SEQUENTIAL LOGIC

Satish Chandra
Assistant Professor
Department of Physics
P P N College, Kanpur
www.satish0402.weebly.com

OSCILLATORS

- Oscillators is an amplifier which derives its input from output.
- Oscillators
  - Sinusoidal Oscillators
  - Non-Sinusoidal Oscillators
    - Ringing Oscillators
    - Blocking Oscillators
    - Multivibrators

MULTIVIBRATORS

- A Multivibrator is a regenerative circuit with two active devices, designed so that one device conducts while the other cut off.
- It can store binary numbers, count pulses, synchronize arithmetic operations and perform other essential functions in digital systems.
- Multivibrators
  - Astable Multvibrator
  - Monostable Multivibrator
  - Bistable Multivibrator or Flip-Flop

FLIP-FLOPS

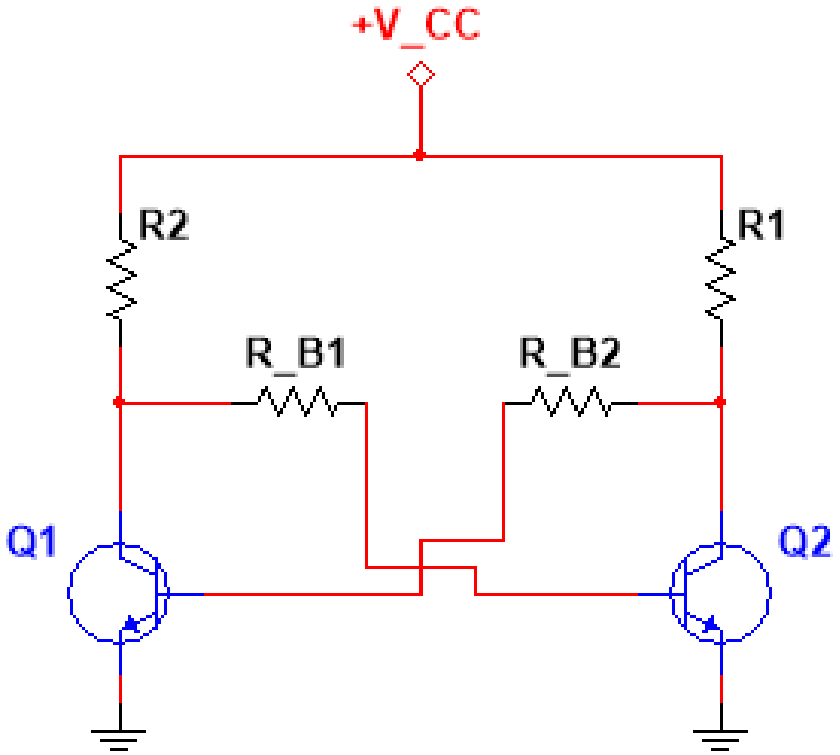
- Flip-Flops (FF) is another name for a bistable multivibrator one whose output voltage is either low or high, a 0 or 1.
- The output stays low or high; to change it, the circuit must be driven by an input called a trigger.
- Until the trigger arrives, the output voltage remains low or high indefinitely.

FLIP-FLOPS

- Flip-Flops (FF)
  - R S FF
  - T FF
  - D FF
  - J K FF
  - Master/Slave J K FF

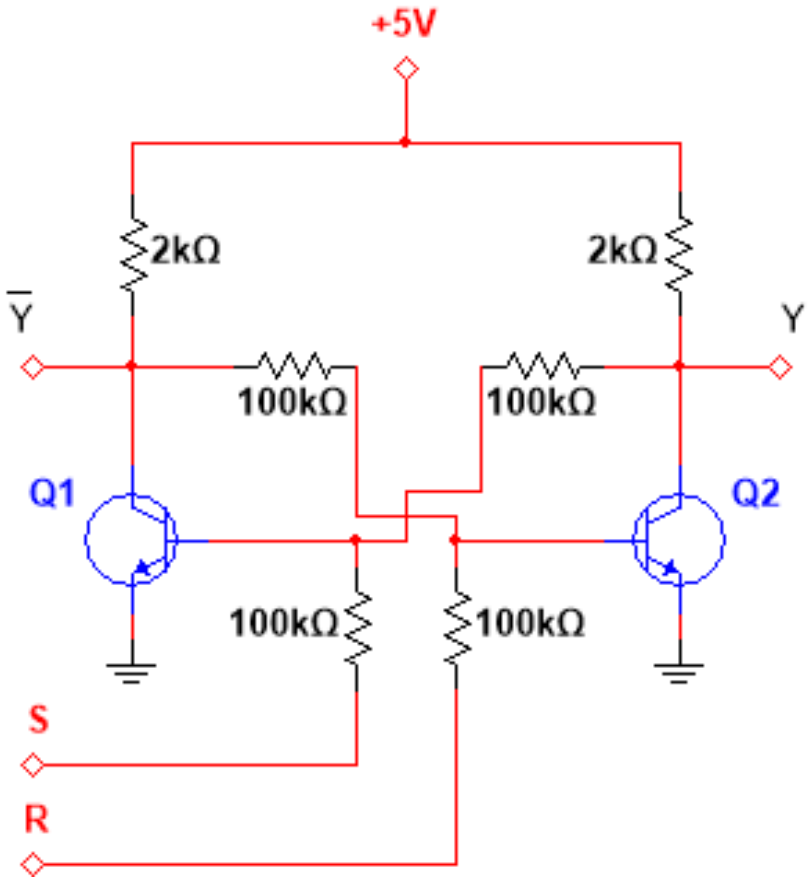
FLIP-FLOPS

- The cross-coupling from each collector to the opposite base. The cross-coupling results in positive feedback.
- If Q_1 is saturated, the low Q_1 collector voltage will force Q_2 to cut off.
- Similarly, if Q_2 get saturated, it will force Q_1 to go into cut off.
- So there are two stable operating conditions: Q_1 saturated, Q_2 cut off; or Q_1 cut off, Q_2 saturated.



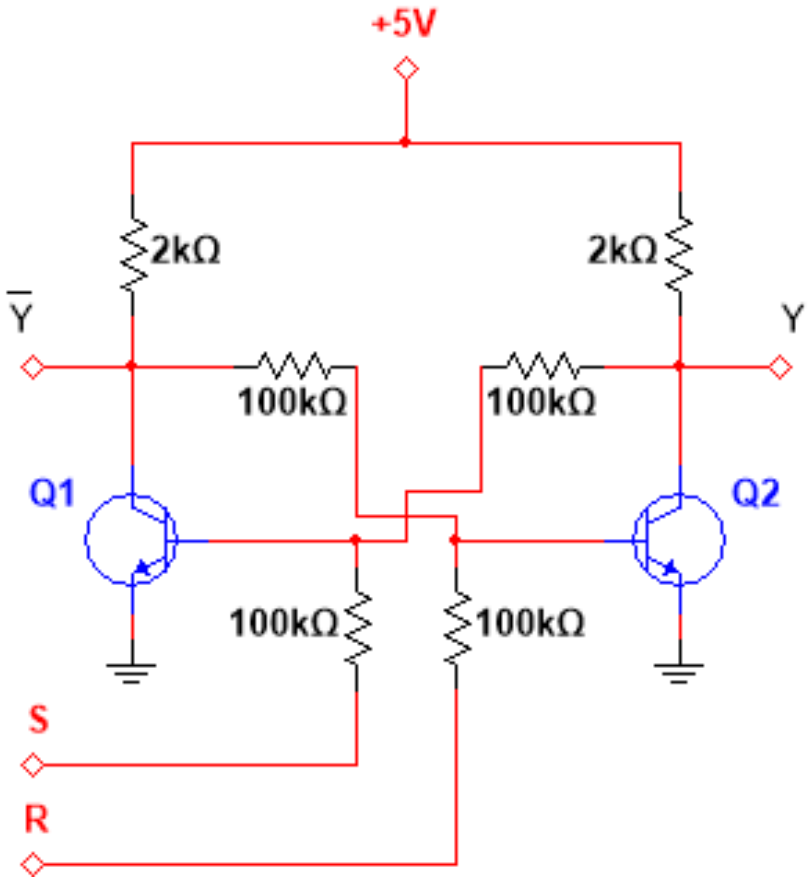
RS FLIP-FLOP

- To control the state of a FF, we have to add trigger inputs.
- If a high voltage is applied to the S (SET) input, then Q_1 saturates; this forces Q_2 into cut off.
- Likewise , a high voltage can be applied to the R (RESET) input; this saturates Q_2 and forces Q_1 into cut off.



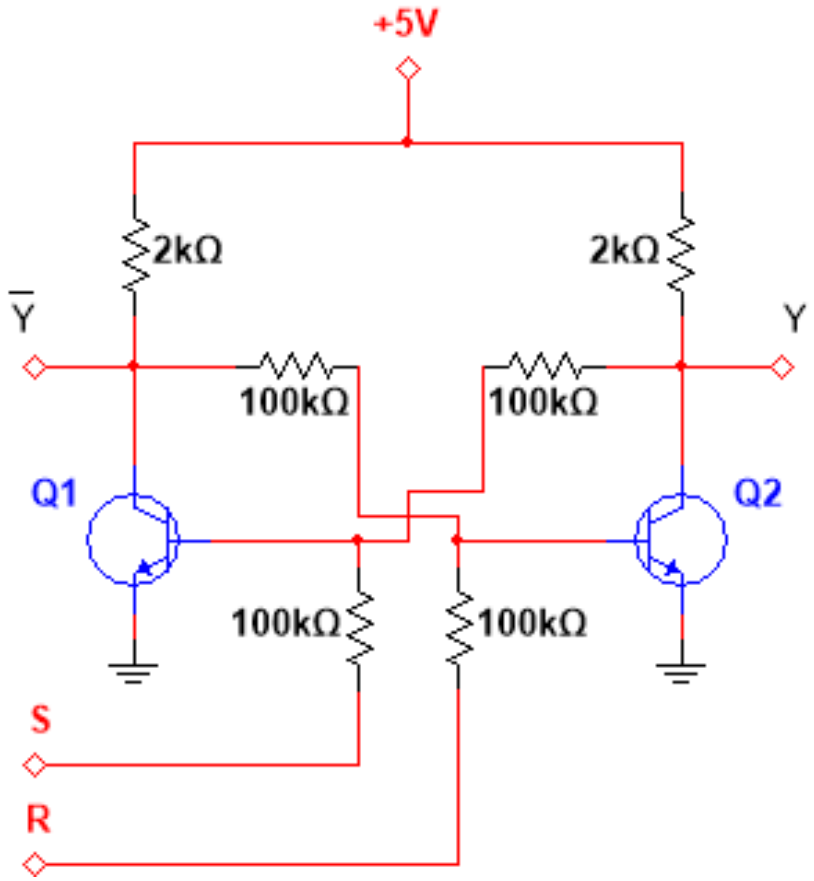
RS FLIP-FLOP

- Applying a high voltage to the S input is called setting and results ; y = 1
- Applying a high voltage to the R input is called as resetting and results ; y = 0
- So, a FF is a natural circuit for generating a variable and its compliment.



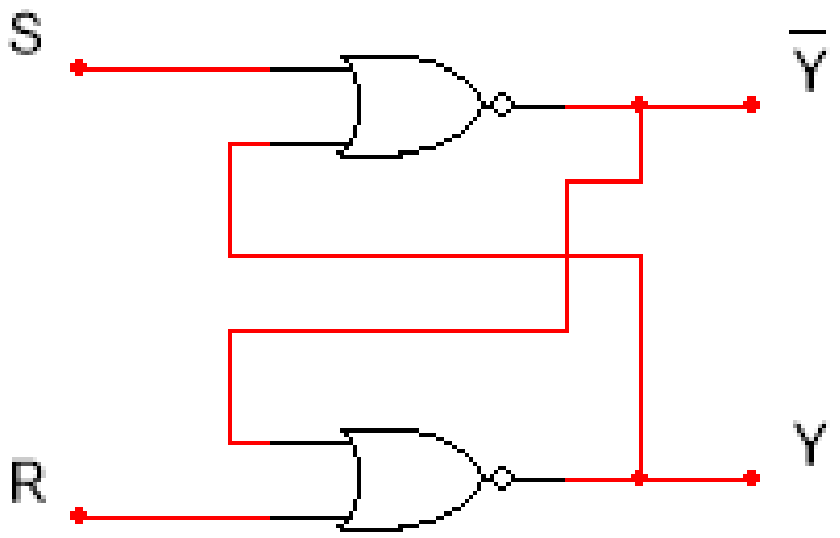
RS FLIP-FLOP

R	S	Y	$\overline{Y}$
0	0	LAST VALUE	
0	1	1	0
1	0	0	1
1	1	FORBIDDEN	



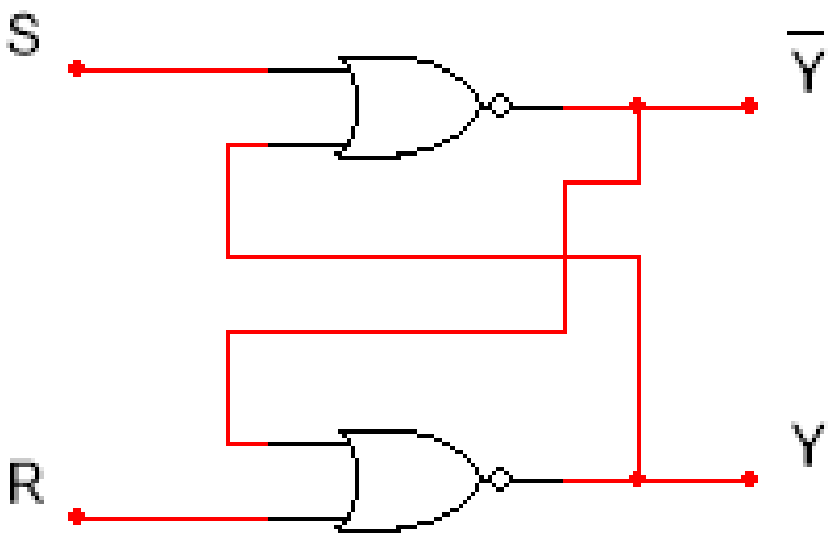
RS FLIP-FLOP – NOR gate

- IC equivalent – NOR gate
- The output of each NOR gate drives one of the inputs on the other NOR gate.
- Also the S and R inputs allows us to set or reset the output y.



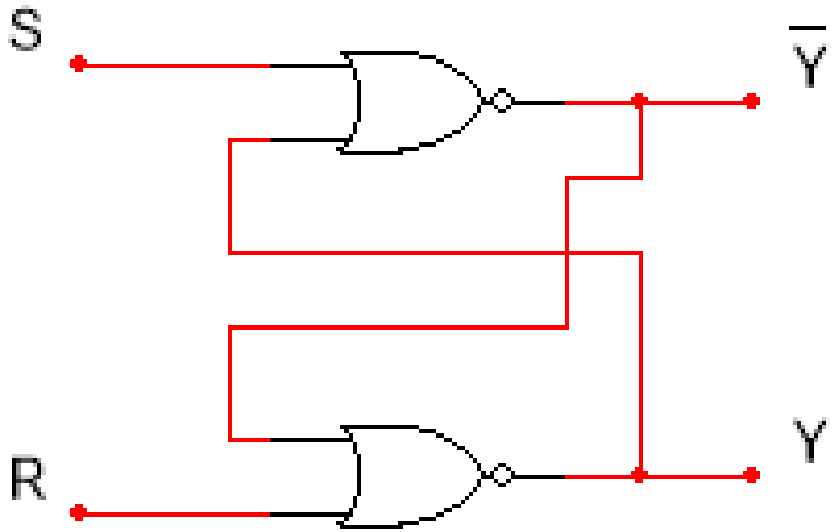
RS FLIP-FLOP

- A high S input sets y to 1.
- A high R input reset y to 0
- If R and S are both low, the output remains *latched* in its last state.
- Contradictory state of R and S both high at the same time is still forbidden.
- An RS FF is therefore, also called a latch.

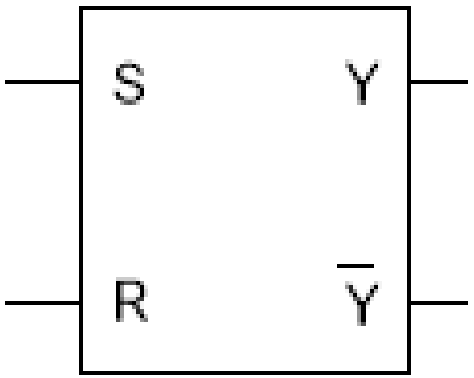


RS FLIP-FLOP

- RS FF, or latches, are said to be transparent, i.e.. any change in input at R and S is transmitted immediately to the output at Y and $\overline{Y}$.

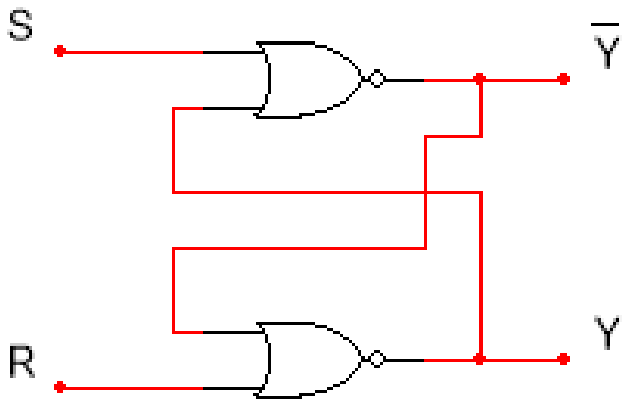


R	S	Y
0	0	LAST VALUE
0	1	1
1	0	0
1	1	FORBIDDEN

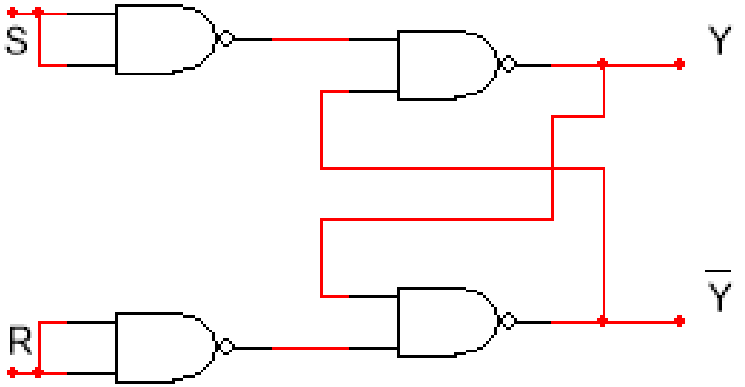


RS FLIP-FLOP

NOR gate RS FF

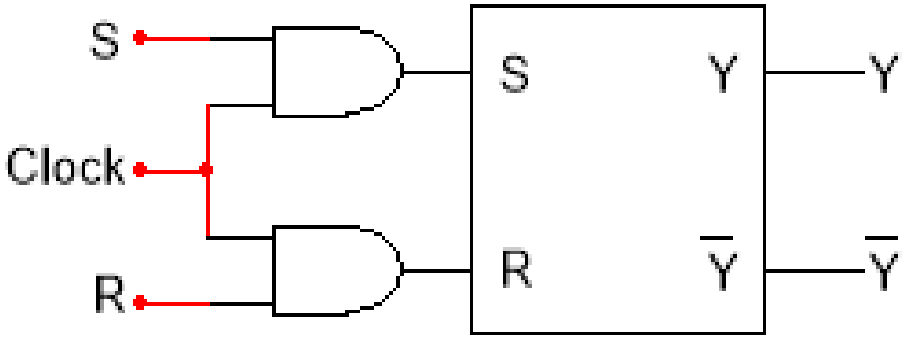


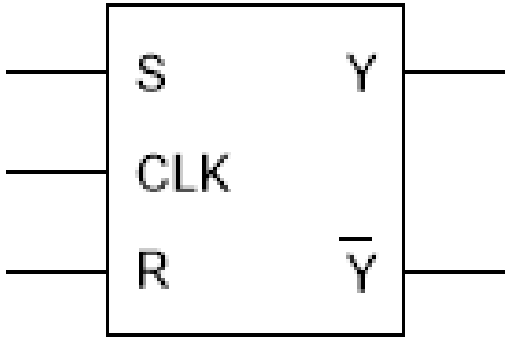
NAND gate RS FF



Clocked RS FLIP-FLOP

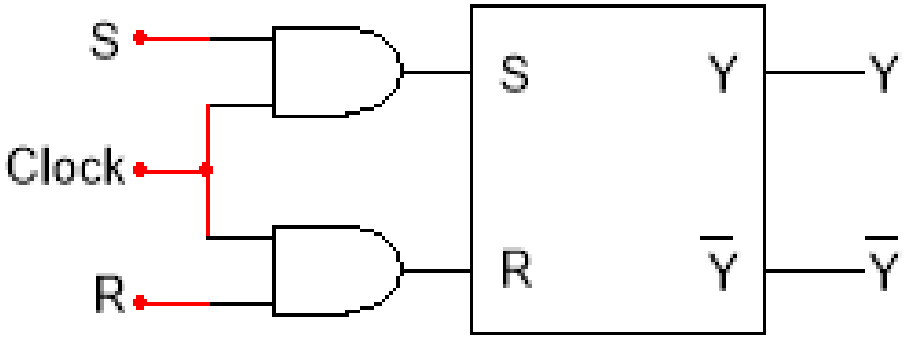
- When the Clock is low both the AND gates are disabled, which means the output Y remains in the last state it was in.
- When the Clock input goes high, however, both AND gates are enabled. This allows the S and R inputs to reach the RS FF.

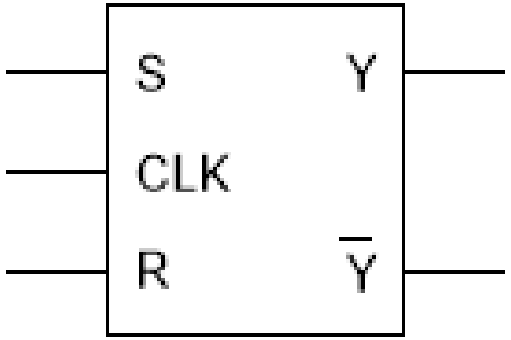




Clocked RS FLIP-FLOP

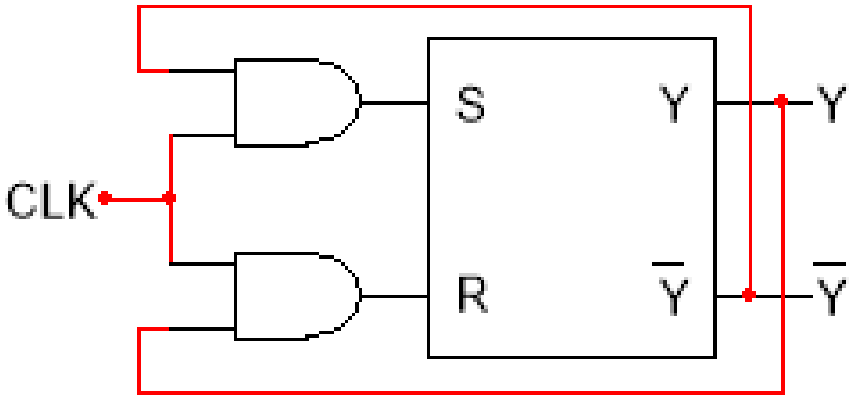
- For the FF to operate properly, there must be a transition from low to high on the Clock input.
- Clocking is important in large digital systems where large number of FF may be interconnected. The single clock is applied to all FF simultaneously; so that they all changes states in unison.

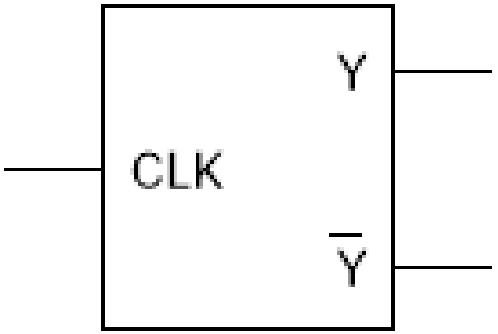




T FLIP-FLOP

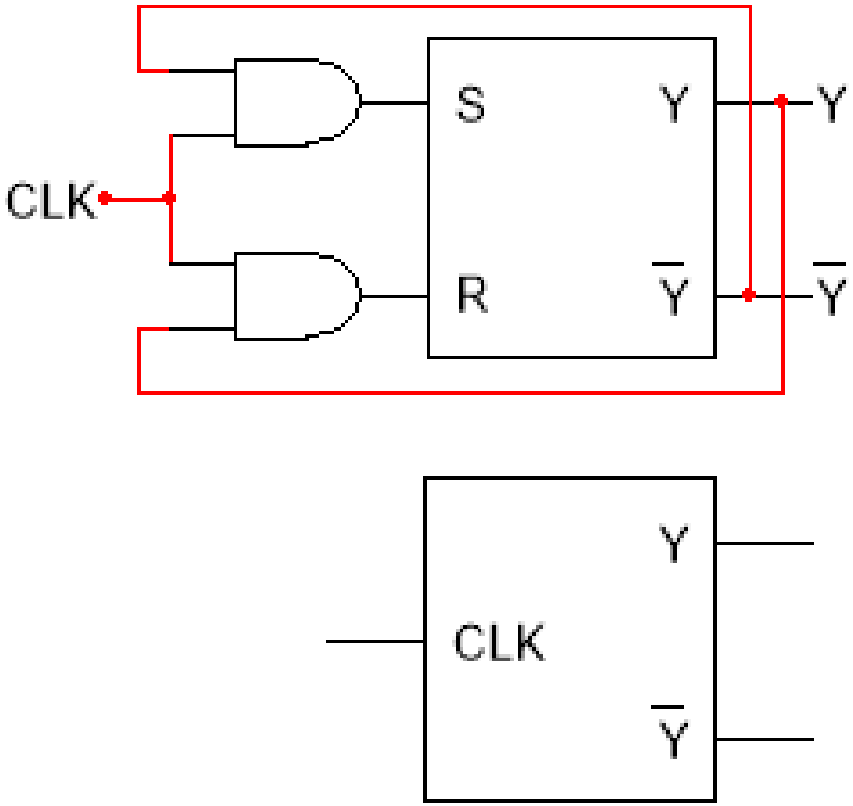
- The output of FF toggles between set and reset condition on arrival of each clock pulse.
- Toggle means to switch to the opposite state. So it is called Toggle or T FF.
- The RS FF can be converted to function as T FF with slight modification.





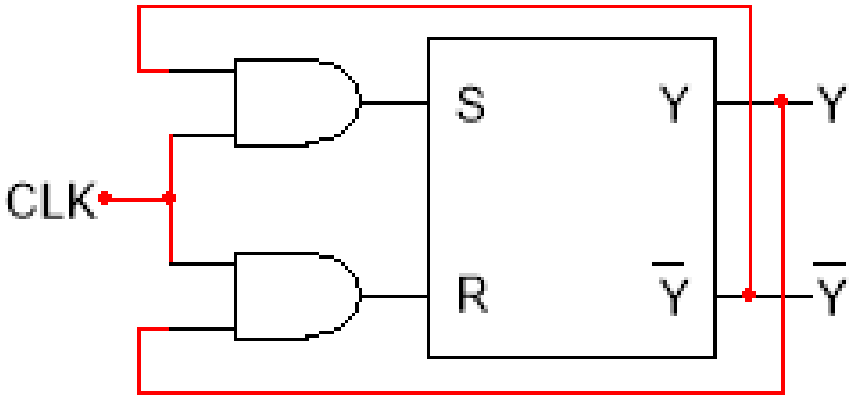
T FLIP-FLOP

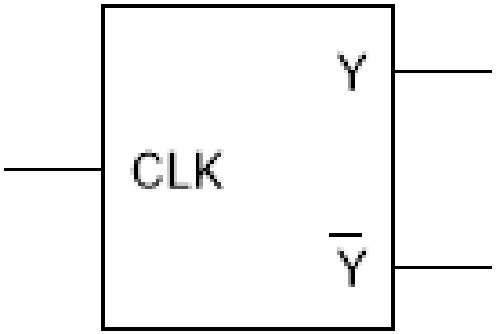
- The S input of RS FF is connected to $\overline{Y}$ and R input is connected to Y output.
- Since Y and $\overline{Y}$ cannot be in same state simultaneously, the output of T FF goes high on alternate clock signals.



T FLIP-FLOP

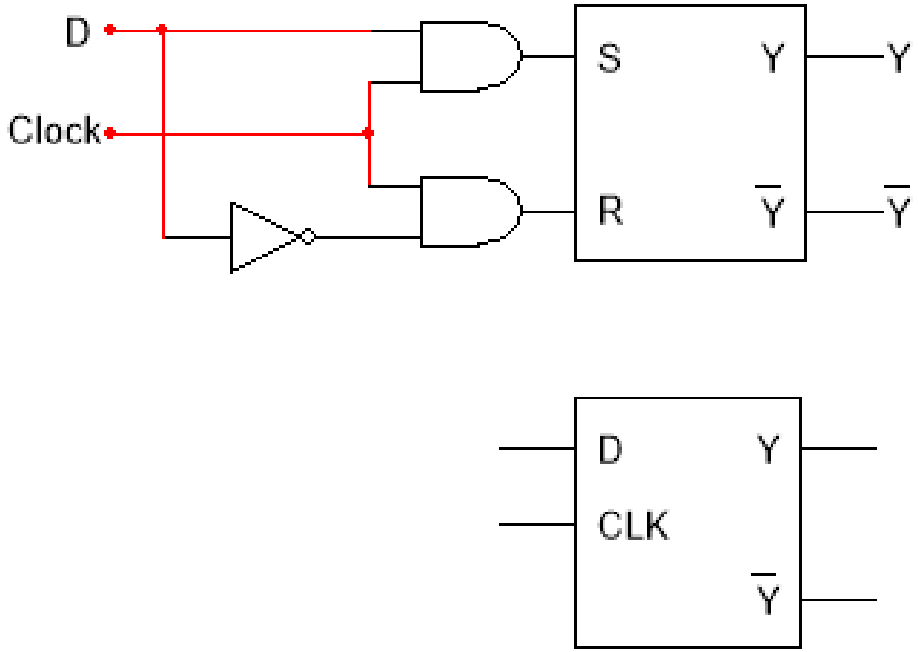
- Since the output frequency of the pulses is the half the frequency of the input pulses. Thus, the T FF operates as frequency divider.
- If Y = 0 and $\overline{Y}$ = 1, then R = 0 and S = 1, so the output is set on arrival of pulse.
- If Y = 1 and $\overline{Y}$ = 0, then R = 1 and S = 0, so the output is reset on arrival of pulse





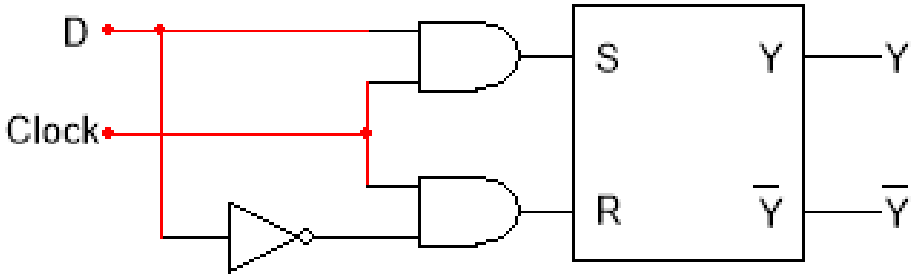
D FLIP-FLOP

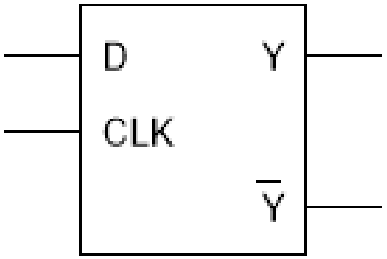
- The RS FF has two data inputs, S and R.
- Generating two input signals to drive a FF is a disadvantage in many application.
- The forbidden condition of both R and S high may occurs undesirably.
- D FF is a circuit that needs only a single data input.



D FLIP-FLOP

- Delay (D) FF prevents the value of D from reaching the Y output until a clock pulse occurs.
- When the clock is low both AND gates are disabled therefore D can change value without affecting the value of Y.
- When the clock is high both AND gates are enabled. Y is forced to equal the value of D.

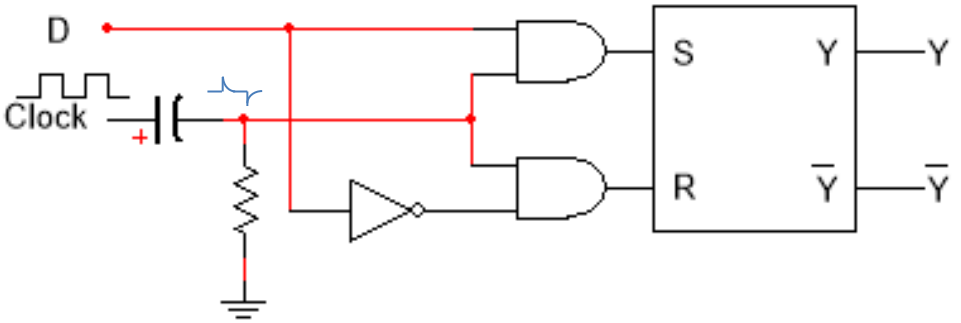


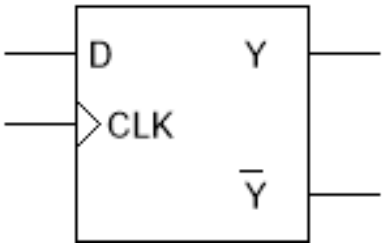


CLK	D	Y
0	X	LAST VALUE
1	0	0
1	1	1

EDGE-TRIGGERED D FLIP-FLOP

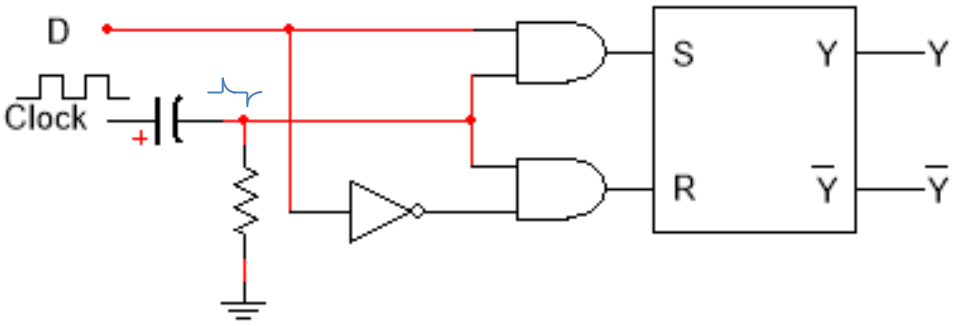
- An RC circuit at the input of a D latch.
- The RC time constant is much smaller than the clock pulse width.
- The C charged fully when the clock goes high (leading edge) produces a narrow positive spike.
- The trailing edge results in a narrow negative spike.





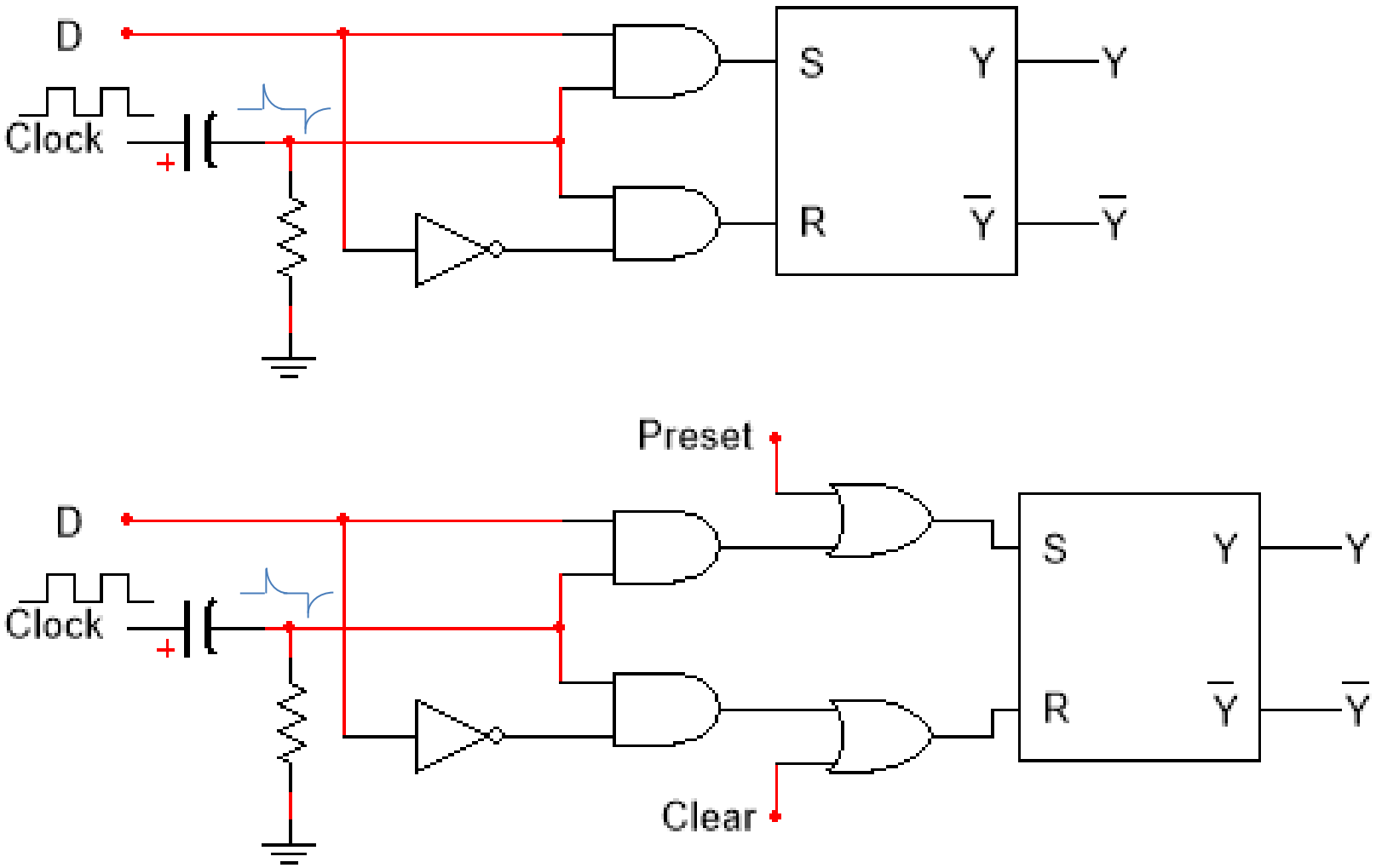
EDGE-TRIGGERED D FLIP-FLOP

- The positive spike enables the AND gates for an *instant*; but the negative spikes does nothing.
- This referred to as positive-edge triggering.
- This kind of operation is called edge triggering because the FF responds only when the clock is in transition between its two voltage states.



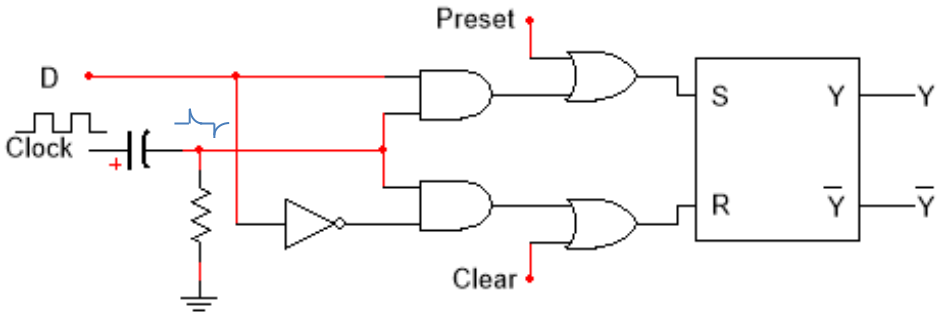
CLK	D	Y
0	X	LAST VALUE
↑	0	0
↑	1	1
↓	X	LAST VALUE

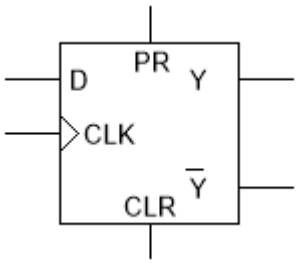
EDGE-TRIGGERED D FLIP-FLOP



EDGE-TRIGGERED D FLIP-FLOP

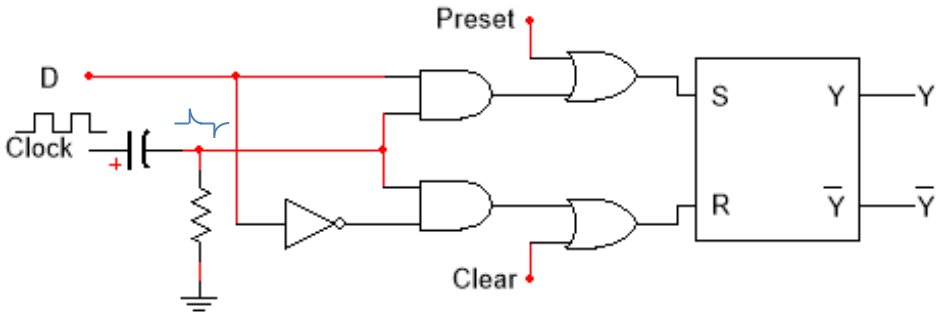
- When power is first applied, FFs come up in random states.
- To reset it operator has to send clear signal all FFs.
- Some systems requires to preset certain FFs.
- OR gates allow us to slip in a high PRESET or a high CLEAR when desired.

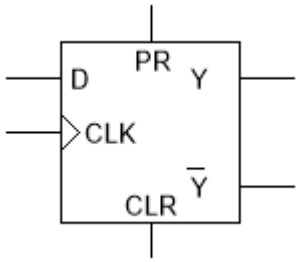




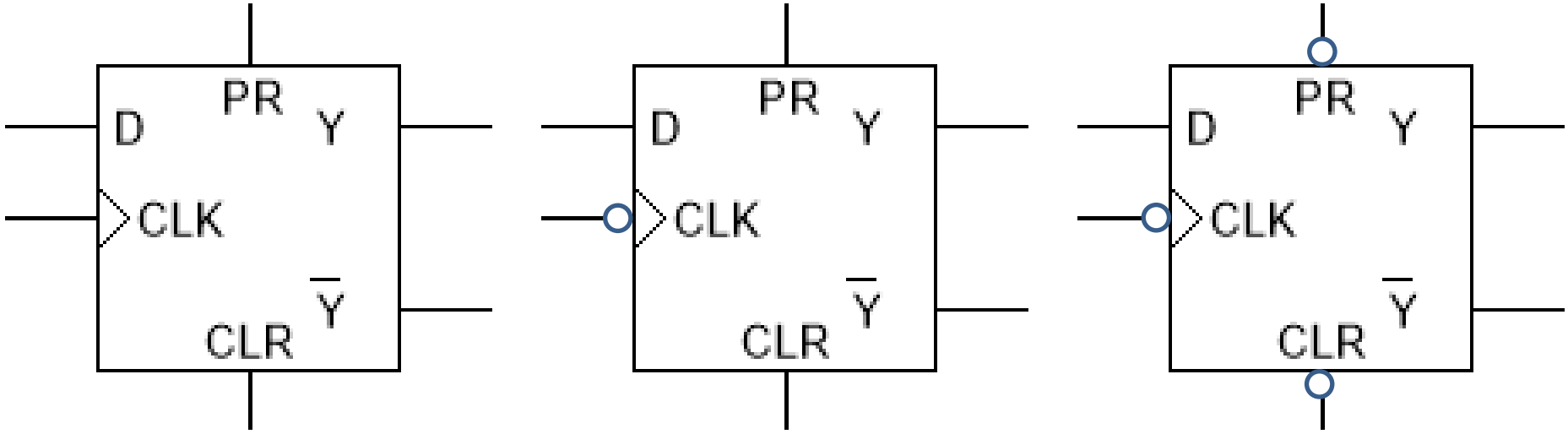
EDGE-TRIGGERED D FLIP-FLOP

- A high PRESET forces Y to set and a high CLEAR forces Y to reset.
- The Preset and Clear are asynchronous inputs, as they activate the FF with out use of clock pulse.
- The D input of FF is a synchronous input, as D activate only when a clock edge occurs.





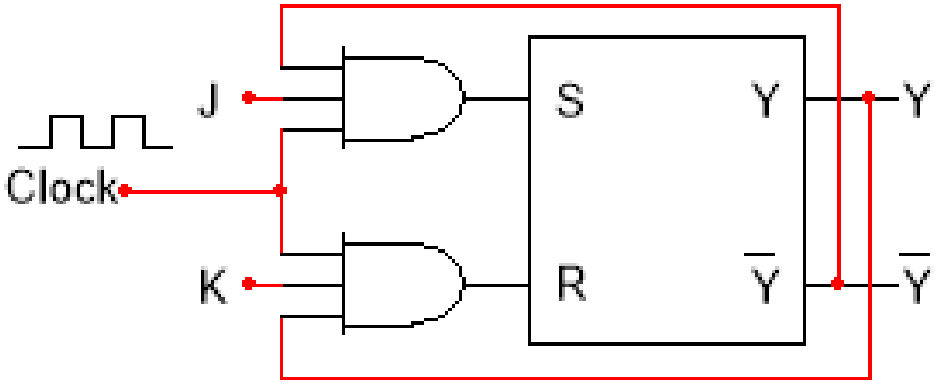
EDGE-TRIGGERED D FLIP-FLOP



a) Positive-edge triggered
b) Negative-edge triggered
c) Negative-edge triggered with inverted preset and clear

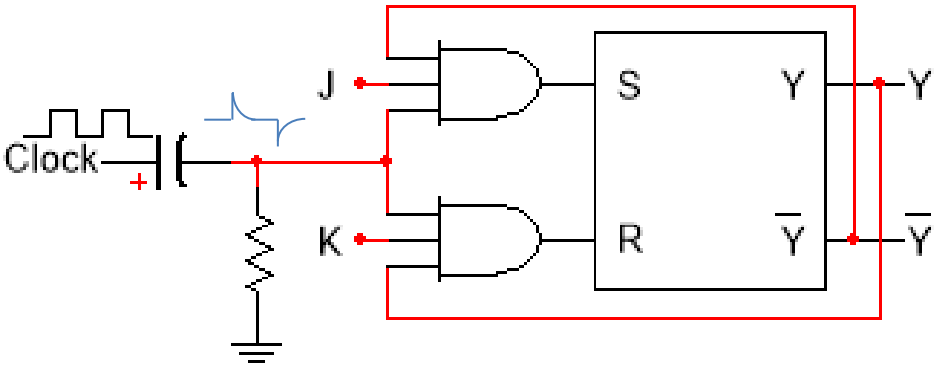
J K FLIP-FLOP

- JK FF is a very important FF normally used to build counter. The JK FF is a modification of T FF as it is also clock driven FF.
- J and K are called *control inputs* because they determine what the FF does when clock pulse arrives.



J K FLIP-FLOP

- The RC circuit has a short time constant, thus converting the rectangular clock pulse into narrow spikes.
- When J & K are low, both AND gates are disabled. Therefore, clock pulses have no effect. The output Y retains its last value.

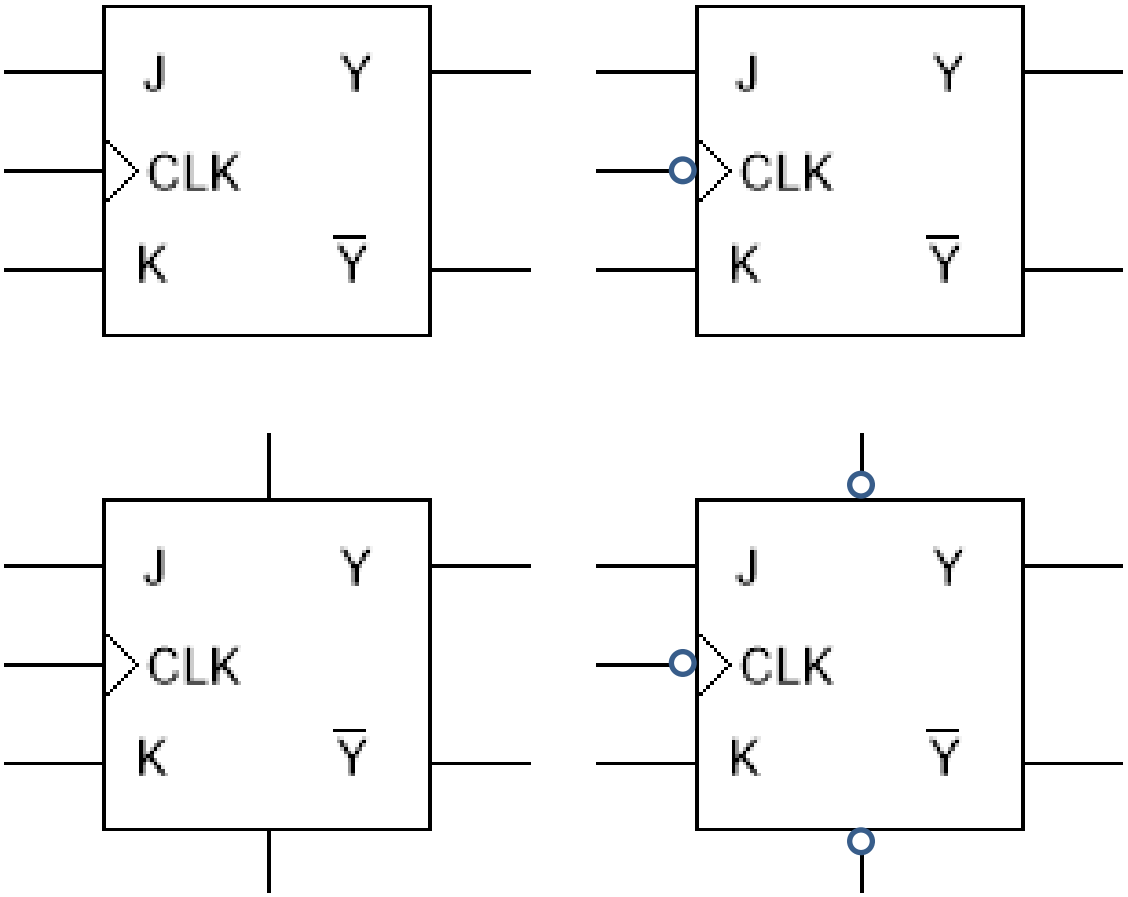


CLK	J	K	Y
X	0	0	LAST VALUE
↑	0	1	0
↑	1	0	1
↑	1	1	$\overline{\text{LAST VALUE}}$

J K FLIP-FLOP

- When J is low and K is high, the upper gate is disabled. When Y is high, the lower gate can pass a reset trigger on arrival of next positive clock edge. This forces Y to become low, i.e. reset the FF.
- When J is high and K is low, the lower gate is disabled. When Y is low but $\overline{Y}$ is high, therefore the upper gate passes a set trigger on arrival of next positive clock edge. This forces Y to become high, i.e. set the FF.
- When J and K are high, both gates are enabled. If Y is high, the lower gate passes a reset trigger on next positive edge. When Y is low, the upper gate passes a set trigger on next positive edge. Either way, Y changes to the complement of the last state.

EDGE-TRIGGERED JK FLIP-FLOP



a) Positive-edge triggered

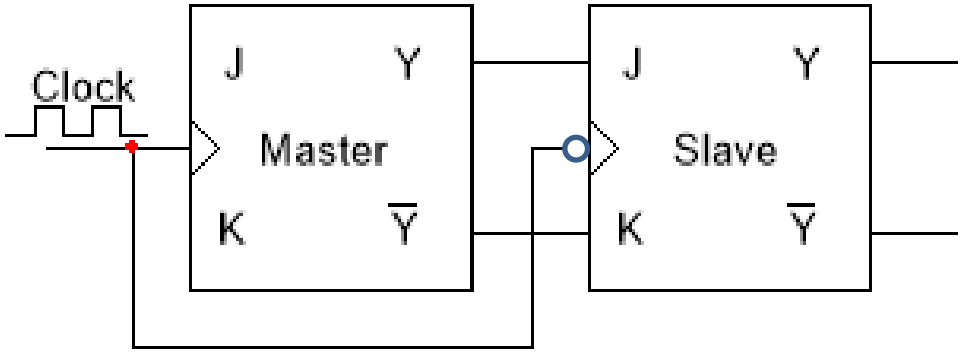
b) Negative-edge triggered

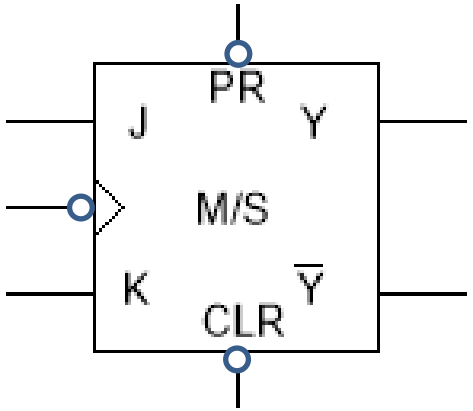
c) Positive-edge triggered with preset and clear

d) Negative-edge triggered with inverted preset and clear

J K MASTER/SLAVE FLIP-FLOP

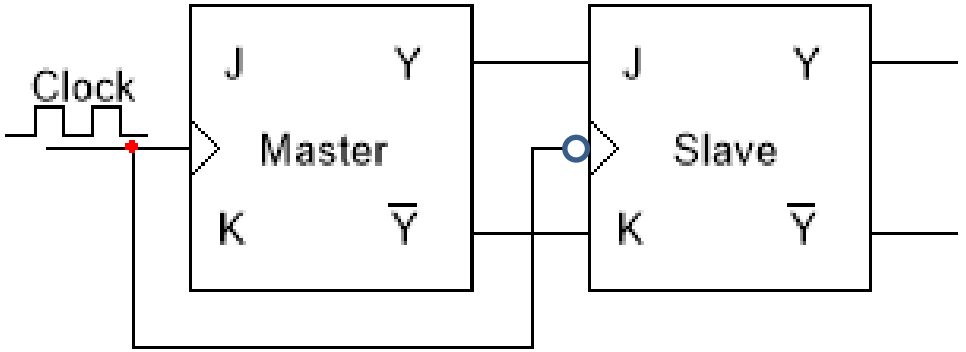
- Widely used as the main counting device.
- Free from race problem i.e., toggling more than once during a negative clock edge.
- The master (M) is positive-edge-triggered and the slave (S) is negative-edge-triggered.
- Therefore the master (M) responds to its J and K inputs before the slave (S).

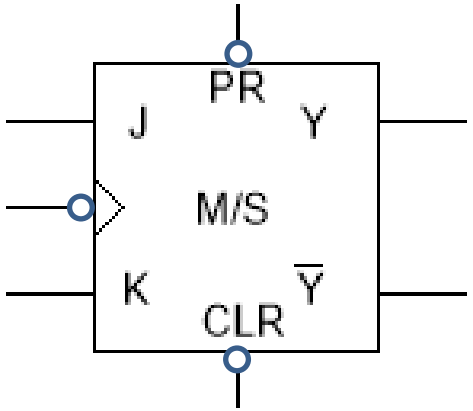




J K MASTER/SLAVE FLIP-FLOP

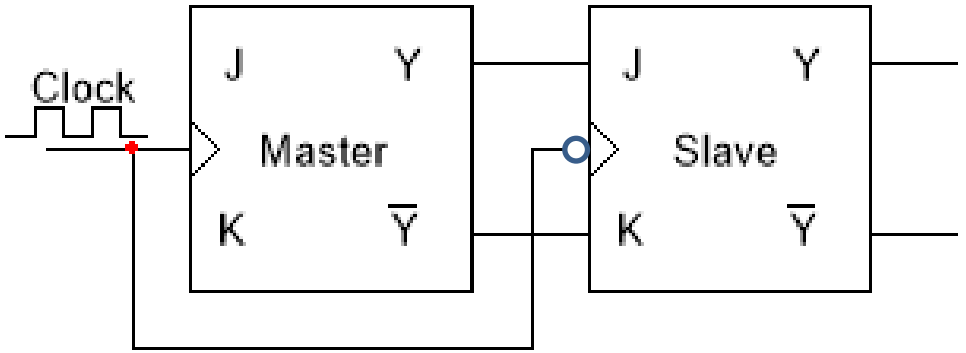
- If J = 1 and K = 0, then M sets on the + clock edge. The high Y of the M drives the J input of the S, so when – clock edge hits, the S sets, copying M.
- If J = 0 and K = 1, then M resets on the + clock edge. The low Y of the M drives the J input of the S, so when – clock edge hits, the S resets, copying M.

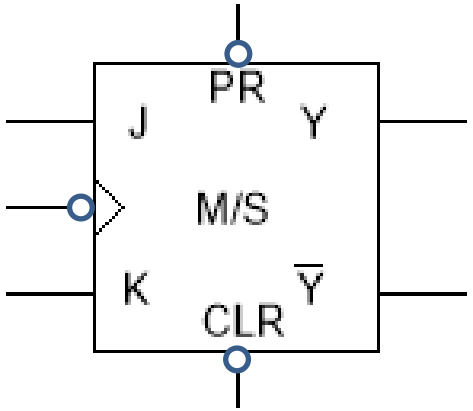




J K MASTER/SLAVE FLIP-FLOP

- If the M's J and K inputs are high, it toggles on the + clock edge and the S toggles on the – clock edge.
- Therefore, no matter what the Master does, the slave copies it.





Sequential Logic

COUNTERS

COUNTERS

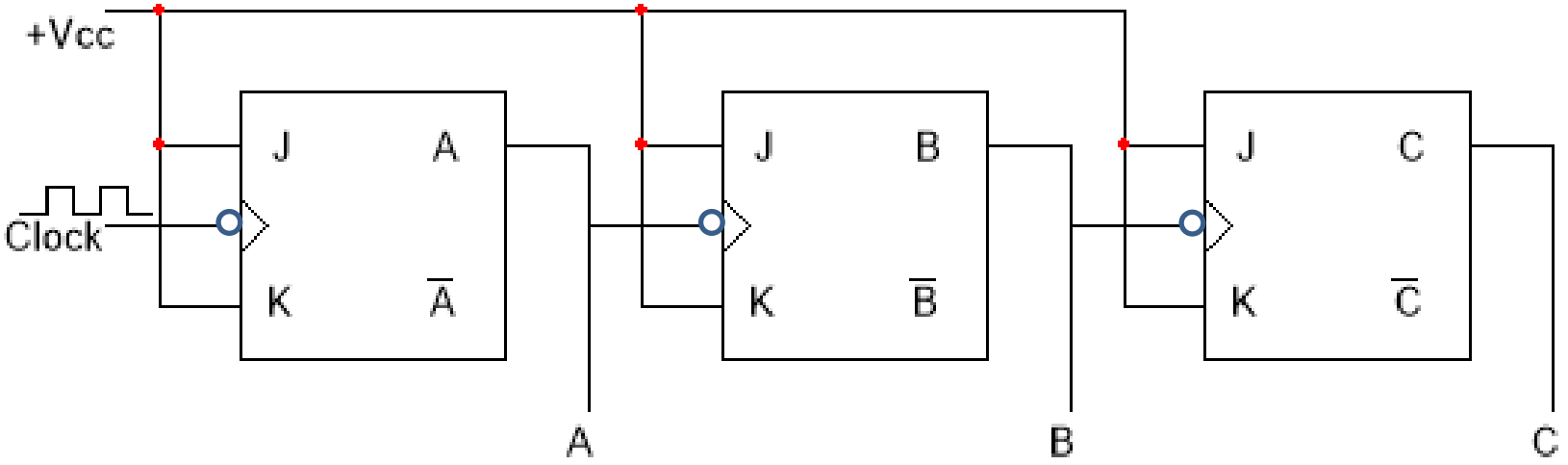
- A counter has the ability to count. It is one of the most useful and versatile subsystems in a digital systems.
- A counter driven by a clock can be used to count the number of clock cycles.
- Since the clock pulses occurs at known intervals, the counter can be used as an instrument for measuring time and therefore period or frequency.
- There are basically two different types of counters
  - Synchronous counters
  - Asynchronous counter

COUNTERS

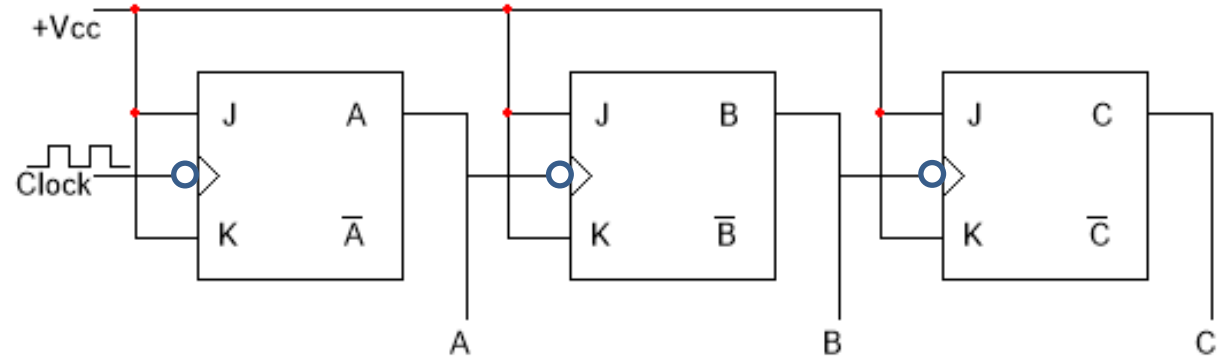
- The ripple counter is simple and straightforward both in operation and construction and usually requires a minimum of hardware.
- It does, however, have a speed limitation. Each FF is triggered by the previous FF, and thus the counter has a cumulative settling time.
- Counter such as these are called *serial* or *asynchronous*.
- In *parallel* or *synchronous* counter, each FF is triggered by the clock (in synchronisation), and thus settling time is simply equal to the delay time of a single FF.
- But that increases the hardware.

3 BIT BINARY RIPPLE COUNTER

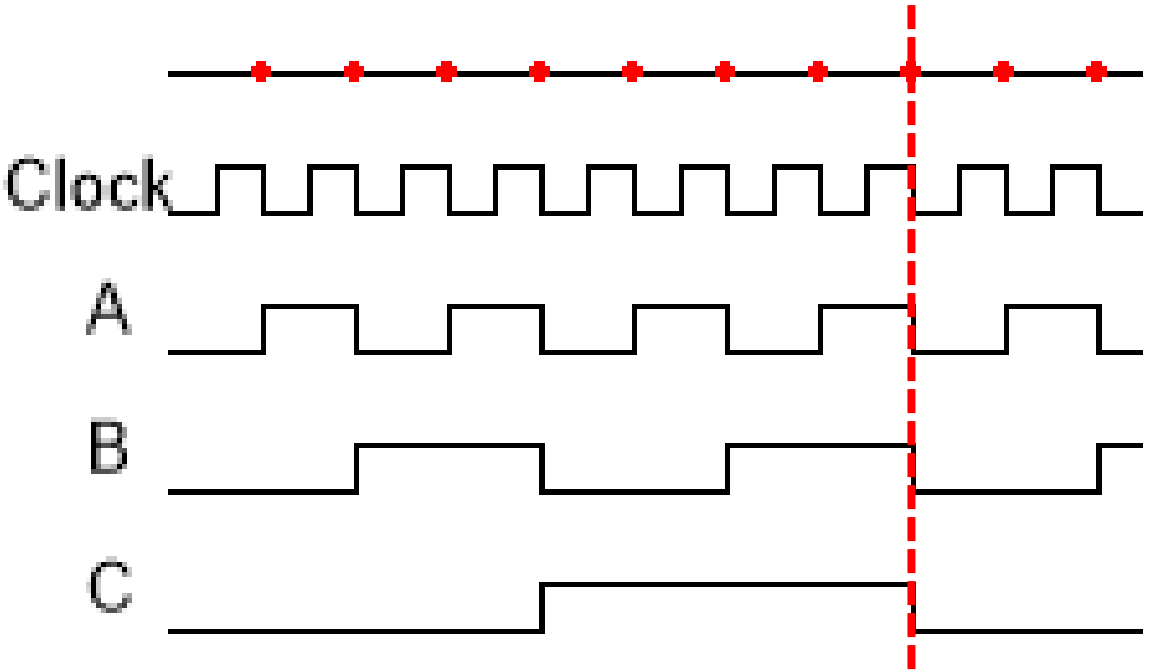
- The three FFs are in cascade. The clock pulse drives the A FF. Output of the A FF drives the B FF. The B FF in turn drives the C FF.
- All the J and K inputs are tied to +Vcc. Thus each FF will change state with a negative transition at its clock input.



3 BIT BINARY RIPPLE COUNTER

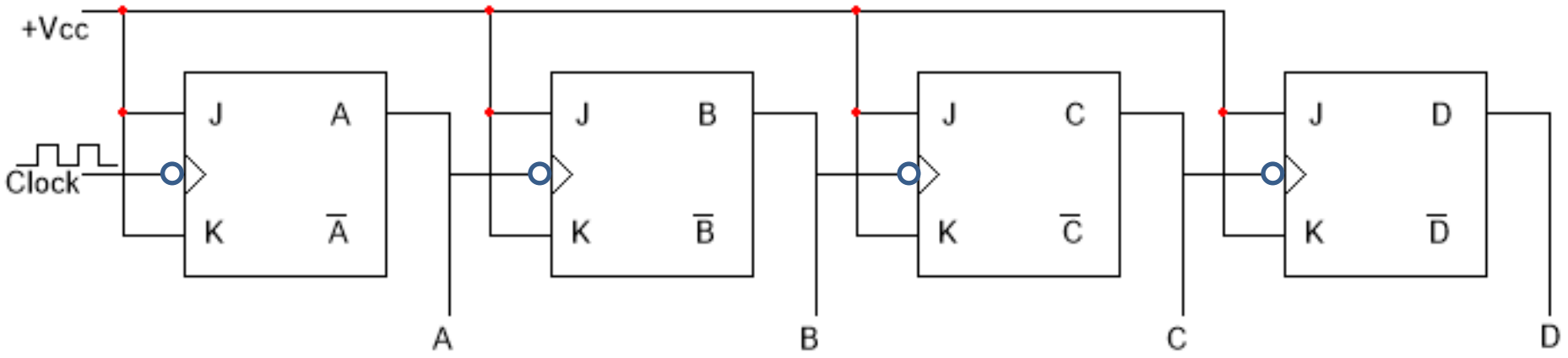


COUNT	C	B	A
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
0	0	0	0

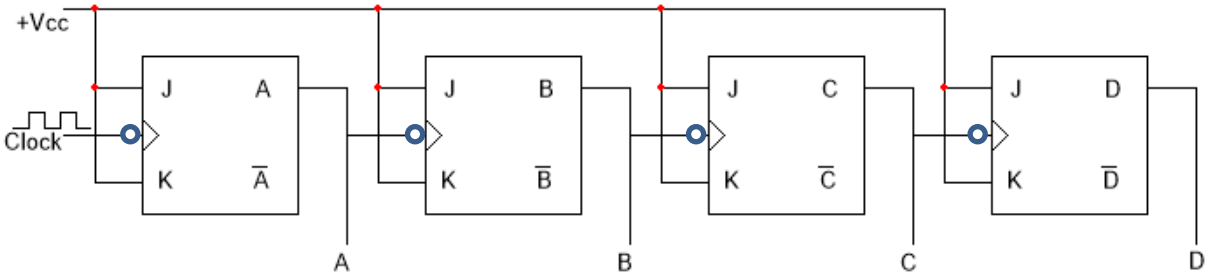


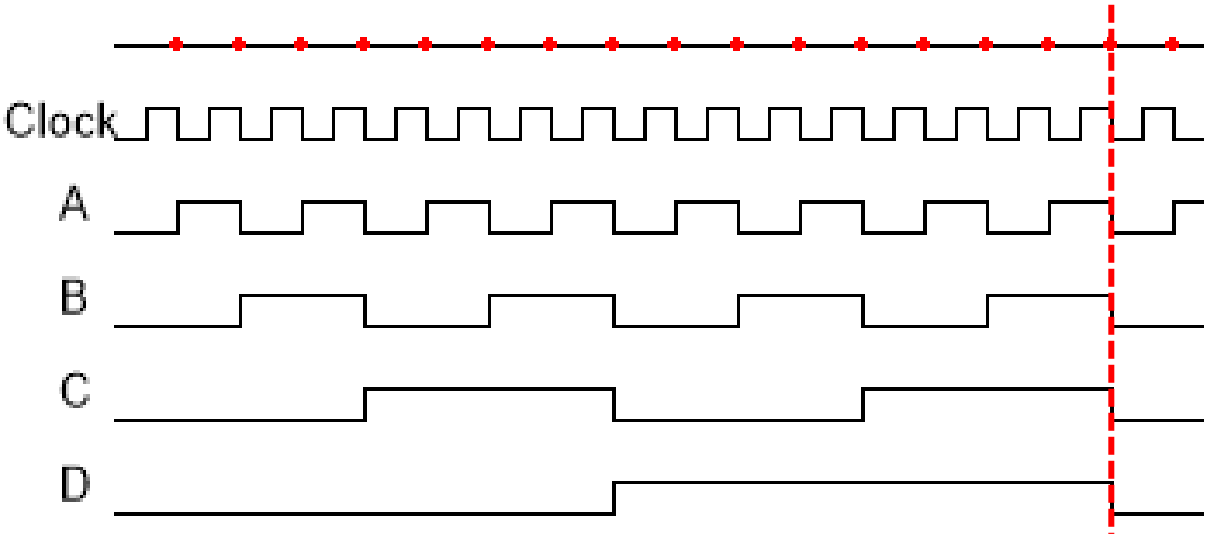
4 BIT BINARY RIPPLE COUNTER

- The three FFs are in cascade. The clock pulse drives the A FF. Output of the A FF drives the B FF. The B FF in turn drives the C FF, which then drives the D FF.
- All the J and K inputs are tied to +Vcc. Thus each FF will change state with a negative transition at its clock input.
- Overall propagation delay time is the sum of the individual delays.



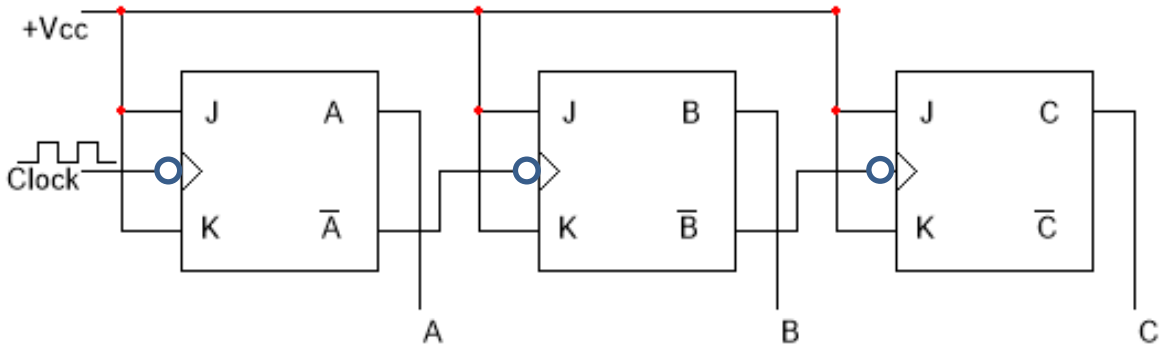
4 BIT BINARY RIPPLE COUNTER

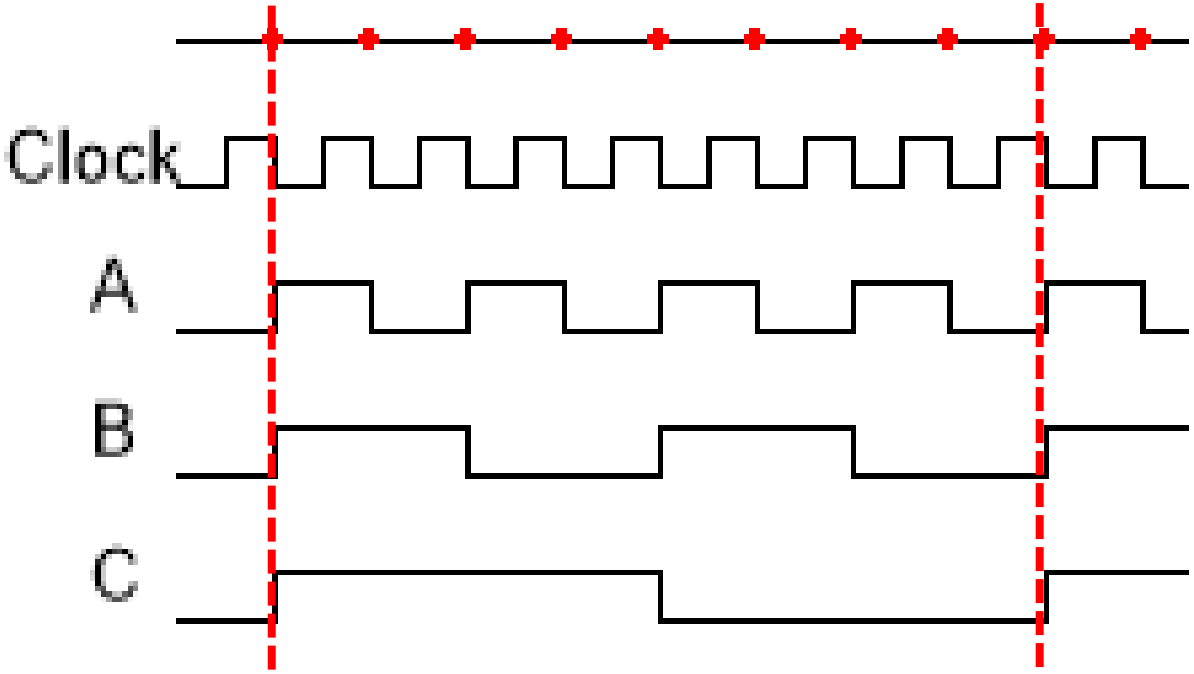




COUNT	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
0	0	0	0	0

3 BIT BINARY DOWN RIPPLE COUNTER

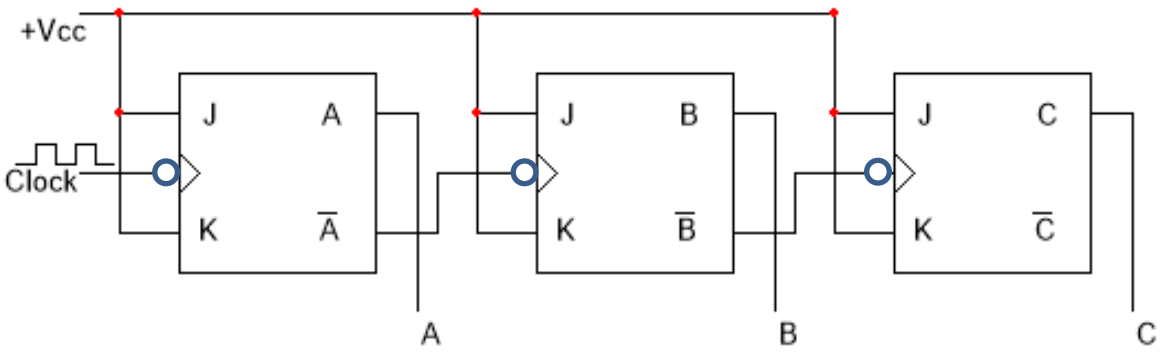




COUNT	C	B	A
7	1	1	1
6	1	1	0
5	1	0	1
4	1	0	0
3	0	1	1
2	0	1	0
1	0	0	1
0	0	0	0
7	1	1	1

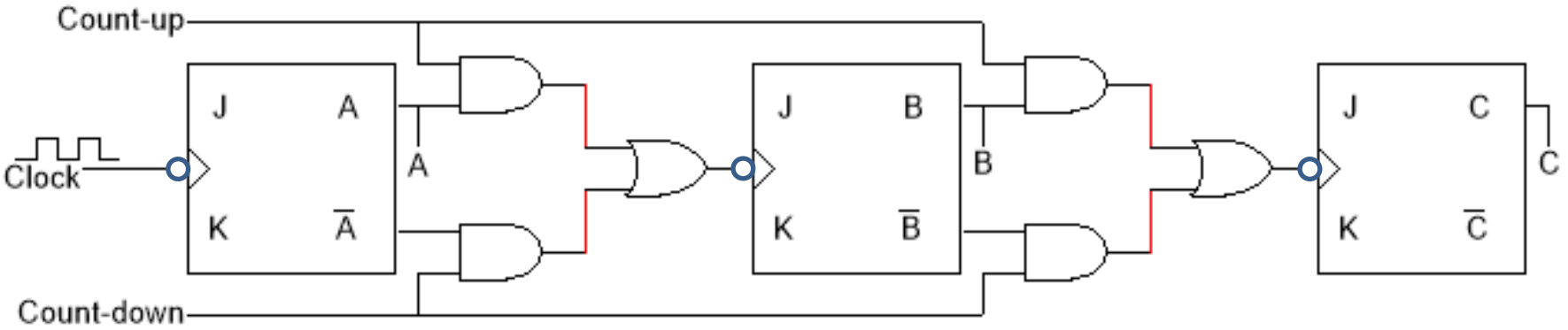
3 BIT BINARY DOWN RIPPLE COUNTER

- The system clock is still used at the clock input to FF A, but the complement of A, is used to drive FF B; likewise, complement of B is used to drive FF C.
- FF A simply toggles with each – edge as before. But FF B will toggle each time A goes high. Similarly, FF C triggered by B and so C will toggle each time B goes high.
- Counter contents are reduced by one count with each clock transition. Thus, counter is operating in count down mode.



3 BIT BINARY UP-DOWN COUNTER

- It is a combination of the two counters; up and down.
- If the count-down control line is low and the count-up control line high, the counter will have count up waveform.
- If the count-down is high and count-up is low; the counter will then be in count-down mode and have count down waveform.



BINARY RIPPLE COUNTER

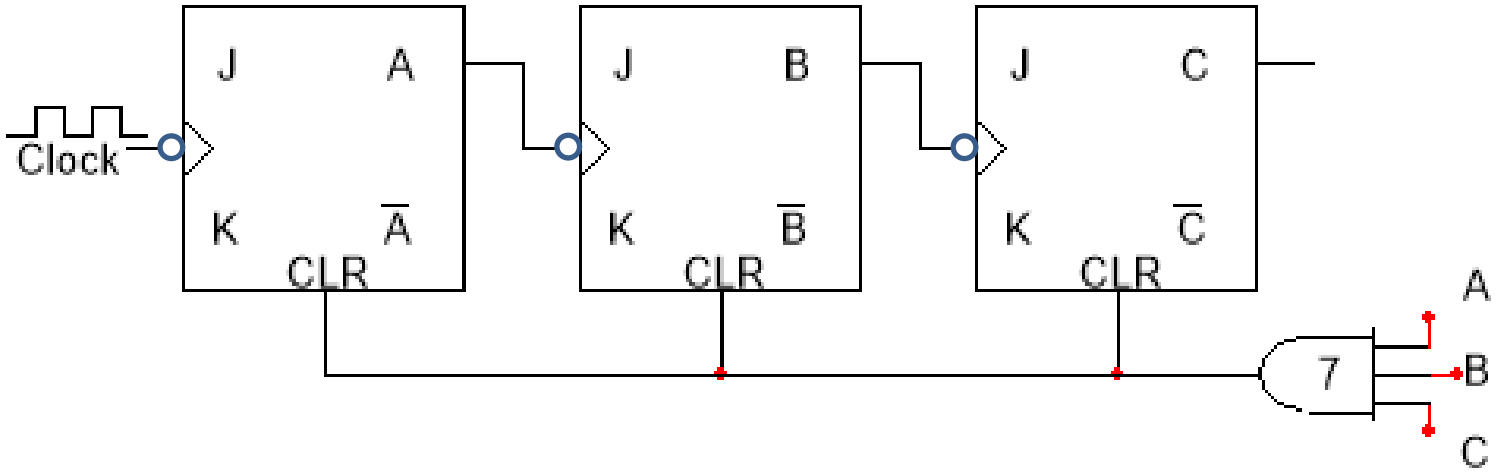
- The total number of counts through which the counter can progress is given by 2^n, where n is the total number of FF used. It is said to have a *natural count of* 2^n.
- Thus, we can build counters which count through 2, 4, 8, 16, 32, etc., states by using proper number of FFs.
- A 3 FF counter is referred to as a modulus-8 (mod-8) counter since it has 8 states. Similarly, a 4 FF counter is a mod-16 counter
- The modulus of a counter is the total number of states through which the counter can progress.

MODIFIED RIPPLE COUNTER

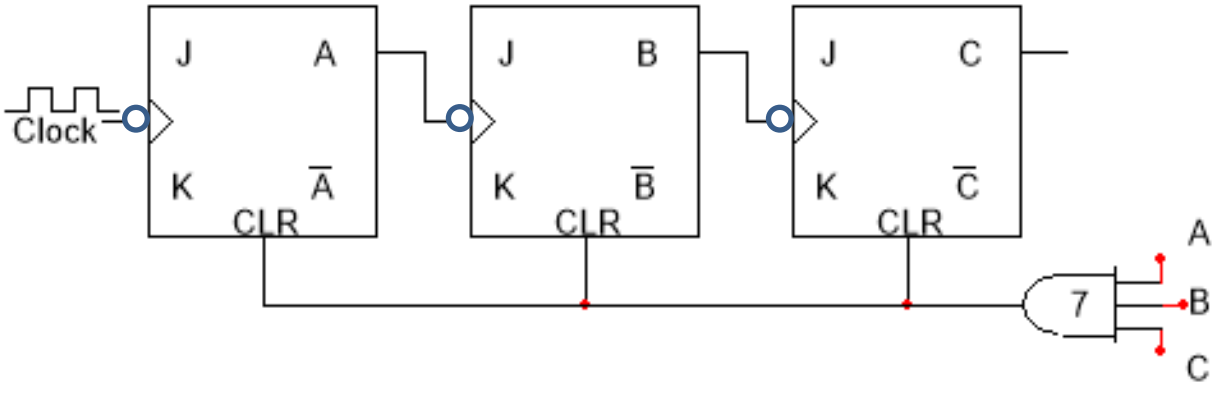
- It is often desirable to construct counters which have moduli other than 2, 4, 8, 16, etc.
- A smaller-modulus counter can always be constructed from a larger-modulus counter by skipping states.
- Such counters are said to have a *modified count*.
- The correct number of FFs is determined by choosing the *lowest natural count* which is greater than the desired *modified count*.
- For example, a mod-7 counter requires 3 FFs, since 8 is the lowest natural count greater than the desired modified count of 7.

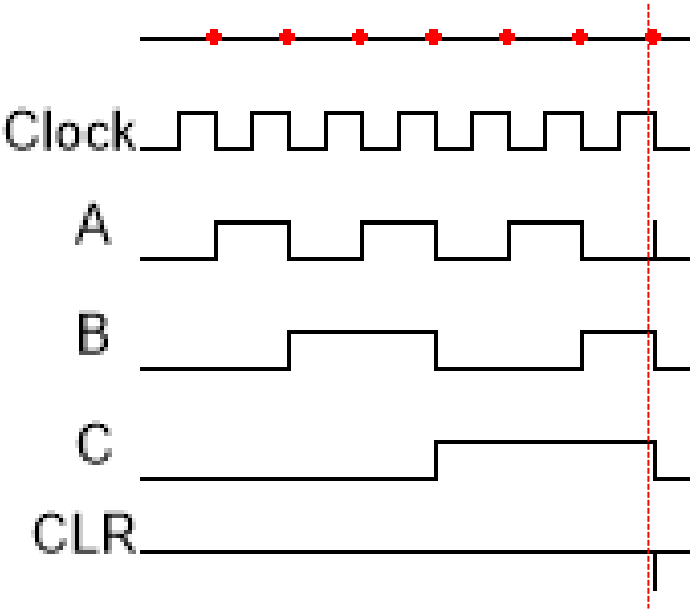
RIPPLE COUNTER USING FEEDBACK

- One method used to cause a counter to skip count is to feedback a signal from some FF to some previous FF.
- Suppose we want to implement a mod-7 counter.
- Since only one count is to be skipped, it would be convenient to let the counter advance through it natural sequence and then reset it one count early.



MOD-7 COUNTER

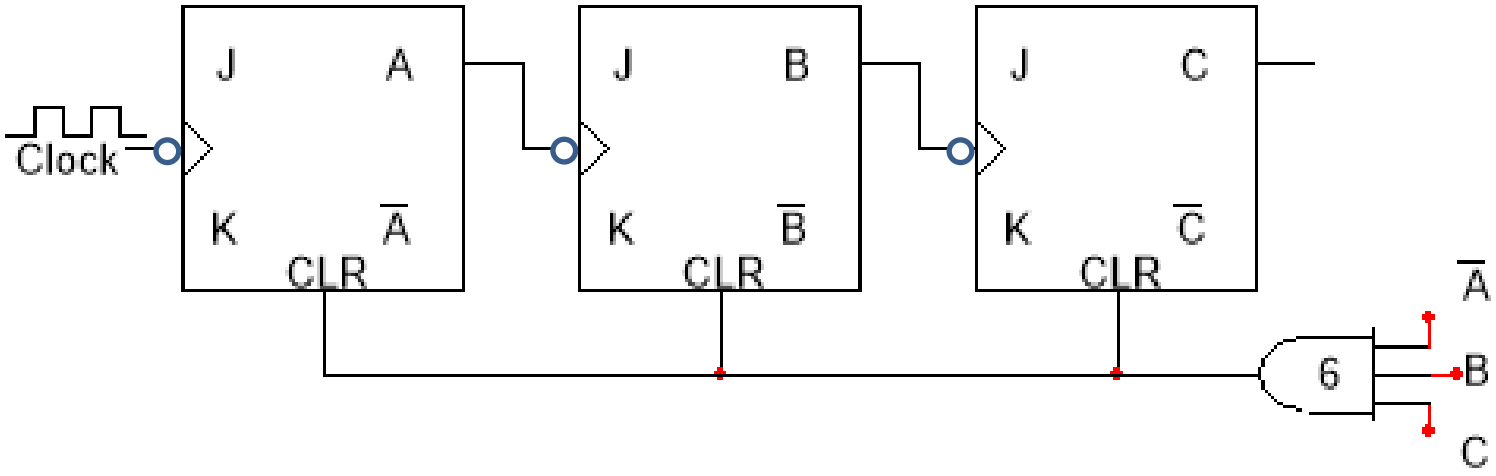




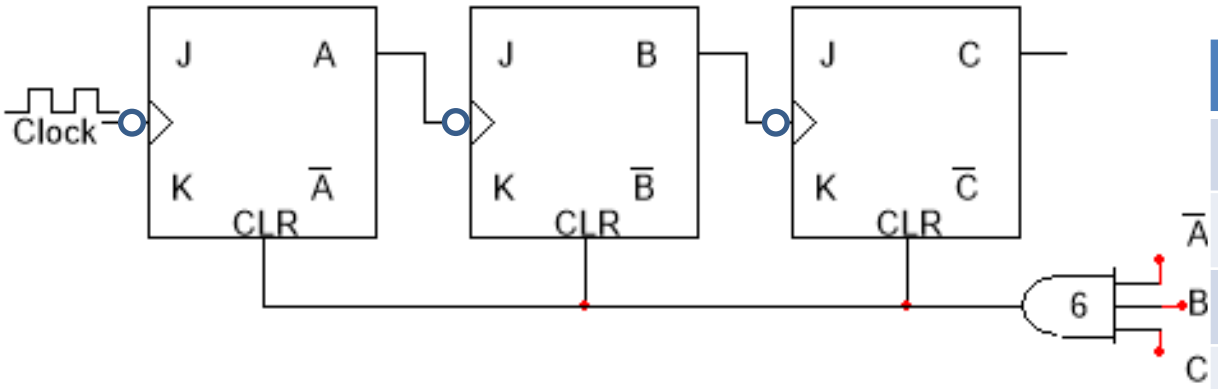
COUNT	C	B	A
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
0	0	0	0

MOD-6 COUNTER

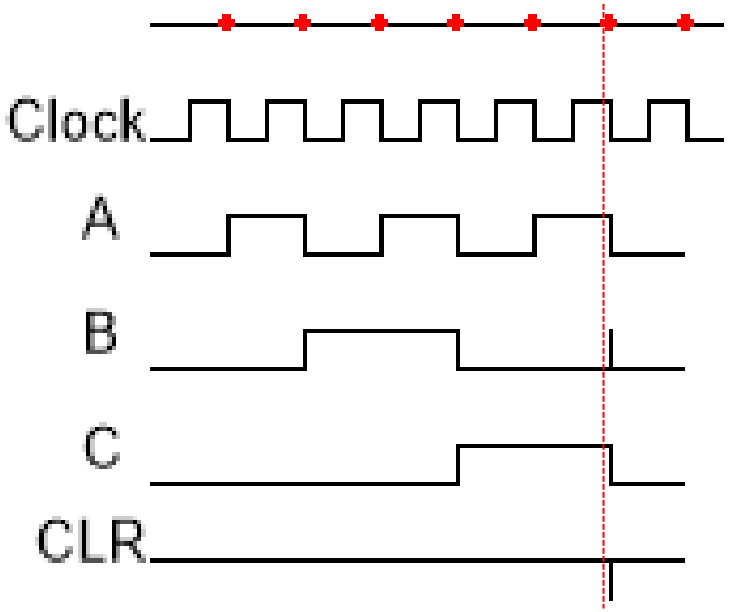
- Suppose we want to implement a mod-6 counter.
- Since two count is to be skipped, it would be convenient to let the counter advance through it natural sequence and then reset it two count early.



MOD-6 COUNTER

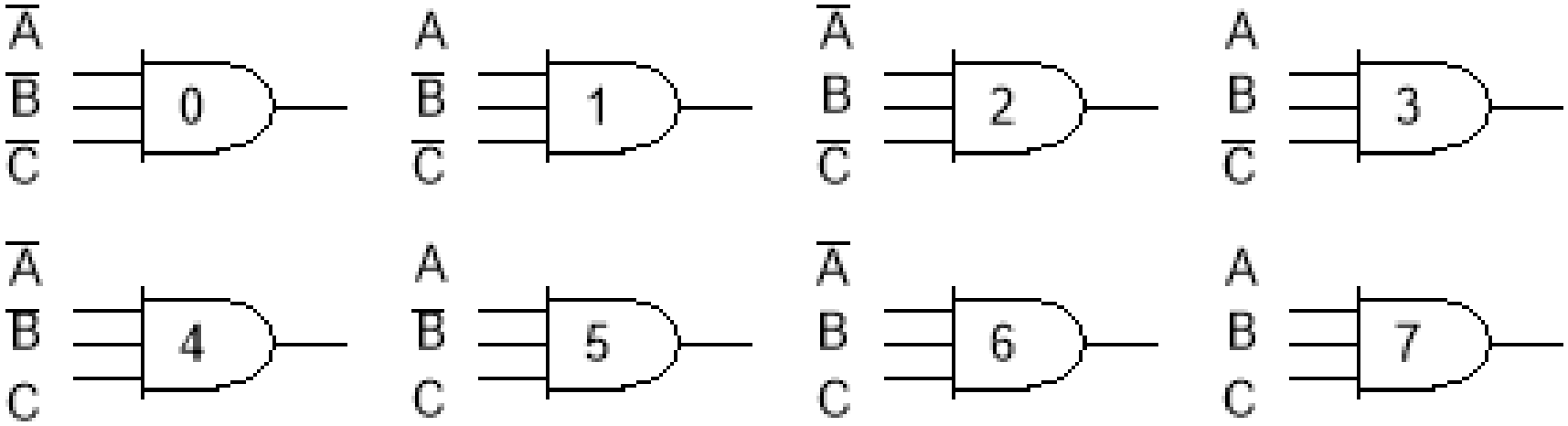


COUNT	C	B	A
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
0	0	0	0



DECODING GATES

- A decoding gate can be connected to the CLR input of all FFs in such a way that the output of the AND gate will be high only when the counter contents are equal to a given state, after which we want to skip all the states.
- All eight gates necessary to decode the eight states of the 3-bit counter are shown.

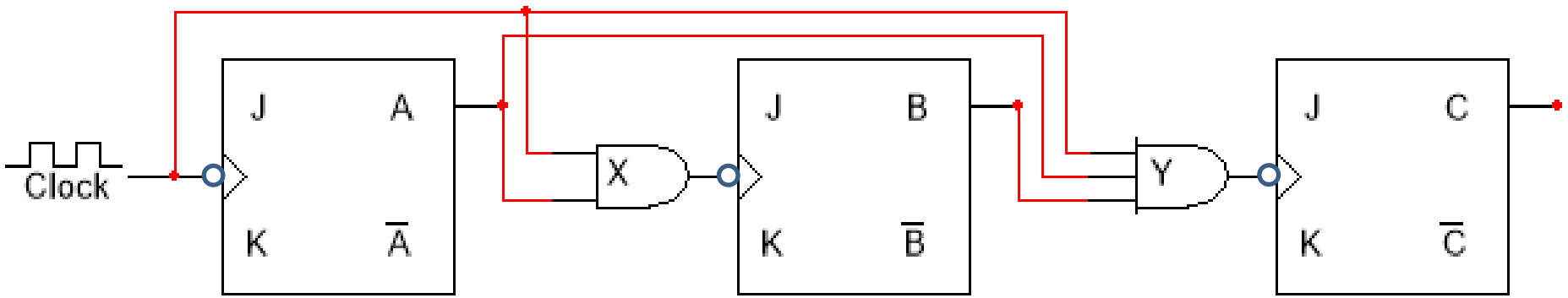


SYNCHRONOUS COUNTER

- The ripple counter is simplest to build, but there is a limit to its highest operating frequency.
- Each FF has a delay time and in a ripple counter these delay times are additive.
- The total *settling time* for the counter is approximately the delay time times the total number of FFs.
- The speed limitation can be overcome by the use of *synchronous* or *parallel counter*.
- Now every FF is triggered by the clock. Thus, they all make their transitions simultaneously.

SYNCHRONOUS COUNTER

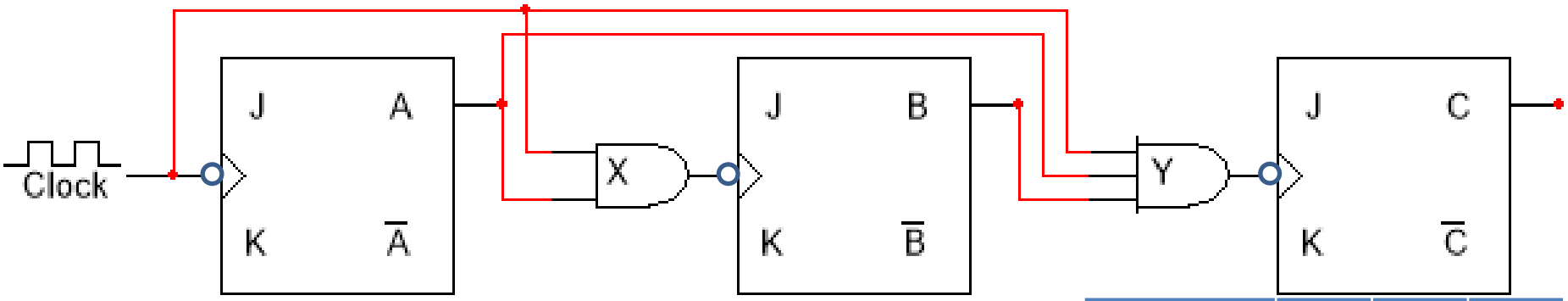
- Keep the J and K inputs of each FF high, such that the FF will toggle with any negative transition at its clock input.
- Then use AND gates to gate every second clock to FF B, every forth clock to FF C, and so on.
- Since the clock pulses are gated to each individual FF, this configuration is referred to as *gate logic*.

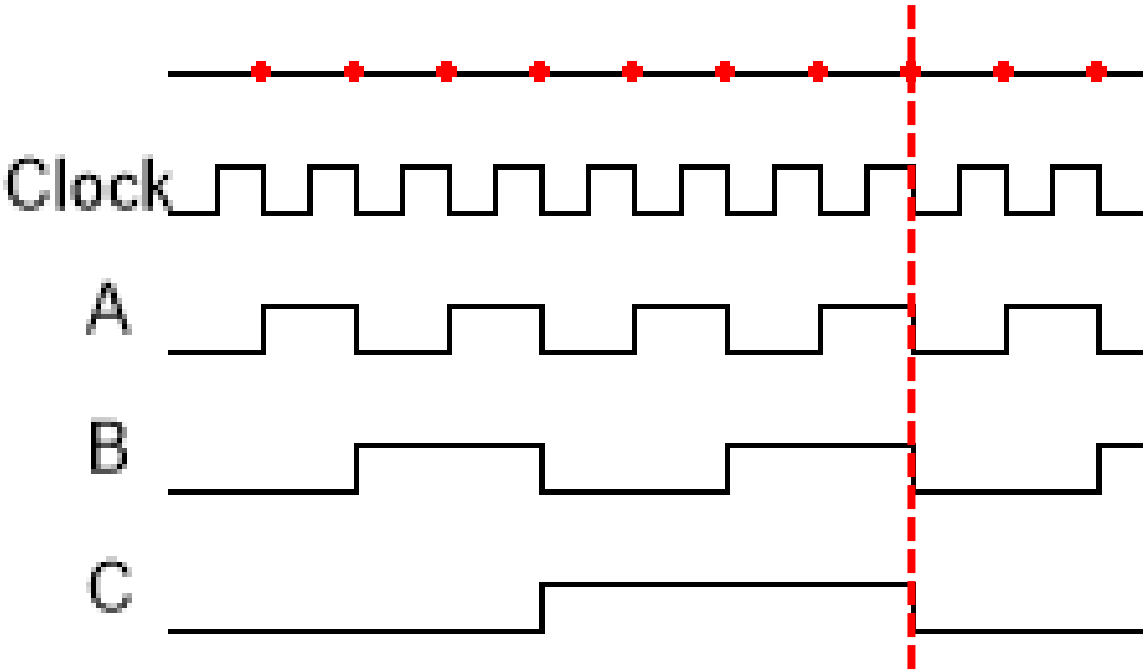


MOD-8 PARALLEL BINARY COUNTER

- The clock is applied directly to FF A which responds to a negative transition at the clock input and toggles. The FF A will change state with each negative clock transition.
- Whenever A is high, AND gate X is enabled and a clock pulse is passed through the gate to the clock input of FF B. The FF B will change state with every alternate negative clock transition.
- Since AND gate Y is enabled and will transmit the clock to FF C only when both A and B are high, the FF C changes state with every forth negative clock transition.

MOD-8 PARALLEL BINARY COUNTER



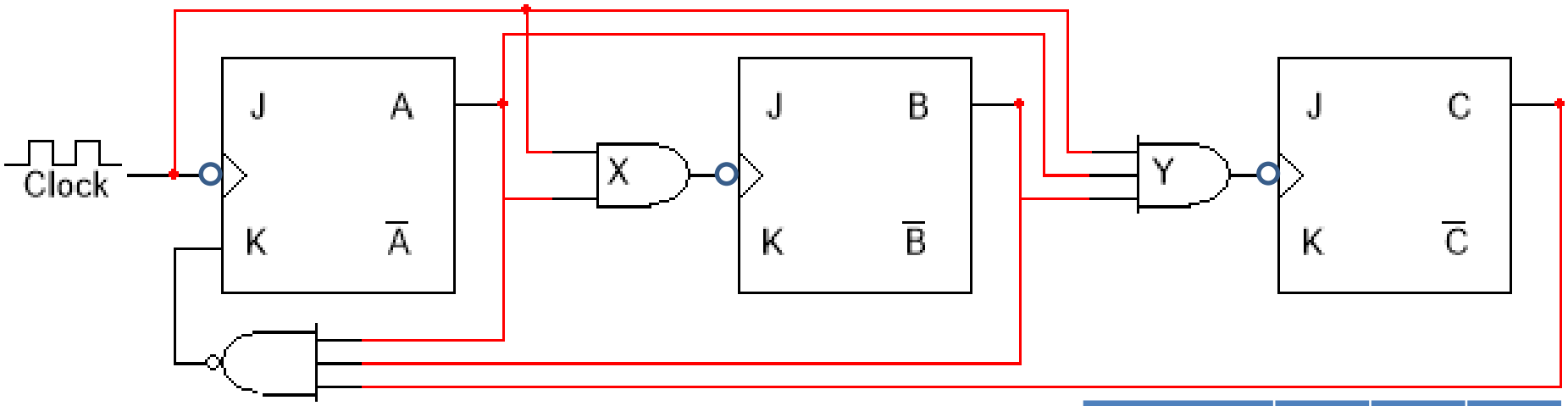


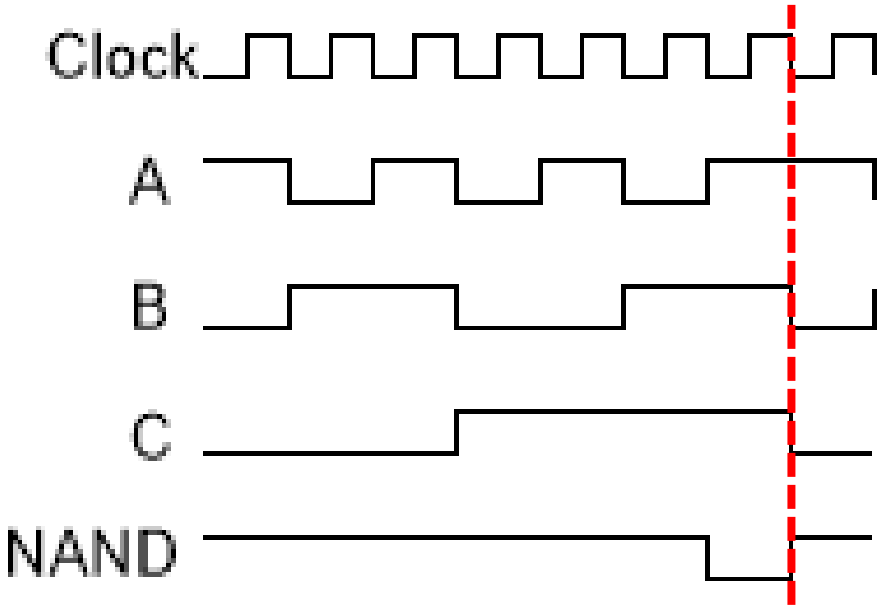
COUNT	C	B	A
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
0	0	0	0

MOD-7 PARALLEL BINARY COUNTER

- The parallel counter can be used as a basis for building counters of other moduli.
- It is necessary to find some means of eliminating desired states from the natural count sequence.
- In Mod-7 counter, all the FFs are high during count 7.
- In changing from count 7 to 0, all FF change to the 0 state
- Prevent FF A from being reset during this transition, with out affecting the FF B and FF C.
- The counter would progress from count 7 to 1.
- Thus, count 0 would be skipped. A Mod-7 counter would be formed.

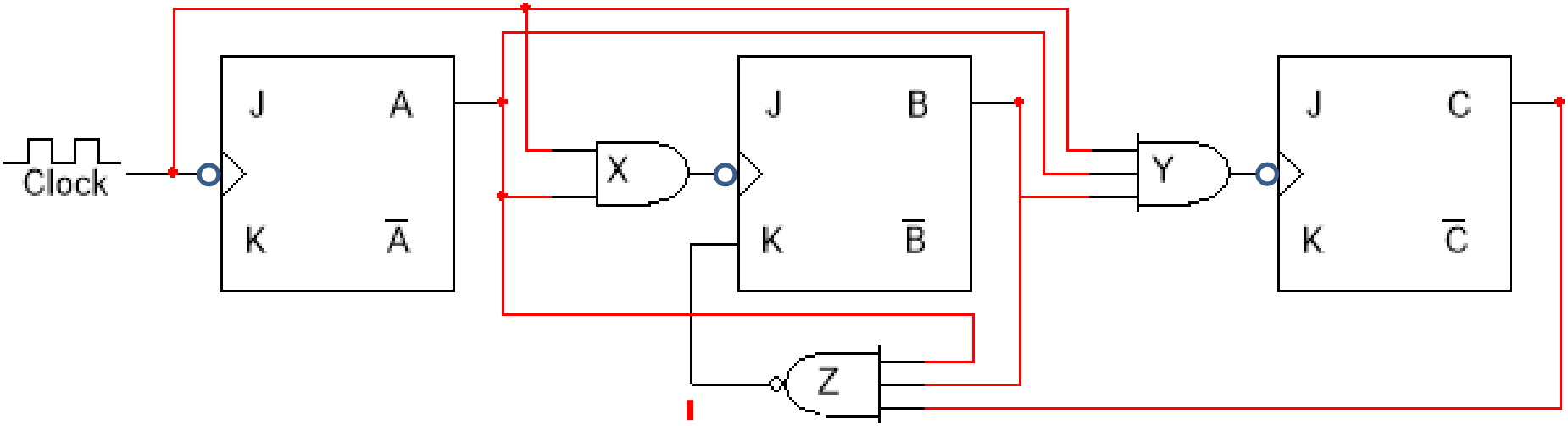
MOD-7 PARALLEL BINARY COUNTER

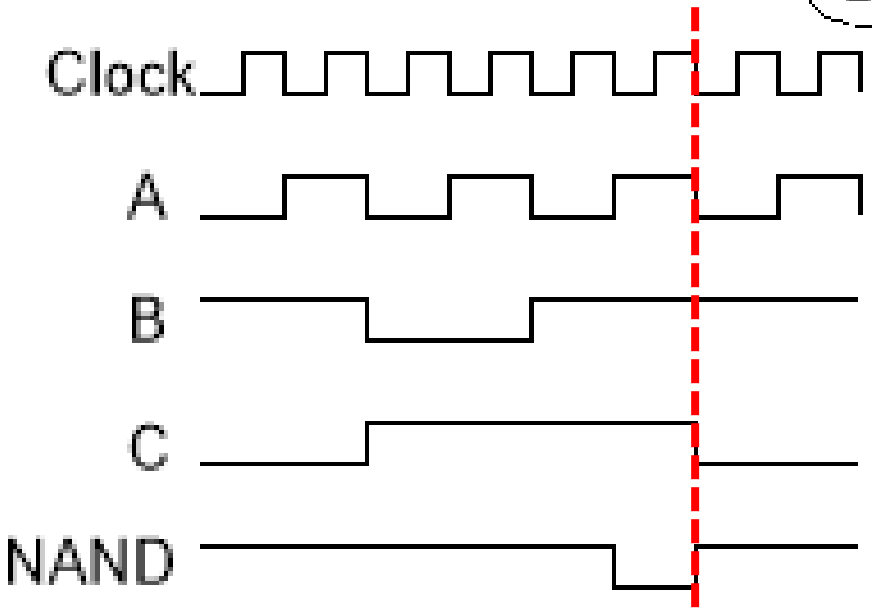




COUNT	C	B	A
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
1	0	0	1

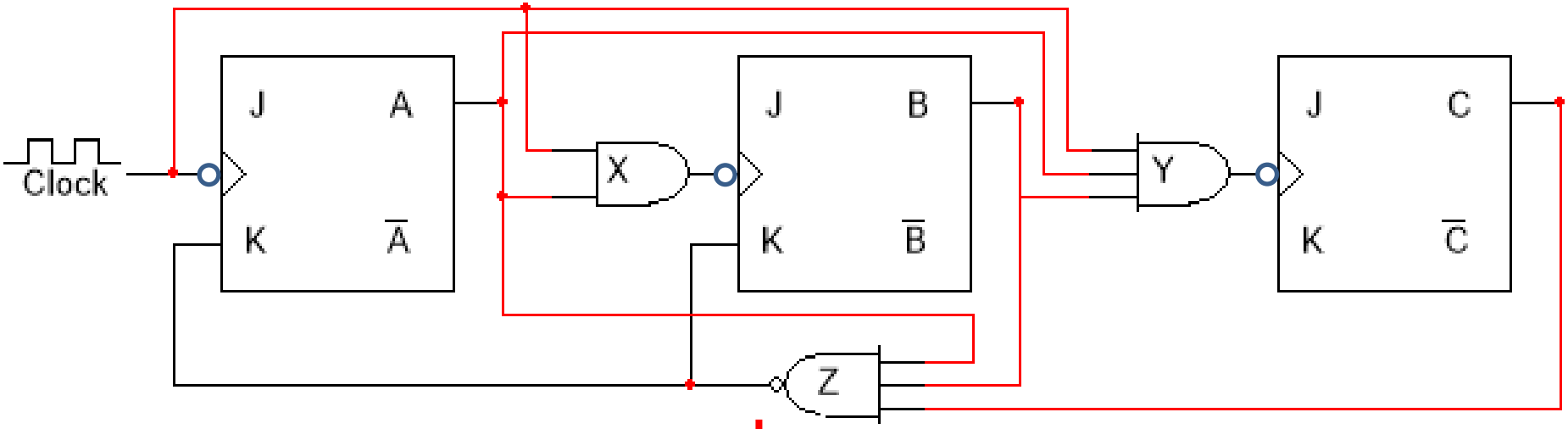
MOD-6 PARALLEL BINARY COUNTER

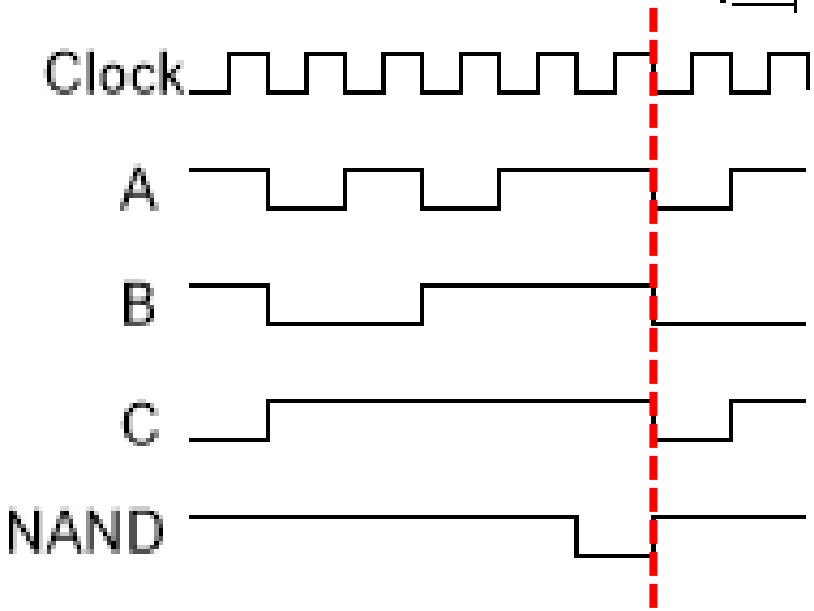




COUNT	C	B	A
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
2	0	1	0

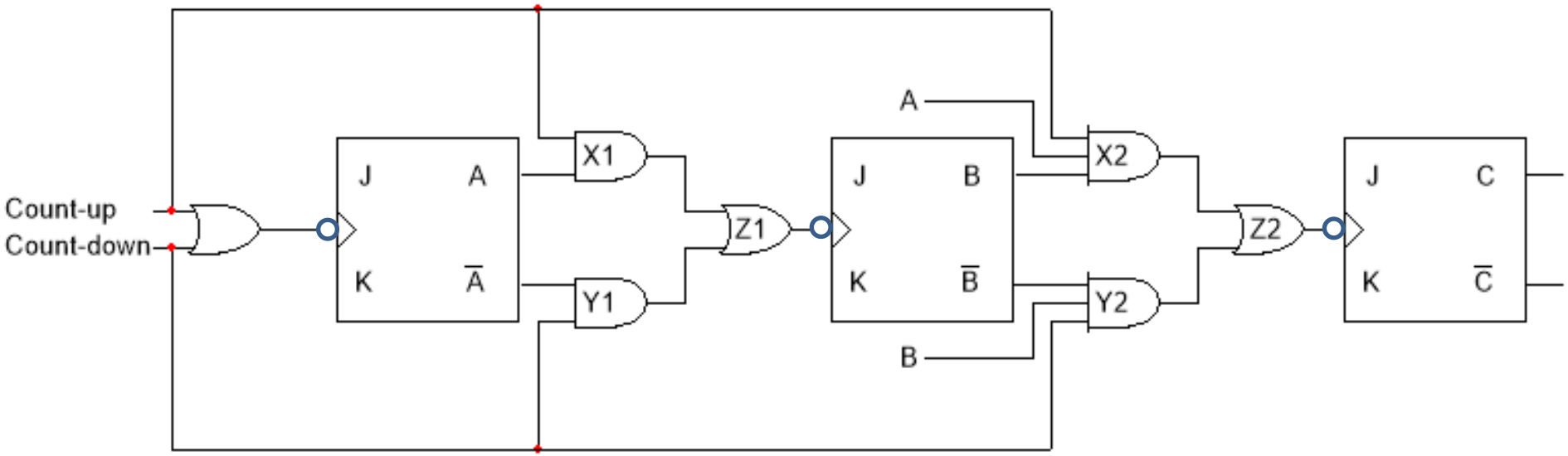
MOD-5 PARALLEL BINARY COUNTER



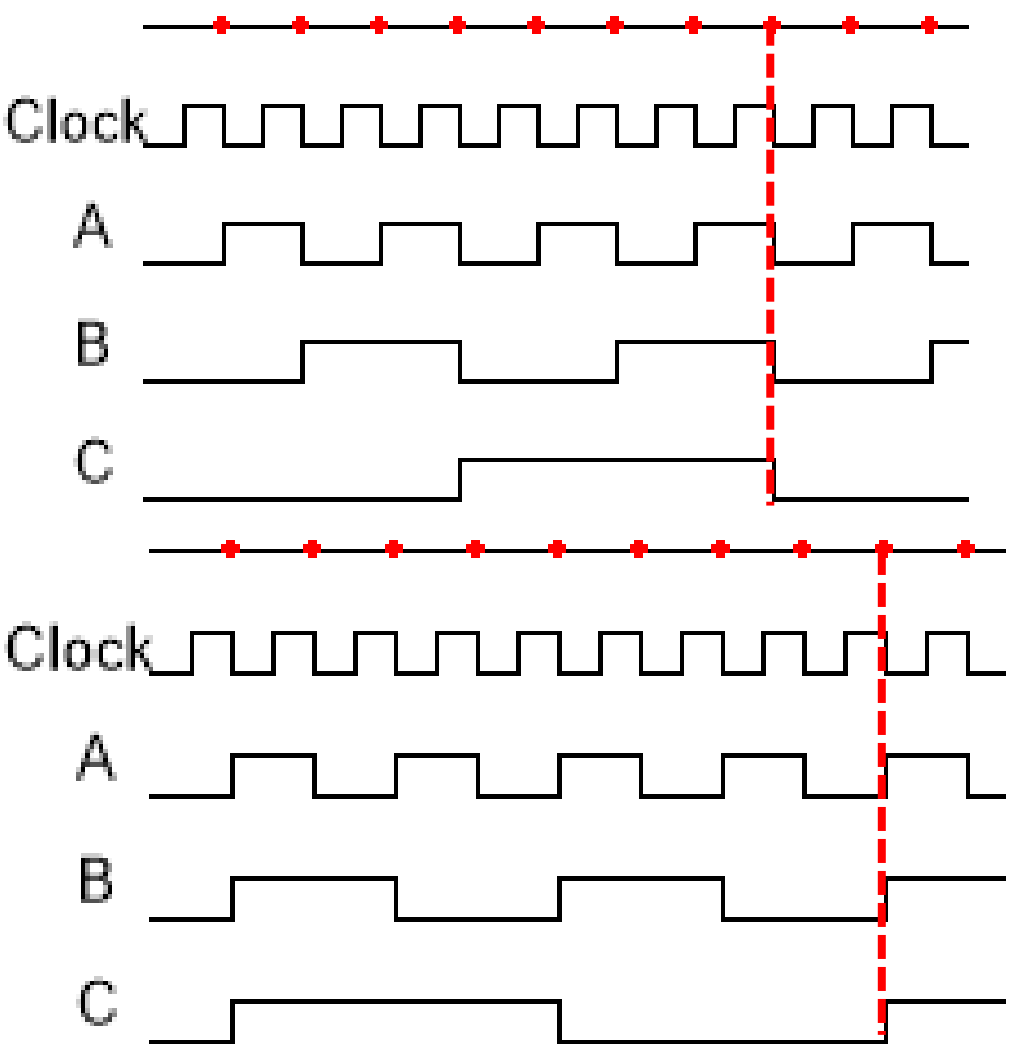


COUNT	C	B	A
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
3	0	1	1

PARALLEL UP/DOWN COUNTER



PARALLEL UP/DOWN COUNTER



COUNT	C	B	A
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
0	0	0	0

PARALLEL UP/DOWN COUNTER

- In count-up mode. A FF must toggle every time all prior FF are in a 1 state, and the clock makes a transition.
- In the count-down mode, FF toggles must occur when all prior FF are in a 0 state.
- In the count-up mode, the system clock is applied at the count-up input, while the count-down input is held low.
- In the count-down mode, the system clock is applied at the count-down input while holding the count-up input low.

Sequential Logic

SHIFT REGISTERS

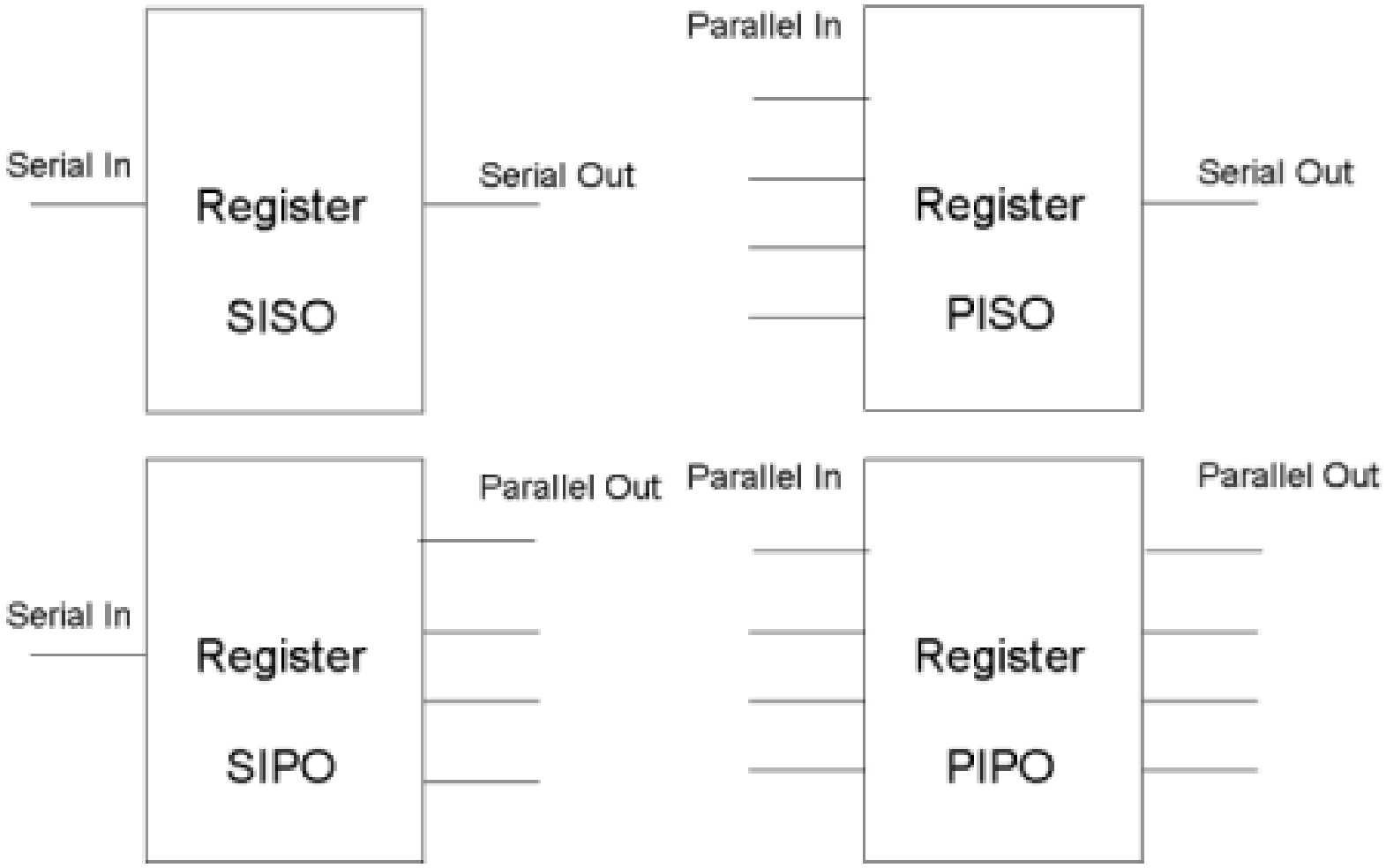
SHIFT REGISTERS

- A register is simply a group of FF that can be used to store a binary number. There must be one FF for each bit in the binary number.
- A register used to store an 8-bit binary number must have eight FFs.
- The FFs must be connected in such a way that the binary number can be entered (shifted) into the register and taken (shifted) out from the register.
- A group of FFs connected to provide either or both of these functions is called shift register.

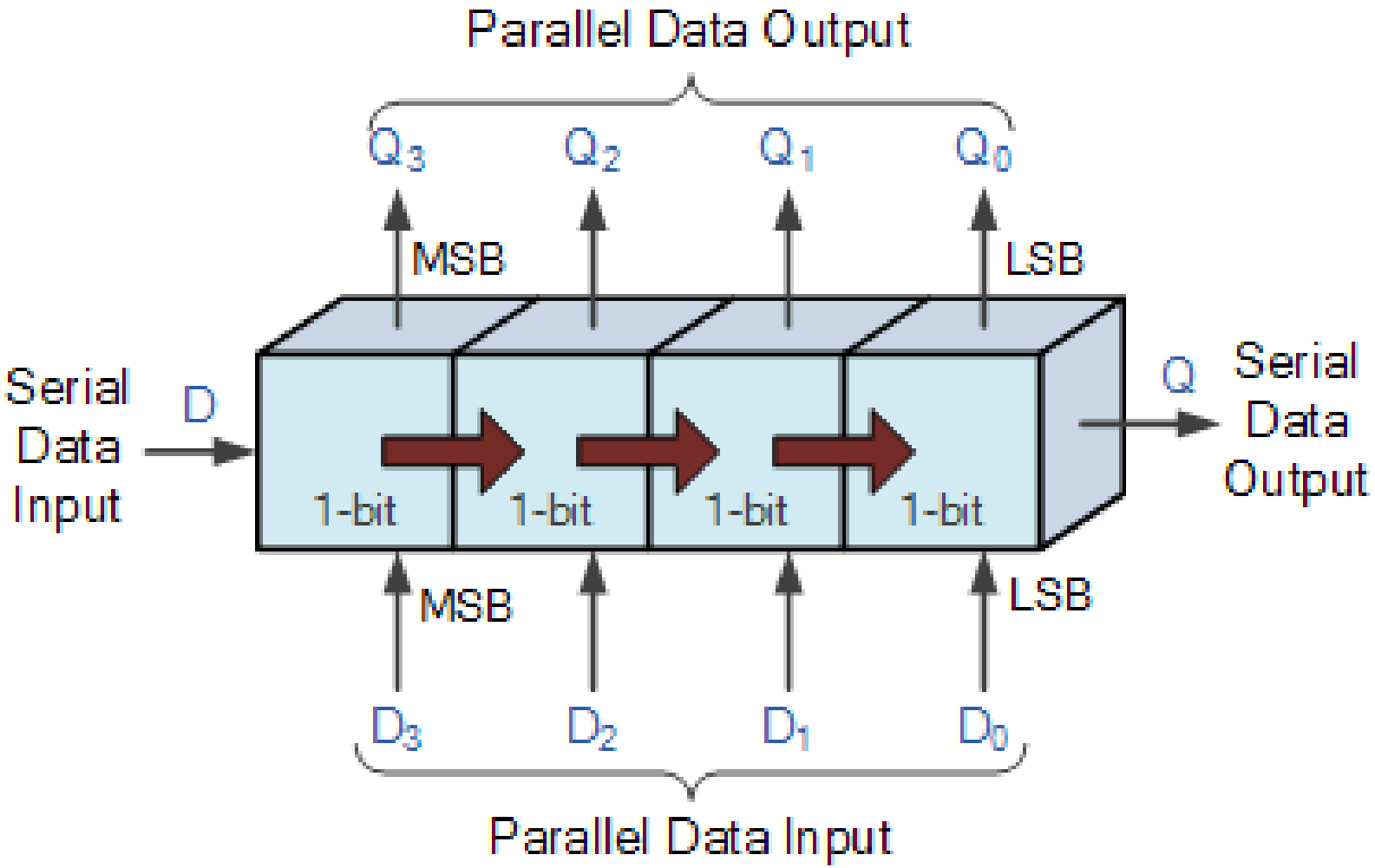
SHIFT REGISTERS

- There are two ways to shift data into a register (serial or parallel) and similarly two ways to shift the data out of the register.
- This leads to the construction of four basic register types;
  - Serial In – Serial Out SISO 54/74L91 8 bits
  - Serial In – Parallel Out SIPO 54/76164 8 bits
  - Parallel In – Serial Out PISO 54/76165 8 bits
  - Parallel In – Parallel Out PIPO 54/74198 8 bits

SHIFT REGISTERS

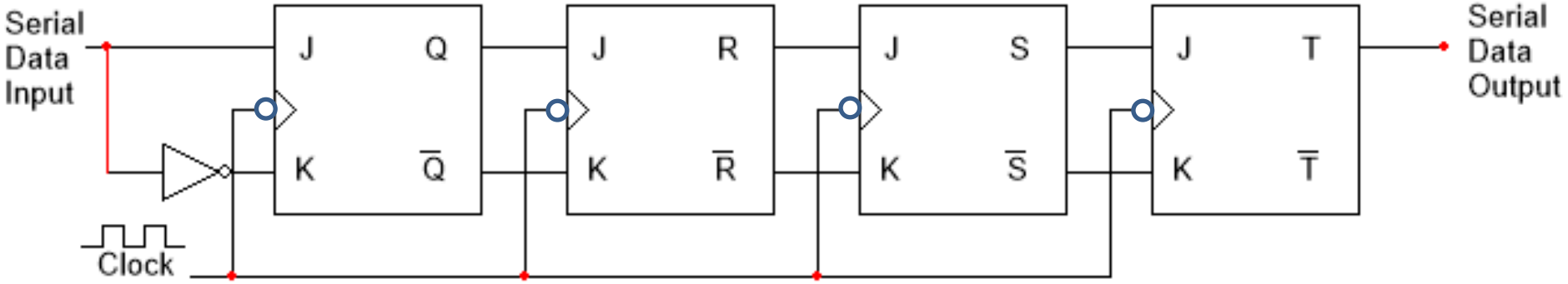


SHIFT REGISTERS



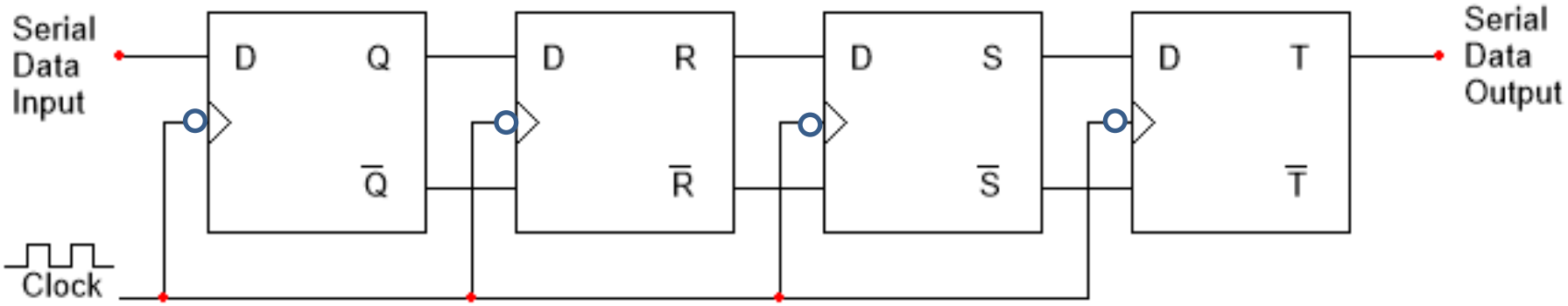
SISO

- The data is allowed to flow straight through the register and out of the other end. Since there is only one output, the data leaves the shift register one bit at a time in a serial pattern.
- Serial-in, serial-out shift registers delay data by one clock time for each stage, therefore, acts as a temporary storage device or it can act as a time delay device.



SISO

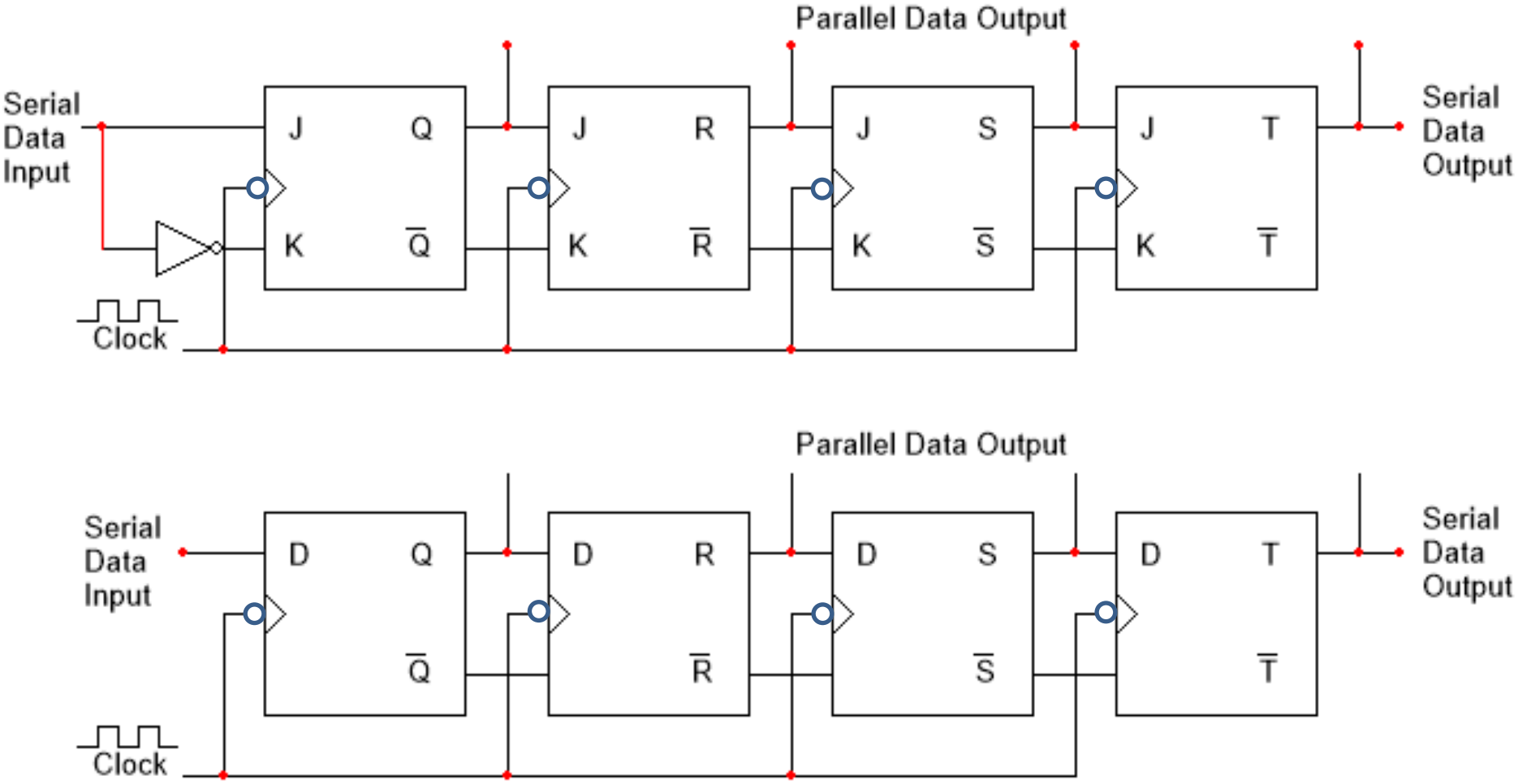
- The SISO shift register is the simplest as it has only three connections, the serial data input which determines what enters the left hand FF, the serial data output which is taken from the output of the right hand FF and the sequencing clock signal.
- The logic circuit diagram below shows a generalized serial-in serial-out shift register.



SIPO

- A serial-in/parallel-out shift register is similar to the serial-in/serial-out shift register in that it shifts data into register at serial-in pin and shifts data out at the serial-out pin.
- It is different in that it makes all the FFs available as outputs. Therefore, a serial-in/parallel-out shift register converts data from serial format to parallel format.
- If four data bits are shifted in by four clock pulses at data-in, the data becomes available simultaneously on the four Outputs Q, R, S, T after the fourth clock pulse.

SIPO


Parallel Data Output
Serial Data Input
J
Q
K
$\overline{Q}$
J
R
K
$\overline{R}$
J
S
K
$\overline{S}$
J
T
K
$\overline{T}$
Serial Data Output
Clock
Parallel Data Output
Serial Data Input
D
Q
$\overline{Q}$
D
R
$\overline{R}$
D
S
$\overline{S}$
D
T
$\overline{T}$
Serial Data Output
Clock

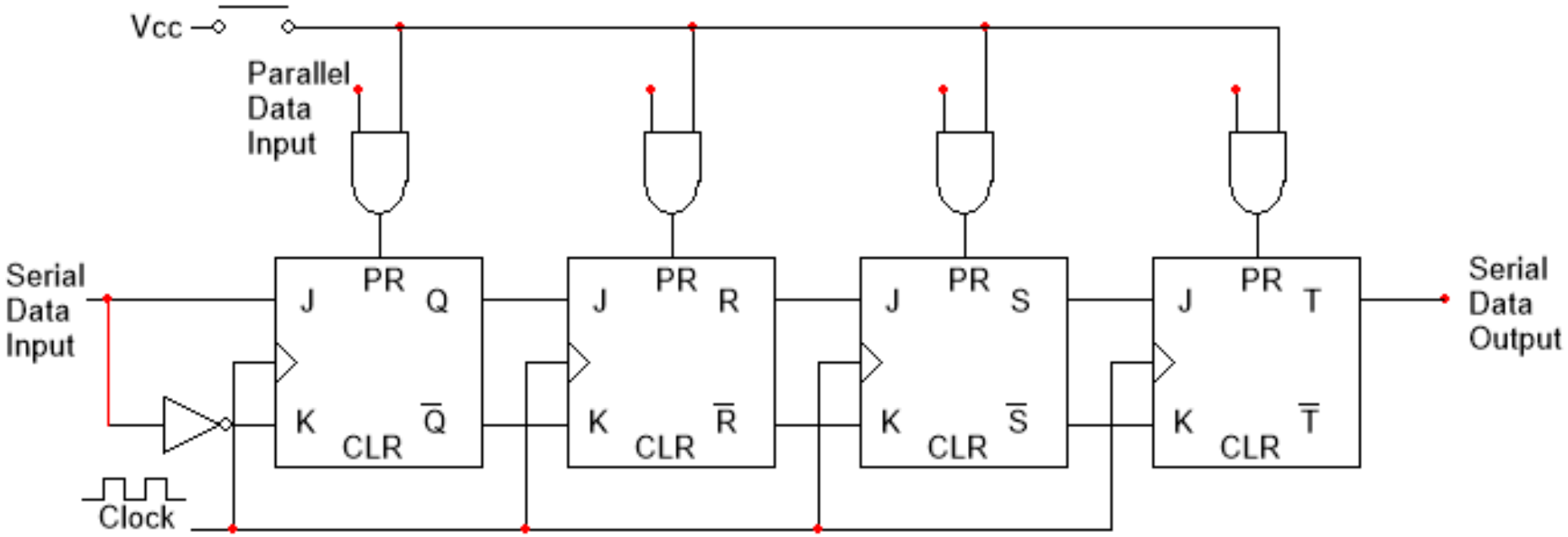
PISO

- The Parallel-in/Serial-out shift register acts in the opposite way to the serial-in/parallel-out.
- The data is loaded into the register in a parallel format in which all the data bits enter their inputs simultaneously, to the parallel input pins of the register.
- The data is then read out sequentially in the normal shift-right mode from the register at T representing the data present at Q to T.
- This data is outputted one bit at a time on each clock cycle in a serial format.

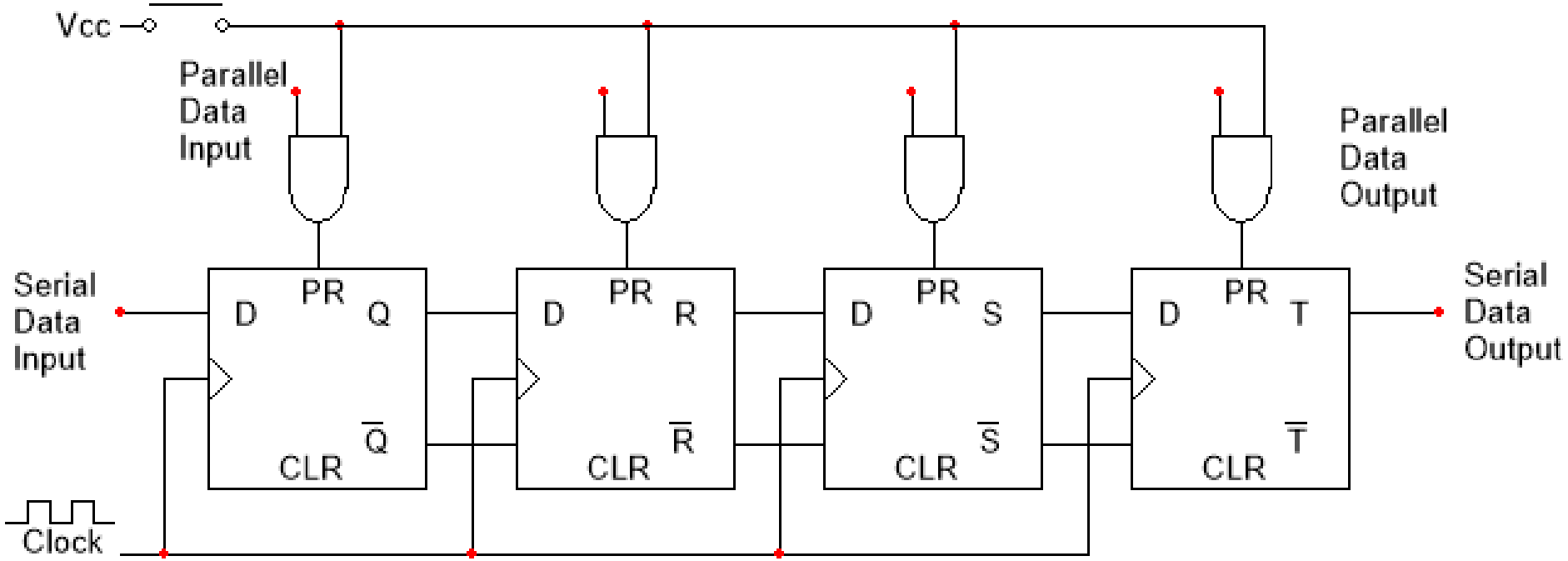
PISO

- It is important to note that with this type of data register a clock pulse is not required to parallel load the register as it is already present, but four clock pulses are required to unload the data.
- As this type of shift register converts parallel data, such as an 8-bit data word into serial format, it can be used to multiplex many different input lines into a single serial DATA stream which can be sent directly to a computer or transmitted over a communications line.

PISO


Vcc
Parallel
Data
Input
Serial
Data
Input
J
PR
Q
K
CLR
$\overline{Q}$
J
PR
R
K
CLR
$\overline{R}$
J
PR
S
K
CLR
$\overline{S}$
J
PR
T
K
CLR
$\overline{T}$
Serial
Data
Output
Clock

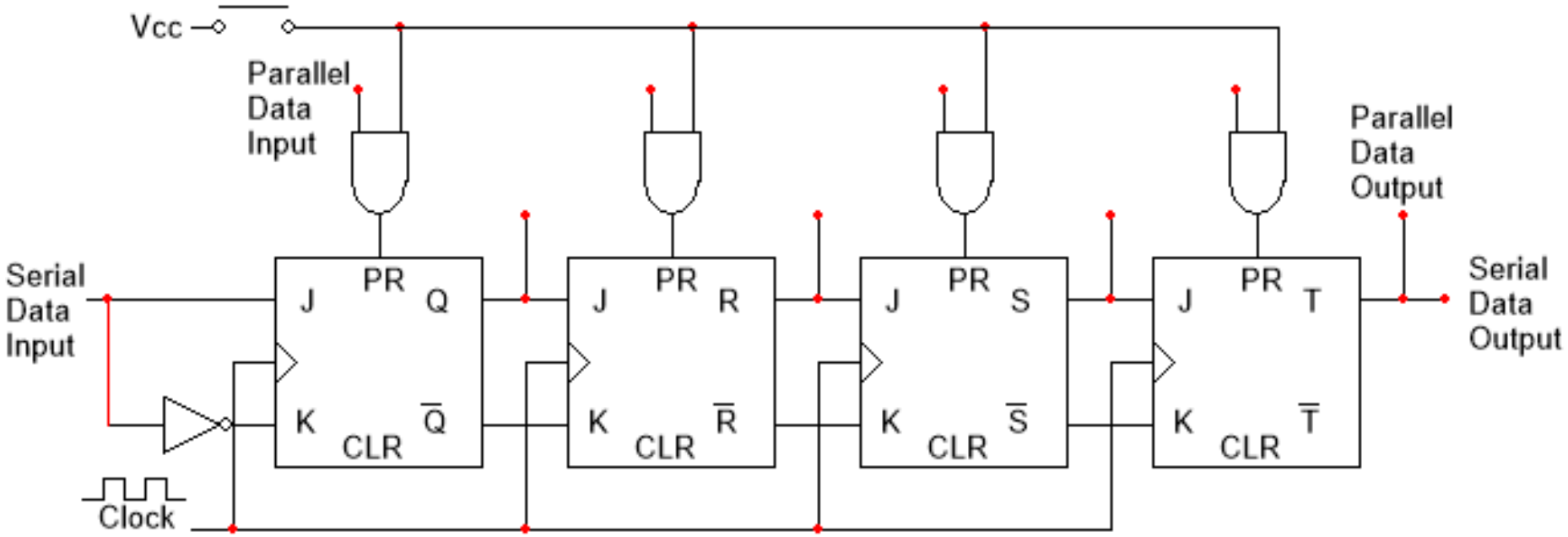
PISO


Vcc
Parallel
Data
Input
Parallel
Data
Output
Serial
Data
Input
D
PR
Q
CLR
Q
D
PR
R
CLR
R
D
PR
S
CLR
S
D
PR
T
CLR
T
Serial
Data
Output
Clock

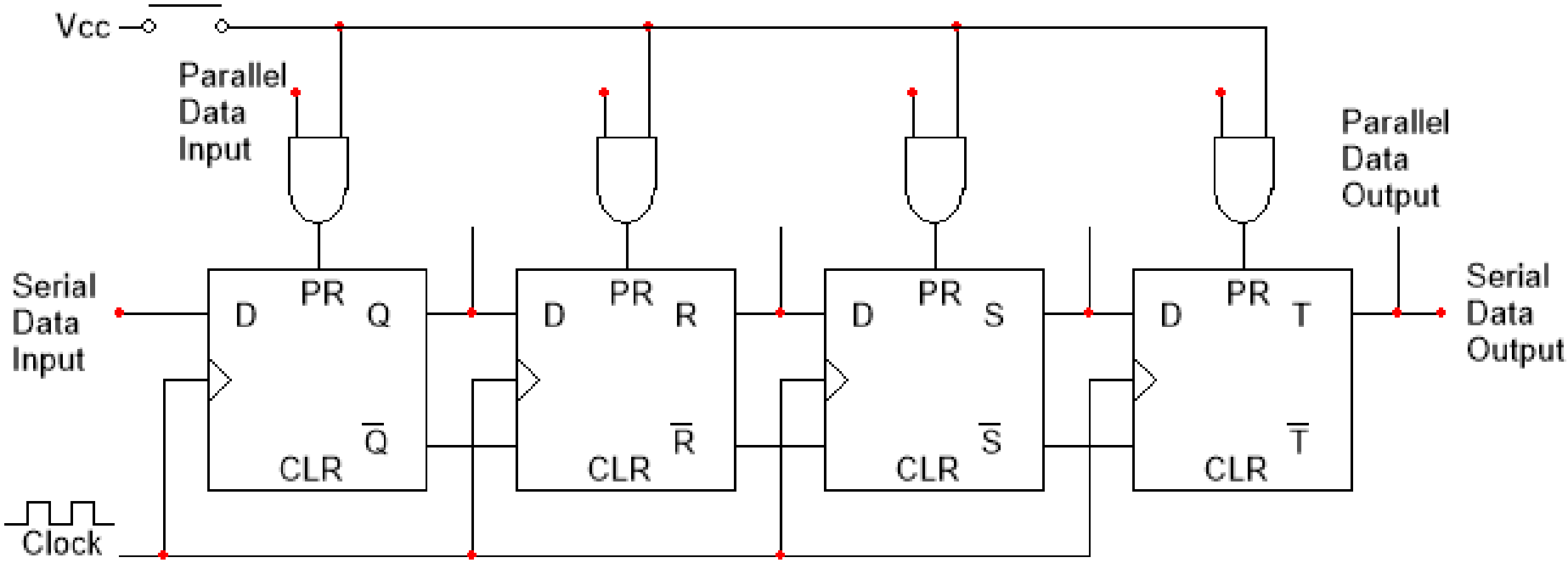
PIPO

- The final mode of operation is the Parallel-in/Parallel-out Shift Register.
- This type of shift register also acts as a temporary storage device or as a time delay device similar to the SISO.
- The data is presented in a parallel format to the parallel input pins and then transferred together directly to their respective output pins by the same clock pulse.
- Then one clock pulse loads and unloads the register.
- The PIPO shift register has only three connections, the parallel input which determines what enters the FF, the parallel output and the sequencing clock signal.

PIPO


Vcc
Parallel
Data
Input
Parallel
Data
Output
Serial
Data
Input
Serial
Data
Output
PR
J
Q
K
CLR
$\overline{Q}$
PR
J
R
K
CLR
$\overline{R}$
PR
J
S
K
CLR
$\overline{S}$
PR
J
T
K
CLR
$\overline{T}$
Clock

PIPO


Vcc
Parallel
Data
Input
Parallel
Data
Output
Serial
Data
Input
Serial
Data
Output
D
PR
Q
CLR
Q
D
PR
R
CLR
R
D
PR
S
CLR
S
D
PR
T
CLR
T
Clock

SISO, SIPO & PISO

CLEAR

FF0	FF1	FF2	FF3
0	0	0	0

CLEAR
1001

Q0	Q1	Q2	Q3
0	0	0	0

CLEAR

Q0	Q1	Q2	Q3
0	0	0	0

SHIFT REGISTERS

Then to summarise a little about **Shift Registers**

- A simple **Shift Register** can be made using only D-type FFs, one FF for each data bit.
- The output from each FF is connected to the D input of the FF at its right.
- Shift registers hold the data in their memory which is moved or “shifted” to their required positions on each clock pulse.
- Each clock pulse shifts the contents of the register one bit position to either the left or the right.

SHIFT REGISTERS

- The data bits can be loaded one bit at a time in a series input (SI) configuration or be loaded simultaneously in a parallel configuration (PI).
- Data may be removed from the register one bit at a time for a series output (SO) or removed all at the same time from a parallel output (PO).
- One application of shift registers is in the conversion of data between serial and parallel, or parallel to serial.
- Shift registers are identified individually as SIPO, SISO, PISO, PIPO, or as a Universal Shift Register with all the functions combined within a single device.