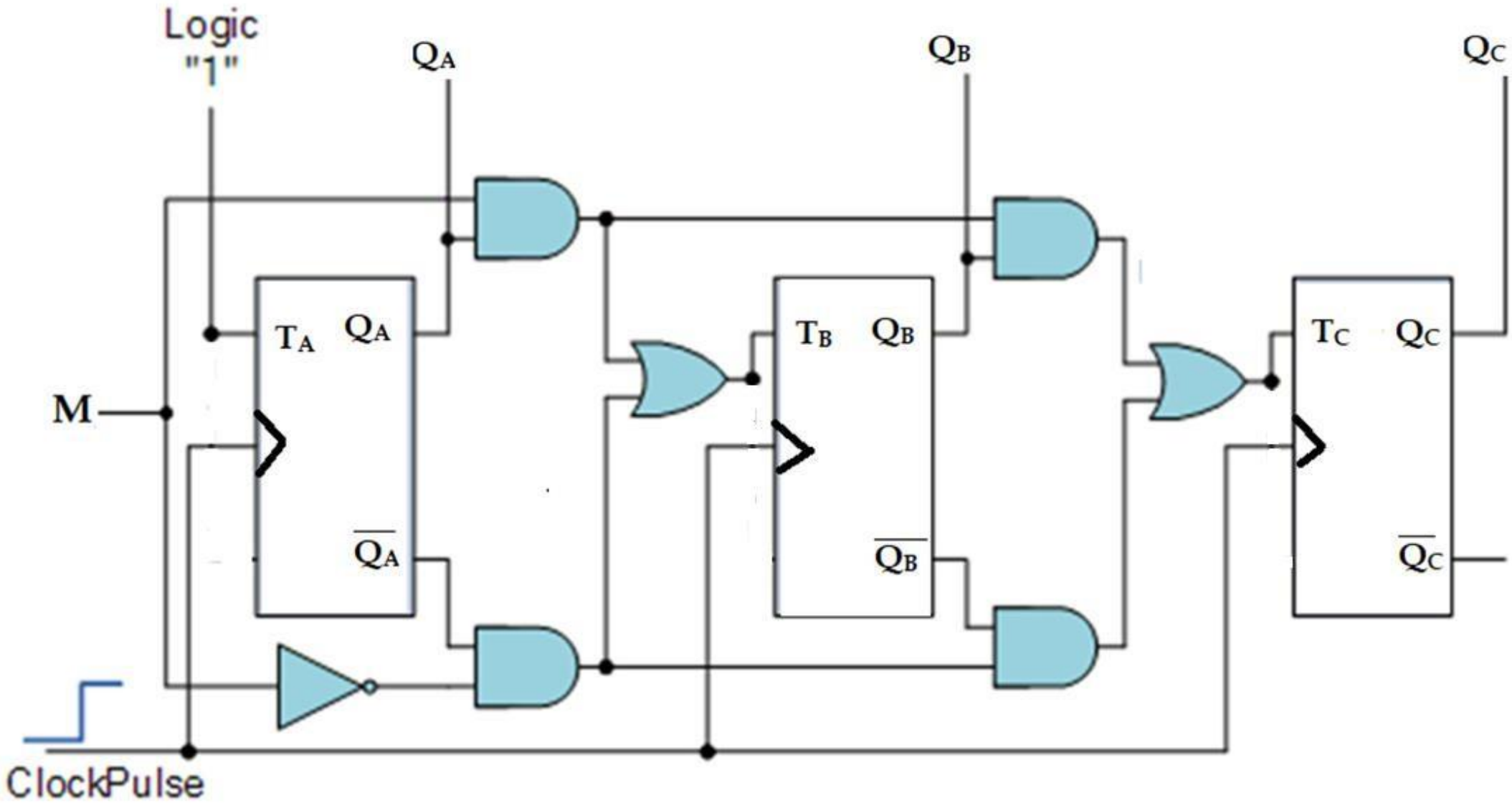
$T_A = 1$

$T_A = 1$

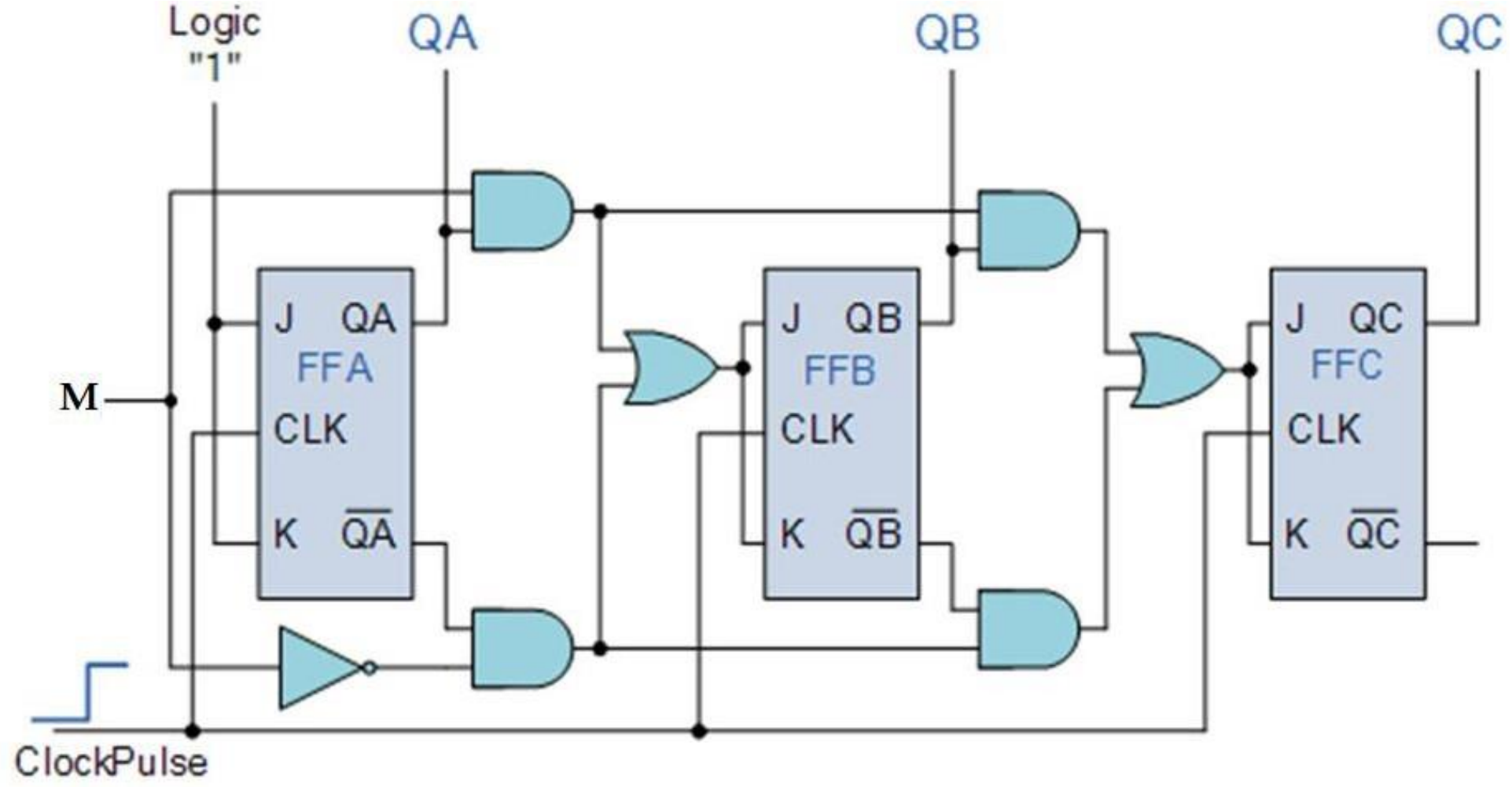
$T_B = M\ Q_A + M'Q'_A$

$T_C = M\ Q_B\ Q_A + M'\ Q'_B\ Q'_A$

3 bit Up/Down Synchronous counter

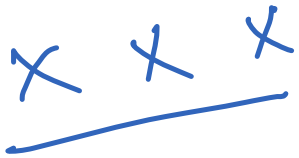


3 bit Up/Down Synchronous counter



Modulo, Ring and Johnson Counters

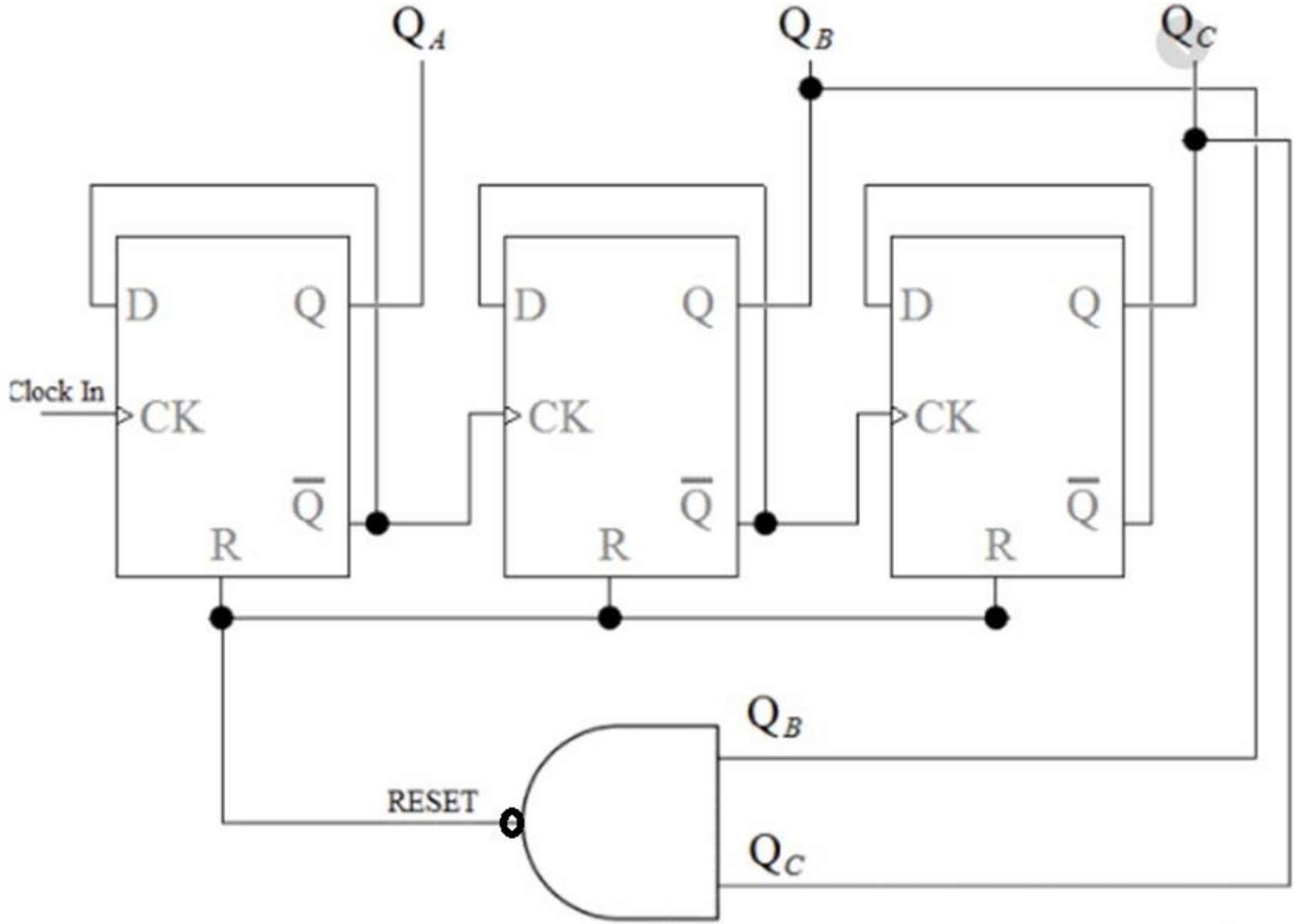
Modulo – N Counter



- Counters can be designed to generate any desired sequence of states.
- A divide by N counter is a counter that goes through a repeated sequence of N states also known as Modulo- N counter.
- The sequence may follow the binary count or may be any other arbitrary sequence.
- Eg. 2 bit ripple counter (4 states) is called as MOD- 4 or modulus 4 counter
- 3 bit ripple counter (8 states) is called as MOD-8 or modulus 8 counter.
- Modulus will give the number of states in the counter.
- 2 bit up/down counter – MOD -4
- 3 bit Up/Down counter – MOD-8

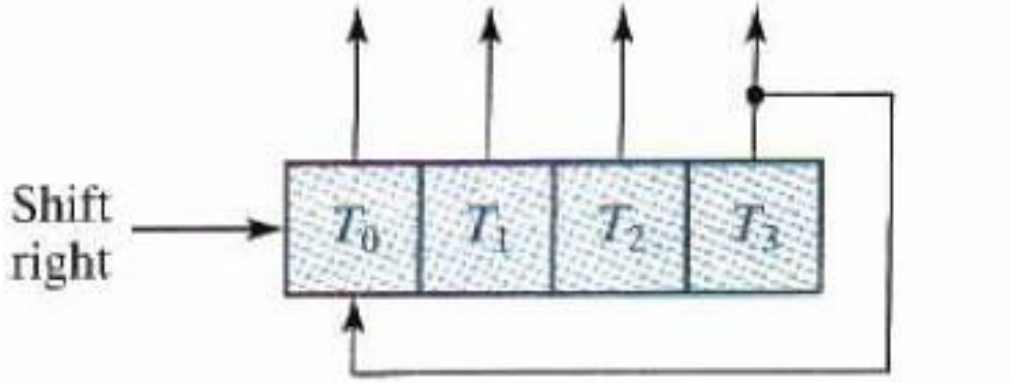
MOD- 6 counter

- Design of MOD-6 counter (using MOD-8 counter)
- States = 6
- Number of FF = 3
- Maximum count = 6; Range = (0,1,2,3,4,5)
- Choose D FF
- 000---→ 001--→ 010--→ 011--→100---→101 --→(110 & 111—Not required---Should be Reset).


Q_A
Q_B
Q_C
D
Q
CK
$\overline{Q}$
R
Clock In
Q_B
RESET
Q_C

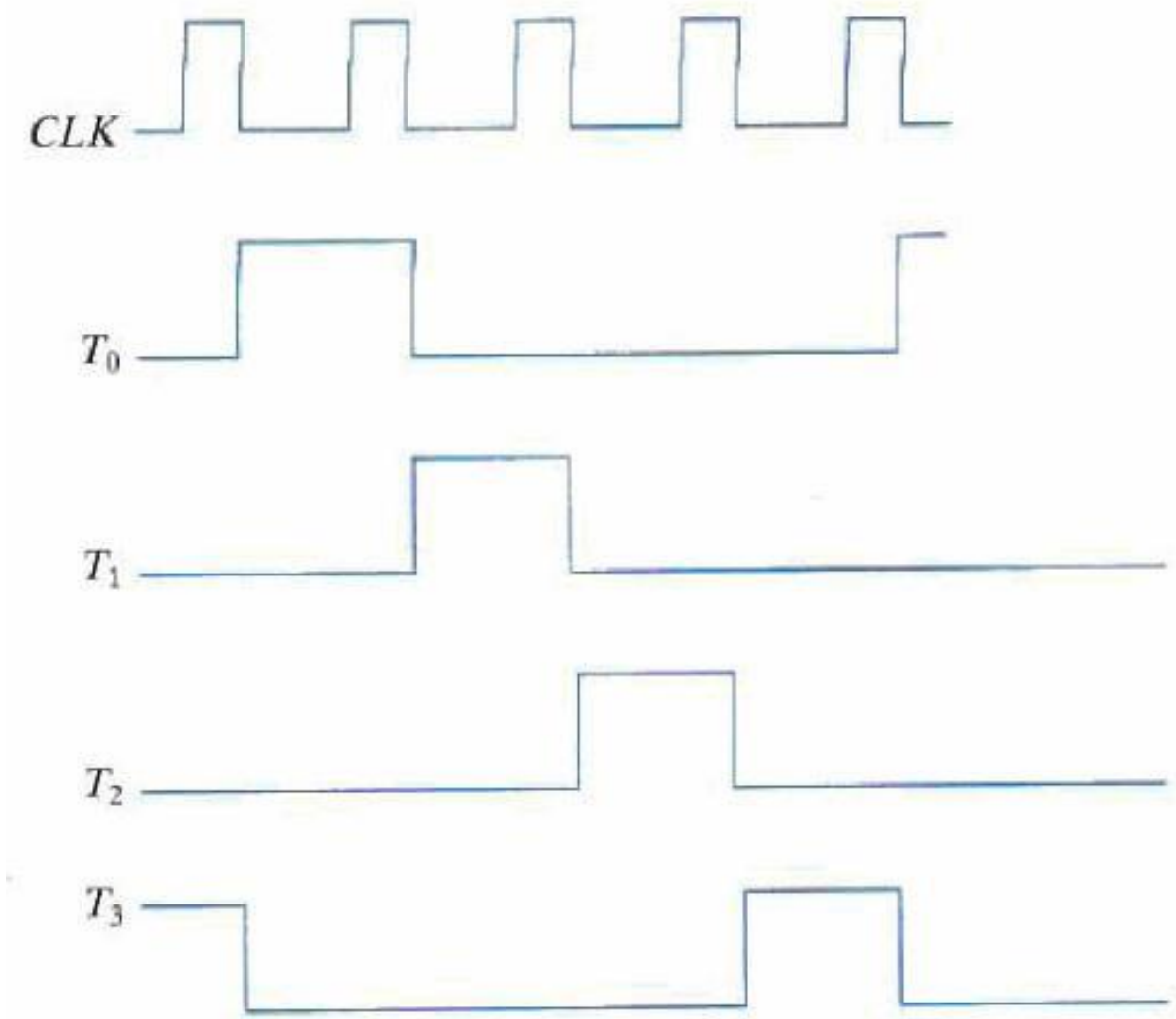
Ring Counter

- A ring counter is a circular shift register with only one flip flop being set at any particular time; all others are cleared.
- The single bit is shifted from one flip flop to the next to produce the sequence of timing signals.
- To generate 2^n timing signals, we need either a shift register with 2^n flip flops or an n-bit binary counter together with an n to 2^n decoder.
- Number of states = Number of flip flops

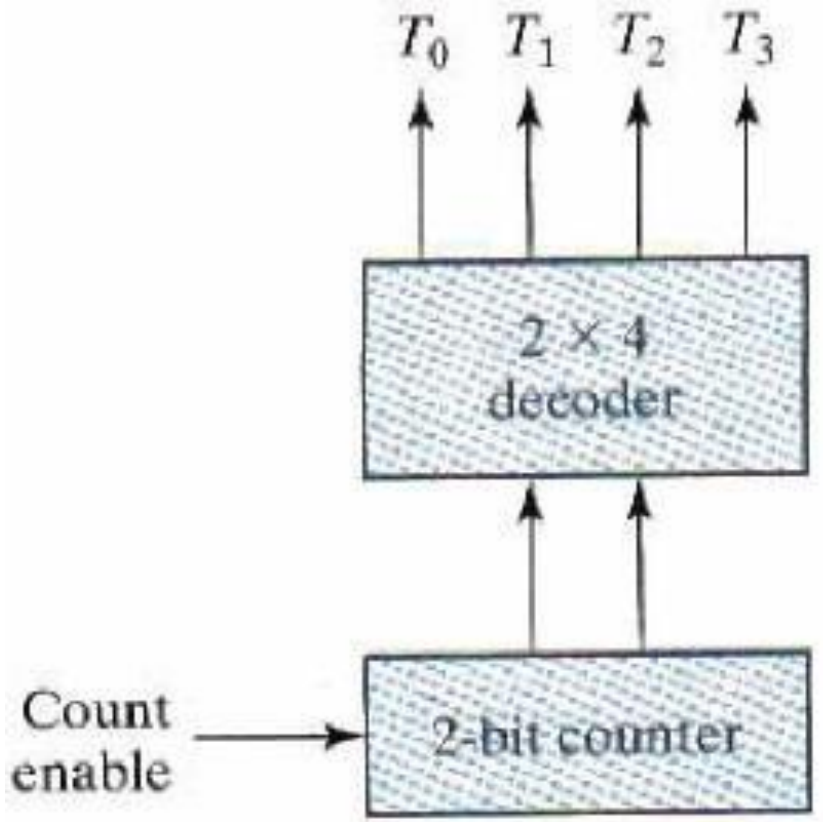


(a) Ring-counter (initial value = 1000)

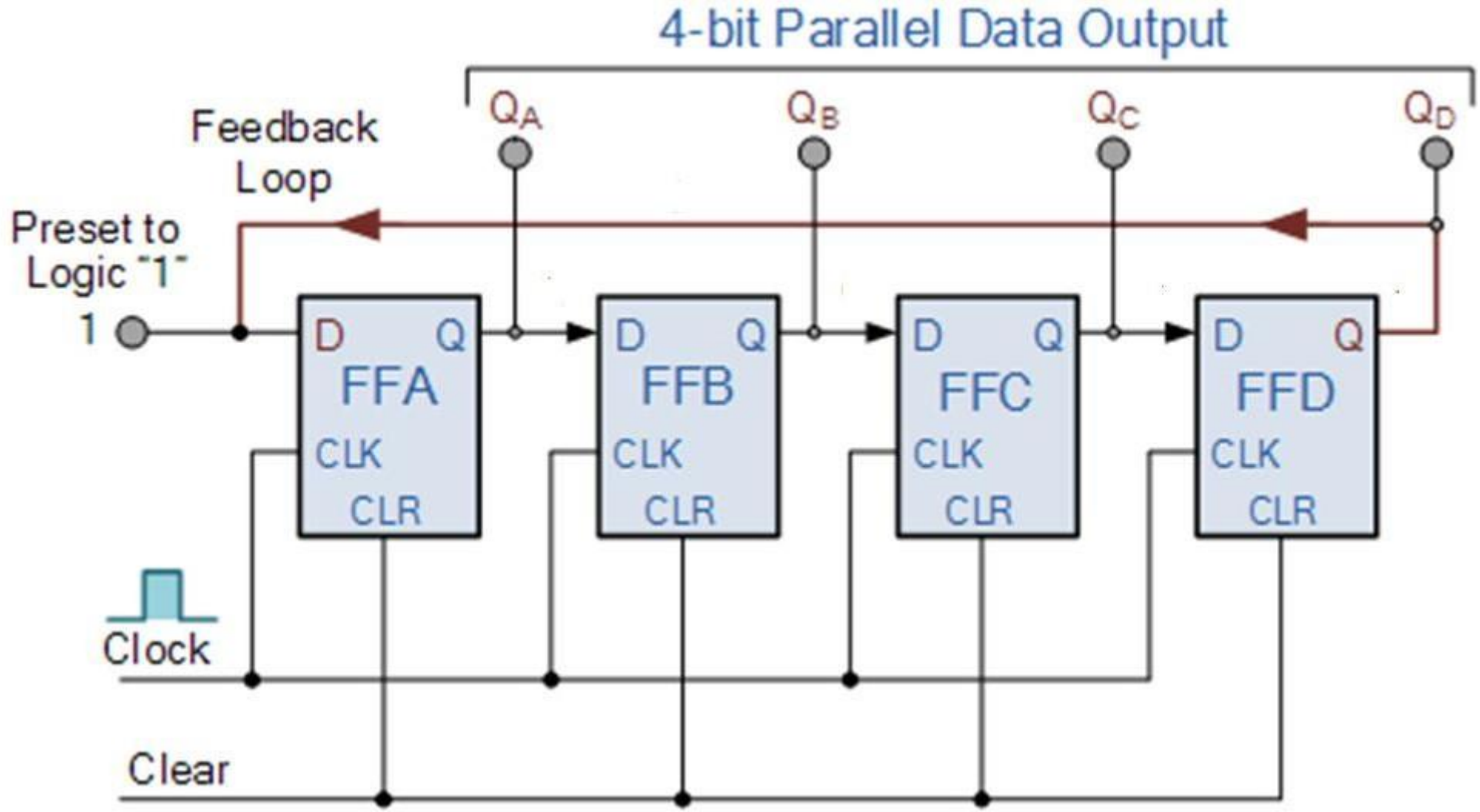
Alternative Design:

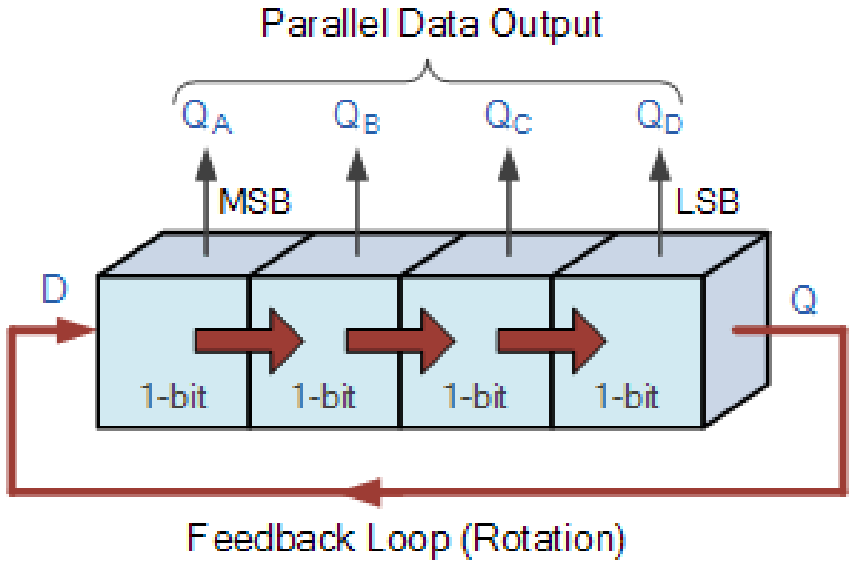


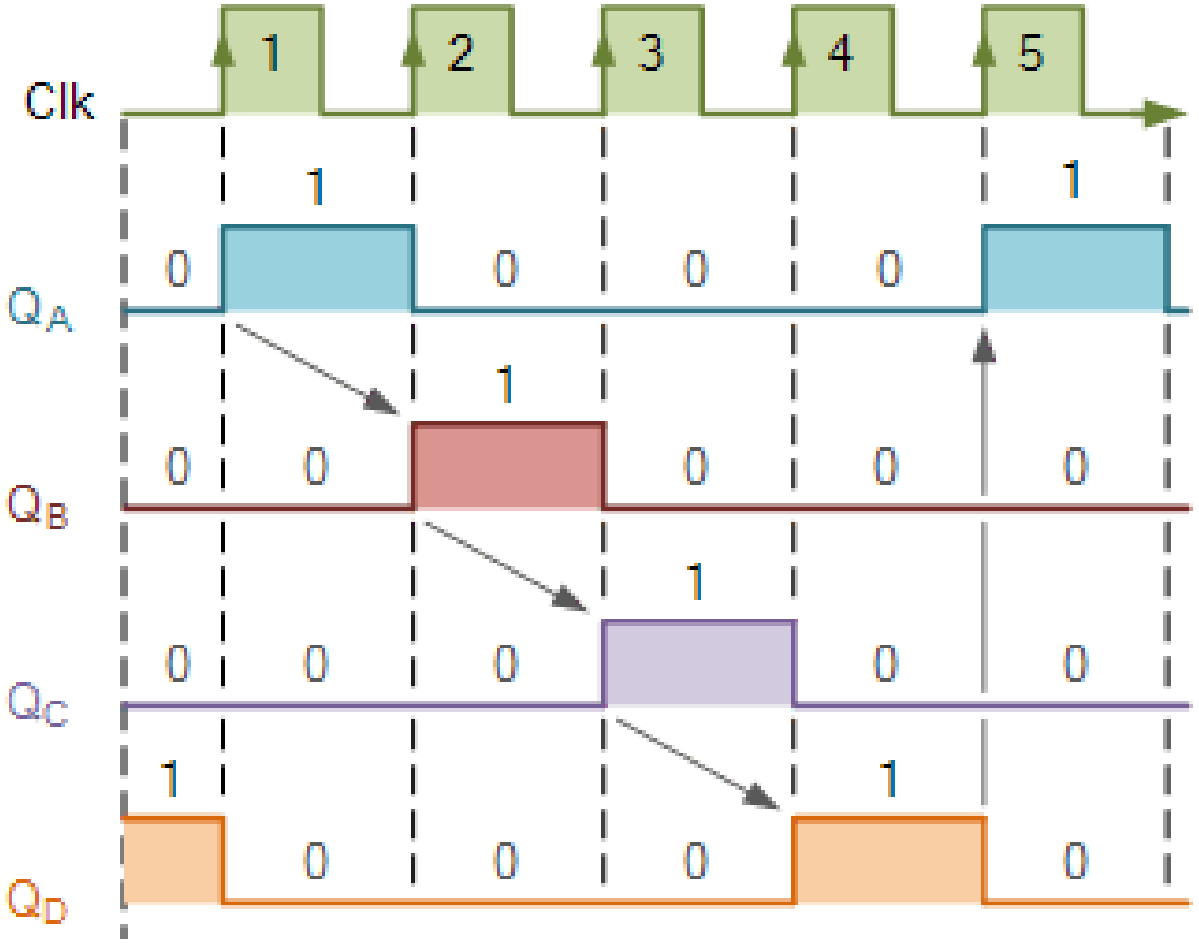
(b) Sequence of four timing signals



(c) Counter and decoder


4-bit Parallel Data Output
Q_A
Q_B
Q_C
Q_D
Feedback Loop
Preset to Logic "1"
1
D
Q
FFA
CLK
CLR
FFB
FFC
FFD
Clock
Clear


Parallel Data Output
Q_A
Q_B
Q_C
Q_D
MSB
LSB
D
Q
1-bit
1-bit
1-bit
1-bit
Feedback Loop (Rotation)


Clk
1
2
3
4
5
Q_A
Q_B
Q_C
Q_D

```
// 4 bit ring counter example
module four_bit_ring_counter (
input clock,
input reset,
output [3:0] q);
reg[3:0] a;

always @(posedge clock)
if (reset)
a = 4'b0001;
else
begin
a <= a<<1; // Notice the
blocking assignment
a[0]<=a[3];
end
assign q = a;

endmodule
```

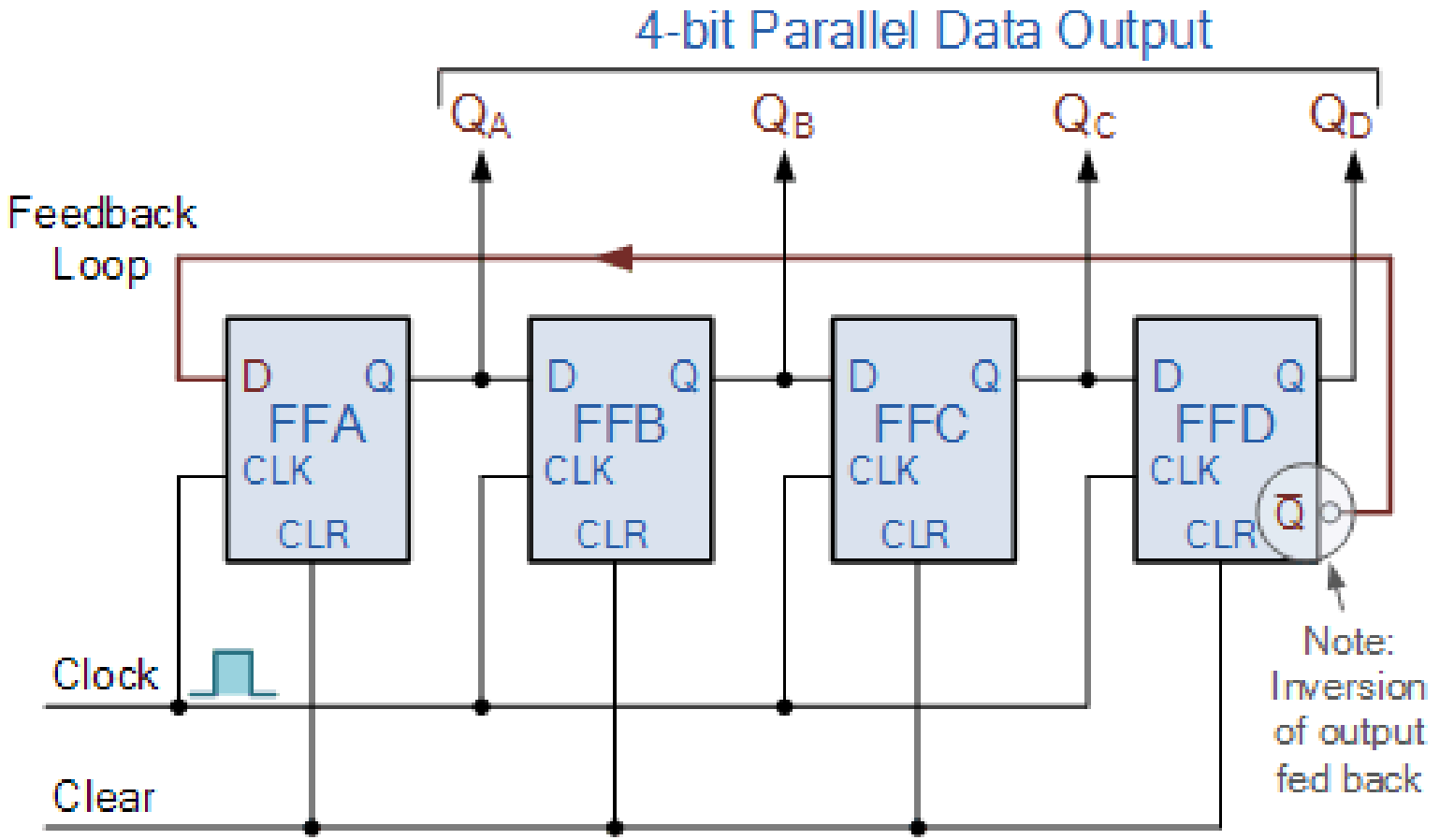
```
module tb;
// Inputs
reg clock;
reg reset;
// Outputs
wire[3:0] q;
// Instantiate the Unit Under Test (UUT)
four_bit_ring_counter r1 (.clock(clock),
.reset(reset),.q(q));
always #10 clock = ~clock;
initial begin
// Initialize Inputs
clock = 0;
reset = 0;
#5 reset = 1;
#20 reset = 0;
#500 $finish;
end
initial begin
$monitor($time, " clock=%1b,reset=%1b,q=%4b",
clock,reset,q);
end
endmodule
```

8-Bit Up-Down Counter

```
module up_down_counter    (  out ,  // Output of the counter
  up_down  ,  // up_down control for counter
 clk      ,  // clock input
 data     ,  // Data to load
 reset       // reset input
 );
 //----------Output Ports--------------
 output [7:0] out;
 //------------Input Ports--------------
 input [7:0] data;
 input up_down, clk, reset;
 //------------Internal Variables--------
 reg [7:0] out;
 //-------------Code Starts Here-------
 always @(posedge clk)
 if (reset) begin // active high reset
   out <= 8'b0 ;
 end else if (up_down) begin
   out <= out + 1;
 end else begin
   out <= out - 1;
 end
 endmodule
```

Johnson counter

- Johnson's counter is also known Twisted/Switch tail ring counter.
- Number of states = 2 x number of flip flops.



Truth Table for a 4-bit Johnson Ring Counter

Clock Pulse No	Q_A	Q_B	Q_C	Q_D
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

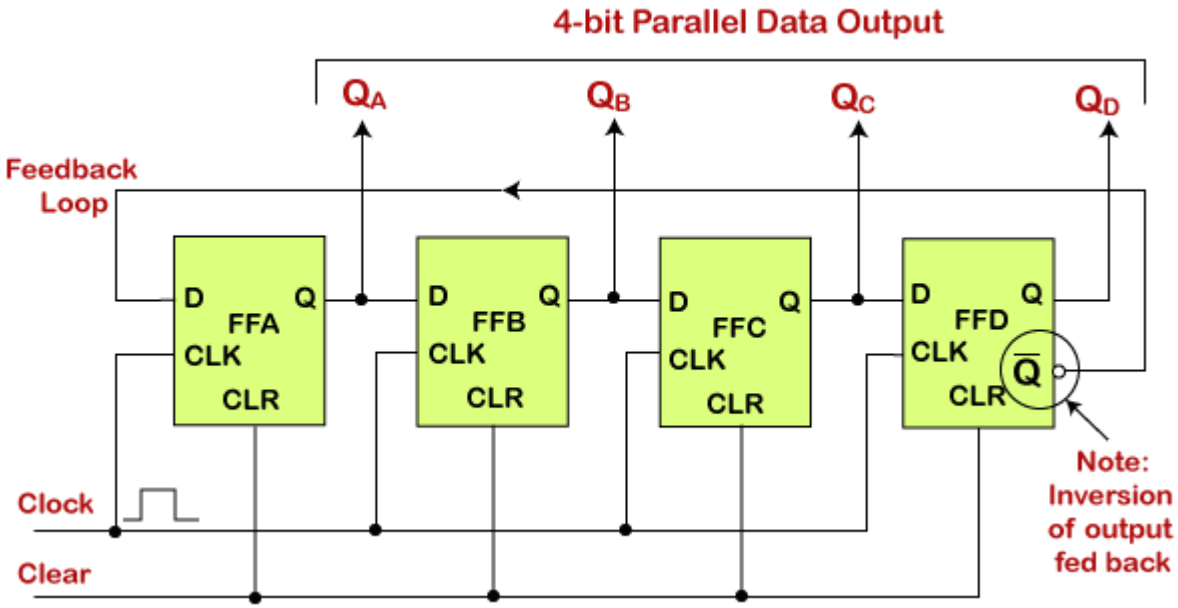
Verilog Johnson Counter

A Johnson counter is a ***digital circuit with a series of flip flops connected in a feedback manner.*** Verilog Johnson counter is a counter that counts 2N states if the number of bits is N.

The circuit is a special type of shift register where the last flip flop's complement output is fed back to the first flip flop's input. This is almost similar to the ring counter with a few extra advantages.

The Johnson counter's main advantage is that it only ***needs half the number of flip-flops compared to the standard ring counter,*** and then it's modulo number is halved. So an n-stage Johnson counter will circulate a single data bit, giving a sequence of 2n different states and can therefore be considered a mod-2n counter.

The inverted output Q of the last flip-flop is connected back to the input D of the first flip-flop. Below is the circuit diagram for a 4-bit Johnson counter:



This inversion of Q before it is fed back to input D causes the counter to count differently. Instead of counting through a fixed set of patterns just like the normal ring counter such as for a 4-bit counter, "0001"(1), "0010"(2), "0100"(4), "1000"(8) and repeat.

The Johnson counter counts up and then down as the initial logic "1" passes through it to the right replacing the preceding logic "0".

A 4-bit Johnson ring counter passes blocks of four logic "0" and then four logic "1" thereby producing an 8-bit pattern.

As the inverted output Q is connected to the input D, this 8-bit pattern continually repeats. For example, "1000", "1100", "1110", "1111", "0111", "0011", "0001", "0000". This is demonstrated in the following table:

Clock Pulse No	FFA	FFB	FFC	FFD
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

As well as counting or rotating data around a continuous loop, ring counters can also be used to detect or recognize various patterns or number values within a set of data. By connecting simple logic gates such as the OR gates to the flip-flops' outputs, the circuit can be made to detect a set number or value.

Standard 2, 3 or 4-stage Johnson Ring Counters can also be used to divide the clock signal frequency by varying their feedback connections, and divide-by-3 or divide-by-5 outputs are also available.

For example, a 3-stage Johnson Ring Counter could be used as a 3-phase, 120-degree phase shift square wave generator by connecting to the data outputs at A, B and NOT-B.

The standard 5-stage Johnson counter, such as the commonly available CD4017, is generally used as a synchronous decade counter/divider circuit.

Other combinations, such as the smaller 2-stage circuit, which is also called a Quadrature Oscillator or Generator, can be used to produce four individual outputs that are each 90 degrees out-of-phase for each other to produce a 4-phase timing signal.

Johnson counter 4 -bit

```
module johnson_ctr #(parameter WIDTH=4)
  (input clk,
   input rstn,
   output reg [WIDTH-1:0] out
  );

  always @ (posedge clk) begin
     if (!rstn)
       out <= 1;
     else begin
      out[WIDTH-1] <= ~out[0];
      for (int i = 0; i < WIDTH-1; i=i+1) begin
        out[i] <= out[i+1];
      end
     end
  end
endmodule
```

```
 module tb;
 parameter WIDTH = 4;
 reg clk;
 reg rstn;
 wire [WIDTH-1:0] out;
 johnson_ctr   u0 (.clk (clk),
                   .rstn (rstn),
                   .out (out));

 always #10 clk = ~clk;

 initial begin
  {clk, rstn} <= 0;

  $monitor ("T=%0t out=%b", $time, out);
  repeat (2) @(posedge clk);
  rstn <= 1;
  repeat (15) @(posedge clk);
  $finish;
 end
endmodule
```

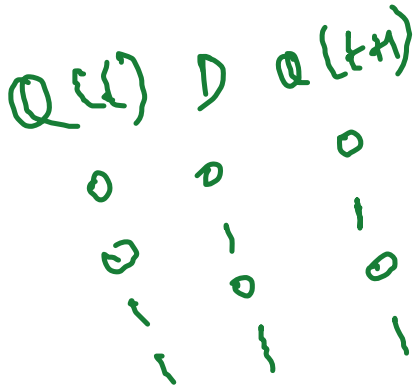
Johnson Counter Verilog Code

```
module johnson_counter( out,reset,clk);
input clk,reset;
output [3:0] out;
reg [3:0] q;

always @(posedge clk)
begin
if(reset)
 q=4'd0;
 else
            begin
                        q[3]<=q[2];
                        q[2]<=q[1];
                        q[1]<=q[0];
                        q[0]<=(~q[3]);
            end
 end
assign out=q;
endmodule
```

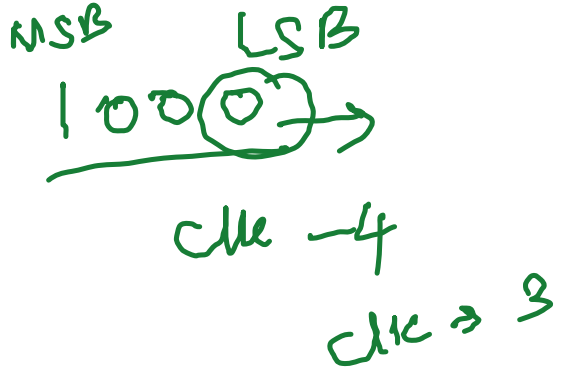
Test Bench

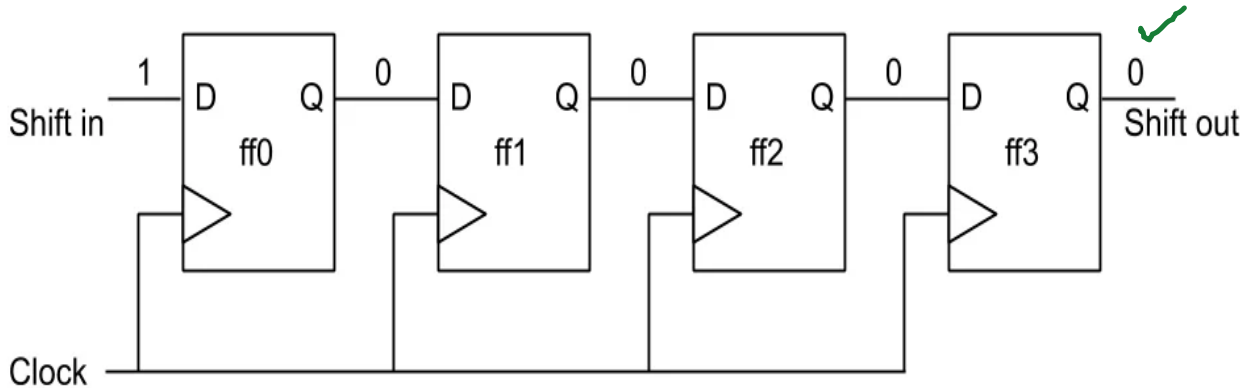
```
 module jc_tb;
 reg clk,reset;
 wire [3:0] out;
 johnson_counter dut (.out(out), .reset(reset), .clk(clk));
 always
  #5 clk =~clk;
  initial begin
  reset=1'b1; clk=1'b0;
 #20 reset= 1'b0;
 end

 initial
  begin
 $monitor( $time, " clk=%b, out= %b, reset=%b", clk,out,reset);
 #105  $stop;
 end

endmodule
```

```
//timescale 1ns / 1ps
module siso_design(input clk,b,output q);
wire w1,w2,w3;
d_ff dut1(.clk(clk),.d(b),.q(w1),.rst());
d_ff dut2(.clk(clk),.d(w1),.q(w2),.rst());
d_ff dut3(.clk(clk),.d(w2),.q(w3),.rst());
d_ff dut4(.clk(clk),.d(w3),.q(q),.rst());
endmodule
// d flip flop
module d_ff (
  input clk,
  input d,
  input rst,
  output reg q);
  always @(posedge clk)
  begin
    if (rst)
      q <= 1'b0;
    else
      q <= d;
  end
endmodule
```




```
// testbench
module siso_tb();
reg clk,b;
wire q;
siso_design uut(.clk(clk),.b(b),.q(q));
initial
begin
clk=1'b0;
forever #5clk=~clk;
end
initial
begin
$monitor("clk=%d,b=%d,q=%d",clk,b,q);end
initial
begin
b=1;
#10; b=0;
#10; b=0;
#10; b=0;
#50;
$finish;
end
endmodule
```





SISO

```
module sisomod(clk,clear,si,so);
input clk,si,clear;
output so;
reg so;
reg [3:0] tmp;
always @(posedge clk )
begin
if (clear)
tmp <= 4'b0000;
else
tmp <= tmp << 1;
tmp[0] <= si;
so = tmp[3];
end
endmodule
```

TEST BENCH

```
module sisot_b;
reg clk;
reg clear;
reg si;
wire so;
sisomod uut (.clk(clk), .clear(clear),.si(si),.so(so));
initial begin
clk = 0;
clear = 0;
si = 0;
#5 clear=1'b1;
#5 clear=1'b0;
#10 si=1'b1;
#10 si=1'b0;
#10 si=1'b0;
#10 si=1'b1;
#10 si=1'b0;
#10 si=1'bx;
end
always #5 clk = ~clk;
initial #150 $stop;
endmodule
```

VERILOG CODE FOR 4-BIT SHIFT REGISTER

```
module shift4mod(R, L, w, Clock, Q);
input [3:0] R;
input L, w, Clock;
output [3:0] Q;
reg [3:0] Q;
always @(posedge Clock)
if (L)
Q <= R;
else
begin
Q[0] <= Q[1];
Q[1] <= Q[2];
Q[2] <= Q[3];
Q[3] <= w;
end
endmodule
```

VERILOG CODE FOR GENERIC N-BIT SHIFT REGISTER

```
module shiftnmod(R, L, w, Clock, Q);
parameter n = 16;
input [n-1:0] R;
input L, w, Clock;
output [n-1:0] Q;
reg [n-1:0] Q;
integer k;
always @(posedge Clock)
if (L)
Q <= R;
else
begin
for (k=0; k<n-1; k=k+1)
Q[k] <= Q[k+1];
Q[n-1] <= w;
end
endmodule
```

SIPO

```
module sipomod(clk,clear, si, po);
input clk, si,clear;
output [3:0] po;
reg [3:0] tmp;
reg [3:0] po;
always @(posedge clk)
begin
if (clear)
tmp <= 4'b0000;
else
tmp <= tmp << 1;
tmp[0] <= si;
po = tmp;
end
endmodule
```

TEST BENCH

```
module sipot_b;
reg clk;
reg clear;
reg si;
wire [3:0] po;
sipomod uut (.clk(clk),.clear(clear),   .si(si),.po(po) );
initial begin
clk = 0;
clear = 0;
si = 0;
#5 clear=1'b1;
#5 clear=1'b0;
#10 si=1'b1;
#10 si=1'b0;
#10 si=1'b0;
#10 si=1'b1;
#10 si=1'b0;
#10 si=1'bx;
end
always #5 clk = ~clk;
initial #150 $stop;
endmodule
```

VERILOG CODE FOR PIPO AND TESTBENCH

PIPO

```
module pipomod(clk,clear, pi, po);
input clk,clear;
input [3:0] pi;
output [3:0] po;
wire [3:0] pi;
reg [3:0] po;
always @(posedge clk)
begin
if (clear)
po<= 4'b0000;
else

po <= pi;
end
endmodule
```

TEST BENCH

```
module pipot_v;
reg clk;
reg clear;
reg [3:0] pi;
wire [3:0] po;
pipomod uut (.clk(clk),.clear(clear),.pi(pi),.po(po) );
initial begin
clk = 0;
clear = 0;
pi = 0;
#5 clear=1'b1;
#5 clear=1'b0;
#10 pi=4'b1001;
#10 pi=4'b1010;
#10 pi=4'b1011;
#10 pi=4'b1110;
#10 pi=4'b1111;
#10 pi=4'b0000;
end
always #5 clk = ~clk;
initial #150 $stop;
endmodule
```

```
module Shiftregister_PISO(Clk, Parallel_In,load,
Serial_Out);
input Clk,load;
input [3:0]Parallel_In;
output reg Serial_Out;
reg [3:0]tmp;
always @(posedge Clk)
begin
if(load)
tmp<=Parallel_In;
else
begin
Serial_Out<=tmp[3];
tmp<={tmp[2:0],1'b0};
end
end
endmodule
```