MODULE – 4

FLIP-FLOPs, REGISTERS & COUNTERS

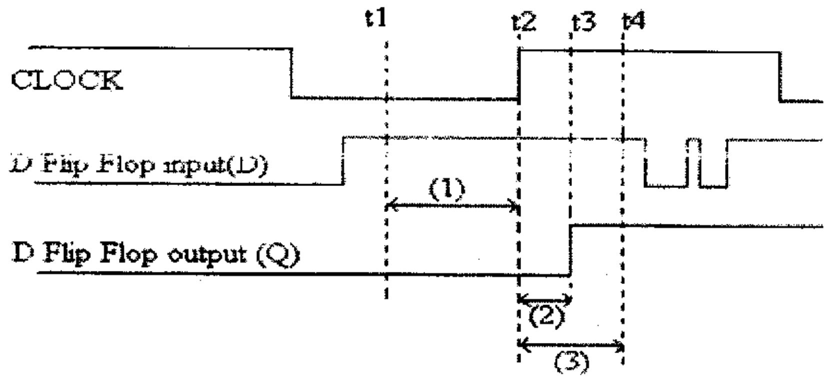
FLIP-FLOPS

FLIP-FLOP TIMING:

The *propagation delay (t_p)* represents the amount of time it takes for the output of a gate or flip-flop to change states after the input changes. For example, if data sheet of an edge-triggered d flip-flop lists t_p = 10 ns, it takes about 10 ns for the Q to change states after D has been sampled by the clock edge.

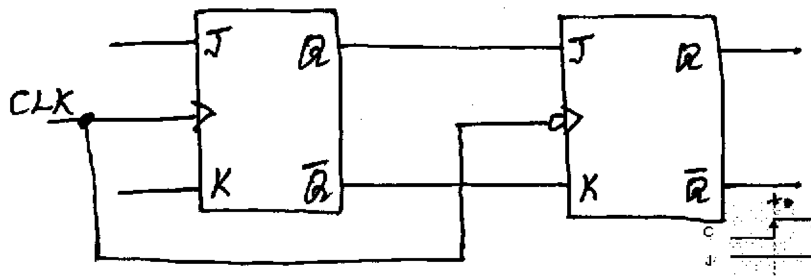
The *setup time (t_{setup})* is the minimum amount of time that the data bit must be present at the input, before the clock edge arrives. For example, if a D flip-flop has a setup time of 15 ns, the data bit to be stored must be at the D input at least 15 ns before the clock edge arrives; otherwise, correct sampling and storing is not guaranteed.

The *hold time (t_{hold})* is the minimum amount of time that data bit D must be present after the positive transition of the clock. For example, if t_{setup} = 15 ns and t_{hold} = 5 ns, the data bit has to be at the D input at least 15 ns before the clock edge arrives and held at least 5 ns after the clock PT.

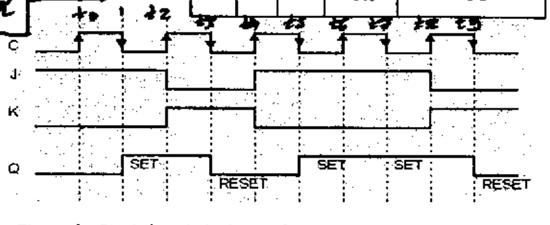


(1) Setup Time, (2) Propagation Delay, & (3) Hold Time

JK MASTER-SLAVE FLIP-FLOPS:



C	J	K	Q_{n+1}	Action
⊓	0	0	Q_n	No Change
⊓	0	1	0	RESET
⊓	1	0	1	SET
⊓	1	1	$\overline{Q_n}$	Toggle



JK Master-Slave Flip-Flop, Symbol, Truth Table & Waveforms

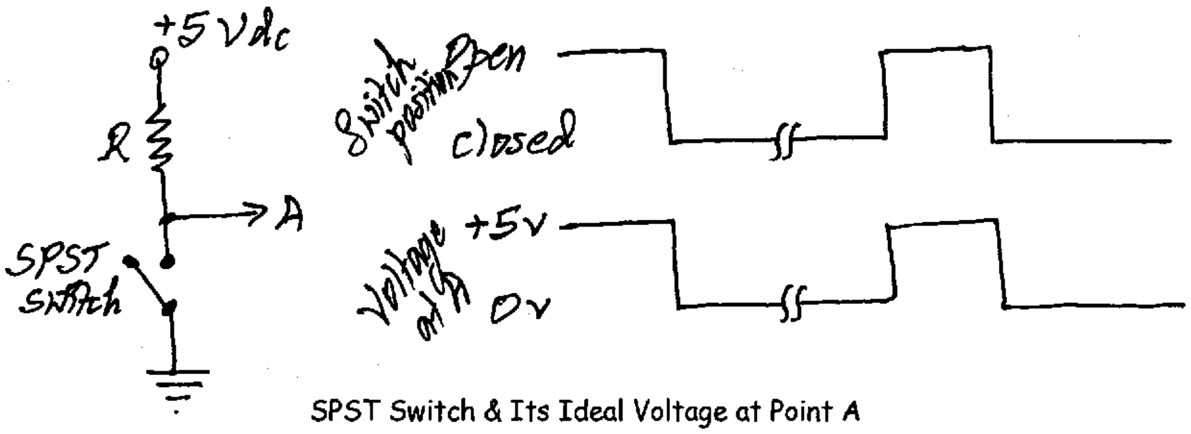
The master is positive-edge-triggered and the slave is negative-edge-triggered. The master responds to its J and K inputs before the slave.

- If J = 1 and K = 0, the master sets on the positive clock transition. The high Q output of the master drives the J input of the slave, hence, on negative clock transition, the slave also sets.
- If J = 0 and K = 1, the master resets on the PT of the clock. The low Q of the master goes to J input of the slave. Therefore, the NT of the clock forces the slave to reset.
- If J = 1 and K = 1, the master toggles on the PT of the clock and the slave then toggles on the NT of the clock.

To summarize, if the master sets, the slave sets; if the master resets, the slave resets – regardless of what the master does, the slave copies it. This particular flip-flop might be referred to as *pulse-triggered (⊓)*.

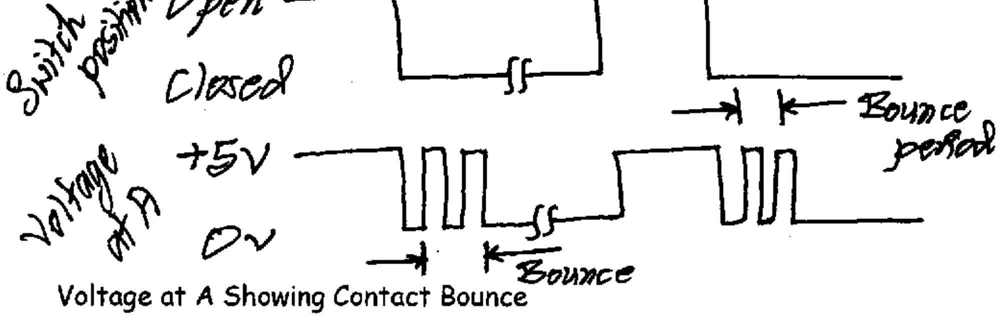
SWITCH CONTACT BOUNCE CIRCUITS:

In digital systems, sometimes, mechanical contacts are used for conveying an electrical signal. For example, the switches used in a computer system. A single-pole-single-throw (SPST) switch (shown in the following Fig) can be used to show a high logic level (usually +5 V) or a low logic level (0 V). When the switch is open, the voltage at point A will be + 5 V; and when the switch is closed, the voltage at A will be 0 V. Ideally, the voltage waveform at A should appear as shown below.



SPST Switch & Its Ideal Voltage at Point A

Actually, the waveform at point A will appear, as shown in the following Fig, as the result of a phenomenon, known as *contact bounce*.



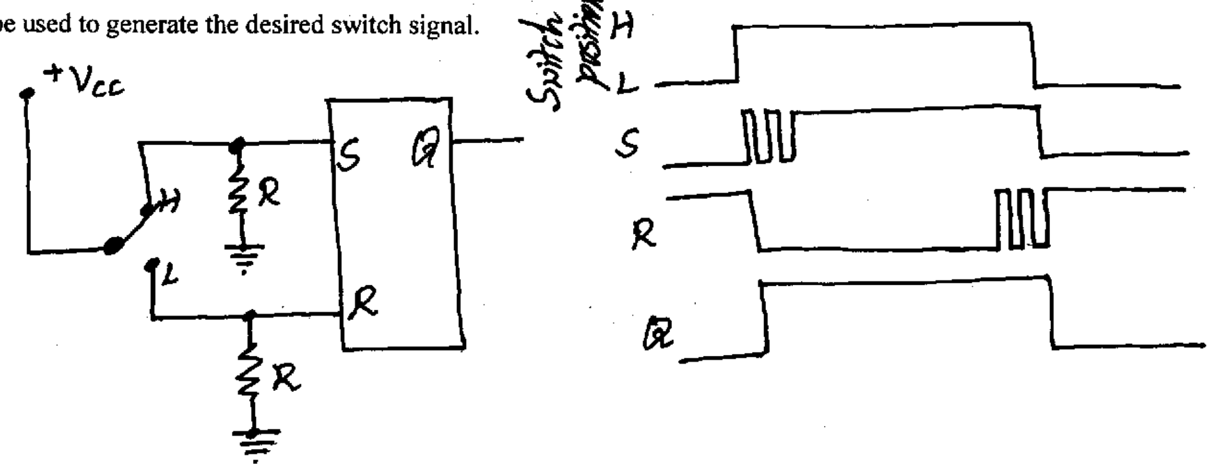
Voltage at A Showing Contact Bounce

Any mechanical switching device consists of a moving contact arm restrained by some sort of spring system. As a result, when the arm is moved from one stable position to the other, the arm bounces (as a hard ball bounces when dropped on a hard surface). The number of bounces that occur and the period of the bounce differ for each switching device.

If the voltage at point A is applied to the input of a TTL circuit, the circuit will respond (when the switch is closed) as if multiple signals were applied, rather than a single switch closure. Hence, contact bounce problem must be addressed / eliminated in an electronic circuit.

SR Latch Debounce Circuit:

The SR latch (shown in the following Fig) will remove any contact bounce due to the switch. The output Q can be used to generate the desired switch signal.



Debounce Circuit

When the switch is moved to position H, S = 1 and R = 0. Bouncing occurs at the S input (as shown in the above Fig) due to the switch closure. The flip-flop "sees" this as a series of high and low inputs, settling with a high level. At the first high level on S, the flip-flop will immediately be set with Q = 1. When the switch bounces, the input signals are S = R = 0; therefore, the flip-flop remains set (Q = 1). When the switch regains S = 1 and R = 0; causes an attempt to again set the flip-flop, But, since the flip-flop is already set, no changes occur at Q. The result is that, the flip-flop responds to the first, and to the first, high level at its S input, resulting in a "clean" low-to-high signal at its output (Q).

When the switch is moved to position L, S = 0 and R = 1.Bouncing occurs at R input (as shown in the above Fig) due to the switch closure. Again, the flip-flop "sees" this as a series of high and low inputs, and it simply responds to the first high level, and ignores all following transitions. The result is a "clean" high-to-low signal at its output (Q).

VARIOUS REPRESENTATIONS OF FLIP-FLOPS:

There are various ways a flip-flop can be represented, each one suitable for certain application.

Characteristic Equations of Flip-Flop:

The *characteristics equations* of flip-flops are useful in analyzing circuits made of them. Here, next output, Q_{n+1}, is expressed as a function of present output Q_n and the input to the flip-flops. Karnaugh map can be used to get the optimized expression.

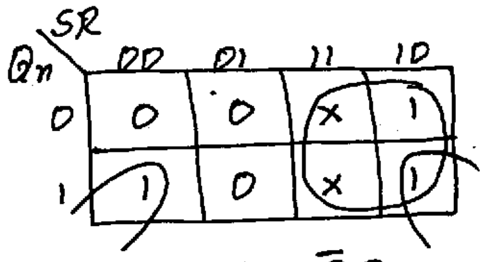
S	R	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	?

D	Q_{n+1}
0	0
1	1

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q_n}$

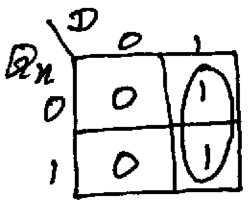
T	Q_{n+1}
0	1
1	0

SR Flip-Flop:



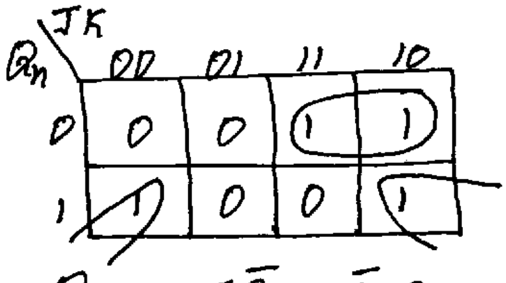
$$Q_{n+1} = S + \bar{R}Q_n.$$

D Flip-Flop:



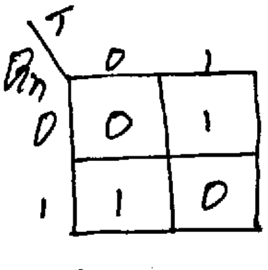
$$Q_{n+1} = D.$$

JK Flip-Flop:



$$Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n.$$

T Flip-Flop:



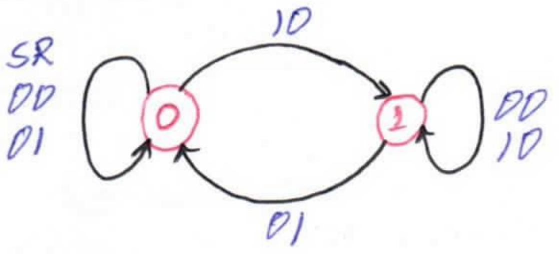
$$Q_{n+1} = T\bar{Q}_n + \bar{T}Q_n.$$

Characteristic Equations of SR, D, JK & T Flip-Flops

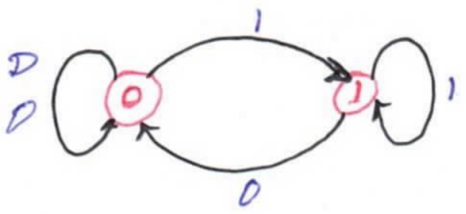
Flip-Flops as Finite State Machine:

In a sequential logic circuit, the value of all memory elements at a given time defines the *state* of that circuit at that time. *Finite State Machine (FSM)* concept offers a better alternative to truth table in understanding progress of sequential logic with time.

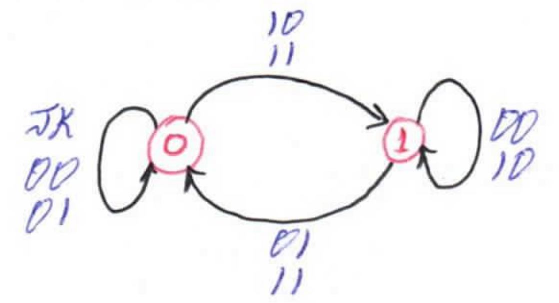
SR Flip-Flop:



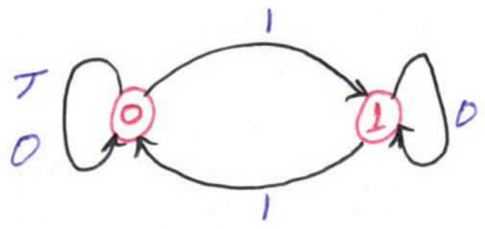
D Flip-Flop:



JK Flip-Flop:



T Flip-Flop:



State Transition Diagrams of SR, D, JK & T Flip-Flops

Flip-Flop Excitation Table:

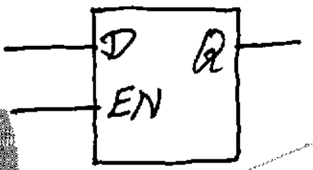
In synthesis or design problem, excitation tables are very useful. *Excitation table* of a flip-flop is looking at its truth table in a reverse way; here, flip-flop output is presented as a dependent function of transition $Q_n \rightarrow Q_{n+1}$ and comes later in the table.

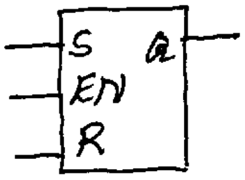
$Q_n \rightarrow$	Q_{n+1}	S	R	J	K	D	T
0	0	0	x	0	x	0	0
0	1	1	0	1	x	1	1
1	0	0	1	x	1	0	1
1	1	x	0	x	0	1	0

Excitation Table of Flip-Flops

HDL IMPLEMENTATION OF FLIP-FLOP:

Behavioral model is preferred for sequential circuits, and ***always*** keyword is used. In an SR or D latch, if EN = 1, the output changes according to characteristic equation; and if EN = 0, output does not change (remains latched to the previous value).





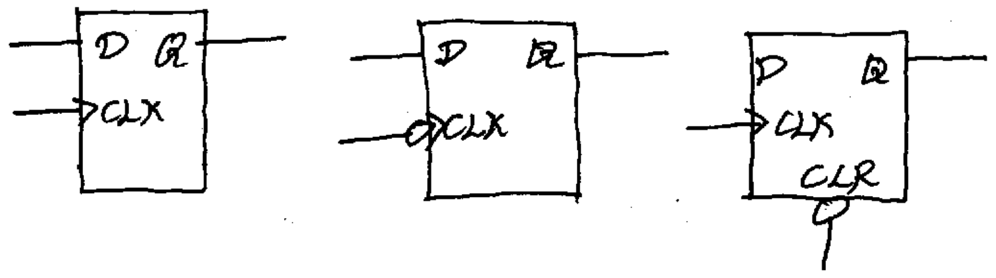
```
module DLatch (D,EN,Q);
input D,EN;
output Q;   reg Q;
  always @ (EN or D)
    if (EN) Q=D;
endmodule
```

```
module SRLatch (S,R,EN,Q);
input S,R,EN;
output Q;  reg Q;
  always @ (EN or S or R)
    if (EN)  Q = S|(~RQ);
endmodule
```

For clocked flip-flops, we use the keyword ***posedge*** (for positive-edge-triggered) and ***negedge*** (for Negative-edge-triggered).

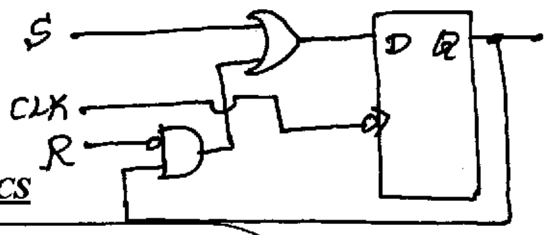
```
module DFF (D,CLK,Q);
input D,CLK;
output Q;   reg Q;
  always @ (posedge CLK)
     Q = D;
endmodule
```

```
module DFF_CLR (D, CLK, CLR, Q);
input D, CLR, CLK;
output Q;   reg Q;
always @ (posedge CLK or negedge CLR)
  if (~CLR) Q = 1'b0;
  else Q = D
endmodule
```



Problem: *Write Verilog code that converts an D flip-flop to an SR flip-flop.*

Solution:



```
module D_SRFF (S, R, CLK, Q);
input S, R, CLK;    output Q;  wire DSR;
assign DSR = S | (~R & Q);
DFFneg D1 (DSR, CLK, Q);
endmodule
      module DFFneg (D, CLK, Q);
      input D, CLK;
      output Q;   reg Q;
        always @ (negedge CLK)
            Q = D;
      endmodule
```

REGISTERS

*A **register** is a group of flip-flops that can be used to store a binary number.* There must be one flip-flop for each bit in the binary number. For example, a register used to store an 8-bit binary number must have eight flip-flops.

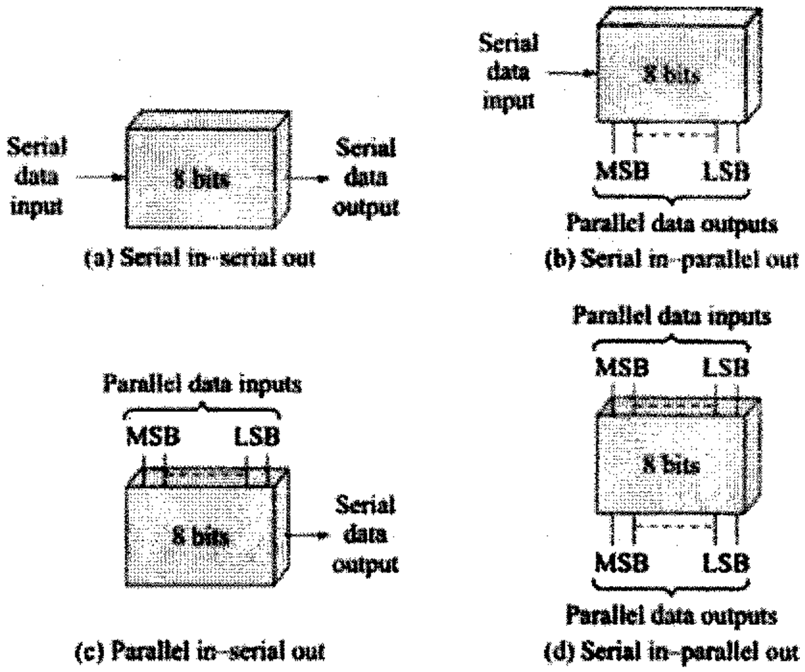
*A group of flip-flops connected such that, it will shift the data right or left is called a **shift register**.* Bits in a binary number (data) can be moved from one place to another.

Shifting the data 1 bit at a time in a serial fashion, beginning with either MSB or LSB, is referred to as ***serial shifting***. Shifting all the data bits simultaneously is referred to as ***parallel shifting***.

TYPES OF REGISTERS:

There are two ways to shift data into a register – serial or parallel; and there are two ways to shift the data out of the register – serial or parallel. This leads to the construction of four basic types of registers, as shown in the following Fig. All of these configurations are commercially available as TTL MSI/LSI circuits.

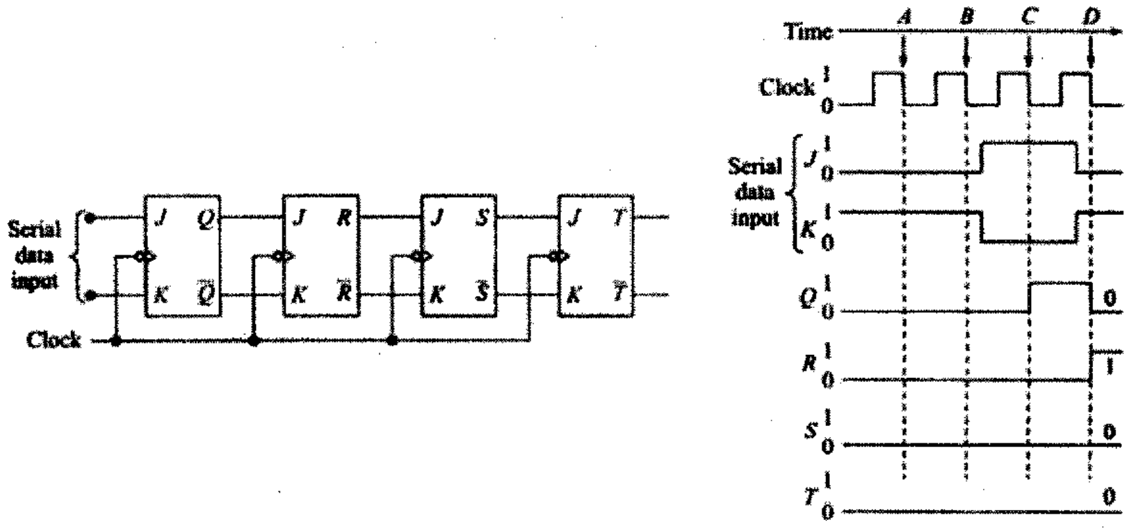
Examples:	Serial in – serial out (SISO): 54/74LS91, 8-bits
	Serial in – parallel out (SIPO): 54/74164, 8-bits
	Parallel in – serial out (PISO): 54/74165, 8-bits
	Parallel in – parallel out (PIPO): 54/74194, 4-bits & 54/74168, 8-bits.



Types of Shift Registers

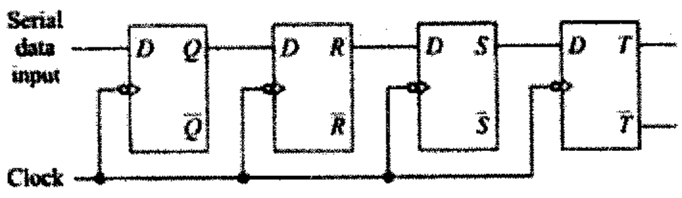
SERIAL IN – SERIAL OUT:

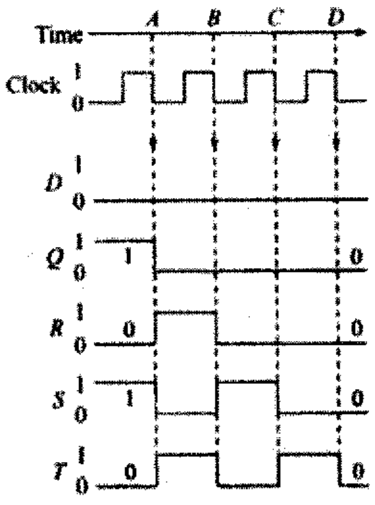
Data is entered or exited from the shift register serially. The flip-flops used to construct registers are usually edge-triggered JK, SR, or D type. Consider four D type flip-flops connected as shown in the following Fig. A common clock provides trigger at its negative edge to all the flip-flops. As output of one D flip-flop is connected to the input of the next at every clock trigger, data stored in one flip-flop is transferred to the next. For this circuit, the transfer takes place like Q → R, R → S, S → T, and serial data input is transferred to Q.



4-bit SISO (JK-type) Shift Register

NOTE: A shift register made up of JK or SR flip-flop has non-inverting out Q of one flip-flop connected to J or S input of the next flip-flop and inverting output Q connected to K or R input respectively. For the first flip-flop, between J and K (or S and R), an inverter is connected and J (or S) input is treated as serial data in. Note that, in this configuration, both JK and SR flip-flops effectively act like a D flip-flop.





Before time A: QRST = 1010

At time A: 1010 is shifted one FF to right
QRST = 0101

At time B: QRST = 0010

At time C: QRST = 0001

At time D: QRST = 0000.

4-bit SISO (D-type) Shift Register

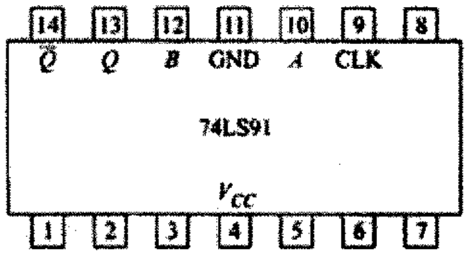
Problem: *Show how a number 0100 is entered serially in a shift register, using state table.*

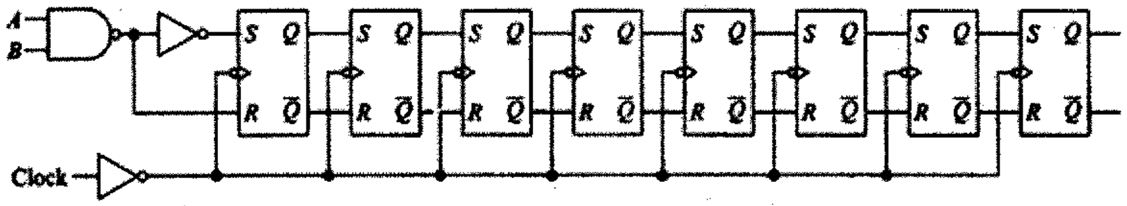
Solution:

Clock	Serial Input	Q	R	S	T
0	0	0	0	0	0
1	1	0	0	0	0
2	0	1	0	0	0
3	0	0	1	0	0
4		0	0	1	0

The 74LS91: is an 8-bit TTL MSI chip. There are eight SR flip-flops connected to provide a serial input as well as a serial output. The clock input at each flip-flop is negative-edge-trigger sensitive. Since, the applied clock is passed through an inverter; data will be shifted on the positive edges of the input clock pulses.

The inverter connected between S and R on the first flip-flop means that this circuit functions as a D-type flip-flop. Notice that, a data level at A (or B) is complemented by the NAND gate and then applied to the R input of the first flip-flop. The same data level is complemented by the NAND gate and then complemented again by the inverter before it appears at the S input. So, a 1 at input A will set the first-flip.

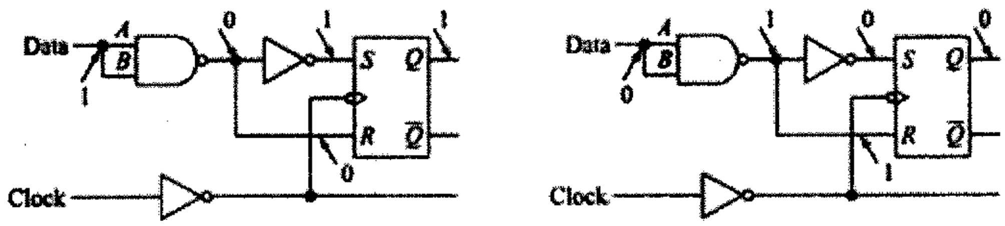




74LS91 8-bit SISO Shift Register

Problem: *Examine the logic levels at the input of a 74LS91 and show how a 1 and then 0 are shifted into the register.*

Solution:



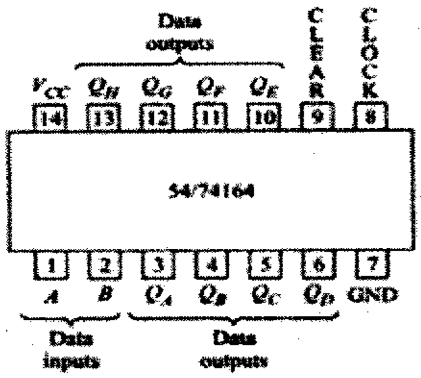
(a) Logic levels shown by arrows will set the flip-flop

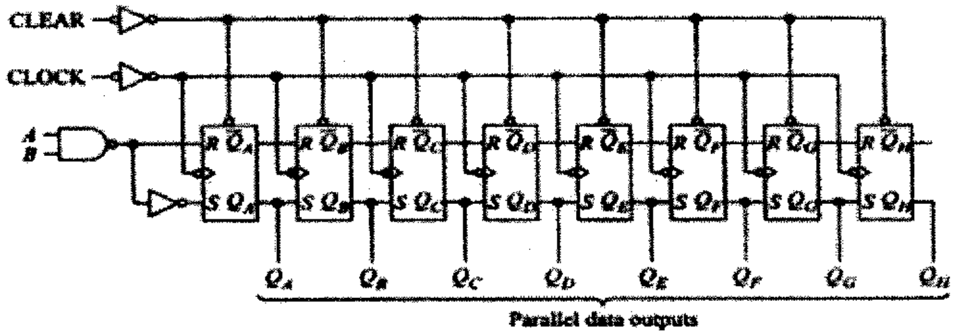
(b) Logic levels shown by arrows will reset the flip-flop

The input logic and the first flip-flop of 74LS91 are shown above. A data bit 1 is applied at the data input. The input R is 0, the S input is 1, and the flip-flop will clearly be set when the clock goes high. In other words, the 1 at the data input will shift into the flip-flop. In the second Fig above, a 0 is applied at the data input A. The R input is 1, the S input is 0, and the flip-flop will be reset when the clock goes high. Thus, the input 0 is shifted into the flip-flop.

SERIAL IN – PARALLEL OUT:

The second type of register is one in which, data is shifted in serially, but shifted out in parallel. For example, an 8-bit shift register would have eight output lines – one for each flip-flop in the register. The basic configuration is shown below.





74/74164 8-bit SIPO Shift Register

The 54/74164 is an 8-bit SIPO shift register. The pin-out and the logic diagram for this device are given in the above Fig. The logic diagram will reveal that this register is exactly like 74LS91, with two exceptions:

(1) the true side of each flip-flop is available as an output,

(2) each flip-flop has an asynchronous clear input.

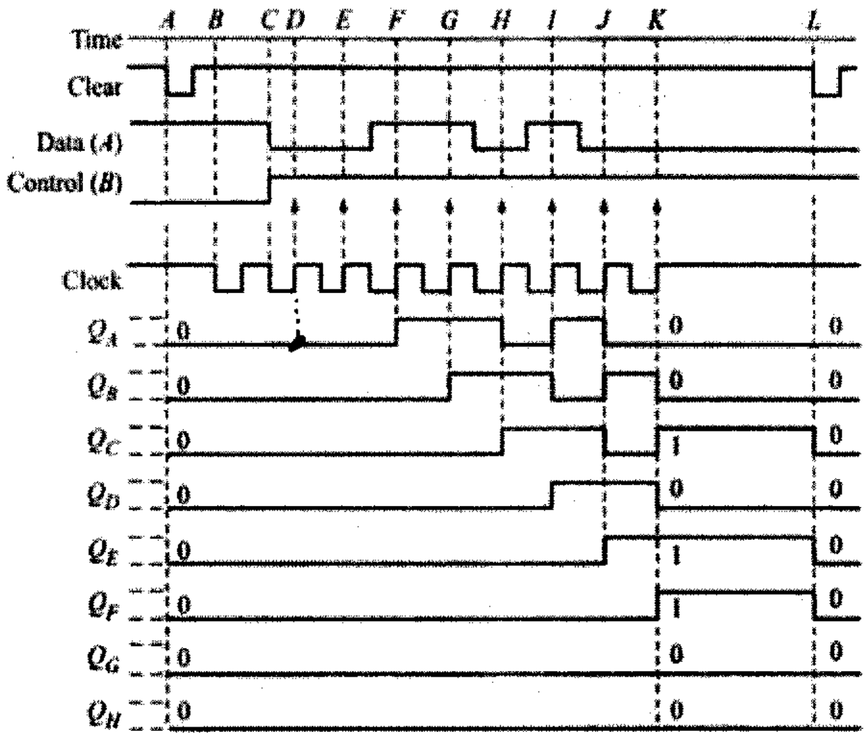
A low level at the clear input to the chip will reset every flip-flop. This is asynchronous signal and is level sensitive. As long as the clear input to the chip is held low, the flip-flop outputs will all remain low.

Shifting data serially into the register is exactly same as previously discussed 74LS91. The serial data is input at A (pin 1), while a gating control signal is applied at B (pin 2). The first clear pulse occurs at time A and simply resets all flip-flops to 0. The waveforms are shown below:

A	B	NAND
0	0	1
0	1	1
1	0	1
1	1	0

$B = 0$; Output $= 1$

$B = 1$; Output $= \bar{A}$



The clock begins at time B, but the first PT does nothing since the control line is low. At time C, the control line goes high, and the first data bit (a 0) is shifted into the register at time D.

The next 7 data bits are shifted in, in order, at times, E, F, G, H, I, J and K. The clock remain high after time K, and the 8-bit number 0010 1100 now resides in the register and is available on eight output lines. The LSB was shifted in first and appears at Q_H. Notice that, the clock must be stopped after its positive transition at time K, otherwise, shifting will continue and the data bits will be lost.

Finally, another clear pulse occurs at time L, the flip-flops are all reset to zero, and another shift sequence may begin.

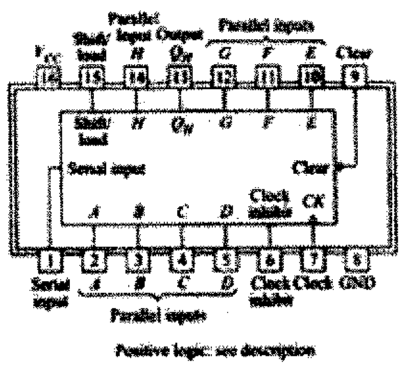
SIPO

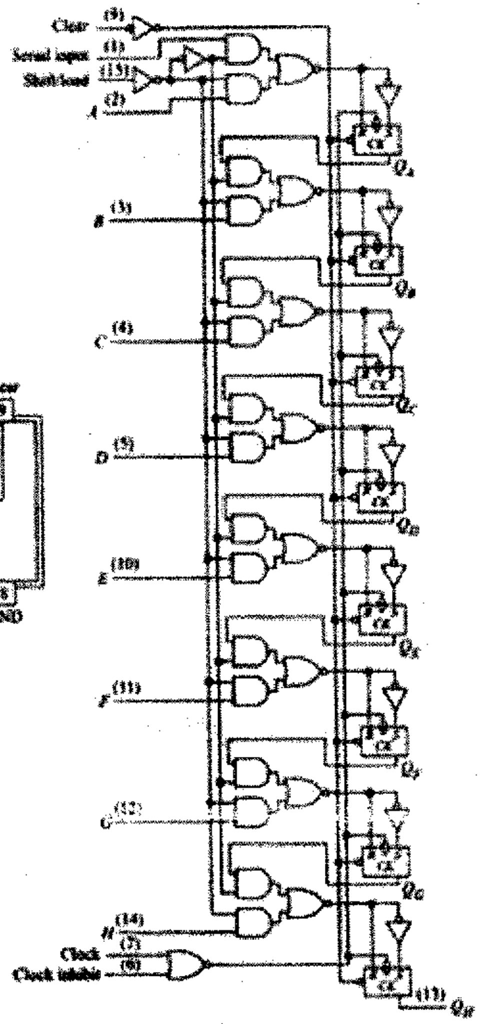
Problem: *How long will it take to shift an 8-bit number into a 54164 shift register if the clock is et to 10 MHz?*

Solution: A minimum of eight clock periods will be required, since the data is entered serially. One clock period is 1/10 MHz = 100 Ns. So, it will require 800 ns minimum.

PARALLE IN – SERIAL OUT:

The pin-out and the logic diagram for a 54/74166 are given in the following Fig. Notice that there are eight SR flip-flops, each with some attached logic circuitry.





54/74166 8-bit PISO Shift Register

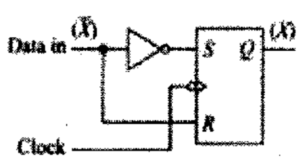
First consider the clocked SR flip-flop and the attached inverter given in Fig (a). This combination forms a type D flip-flop. If a data bit X is to be clocked into the flip-flop, the complement of X must be present at the input. For example, If X = 0, then S = 1 and R = 0, and a 1 will be clocked into the flip-flop.

Now, add a NOR gate as shown in Fig (b). If one leg of this NOR gate is at ground level, a data bit X at the other leg is simply inverted by the NOR gate. For Example, if X = 1, then at the output of NOR gate $\bar{X} = 0$, allowing a 1 to be clocked into the flip-flop. This NOR gate offers the option of entering data from two different sources, either X_1 or X_2. Holding X_2 at ground will allow the data at X_1 to be shifted into the flip-flop; conversely, holding X_1 at ground will allow data at *X2* to be shifted in.

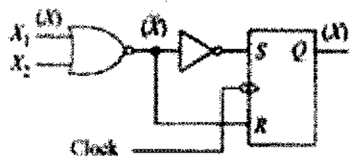
The addition of two AND gates and two inverters, as shown in Fig (c), will allow the selection of data X_1 or data X_2. If the control line high, the upper AND gate is enabled and the lower AND gate is disabled. Thus, X_1 will appear at the upper leg of the NOR gate, while the lower leg of the NOR gate will be at ground level. On the other hand, if the control line low, the upper AND gate is disabled, while the wer AND gate is enabled. This allows X_2 to appear at the lower leg of the NOR gate while the upper leg of the NOR gate is at ground level. Thus, to summarize –

Control line is high: Data bit at X_1 will be shifted into the flip-flop at the next clock transition.

Control line is low: Data bit at X_2 will be shifted into the flip-flop at the next clock transition.

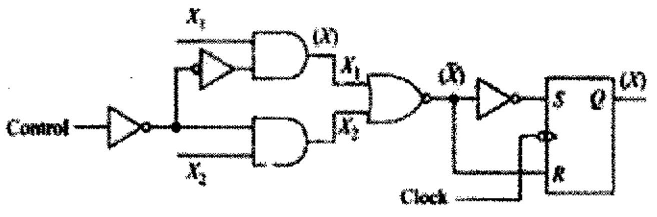


(a) Type D flip-flop



(b) NOR-gate added

X_1	X_2	NOR
→ 0	0✓	1
0	1	0
→ 1	0✓	0
1	1	0



(c) Control logic added

A careful examination will reveal that, exactly eight of the circuits given in Fig (c) are connected together to form the 54/74166 shift register. They are connected to allow two different operations: (1) the parallel entry of data and (2) the operation of shifting data serially through the register from the first flip-flop, Q_A, towards the last flip-flop, Q_H.

If the data input labeled X2 in above Fig (c) is brought out individually for each flip-flop, these eight inputs will serve as the parallel data entry inputs for an 8-bit number ABCD EFGH. The control line is labeled shift/load. Holding this shift/load control line low will enable the lower AND gate for each flip-flop and the 8-bit number will be loaded into the flip-flops with a single clock transition – *parallel input.*

Holding the shift/load control line high will enable the upper AND gate for each flip-flop. If the input from this upper AND gate received its data from the prior flip-flop in the register, each clock transition will shift a data bit from one flip-flop into the following flip-flop – processing in a direction from Q_A towards Q_H. In other words, data will be shifted through the register serially.

To summarize:

Shift/load is low: A single clock transition loads 8 bits of data (ABCD EFGH) into the register in parallel.

Shift/load is high: Clock transitions will shift data through the register serially, with entering data applied at the serial input.

Notice that, the clock input is applied through a two-input NOR gate. When the clock inhibit is held low, the clock signal passes the NOR gate inverted; data will shift into regist on the PTs of the clock. When the clock inhibit is high, the NOR gate output is held low, and the clock is prevented from reaching the flip-flops.

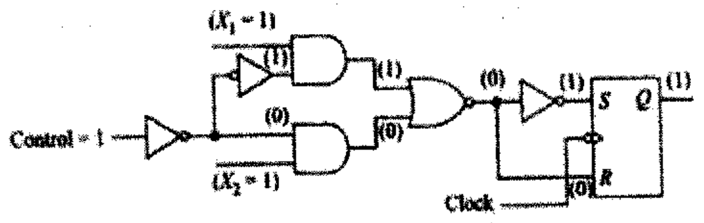
Inputs						Internal Levels		Outputs
Clear	Shift/ load	Clock inhibit	Clock	Serial	Parallel $A \ldots H$	Q_A and Q_B		Q_H
L	X	X	X	X	X	L	L	L
H	X	L	L	X	X	Q_{AO}	Q_{BO}	Q_{HO}
H	L	L	$\uparrow$	X	$a \ldots h$	a	b	h
H	H	L	$\uparrow$	H	X	H	Q_{An}	Q_{Gn}
H	H	L	$\uparrow$	L	X	L	Q_{An}	Q_{Gn}
H	X	H	$\uparrow$	X	X	Q_{AO}	Q_{BO}	Q_{HO}

X = Irrelevant, H = High level, L = Low level
$\uparrow$ = Positive transition
$a \ldots h$ = Steady state input level at $A \ldots H$ respectively
Q_{AO}, Q_{BO} = Level at Q_A, $Q_B \ldots$ before steady state
Q_{An}, Q_{Gn} = Level of Q_A or Q_B before most recent transition ( ) $\uparrow$

54/74166 Truth Table

Problem: *For the circuit shown in Fig (c), write the logic levels present on each gate leg if CONTROL = 1, X_1 = 1, and X_2 = 1.*

Solution: The correct levels are given in the following Fig. The data value 1 at X1 is shifted into the flip-flop when the clock transitions.



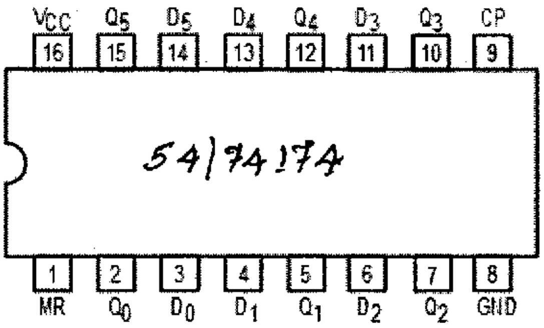
PARALLEL IN – PARALLE OUT:

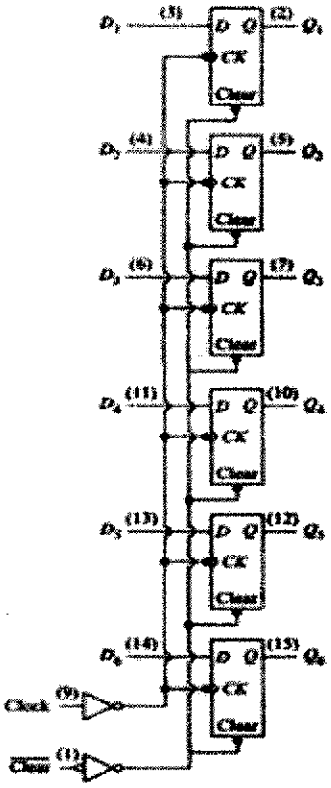
Here, the data can be shifted either into out of the register in parallel.

The 54/74174: The 74174, shown in the following Fig, is an example of PIPO register. It is simply a parallel arrangement of six D-type flip-flops. Each flip-flop is negative-edge-triggered, and thus a PT will shift data into the register.

The six data bits D_1 through D_6 are all shifted into the register in parallel. The stored data is immediately available, in parallel, at the outputs, Q_1 through Q_6. This type of register is simply used to store data, and is sometimes called a *data register*, or *data latch*.

A low level at clear input will immediately reset all the flip-flops low. The clear input is asynchronous – that is, it can be done at any time and it takes precedence over all other inputs.



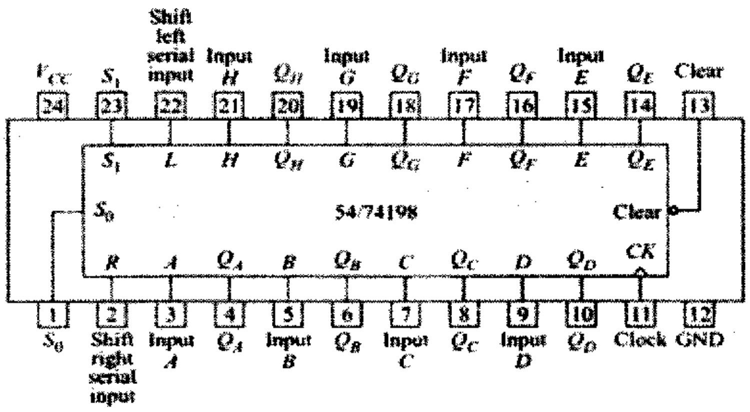


54/74174

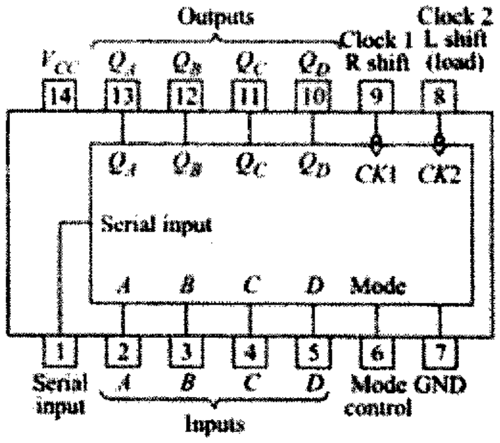
Problem: *The 74LS174 data sheet gives a setup time of 20 ns and hold time of 5 ns. What is the minimum required width of the data input levels ($D_1 \ldots D_6$) for the 74LS174?*

Solution: The data inputs must be steady at least 20 ns before the PT of the clock, and they must be held for minimum of 5 ns after the PT. Thus, the data input levels must be held steady for a minimum of 25 ns.

The 54/74198: is an 8-bit TTL MSI, having both parallel input and parallel output capability. The pin-ou is given below. It uses positive-edge-triggered flip-flops. It can also be used to shift data through the register in either directions – as shift right and shift left.

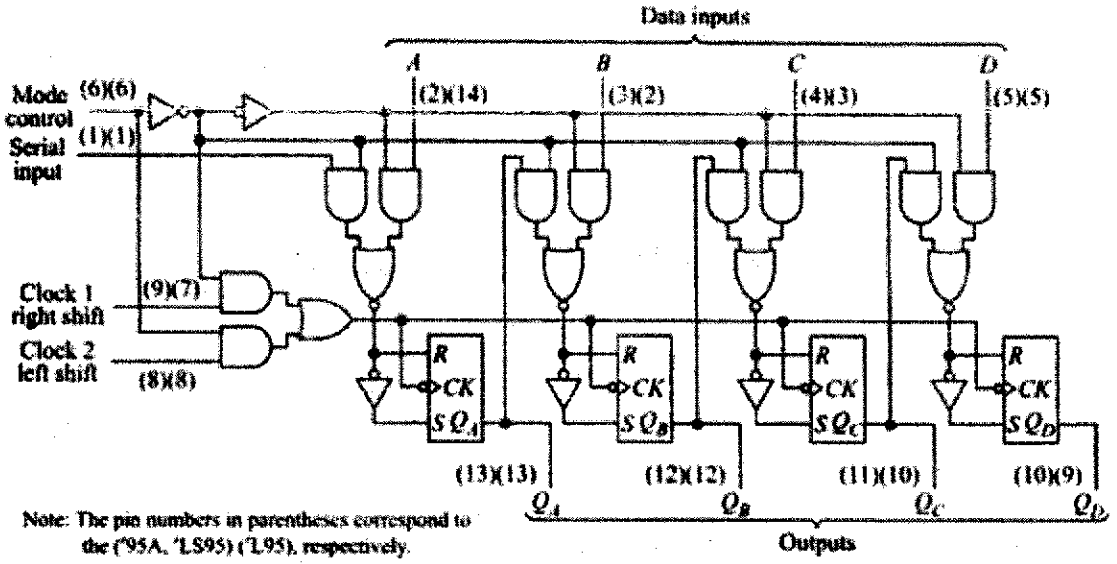


The 54/7495A: is a 4-bit parallel shift register. It also has serial data input and can be used to shift data to the right (from Q_A toward Q_B) and to the left. The DIP pin-out and the logic diagram are given in the following Fig.



MC	C1	C2	Operation
0	↓	X	SISO: Right Shift
			SIPO
1	X	↓	PISO
0	↓	X	
1	X	↓	PIPO

The parallel data outputs are the Q sides of each of the four flip-flops in the register. Note that, the output Q_D could be used as a serial output when data is shifted from left to right through the register (right shift).



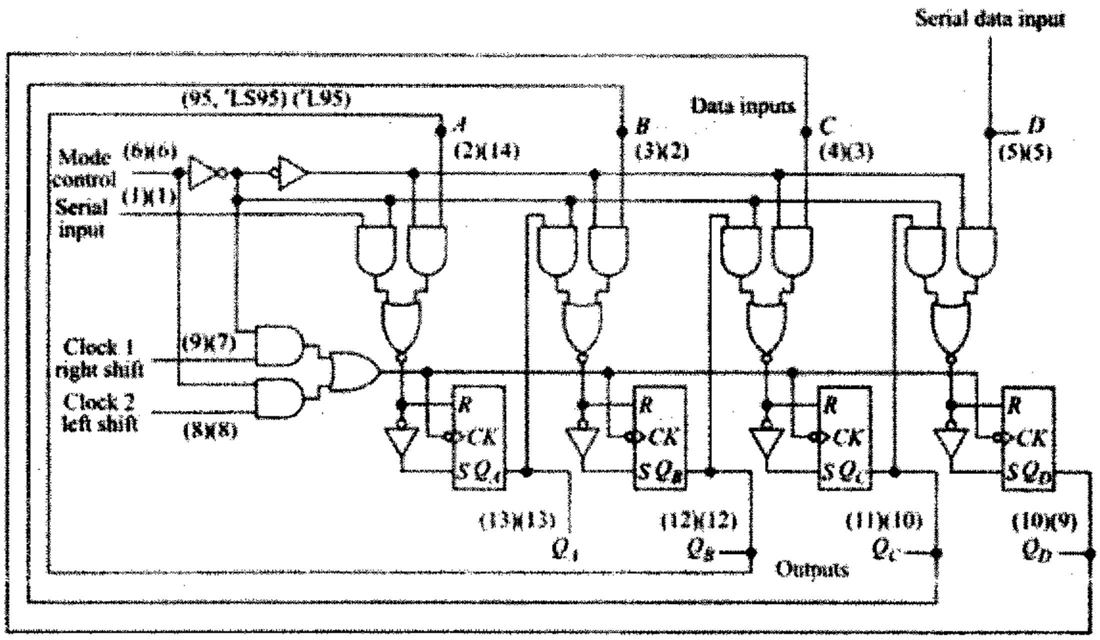
54/7495A

When the mode control line is held high, the AND gate on the right input to each NOR gate is enabled, while the left AND gate is disabled. The data at inputs A, B, C and D will then be loaded into the register on a negative transition of the clock – this is parallel data input.

When the mode control line is low, the AND gate on the right input to each NOR gate is disabled, while the left AND gate is enabled. The data input to the flip-flop Q_A is now at serial input; the data input to Q_B is Q_A and so on. On each clock NT, a data bit is entered serially into the register at the first flip-flop

…, and each stored data bit is shifted one flip-flop to the right (toward the last flip-flop Q_D) – this is the serial input of data and also the right shift operation.

In order to affect the left shift operation, the input data must be connected to the D data input as shown in the following Fig. It is also necessary to connect Q_D to C, Q_C to B, and Q_B to A, as shown below.



54/7495A Wired for Shift-Left

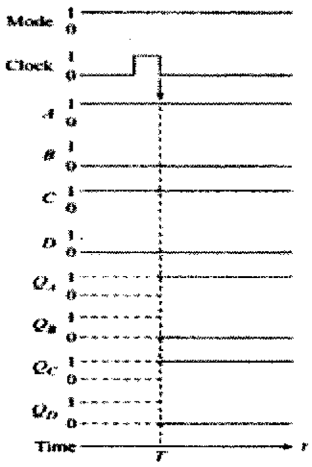
Now, when the mode control line is held high, data bit will be entered into flip-flop Q_D, and each stored data bit will be shifted to the left on each clock NT. This is also serial input of data (but at input D), and is the left-shift operation.

There are two clock inputs – clock 1 and clock 2. This is to accommodate requirements where the clock used to shift data to the right is separate from the clock used to shift data to the left.

Problem: *Draw the waveforms, if the 4-bit binary number 1010 were shifted into a 54/7495A in parallel.*

Solution: The mode control must be high.

A single clock NT will enter the data.



UNIVERSAL SHIFT REGISTER:

Self Study

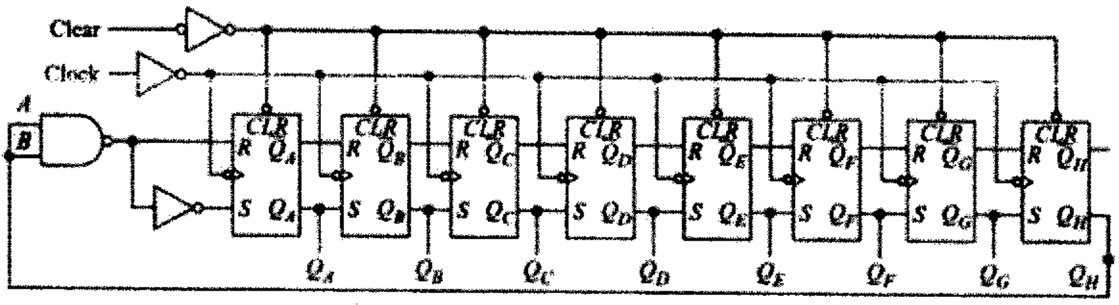
APPLICATIONS OF SHIFT REGISTERS:

Shift registers are used in almost every sphere of a digital logic system. To name a few –

1. Shift registers can be used to count number of pulses entering into a system as ring counter or switched-tail counter
2. As ring counter, it can generate various control signals in a sequential manner
3. Shift register can generate prescribed sequence repetitively or detect a particular sequence from data input
4. Shift registers can help in reduction of hardware by converting parallel data feed to serial one. Serial adder is one such application.

Ring Counter:

Consider a serial shift register, such as 54/74164. The output of the last flip-flop is connected back to the D input (A and B data inputs) of the first flip-flop, as shown in the following Fig.

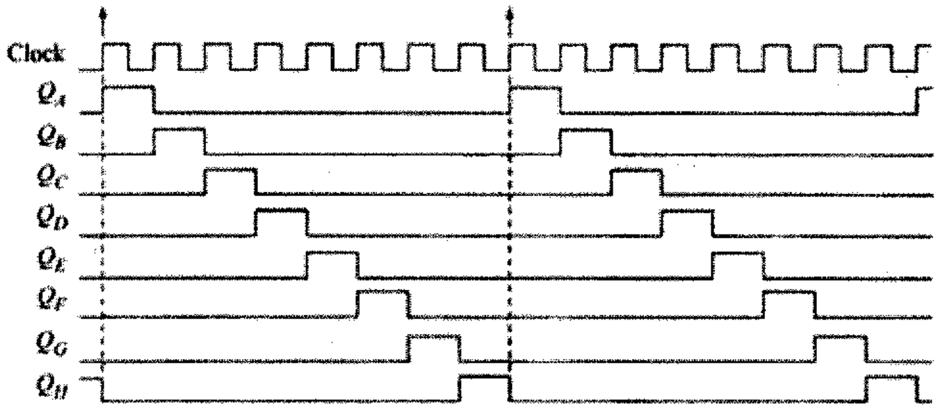


54/74164 8-Bit Shift Register with Feedback from Q_H to A-B

Suppose that, QA is high and all other flip-flops are low, and then allow the clock to run. On the first clock PT, the 1 in A will shift in B, and A will be reset, since the 0 in H will shift into A. All other flip-flops will still contain 0s. The second clock pulse will shift the 1 from B to C, while B resets. The third clock PT will shift the 1 from C to D, and so on.

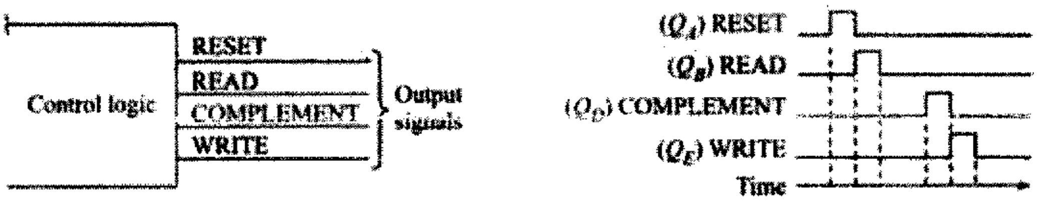
Thus, this single 1 will shift down the register, moving travelling from one flip-flop to the next flip-flop each time the clock goes high. When it reaches flip-flop H, the next clock PT will shift it into flip-flop A by means of the feedback connection.

Thus the 1 is simply circulated around the register in a clockwise direction, moving ahead one flip-flop with each clock PT. This configuration is referred to as *circulating register* or a *ring counter*. The waveform of this ring counter is given below.



Waveforms of Ring Counter

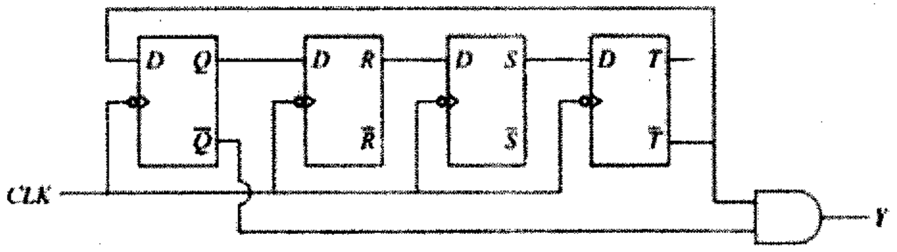
Applications: Waveforms of this type are frequently used in control section of a digital system. They are ideal for controlling events that must occur in a strict time sequence – that is, event A, then event B, then C, and so on. For example, the logic diagram of the following Fig, shows how to generate RESET, READ, COMPLEMENT, and WRITE (set of control signals), that occur one after the other sequentially.



Disadvantages: In order to produce the waveform shown, the counter should have one, and only one, 1 in it. The chances of this, occurring naturally, when power is first applied are very remote.

Switched-Tail Counter:

In ring counter, the non-inverting output of the last flip-flop is fed back to the first flip-flop of the shift register. If we instead, feed inverting output back (or switch the tail) as shown in the following Fig for a 4-bit register, we get a *switched-tail counter* also known as *twisted-tail counter* or *Johnson counter*.



Assume, all the flip-flops are cleared in the beginning. Then, all the flip-flop inputs have 0, except the first one, which is complement of the last flip-flop, i.e. 1. When clock trigger occurs, flip-flop stores QRST as 1000. This makes 1100 at the input of QRST, when the next clock trigger comes and that gets sferred to output at clock NT. Proceeding this way, we complete state table, given below. Note that, output Y = $\bar{Q}\bar{T}$ and state of the circuit repeats every eighth clock cycle. Thus, 4-bit shift register circuit can count 8 clock pulses or called modulo-8 counter.

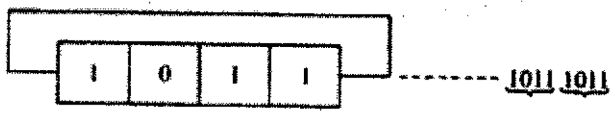
Clock	Serial in = $\bar{T}$	Q	R	S	T	Y = $\bar{Q}\bar{T}$
0	1	0	0	0	0	1
1	1	1	0	0	0	0
2	1	1	1	0	0	0
3	1	1	1	1	0	0
4	0	1	1	1	1	0
5	0	0	1	1	1	0
6	0	0	0	1	1	0
7	0	0	0	0	1	0
8	1	0	0	0	0	1
9	1	1	0	0	0	0
						repeats

Following above logic and preparing sate table for any *N*-bit shift register, we see switched-tail configuration can count up to *2N* number of clock pulse and gives modulo-*2N* counter. The output Y, derived similarly by AND operation of first and last flip-flop inverting outputs gives a logic high at every *2N*-th clock cycle; i.e. Y = 1 only once during *2N* clock cycles. Note that, for ring counter we don't need any decoding gate.

Disadvantages: The counter is to be initialized with one of the valid state of the counting sequence on which the design is based. Otherwise, the counter will follow a completely different count sequence.

Sequence Generator:

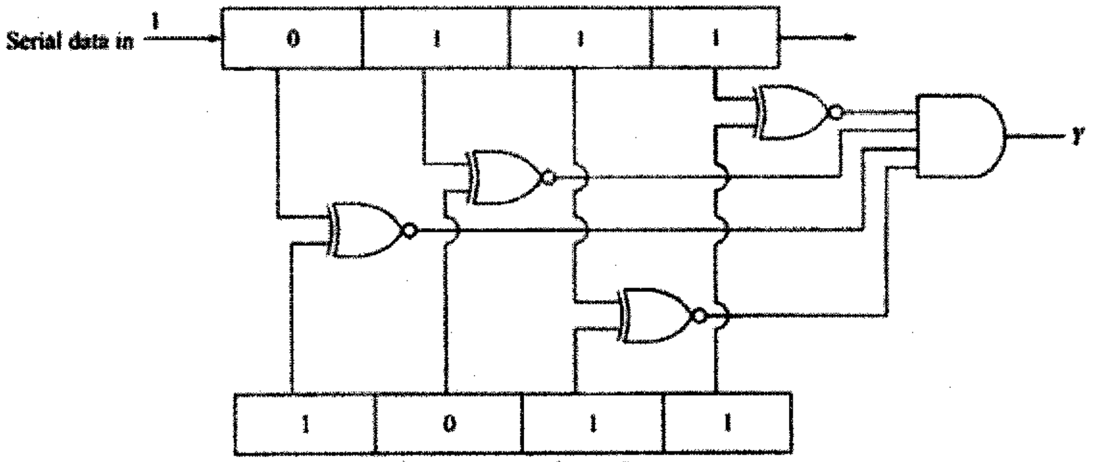
Sequence generator is useful in generating a sequence pattern repetitively. The following Fig gives the basic block diagram of a sequence generator, where shift register is presented as pipe full of data and each flip-flop represents one compartment of it.



The leftmost flip-flop is connected to serial data in and rightmost provides serial data out. The data transfer takes place only when a clock trigger arrives. Note that the shift register is connected like a ring counter and with triggering of clock the binary word stored in the last flip-flop is fed back as serial in to the register all over again. Sequence generated for the binary word 1011 is shown in the above Fig. For any n-bit long sequence to be generated, we need to store the sequence in an n-bit shift register.

Sequence Detector:

The circuit that can detect a 4-bit binary sequence is shown in the following Fig. It has one register to store the binary word that we want to detect from the data stream. Input data stream enters a shift register as serial data in and leaves as serial out. At every clocking instant, bit-wise comparisons of these two registers are done through Ex-NOR gate, as shown below.



Sequence to be detected

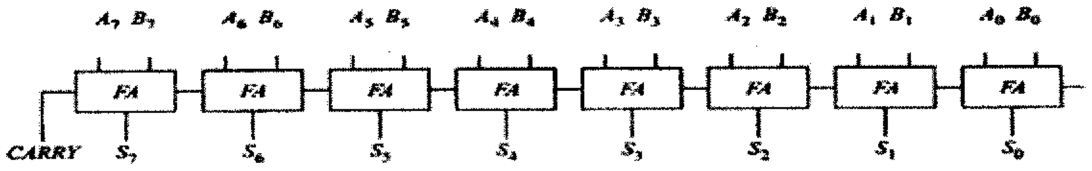
Two input Ex-NOR gate gives logic high when both inputs are low or both of them are high; i.e. when both are equal. The final output is taken from a four input AND gate, which becomes 1 only when all its inputs are 1; i.e. all the bits are matched.

The above Fig shows a situation when data received so far is 0111 and the word to be matched is 1011. The first two bits are mismatched and the output is 0. Now, as the next bit in the serial data stream is 1, when a clock trigger comes, the first flip-flop of the shift register stores 1 and 011 gets shifted to 2nd to 3rd flip-flops. With this, both registers store 1011 and Y = 1, which completes sequence detection.

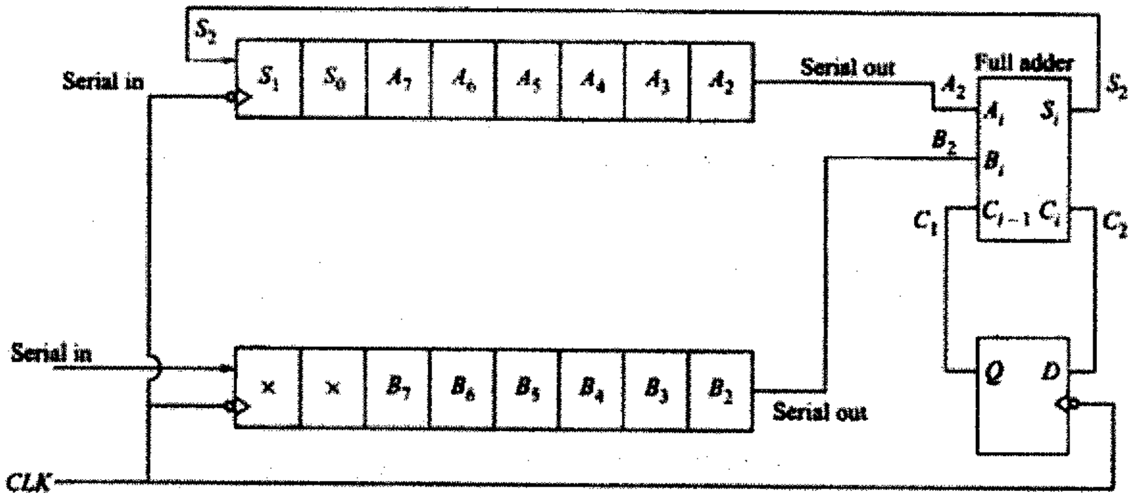
The circuit shown in the above Fig can be used as a *programmable sequence detector*, i.e. if we want to change the binary word to be detected; we simply load that in the bottom register.

Serial Adder:

Using full adder (FA) circuit, for 8-bit addition, we need 8 FA units (as shown below).



e, the addition is done in parallel. Using shift register, we can convert this parallel addition to serial one and reduce number of FA units. The following Fig shows how serial addition takes place in a time-multiplexed manner and also provides a snapshot of the register values at 3rd clock cycle.



Serial Addition of Two 8-Bit Numbers

Two 8-bit numbers, to be added ($A_7A_6 \ldots A_1A_0$ and $B_7B_6 \ldots B_1B_0$) are loaded in two 8-bit shift registers A and B. The LSB of each number appears in the rightmost position in two registers. Serial data out of A and B are fed to data inputs of full adder. The carry-in is fed from its own carry output delayed by one clock period by a D flip-flop, which is initially cleared. Both registers and D flip-flop are triggered by same clock. The sum (S) output of FA is fed to serial data in of shift register A.

The LSB of two numbers (A_0 and B_0) appearing at serial out of respective registers are added by FA during 1st clock cycle and generate sum (S_0) and carry (C_0). S_0 is available at serial data input of shift register A and C_0 at the input of D flip-flop. At NT of clock, shift registers shift its content to right by one unit. Now, S_0 becomes MSB of A and C_0 appears at D flip-flop output. Therefore, in the second clock cycle, FA is fed by second bit (A_1 and B_1) of two numbers and previous carry (C_0).

In 2nd clock cycle, S_1 and C_1 are generated and made available at serial data in of A register and input of D flip-flop respectively. At NT of clock, S_1 becomes MSB of A and S_0 occupies next position. Now, A_2 and B_2 appear at FA data input and the carry input is C_1.

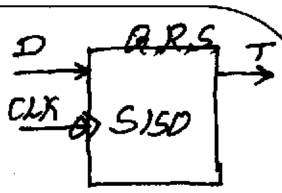
In 3rd clock cycle, S_2 and C_2 are generated and they get transferred similarly to the register and flip-flop. This process goes on and is stopped by inhibiting the clock after 8 clock cycles. At that time, shift register A stores the sum bits, S_7 in leftmost (MSB) position and S_0 in rightmost (LSB) position. The final carry is available at D flip-flop output.

Limitations: The final addition result is delayed by eight clock cycles. In parallel adder, the result is obtained almost instantaneously.

REGISTER IMPLEMENTATION IN HDL:

Now, we will write the Verilog code for SISO register, where T is the final output and Q, R, S are internal outputs. Since, they are outputs of always block, they have to be defined as ***reg*** and not as ***wire***.

```
module SISO (D, CLK, T);
input CLK, D;
output T;
reg T;   reg Q, R, S;
always @ (negedge CLK)
begin
  Q <= D;
  R <= Q;
  S <= R;
  T <= S;
end
endmodule
```



Note, we can use a new assignment operator <= within always block which (unlike = operator), executes all associated statements concurrently. If we had used = instead of <=, the D input through sequential execution would have reached final output in one clock cycle; also all the flip-flops within the register will have same value that of serial data input. Use of = operator is called *blocking mode operation* and use of <= is called *non blocking mode.*

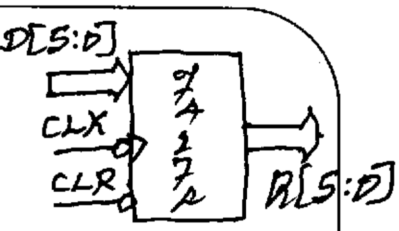
Now, we will see a 4-bit SIPO right shift register, where all the flip-flop outputs are available externally.

```
module SIPO (D, CLK, Q);
input CLK, D;
output [3:0] Q;  reg [3:0] Q;
always @ (negedge CLK)
  begin
    Q[0] <= D;
    Q[1] <= Q[0];
    Q[2] <= Q[1];
    Q[3] <= Q[2];
  end
endmodule
```

D
CLK
SIPO
Q[3:0]

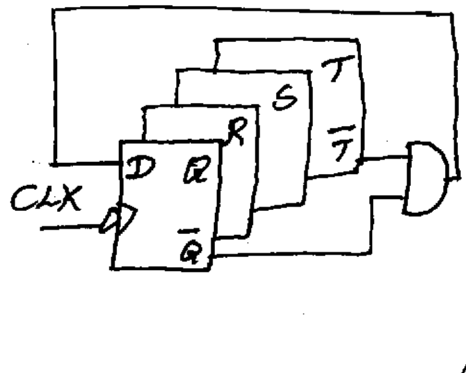
The PIPO register, of IC 74174, in Verilog code is as given below:

```
module Reg74174 (D, CLK, CLR, Q);
input CLK, CLR; input [5:0] D;
output [5:0] Q; reg [5:0] Q;
always @ (negedge CLK or negedge CLR)
if (~CLR) Q = 6'b0;
else Q = D;
endmodule
```



Verilog Code for Switched-Tail Counter:

```
module STC (CLK, Y);
input CLK, output Y; reg Q, R, S, T;
assign Y = (~Q) & (~T);
always @ (negedge CLK)
begin
    Q <= ~T; // Tail is switched & connected to i/p
    R <= Q;
    S <= R;
    T <= S;
end
endmodule
```



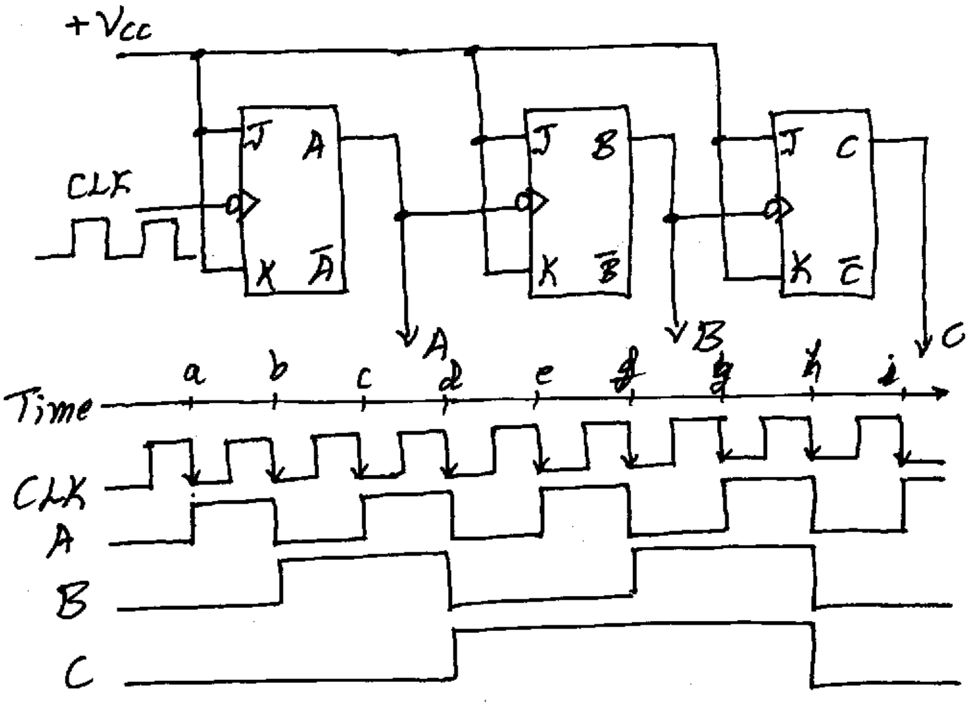
COUNTERS

A *counter* is a sequential circuit that goes through a prescribed sequence of states up on the application of input pulse. Counters are in two categories –

- *Ripple (Asynchronous) Counter* – consists of a series connection of complementing flip-flops, with the output of each flip-flop connected to the clock-pulse input of the next higher-order flip-flop. The flip-flop holding the LSB receives the clock-pulse.
- *Synchronous Counter* – the input pulses / clock-pulses are applied to all clock-pulse inputs of all the flip-flops simultaneously.

ASYNCHRONOUS COUNTERS:

The following Fig shows three negative-edge-triggered, JK flip-flops connected in cascade to form a 3-bit ripple counter. The system clock (a square wave), drives flip-flop A. The output of flip-flop A drives B, and the output of B drives flip-flop C. All the J and K inputs are tied to $+V_{CC}$. Hence, flip-flops will toggle with a negative transition at its clock input.



Negative Clock	C	B	A	State or Count
-	0	0	0	0
a	0	0	1	1
b	0	1	0	2
c	0	1	1	3
d	1	0	0	4
e	1	0	1	5
f	1	1	0	6
g	1	1	1	7
h	0	0	0	0 repeats

3-Bit Binary Ripple Counter, Waveforms & Truth Table sume that, the flip-flops are all initially reset to 0 outputs. For every clock NT, flip-flop A will change state. Notice that, the waveform at the output of flip-flop A is one-half the clock frequency.

Since, A acts as clock for B, each time the waveform at A goes low, flip-flop B will toggle. Notice that, the waveform at the output of flip-flop B is one-half the frequency of A and one-fourth the clock frequency.

Since, B acts as clock for C, each time the waveform at B goes low, flip-flop C will toggle. The waveform at the output of flip-flop C is one-half the frequency of B and one-eighth the clock frequency.

Problem: *What is the clock frequency of a 3-bit ripple counter, if the period of the MSB waveform is 24 μs?*

Solution: Since there are eight clock cycles in one cycle of MSB, the period of the clock must be 24/8 = 3 μs. The clock frequency must be $1/(3*10^{-6})$ = 333 KHz.

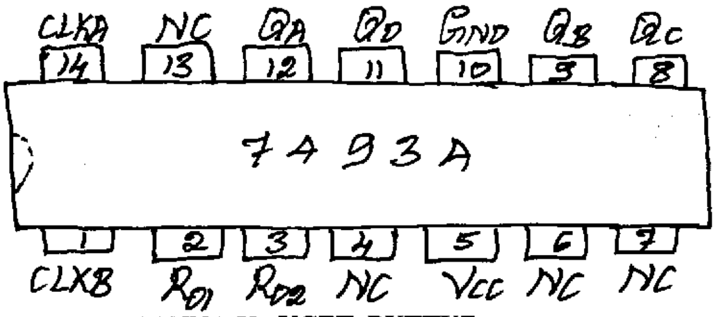
NOTE:

1. A binary ripple counter in straight binary sequence will be as shown in the above table. A ripple counter having *n* flip-flops will have 2^n output conditions. For example, the three-flip-flop counter has $2^3 = 8$ output conditions (000 to 111).
2. A three-flip-flop counter is often referred to as modulus-8 (or Mod-8) counter, since it has eight states. The *modulus* of a counter is the total number of states through which the counter can progress.

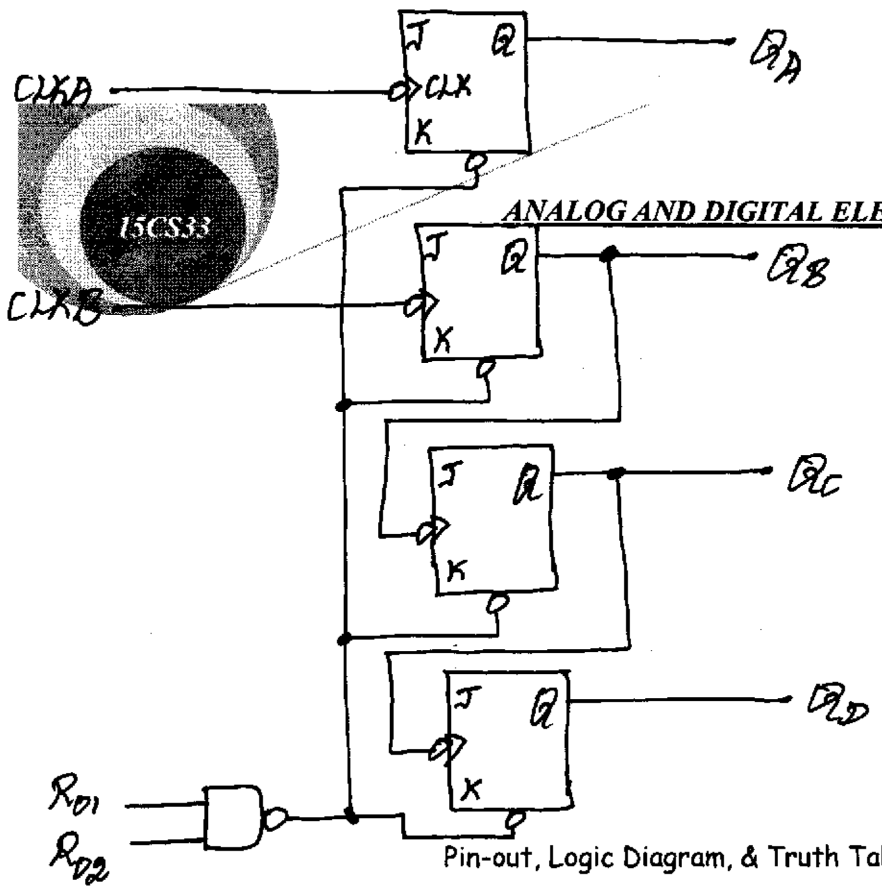
Problem: *How many flip-flops are required to construct a mod-128 counter? A mod-32 counter? What is the largest decimal number that can be stored in a mod-64 counter?*

Solution: A mod-128 counter must have seven flip-flops, since $2^7 = 128$. Five flip-flops are needed to construct a mod-32 counter. The largest decimal number that can be stored in a mod-64 (six flip-flops) counter is 111111 = 63.

The 54/7493A: The pin-out, logic diagram, and truth table for a 54/7493A are given below. This TTL MSI circuit is a 4-bit binary counter that can be used in either mod-8 or mod-16 configuration.



Count	Output			
	Q_D	Q_C	Q_B	Q_A
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H



4	L	H	L	L
5	L	H	L	H
6	L	H	H	L
7	L	H	H	H
8	H	L	L	L
9	H	L	L	H
10	H	L	H	L
11	H	L	H	H
12	H	H	L	L
13	H	H	L	H
14	H	H	H	L
15	H	H	H	H

Pin-out, Logic Diagram, & Truth Table of 54/7493A

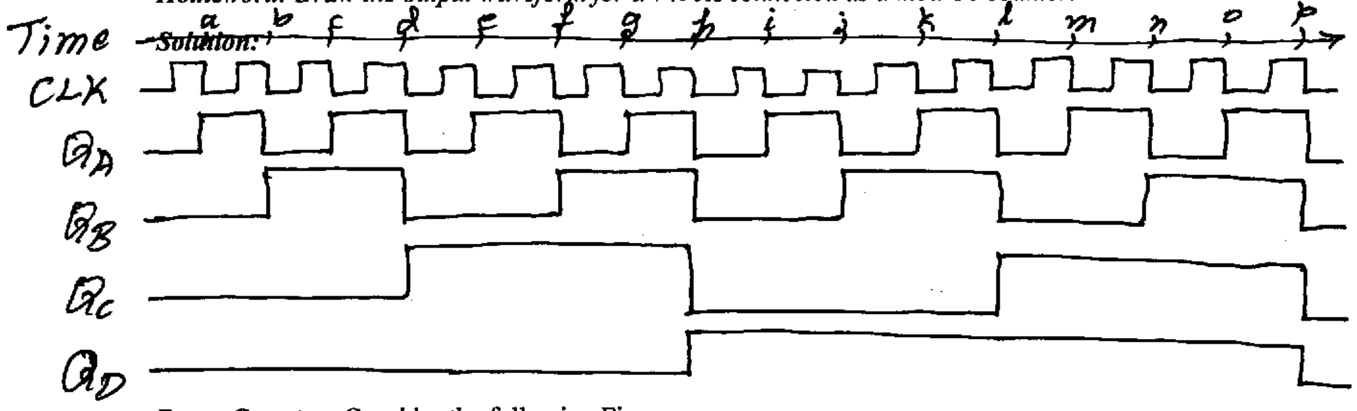
If the clock is applied at input CKB, the outputs will appear at Q_B, Q_C, and Q_D; and this is mod-8 binary ripple counter. In this case, flip-flop Q_A is unused.

On the other hand, if the clock is applied at input CKA, and the flip-flop Q_A is connected to input CKB; a mod-16, 4-bit binary ripple counter is possible. The outputs are Q_A, Q_B, Q_C, and Q_D.

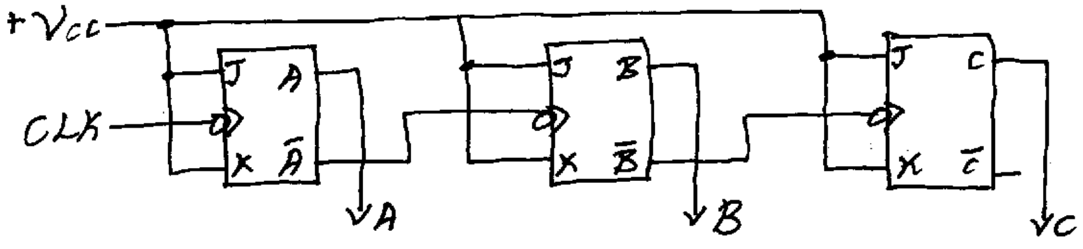
A high level at both reset inputs of the NAND gate, $R_{0(1)}$ and $R_{0(2)}$ will reset all flip-flops simultaneously.

Homework: *Draw the output waveform for a 7493A connected as a mod-16 counter.*

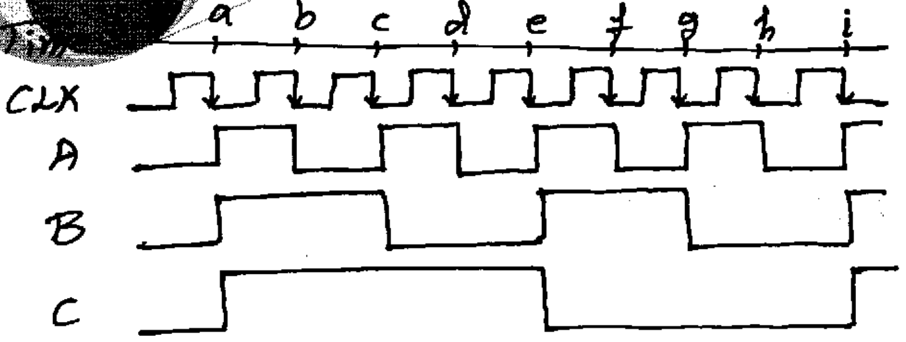
Solution:



Down Counter: Consider the following Fig.



C	B	A	Count
1	1	1	7
1	1	0	6
1	0	1	5
1	0	0	4

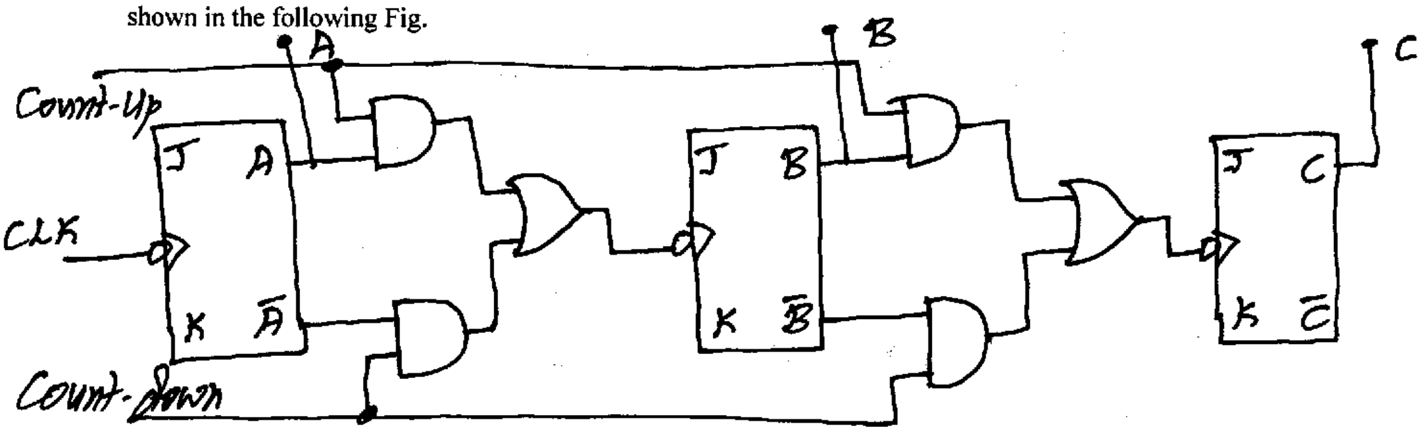


0	1	1	3
0	1	0	2
0	0	1	1
0	0	0	0
1	1	1	7 repeats

A Down Counter

The system clock used at the clock input to flip-flop A, but the complement of A, $\bar{A}$, is used to drive flip-flop B; likewise $\bar{B}$ is used to drive flip-flop C. This is still a mod-8 counter, since it has eight discrete states, but it is configured as a down counter.

UP-Down Counter: A 3-bit asynchronous up-down counter that counts in a straight binary sequence is shown in the following Fig.

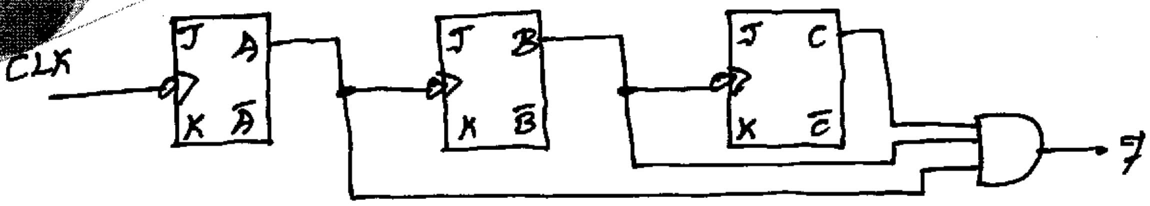


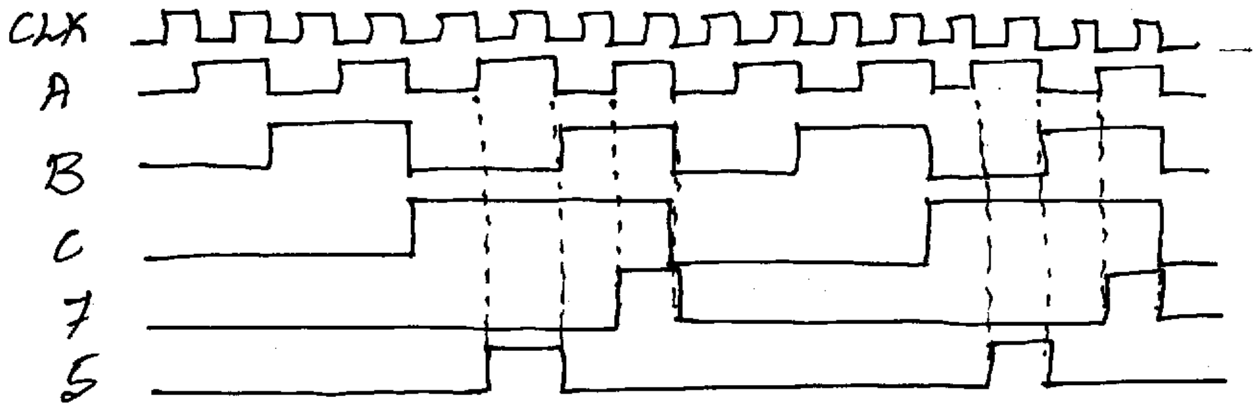
3-Bit Binary Up-Down Counter

If the count-down control line is low, and the count-up control line is high; the counter will count-up. On the other hand, if the count-down control line is high, and the count-up control line is low; the counter will count-down.

DECODING GATES:

A decoding gate can be connected to the outputs of a counter in such a way that the output of the gate will be high (or low) only when the counter contents are equal to the given state. For example, the decoding gate connected to the 3-bit ripple counter, in the following Fig, will decode state 7 (CBA = 111). The gate output will be high only when A = 1, B = 1, and C = 1. The Boolean expression is: 7 = CBA.





The other seven states of the counter can be decoded in a similar fashion. For example, to decode state 5; CBA = 101 is the unique state. The Boolean expression is: 5 = C$\bar{B}$A. Note that, eight gate are necessary to decode the eight states of the 3-bit counter.

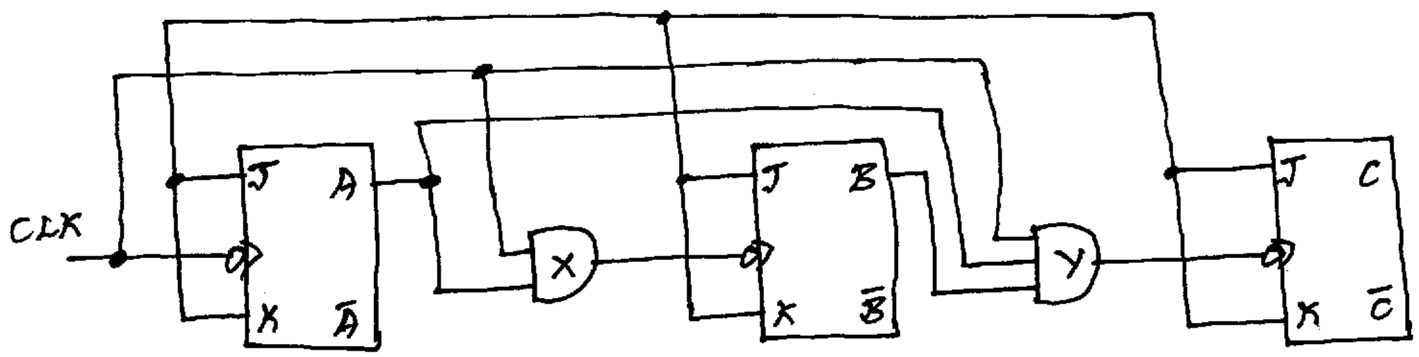
NOTE:

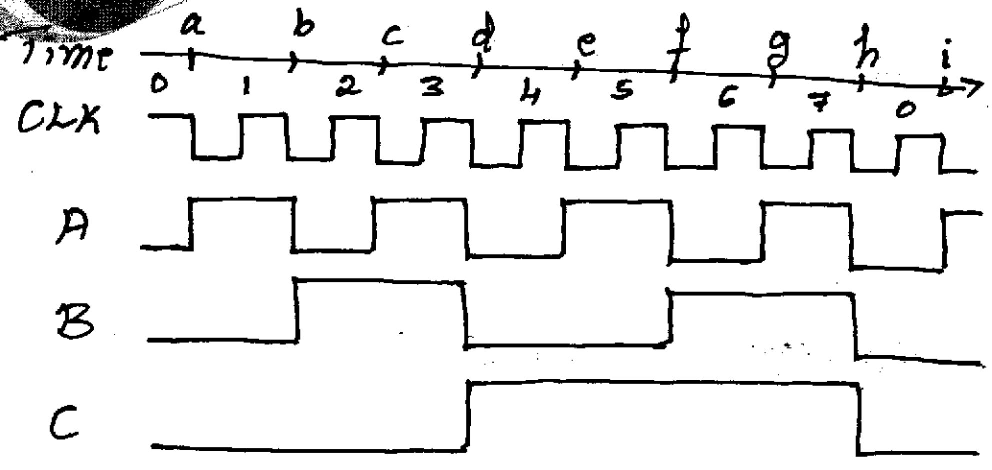
1. The ripple counter is the simplest to build, but there is a limit to its highest operating frequency. Each flip-flop has a delay time. In a ripple counter, these delay times are additive, and the total "settling" time for the counter is approximately the delay times the total number of flip-flops.
2. There is a possibility of glitches occurring at the output of decoding gates used with the ripple counter.

Both of these limitations can be overcome by the use of a synchronous or parallel counter.

SYNCHRONOUS COUNTER:

The construction of parallel binary counter is shown in the following Fig.





C	B	A	Count
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7
0	0	0	0 repeats

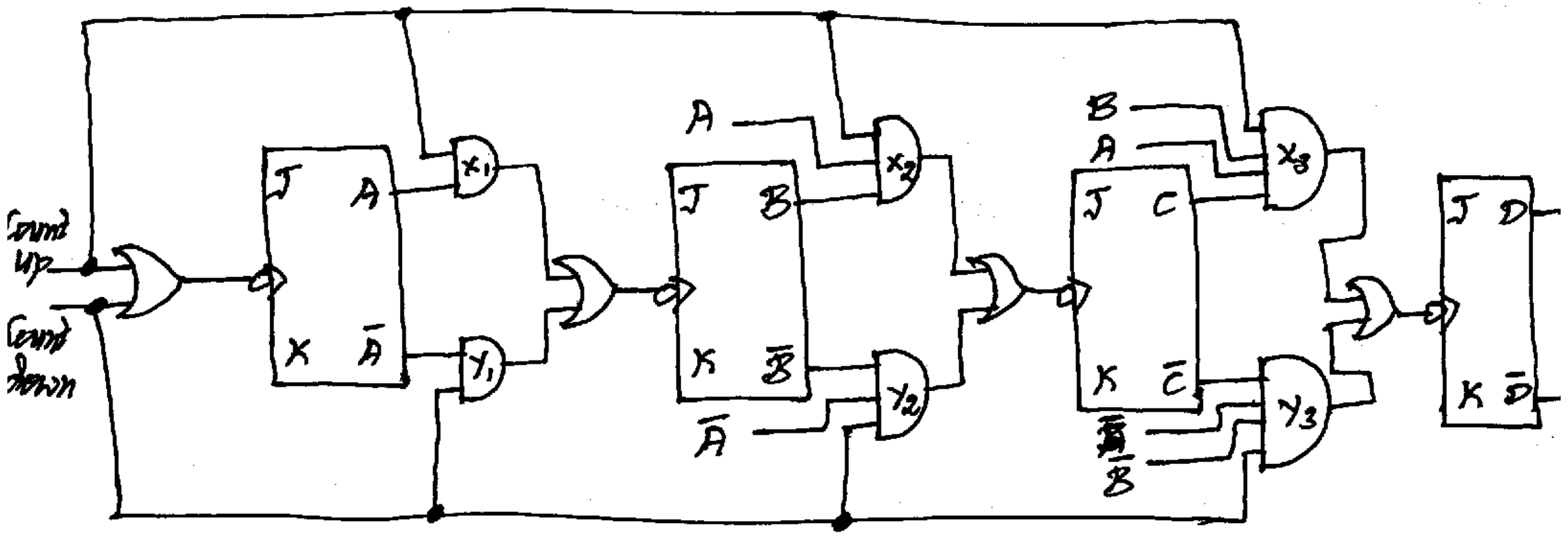
Mod-8 Parallel (Synchronous) Binary Counter

The basic idea here is to keep the J and K inputs of each flip-flop high, such that the flip-flop will toggle with any clock NT at its clock input. We then use AND gates to gate every second clock to flip-flop B, every fourth clock to flip-flop C, and so on.

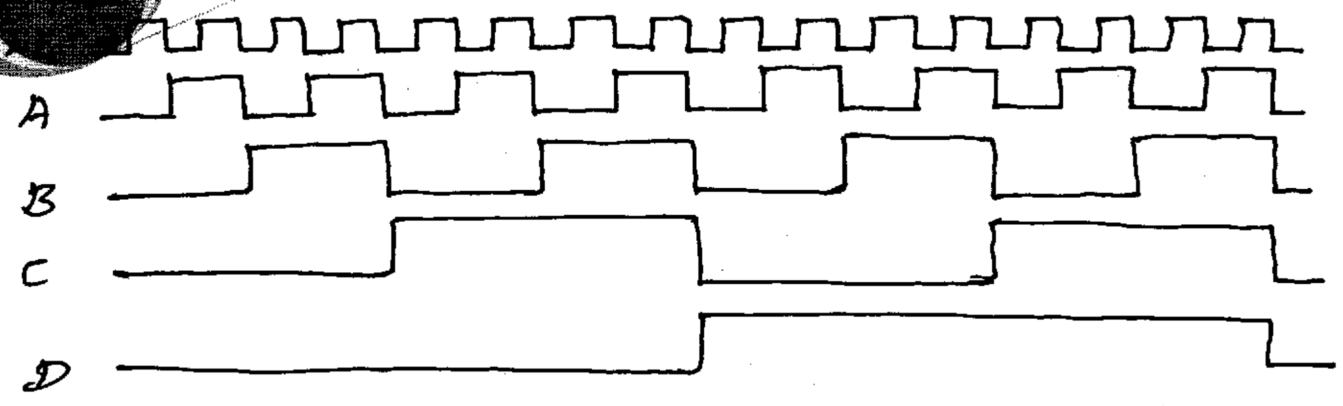
Whenever A is high, AND gate X is enabled and a clock pulse is passes through the gate to the clock input of flip-flop B. Thus B changes state.

Since AND gate Y is enabled and will transmit the clock to the flip-flop C only when both A and B are high, flip-flop C changes state with every fourth clock NT.

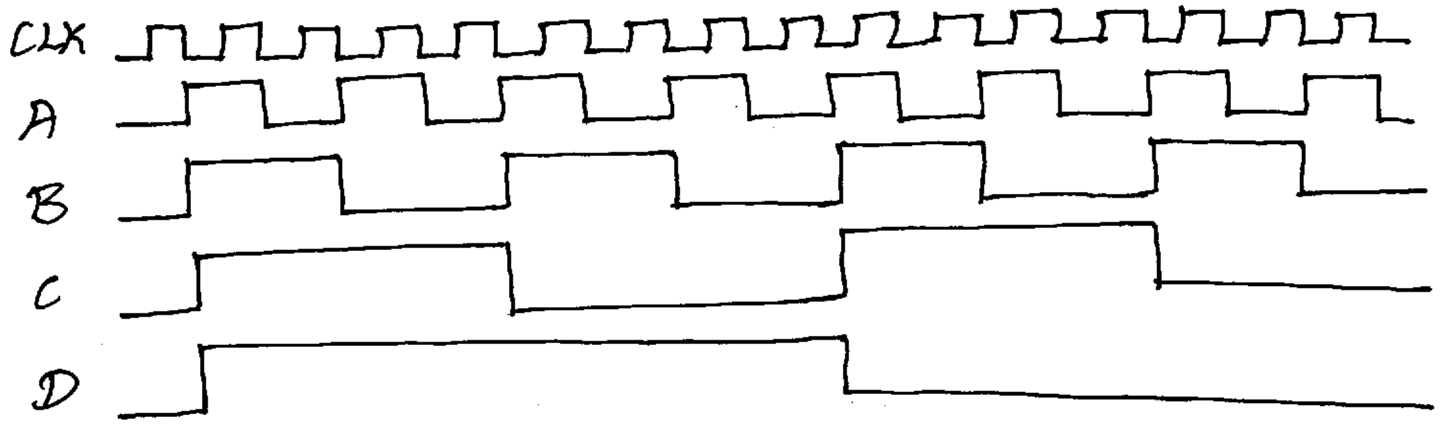
Parallel Up-Down Counter: A parallel up-down counter can be constructed as shown in the following Fig.



Logic Diagram of Parallel Up-Down Counter



Count-Up Waveform



Count-Down Waveform

In parallel counter, the time at which any flip-flop changes state is determined by the states of all previous flip-flops in the counter. In the count-up mode, a flip-flop must toggle every time all previous flip-flops are in a 1 state, and the clock makes a transition. In the count-down mode, flip-flop toggles must occur when all prior flip-flops are in a 0 state.

To operate in the count-up mode, the system clock is applied at the count-up input, while the count-down input is held low; which will disable AND gates Y_1, Y_2, and Y_3. The clock applied at the count-up will then go directly into flip-flop A and will be steered into the other flip-flops by AND gates X_1, X_2, and X_3.

To operate in the count-down mode, the system clock is applied at the count-down input, while the count-up input is held low; which will disable AND gates X_1, X_2, and X_3. The clock applied at the count-down will then go directly into flip-flop A and will be steered into the other flip-flops by AND gates Y_1, Y_2, and Y_3.

54/74193: Self Study.

NOTE:

No. of Flip-Flops	*1*	*2*	*3*	*4*	*5*	*n*
No. of States	*2*	*4*	*8*	*16*	*32*	2^n

HANGING THE COUNTER MODULUS:

No. of FFs	No. of States
1	2
2	4
3	8
4	16
5	32
⋮	⋮
n	2^n

If synchronous / asynchronous counter progress one count at a time in a strict binary progression, then they have a modulus given by 2^n, where, n indicates the number of flip-flops. Such counters are said to have a *natural count* of 2^n.

Example; A mod-2 counter consists of a single flip-flop; a mod-4 counter required two flip-flops; three flip-flops form a mod-8 counter.

Thus, we can construct counters that have a natural count of 2, 4, 8, 16, 32, and son on by using the proper number of flip-flops.

A small modulus counter can be constructed from a larger modulus counter by skipping states. Such counters are said to have a *modified count.*

The correct number of flip-flops, for a modified counter is determined by choosing the lowest natural count that is greater than the desired modified count. For example, a mod-7 counter requires three flip-flops, since 8 is the lowest natural count greater than the desired modified count of 7.

Problem: *Indicate how many flip-flops are required to construct each of the following counters: (a) mod-3, (b) mod-6, (c) mod-9.*

Solution:

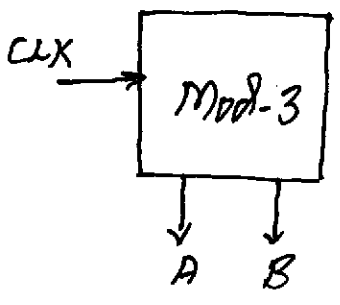
(a) The lowest natural count greater than 3 is 4. Two flip-flops are required to generate a natural count of 4. Therefore, it requires at least two flip-flops to construct a mod-3 counter.

(b) The lowest natural count greater than 6 is 8. Three flip-flops are required to generate a natural count of 8. Therefore, it requires at least three flip-flops to construct a mod-6 counter.

(c) The lowest natural count greater than 9 is 16. Four flip-flops are required to generate a natural count of 16. Therefore, it requires at least four flip-flops to construct a mod-9 counter.

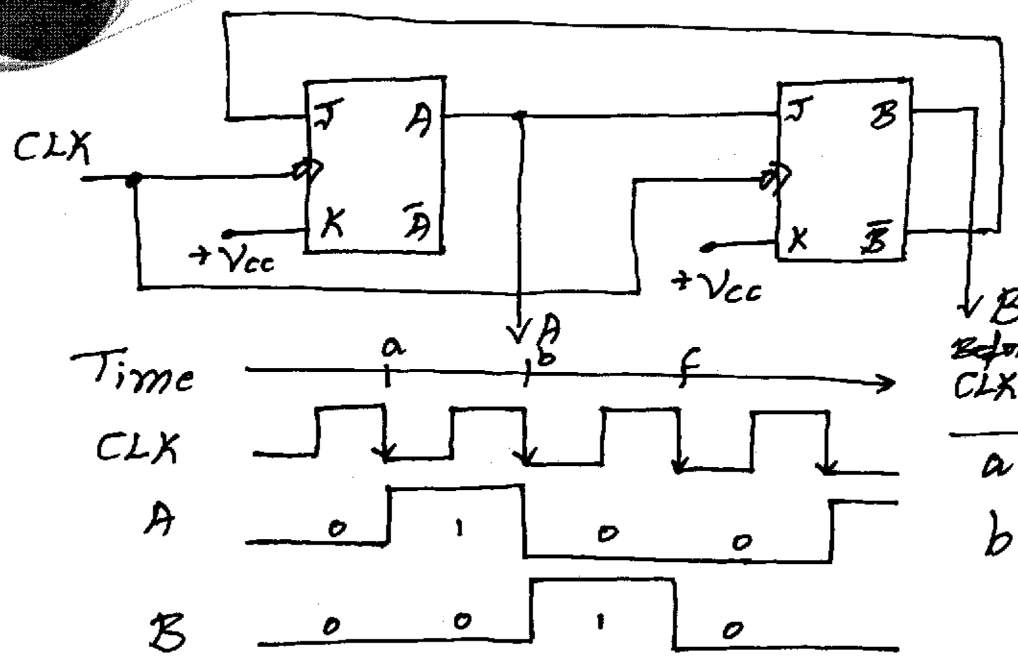
A Mod-3 Counter:

The two flip-flops in the following Fig have been connected to provide a mod-3 counter. Two flip-flops have a natural count of 4, but this counter skips one state.



B	A	Count
0	0	0
0	1	1
1	0	2
0	0	0

B	A	Count
0	0	0
0	1	1
1	0	2
0	0	0 repeat



$J_A = \bar{B}$

$J_B = A$

$K_A = K_B = 1.$

Before CLK	J_A	K_A	J_B	K_B	A FF	B FF
a	1	1	0	1	0	0
b	1	1	1	1	1	0

Mod-3 Counter

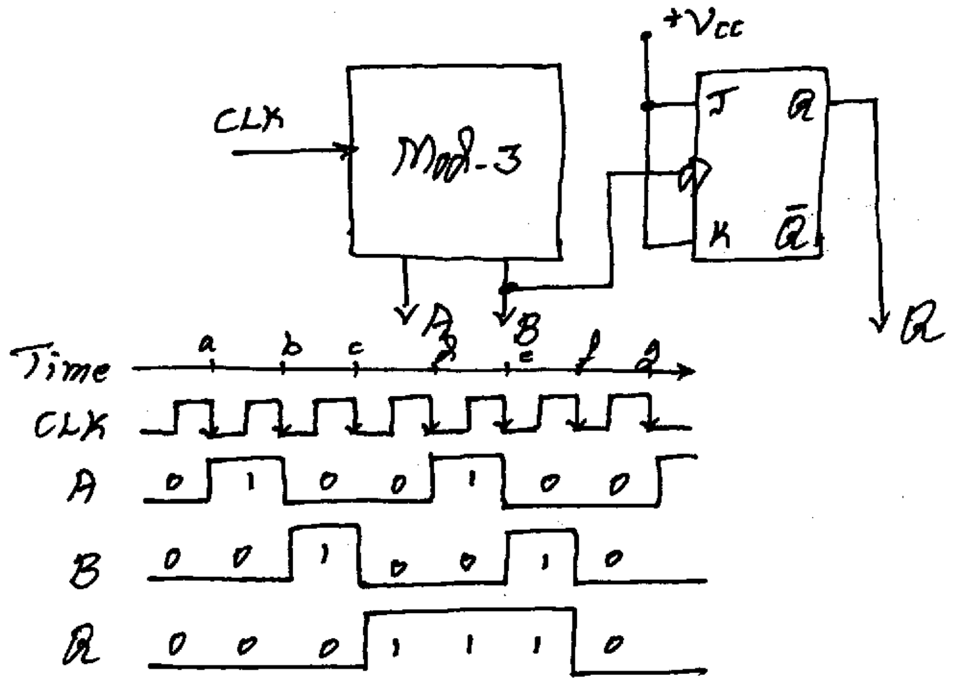
This counter progress through the count sequence 00, 01, 10, and then back to 00. The explanation is as follows:

1. Prior to point *a* on time line, A = 0 and B = 0. A negative clock transition at *a* will cause:
    a. A to toggle to a 1, since its J and K inputs are high
    b. B to reset to 0 (it's already 0), since its J input in low and K input is high
2. Prior to point *b* on the time line, A = 1 and B = 0. A negative clock transition at *b* will cause:
    a. A to toggle to a 0, since its J and K inputs are high
    b. B to toggle to a 1, since its J and K inputs are high
3. Prior to point *c* on the time line, A = 0 and B = 1. A negative clock transition at *c* will cause:
    a. A to reset to 0 (it's already 0), since its J input in low and K input is high
    b. B to rest to 0, since its J input in low and K input is high
4. The counter has now progressed through all three of its states, advancing one count with each negative clock transition.

NOTE: If we consider a basic flip-flop to be amod-2 counter, we see that a mod-4 counter (two flip-flops in series) is simply two mod-2 counters in series. Similarly, a mod-8 counter is simply a 2 x 2 x x connection, and so on. Thus, a greater number of higher-modulus counters can be formed by using the product of any number of lower-modulus counters.

A Mod-6 Counter (3 x 2, Mod-6 Counter):

A 3 x 2, mod-6 counter is shown in the following Fig.

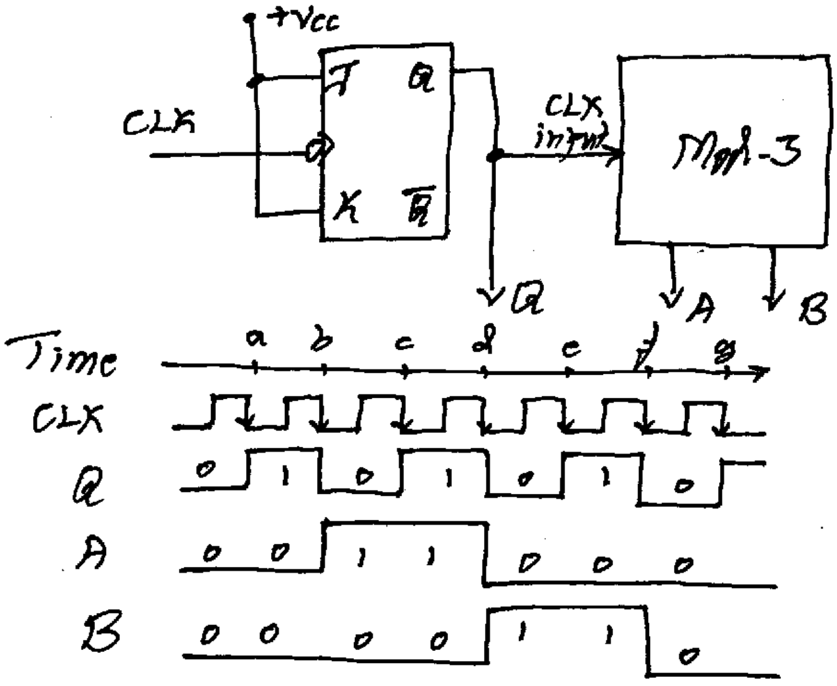


Q	B	A	Count
0	0	0	0
0	0	1	1
0	1	0	2
1	0	0	4
1	0	1	5
1	1	0	6
0	0	0	0 repeats

Bi-Quinary Count Sequence

A Mod-6 Counter (2 x 3, Mod-6 Counter):

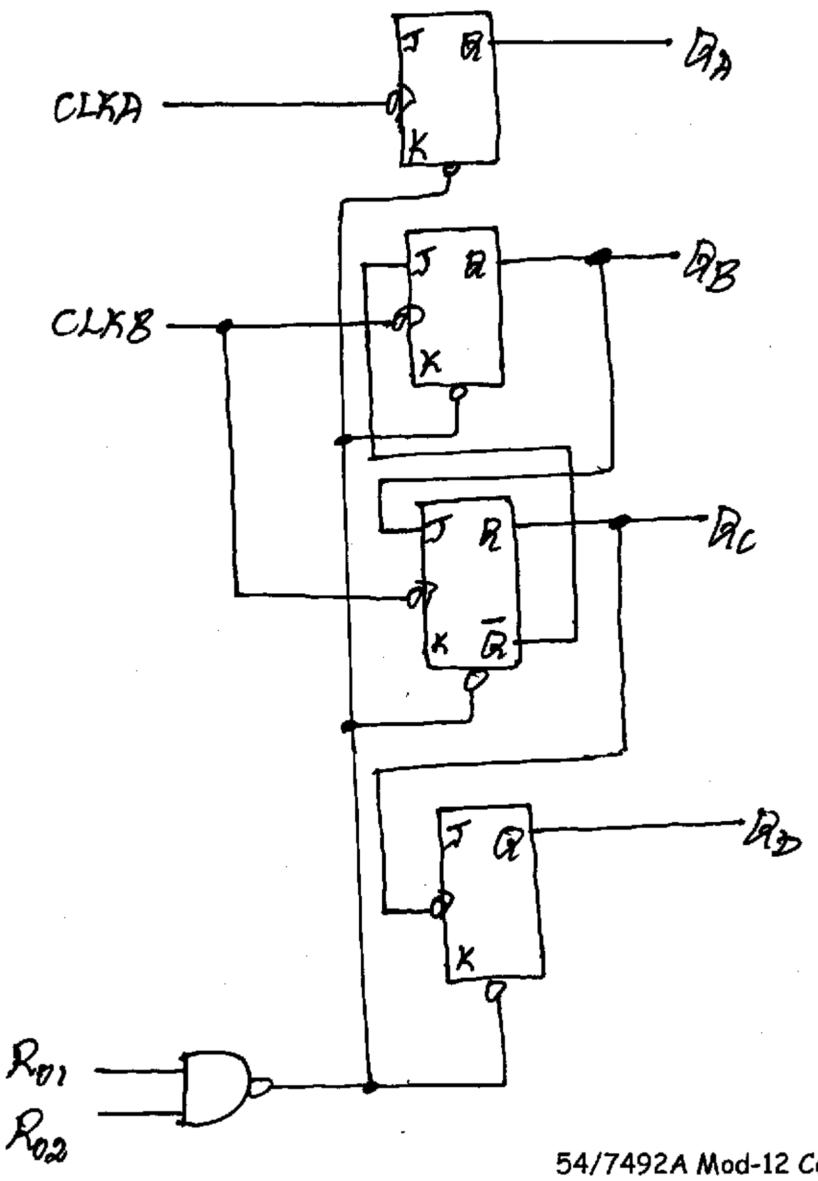
A 2 x 3, mod-6 counter is shown in the following Fig.



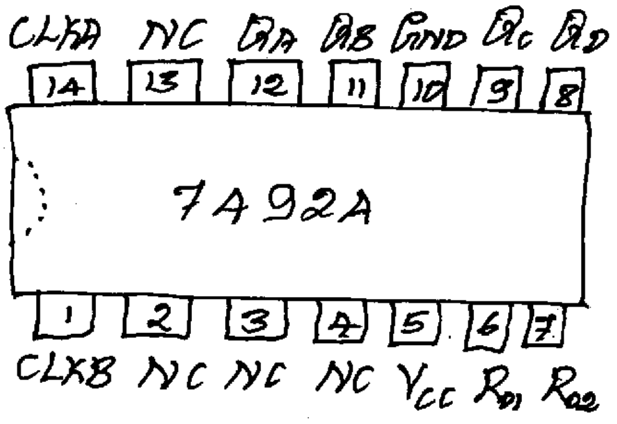
B	A	Q	Count
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
0	0	0	0 repeats

Straight-Binary Count Sequence

The **54/7492A:** is a TTL divide-by-12, MSI counter. If the clock is applied to input B and the outputs are taken at Q_B, Q_C, Q_D, this is a mod-6 counter. On the other hand, if the clock is applied at input A and Q_A is connected input B, we get a 2 x 3 x 2, mod-12 counter. This must be considered as asynchronous counter, since all the flip-flops do not change states at the same time.



Count	Output			
	Q_D	Q_C	Q_B	Q_A
0	L	L	L	L
1	L	L	L	H
2	L	L	H	L
3	L	L	H	H
4	L	H	L	L
5	L	H	L	H
8	H	L	L	L
9	H	L	L	H
10	H	L	H	L
11	H	L	H	H
12	H	H	L	L
13	H	H	L	H



54/7492A Mod-12 Counter

By: **MAHESH PRASANNA K.,**

DEPT. OF CSE, VCET.

****

****