Flip-Flop Timing:

Diodes and transistors cannot switch states immediately. It always takes a small amount of time to turn a diode on or off. It takes time for a transistor to switch from saturation to cutoff and vice versa.

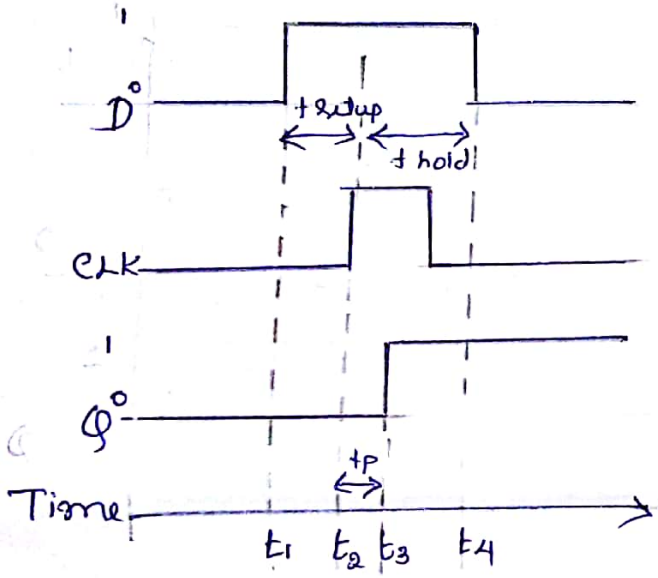
Switching time is the main cause of propagation delay, designated t_p. This represents the amount of time it takes for the output of a gate or flip-flop to change states after the input changes.

For example if the data sheet of an edge triggered D flip-flop lists $t_p = 10ns$, it takes about 10ns for Q_n to change state after D has been sampled by the clock edge.

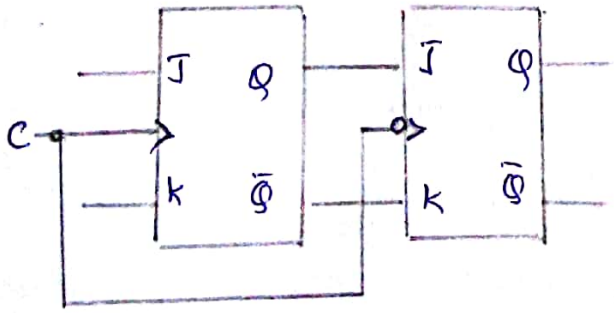
The propagation delay time is used to construct the pulse forming circuit. When flip-flops are used to construct counters, the propagation delay is small & ignored.

Stray capacitance at the D input makes it necessary for data D to be at the input before the clock edge arrives. The setup time t_{setup} is the minimum amount of time that the data bit must be present before the clock edge hits. For example, if a D flip-flop has a setup of time of 15ns, the data bit to be stored must be at the D input at least 15ns before the clock edge arrives.

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



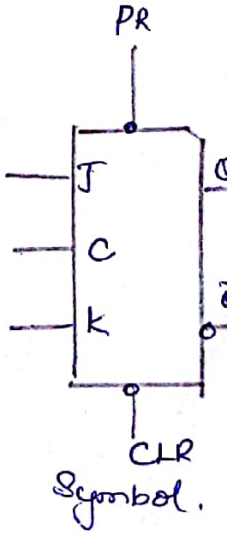
JK Master - Slave Flip-flops.



Master Slave Flip-flop

C	J	K	Q_{n+1}	Action
⊓	L	L	Q_n	No Change
⊓	L	H	L	RESET
⊓	H	L	H	SET
⊓	H	H	$\bar{Q}_n$	Toggle.

Truth table



Symbol.

Master is positive edge triggered and the slave is negative edge triggered. Therefore the master responds to its J & K inputs before the slave.

1) 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, so on the negative clock transition the slave sets, copying the action of the master.

2) If J=0 & K=1 the master resets on the PT of the clock. The high $\bar{Q}$ output of the master goes to the K input of the slave. Therefore, the NT of the clock forces the slave to reset. Again, the slave has copied the master.

3) If the master's J & K inputs are both high, it toggles on the PT of the clock and the slave then toggles on the clock NT. Hence slave copies the master.

4) If J=K=0 the flip-flop is disabled and Q remains unchanged

The master is set according to J and K while the clock is high, the contents of the master are then shifted into the slave when the clock goes low. This flip-flop can be ~~buffered~~ referred as pulse triggered flip-flop.

Race around condition can be avoided by using a Master-Slave JK flip-flop.

Switch Contact Bounce Circuit:

In some occasion digital system uses mechanical contacts for the purpose of conveying an electrical signal. Example: switches used on the keyboard of a computer system.

In each case, the intent is to apply a high logic level (+5Vdc) or a low level (0Vdc). The single pole single throw switch is one such example (fig A)

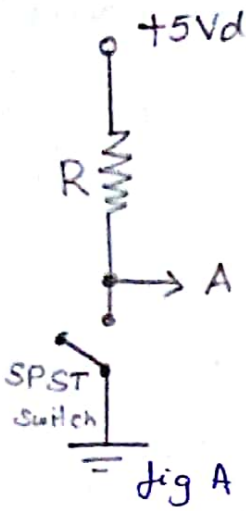


fig A

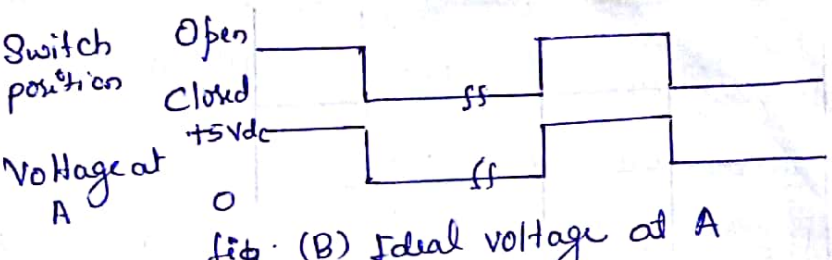


fig. (B) Ideal voltage at A

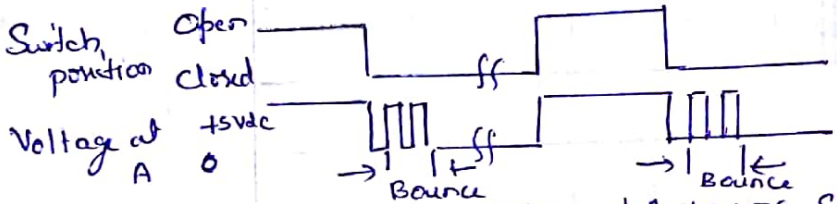


fig (C) Voltage at A showing contact bounce

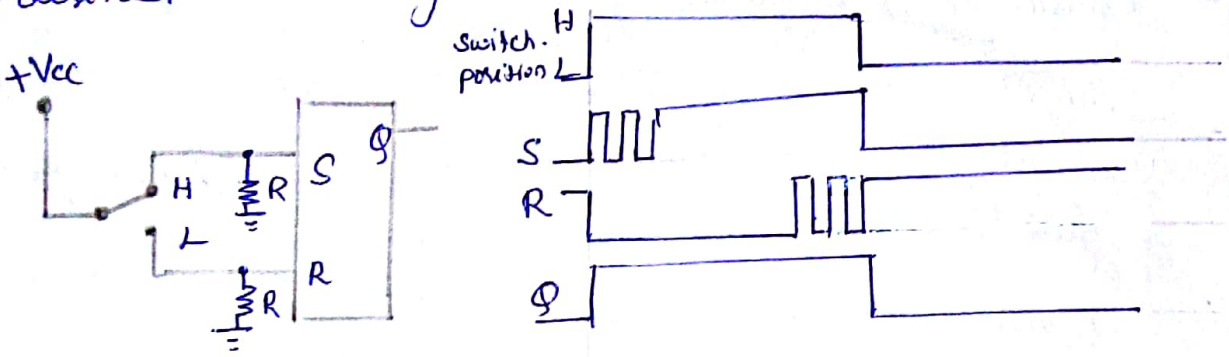
When the switch is open, the voltage at point A is +5 Vdc, when switch is closed the voltage at point A is 0Vdc. Ideally the voltage waveform at A should appear as shown in fig (B) as the switch is moved from open to close or vice versa. But actually, the waveform at point A will appear more or less as shown in fig (C) as result of phenomenon known as ~~coun~~ 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 like hard ball bounce when dropped on the hard surface. The number of bounces that occur & the period of bounce differ from each switching device.

Note: Even though actual physical bounce occurs each time the switch is opened or closed, contact bounce appears in the voltage level at A only when the switch is closed.

A simple RS Latch Debounce Circuit:

The RS latch in below fig will remove any contact bounce due to switch. The output Q is used to generate

the desired switch signal.



Switch contact bounce eliminator

When the switch is moved to position H, R = 0 and S = 1 bouncing occurs at S input due to switch. The flip-flop sees this as a series of high and low inputs, settling with a high level. The flip-flop will immediately be set with Q = 1 at the first high level on S. When the switch bounces the input signals are R = S = 0 therefore the flip-flop remains set. When the switch bounces contact, R = 0 and S = 1, this set the flip-flop.

When the switch is moved to position L, S = 0 and R = 1, Bouncing occurs at the R input due to the switch. Again the flip-flop 'sees' this as a series of high & low inputs. It simply responds to the first high level and ignores all following transition. The result is a clean high to low signal at the flip-flop output.

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-flops

The characteristic 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 input to flip-flops.
K map can be used to get the optimized expression and truth table of each flip-flop is mapped into it.

analysis circuit

RS flip-flop

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

	$\bar{S}\bar{R}$ 00	$\bar{S}R$ 01	SR 11	$S\bar{R}$ 10
$\bar{Q}_n$ 0	0	0	X	1
Q_n 1	1	0	X	1

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

D flip-flop

	$\bar{D}$ 0	D 1
$\bar{Q}_n$ 0	0	1
Q_n 1	0	1

D	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

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

JK flip-flop

Q_n	J	K	Q_{n+1}
0	0	0	0
1	0	0	1
0	0	1	0
1	0	1	0
0	1	0	1
1	1	0	1
0	1	1	1
1	1	1	0

	$\bar{J}\bar{K}$ 00	$\bar{J}K$ 01	JK 11	$J\bar{K}$ 10
$\bar{Q}_n$ 0	0	0	1	1
Q_n 1	1	0	0	1

$$Q_{n+1} = Q_n\bar{K} + \bar{Q}_nJ$$

T flip-flop

	$\bar{T}$ 0	T 1
$\bar{Q}_n$ 0	0	1
Q 1	1	0

T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

$$T = T\bar{Q}_n + \bar{T}Q_n$$
$$= T \oplus Q_n$$

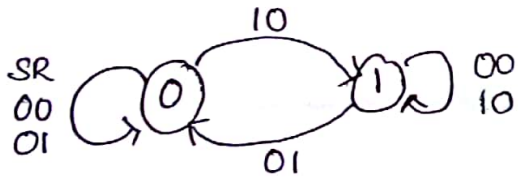
Summary :

SR Flip-flop	$Q_{n+1} = S + \bar{R}Q_n$
JK Flip-flop	$Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n$
D Flip-flop	$Q_{n+1} = D$
T Flip-flop	$Q_{n+1} = T\bar{Q}_n + \bar{T}Q_n = T \oplus Q_n$

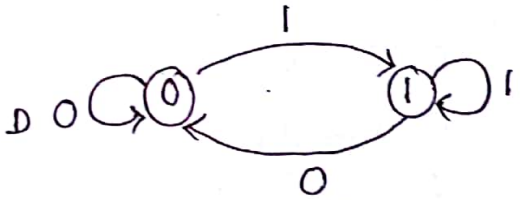
progress of sequential logic with time

Flip-flop as Finite State Machine :

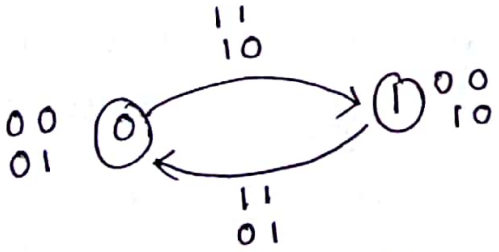
In a sequential logic circuit the value of all the memory elements at a given time define the state of that circuit at that time. Finite State Machine (FSM) concept offers a better alternative to truth table in understanding progress of sequential logic with time



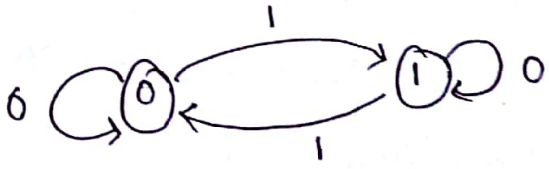
SR flip-flop



D flip-flop



JK flip-flop



T flip-flop.

Let us see how state transition diagram for SR flip-flop is developed from truth table. [illegible] state of flip-flop can be 0 or 1. If current state is 0 and SR = 00, flip-flop will remain same state. If current state is 0 and SR = 01, next state will be 0. If current state

If current state is 0 and SR = 10 next state will be 1.
If current state is 1 and SR = 00 next state will be 1.
If current state is 1 and SR = 10 next state will be 1
If current state is 1 and SR = 01 next state will be 0.

Flip-flop Excitation Table. truth table analysis prob.

Here flip-flop input is presented as a dependent function of transition $Q_n \rightarrow Q_{n+1}$ and comes later in the table. This is derived from flip-flop truth table or characteristic equation but more directly from its state transition diagram.

Q_n	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

From SR flip-flop state transition diagram, if present state is 0 application of SR = 0X does not alter its value. State 0 to 1 transition occurs when SR = 10 is present at the input side while state 1 to 0 transition occurs if SR = 01. Present is maintained if SR = X0.

HDL implementation of Flip-flop

Behavioral model is preferred for sequential circuit and 'always' keyword is used in all those circuits.

Enabled D Flip-flop.

D	EN	Q_{n+1}
0	1	0
1	1	1
0	X	Q_n

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

Enabled RS flip-flop:

$Q = S + \bar{R}Q$

```
module SRLatch (S, R, En, Q)
input S, R, En;
Output Q;
reg Q;
always@(En or S or R)
    if (En) Q = S| (~R &Q);
endmodule
```

We use keyword posedge and negedge for positive edge and negative edge respectively.

positive edge triggered D

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

negative edge triggered D

```
module DFFneg (D, C, Q)
input D, C;
output Q;
reg Q;
always @ (negedge C)
Q=D;
endmodule
```

D flip-flop with positive edge triggered and clear input :

```
module DFFpos_clr (D, C, CLR, Q);
input D, C, CLR;
output Q;
reg Q;
always @ (posedge C or negedge CLR)
    if (~CLR) Q = 1'b0; // Q store 0
    else Q = D;
endmodule
```

positive edge triggered SR flip flop

```
module SRFFpos (S, R, C, Q)
input S, R, C;
output Q;
reg Q;
always @ ( posedge C)
    Q = S | (~R & Q)
endmodule
```

REGISTERS

A register is a very important digital building block.
A data register is used to store (momentarily) binary information appearing at the output of an encoding matrix.

Registers are groups of flip-flops where each flip-flop is capable of storing one bit of information. An n bit register is a group of n flip-flops. The basic function of a register is to hold information in a digital system and make it available to the logic elements for the computing process. Two basic functions of register is data storage & data movement.
Register is capable of storing finite number 0-1 combination in finite number of flip-flops.

Types of Registers:

A register used to store an 8 bit binary number, then it has 8 flop-flops.

A shift register is a register that allows each of the flip flop to pass the stored information to its adjacent neighbour.

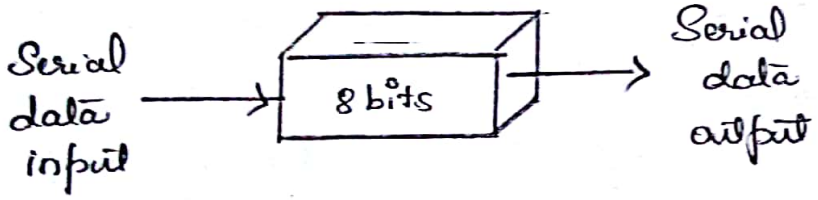
The bits in a binary number can be moved from one location to another in either of 2 ways.

Shifting the data 1 bit at a time in serial fashion i.e beginning with either the MSB or LSB is called serial shifting.

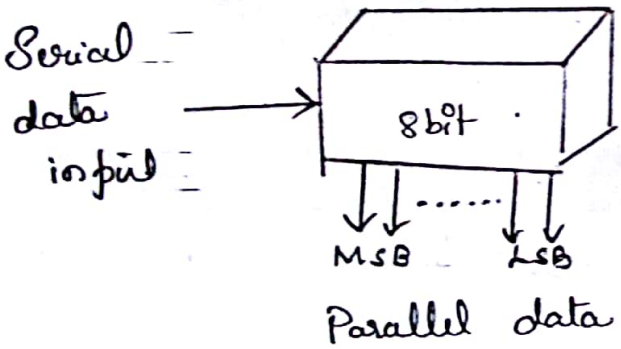
Shifting of all the data bits simultaneously is called parallel shifting.

There are 2 ways to shift data into register and 2 ways to shift data out of the register. So there are 4 basic types of register (shift registers).

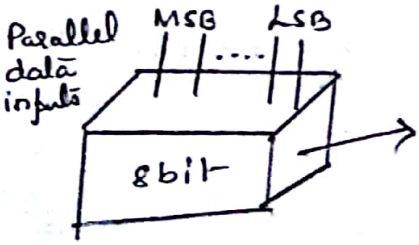
1) Serial in serial out : 54/74LS91 (8 bit)



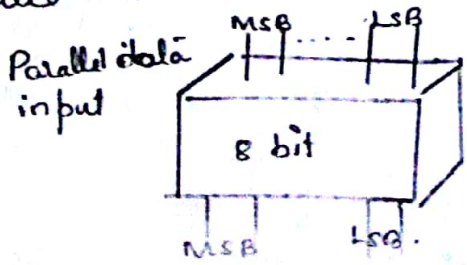
2) Serial in parallel out : 54/74164 (8bits)



3) Parallel in serial out : 54/74165, 8 bits



4) Parallel in Parallel out : 54/74198, 8 bits

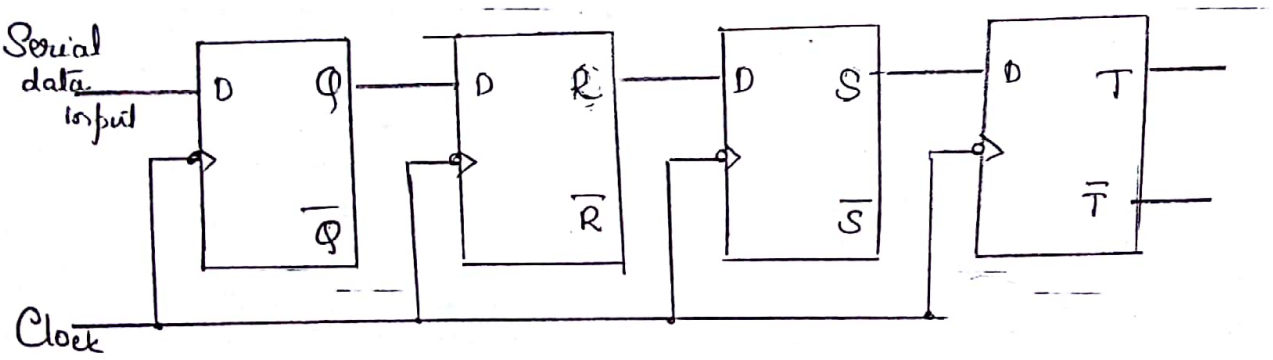


Serial IN Serial Out:

Usually edge triggered JK, SR or D type flip-flops are used to construct registers.

In Serial in Serial out registers data is serially entered or exited from a shift register.

Consider 4 D flip-flop connected as shown below to form a 4 bit shift register:



A common clock provides trigger at its negative edge to all the flip-flops. As output of one D flip-flop connected to input of the next at every clock trigger data stored in one flip-flop is transferred to the next. For this circuit transfer takes place like this $Q \rightarrow R$, $R \rightarrow S$, $S \rightarrow T$ and serial data input is transferred to Q.

Assume all the flip-flop are initially cleared & Serial data to be inputed is 0100.

At clock edge A, flip-flop Q has input 0 from serial data in D flip-flop R has input 0 from output of Q, flip-flop S has input 0 from the output of R and flip-flop T has input 0 from outputs when clock triggers, these inputs get transferred to corresponding flip-flop outputs simultaneously so that QRST = 0000. Thus at clock trigger values at DQRS is transferred

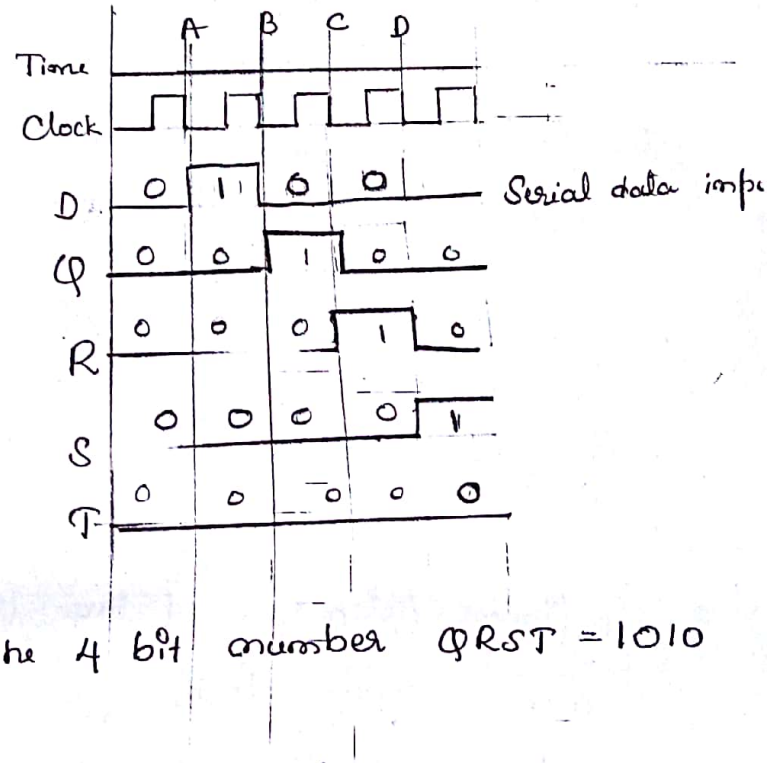
to QRST

At clock edge B : serial data in = 0, i.e DQRS = 0000, So after NT at B QRST = 0000

At clock edge C : Serial data in = 1, i.e DQRS = 1000 and after NT QRST = 1000.

At clock edge D : serial data in = 0, i.e DQRS = 0100 and after NT QRST = 0100

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



Suppose that register has the 4 bit number QRST = 1010 stored in it.

Before Time A : The register stores the number QRST = 1010 and T = 0 (appear at output)

At time A :- The entire number is shifted one flip-flop to the right. The register holds the bits QRST = 0101 and T = 0

At time B : The register holds QRST = 0010 and T = 0 (appears at output)

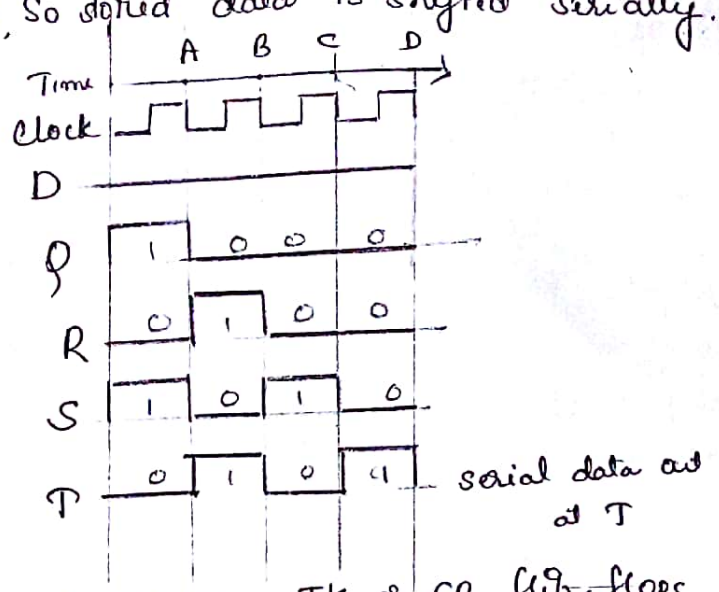
At time C : The register holds QRST = 0001 and T = 1 comes out

At time D : The register holds QRST = 0000, MSB is shifted out the right end last.

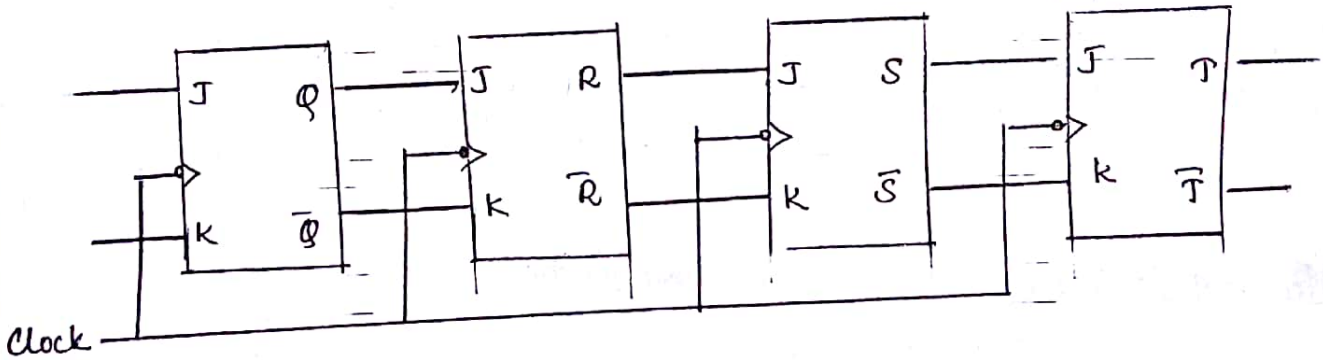
We have caused the number stored in register to appear at T

1 bit at a time beginning with LSB. So stored data is shifted serially.

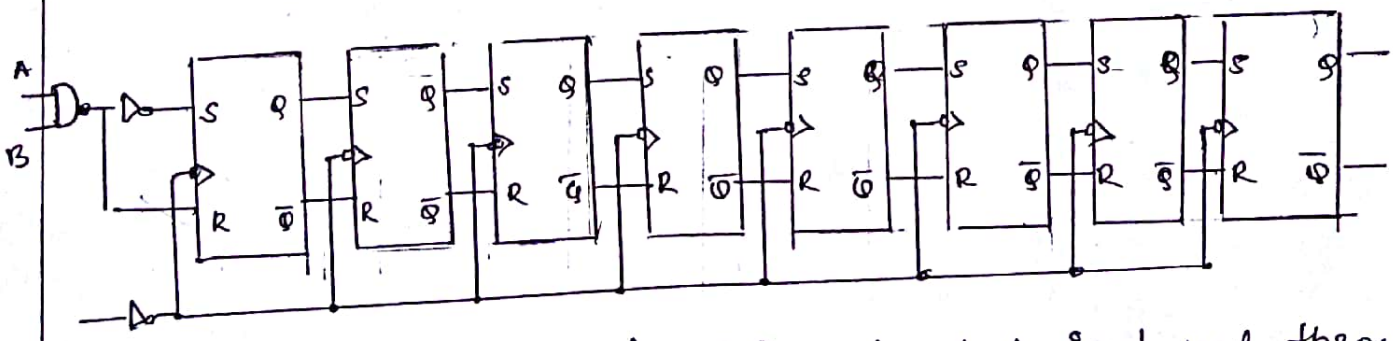
Clock	Q	R	S	T	Serial out
0	1	0	1	0	0
1	0	1	0	1	1
2	0	0	1	0	0
3	0	0	0	1	1
4	0	0	0	0	



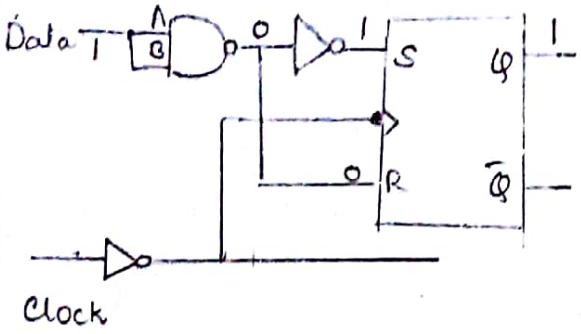
A shift register can be implemented using JK or SR flip-flops also. [Q is connected J next flip-flop & $\bar{Q}$ is connected to K of next FF]



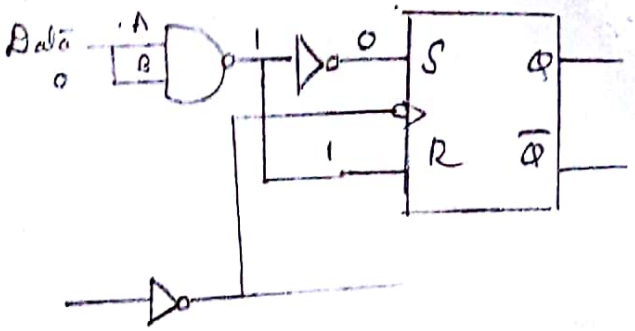
74LS91 is 8 bit shift register using 8 RS flip-flop as shown below:



Each flip-flop is negative-edge-triggered, clock is passed through an inverter, data will be shifted on positive edges. The inverter connected b/w R & S on the 1st flip-flop means that circuit functions as a D type flip-flop. Data passed at (A or B) get complemented at NAND gate & NOT gate [A=1 => S=1 or A=0 => S=0]



When Data input is 1 then S=1 and R=0, flip-flop is set when clock goes high

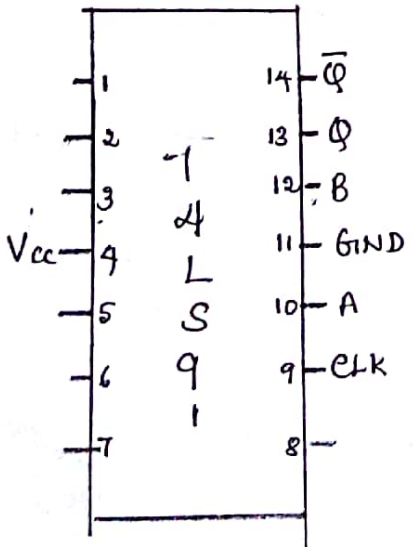


When data input is 0 then S=0 and R=1, flip-flop is reset when clock goes high.

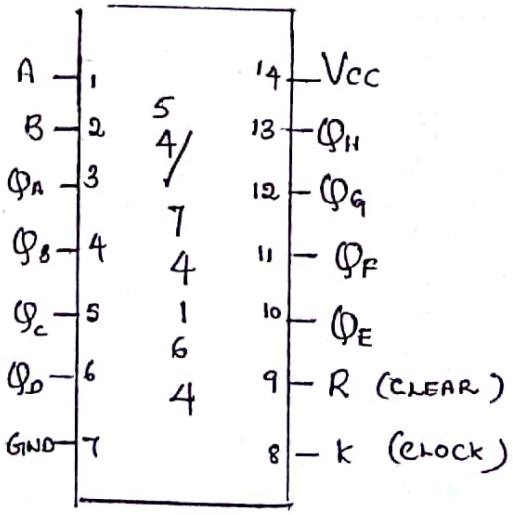
Serial In - Parallel Out:

Data is shifted in serially and shifted out parallelly. To shift the data out in parallel, all the data bits should be available at outputs at the same time; we can achive this by connecting output of each flip to the output pin.

The 54/74164 is an 8 bit serial input - parallel output shift register. It is constructed by using RS flip-flops having clock negative edge triggered.



74LS91 pin diagram



54/74164 pin diagram.

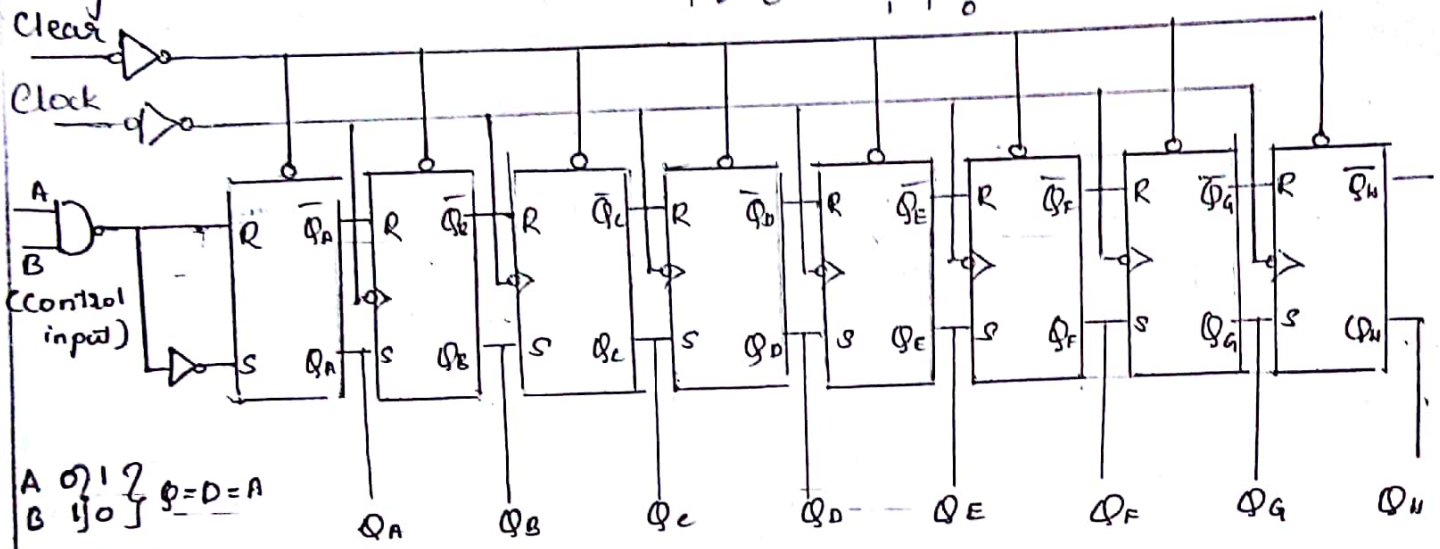
In 54/74164 output of each flip-flop is available simultaneously and each flipflop has an asynchronous clear input. Low to

.To clear will used every flip-flop. When clear is high all flip-flop work normally

Shifting data into register in a serial fashion is exactly the same as previous 74LS91

R	S	Q_n
0	0	Q_n
0	1	1
1	0	0

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



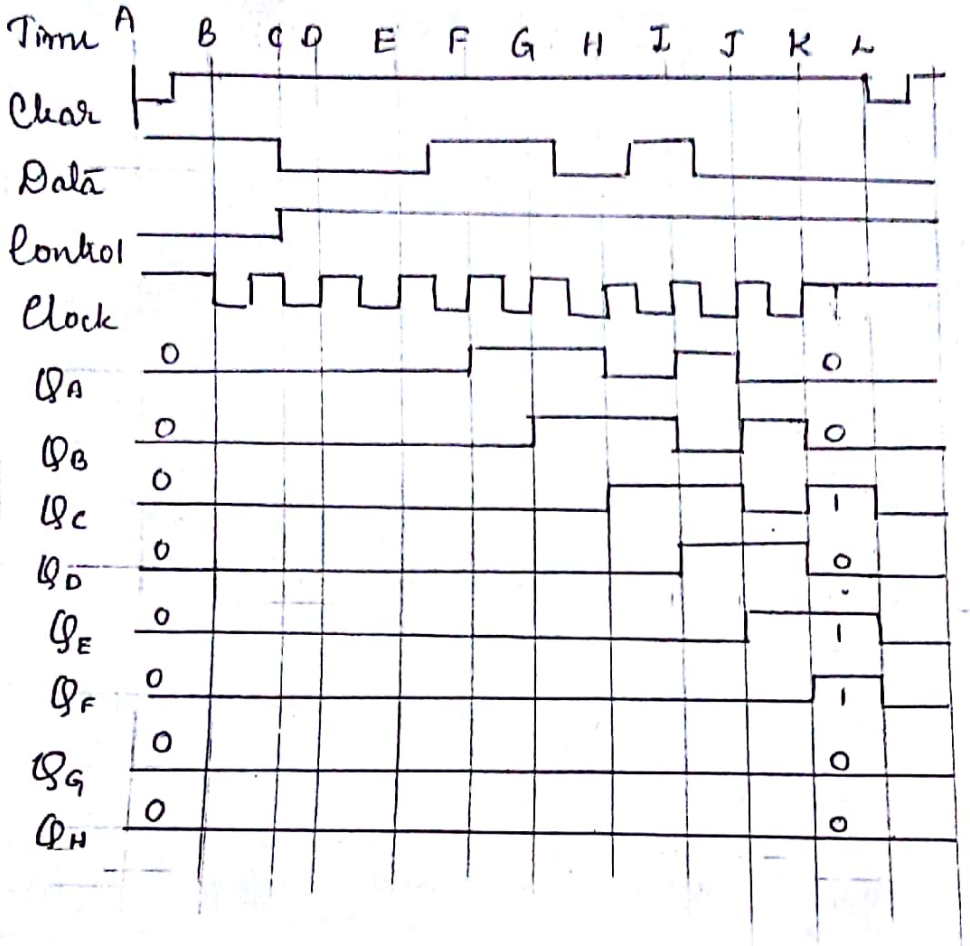
A 0 1 } $Q = D = A$
B 1 0 }

A 1 0 } $Q = 1 = A$
B 1 1 }

Data are shifted into the register on PTs. (because of NOT gate)
B is held High: NAND gate is enabled and the serial input data passes through the NAND gate inverted. The input data is shifted into the register.
B is held Low: The NAND gate is forced high, the input data stream is inhibited.

When clear is low all flip-flop value is reset to 0.
Data input = 00101100

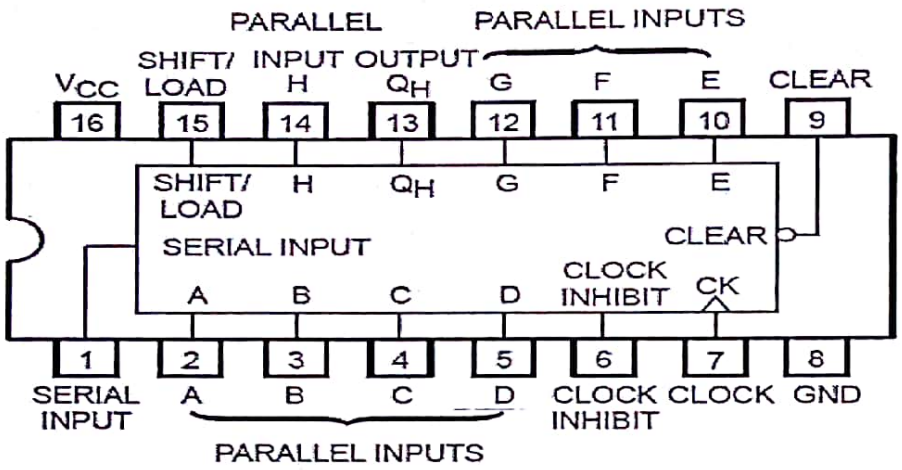
* Clock Begins at time B, no effect on Flip-flop data because control line is low
* First data bit '0' is shifted into register at time D.
* Next 7 data bits are shifted in order at time E, F, G, H, I, J, and K respectively.
* After K clock remains high & bit number 00101100 is available on output lines

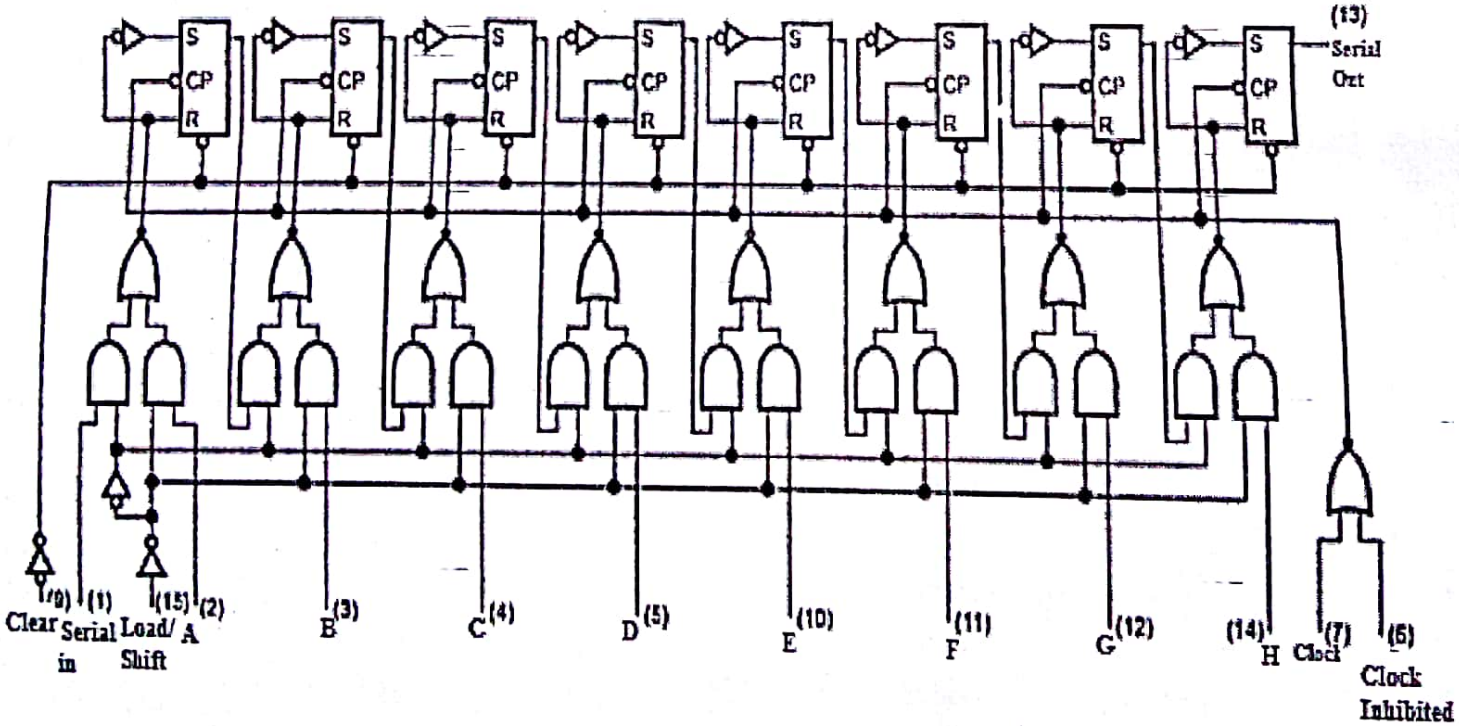


Parallel In Serial Out :

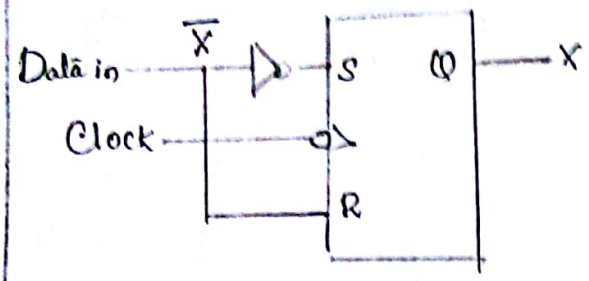
Parallel entry of data into the register an serial removal of data from the register.

54/74166 is commercially available PISO TTL device. This is an 8 bit shift register capable of either serial/parallel data entry and serial data output. It is controlled using eight negative edge triggered RS flip-flop.

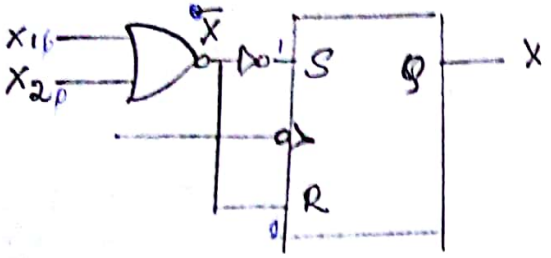




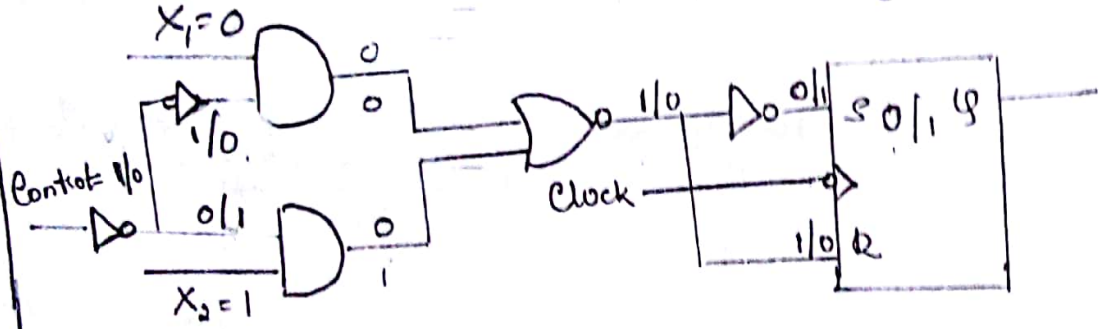
In RS flip-flop R input is inverted and given to S
So this becomes a D-type flip-flop.



X = 1
if $\bar{X} = 0$, then S = 1 and R = 0
1 will be clocked (inputted) to flip-flop



If X = 1, the 1 is clocked into the flip-flop. NOR gate offers to enter data from 2 different source.
i.e. if X_1 is grounded X_2 is shifted in
or if X_2 is grounded X_1 is shifted in.



R	S	AND	NOR	RS
0	0	0	1	Q_n
0	1	0	0	1
1	0	0	0	0
1	1	1	0	F

A control input is introduced by adding 2 AND gates and 2 NOT gates as shown in above fig.

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

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

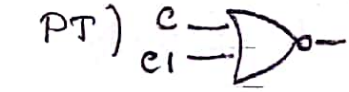
They are connected to allow 2 different operations
1) parallel entry of data 2) serially shifting of data through Q_A to Q_n

Shift/Load is Low: A single clock transition loads 8 bits of data (ABCDEFGH) into the register in parallel (i.e lower AND gates are enabled)

Shift/Load is High: Clock transition will shift data through the register serially, with entering data applied at the SERIAL OUT.

When clear is low all flip-flops in register value is cleared to 0.

Clock is applied through a NOR gate (2 input) when clock inhibit is low, the clock signal passes through NOR gate inverted, hence data will shift into register on the PTs (flip-flops are NTs but NOT of NT is PT)



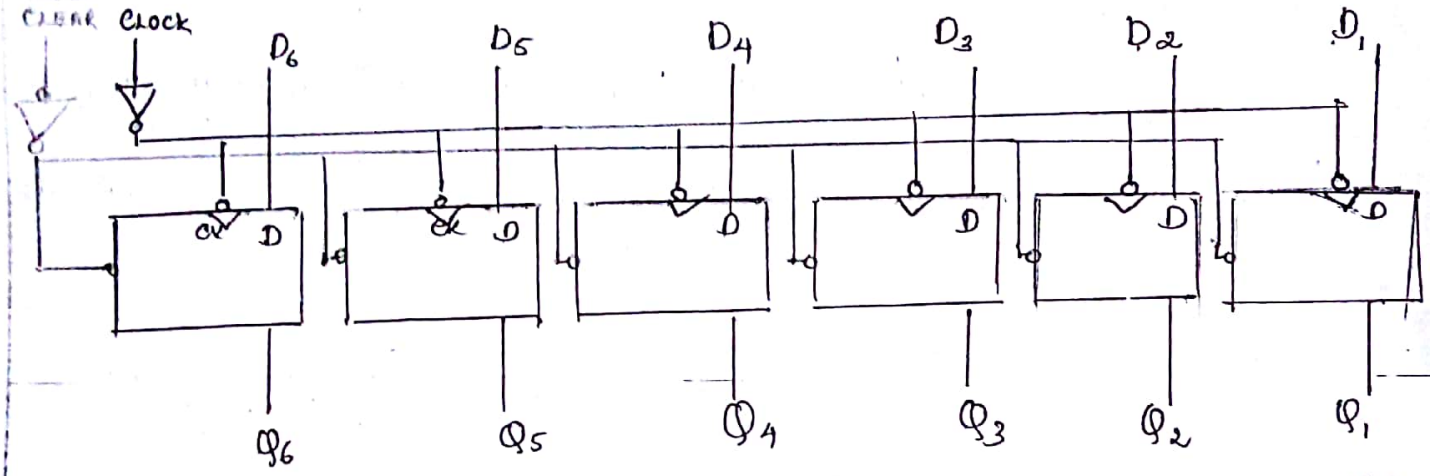
When clock inhibit is high NOR gate output is low, clock is prevented from reaching flip-flop, the register hold its contents.

C	CI	
0	0	1
1	0	0
0	1	0
1	1	0

Parallel in Parallel Out:

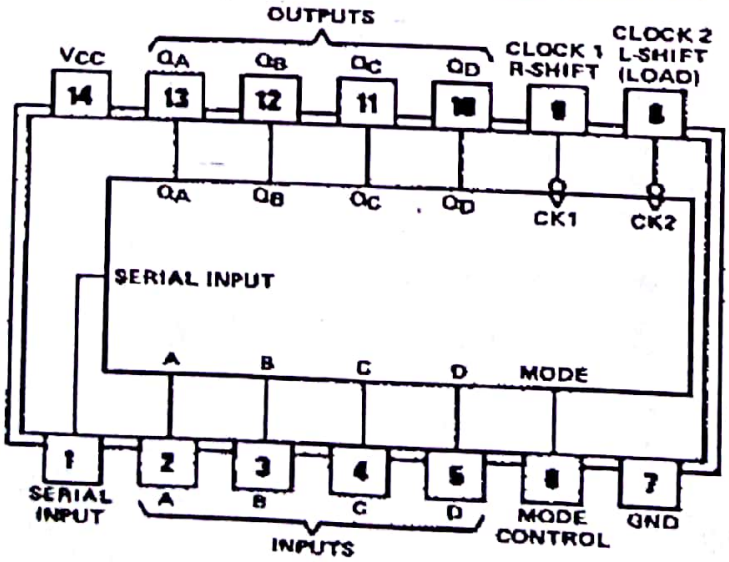
74174 is an example of a parallel in - parallel out register. In 74174 6 negative edge triggered D flip-flops are used. Because of NOT gate present at clock data is shifted in PTs The 6 data bits D_1 through D_6 are all shifted into the register in parallel. The stored data is immediately available in parallel at the outputs Q_1 through Q_6. They are simply used to store data hence called data register / data latch. It is not

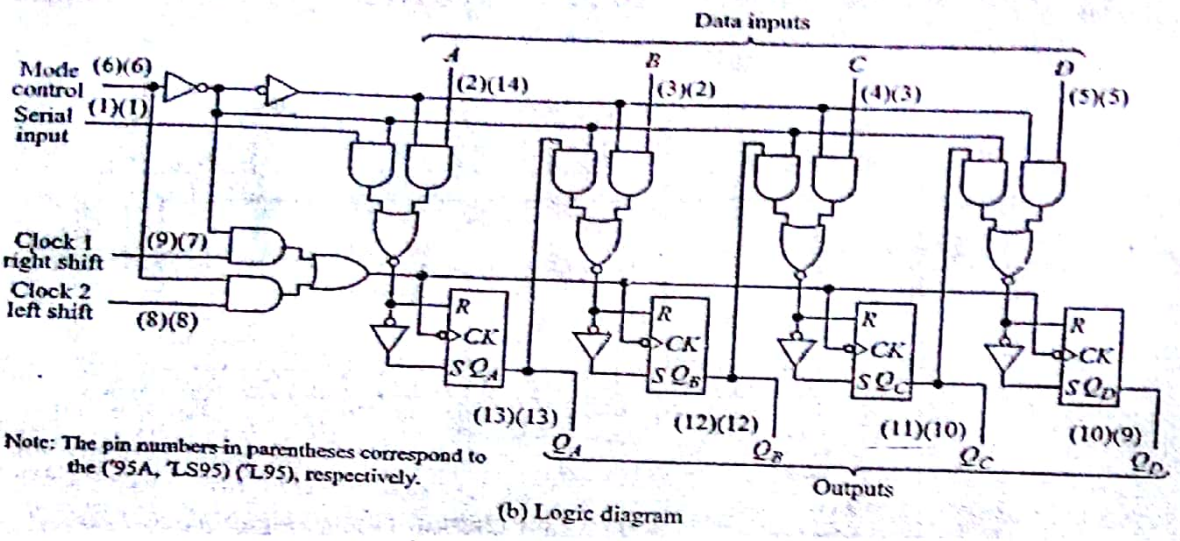
Here it is not possible to shift stored data either to the right or left. Asynchronous clear low will clear all flip flops to low.



54/74198 is an 8 bit TTL having both parallel input and parallel output capacity. It uses positive edge triggered flip-flop. It can also be used to shift data through the register in either direction.

54/7495 is a 4 bit parallel access shift register. It also has serial data input and can be used to shift data to the right and to the left.

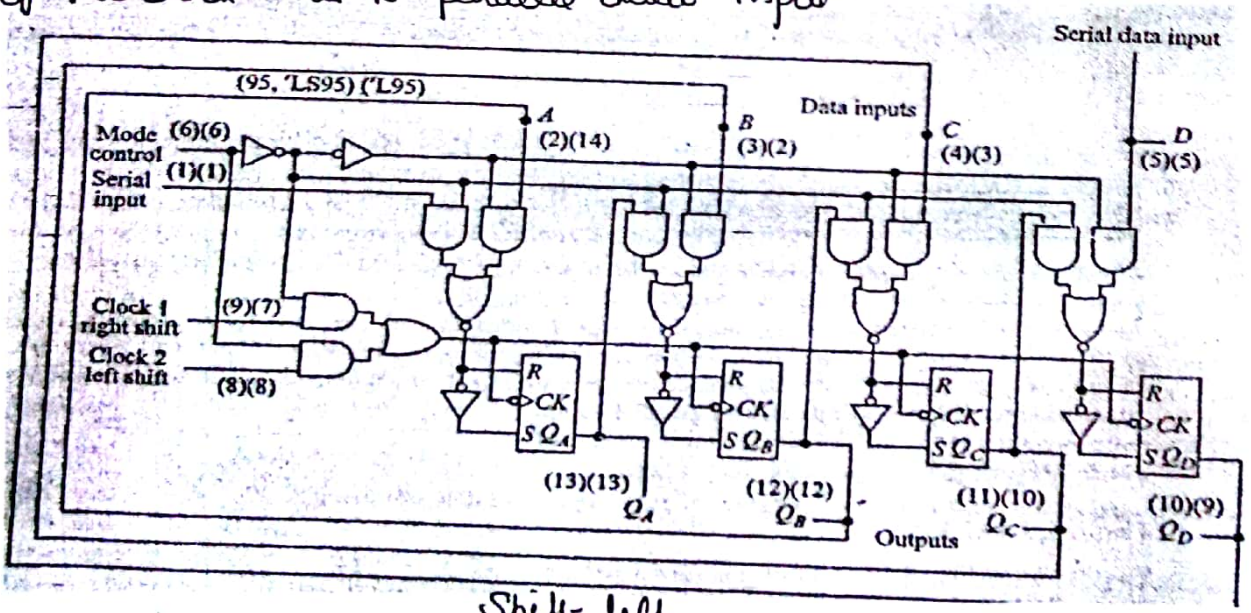




(b) Logic diagram

[fig 1]

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 is disabled The data al inputs A, B, C, D will then be loaded into the -ve transition of the clock this is parallel data input.



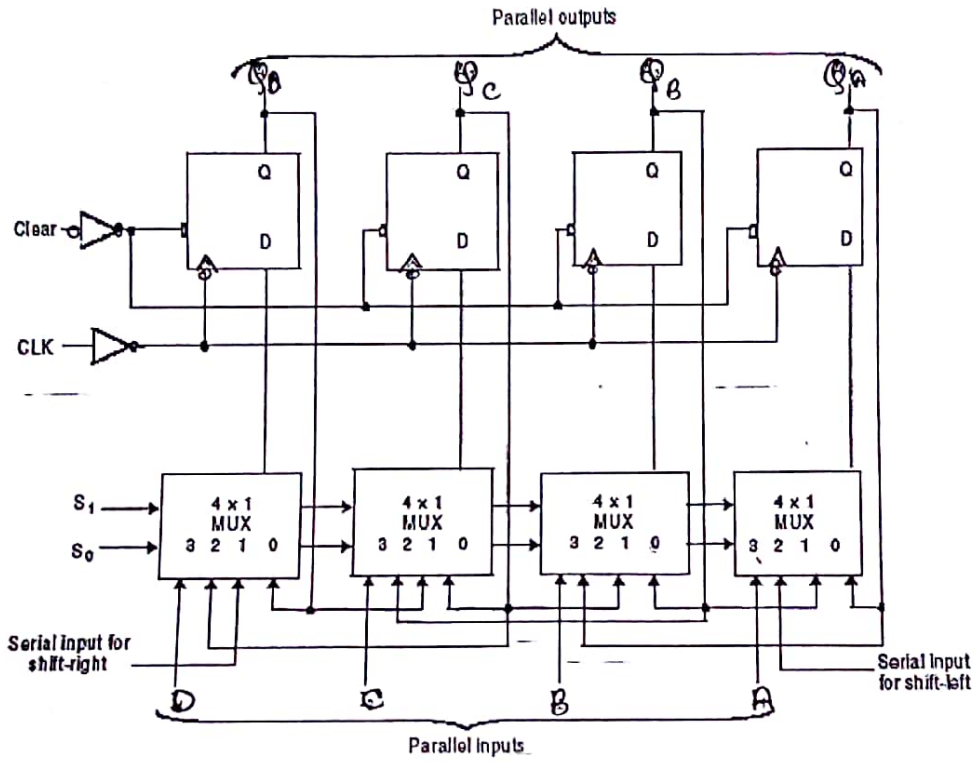
[fig 2]

Shift- left.

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 flip-flop Q_A is now al serial input; the data input to Q_B is Q_A and so down the line. On each clock NT a data bit is entered serially into the register at the 1st flip-flop Q_A and each stored data bit is shifted 1 flip-flop to the right- This is serial input of data and right- shift- operation [Fig 1]

In order to left shift input data must be connected as shown fig 2

Universal Shift Register: A universal shift register can perform all the 4 operations and is also bidirection in nature. (4 operation are SISO, SIPO, PISO, PIPO)



4-bit universal shift register.

74194 is a 4 bit universal shift register. A, B, C, D are four parallel inputs and Q_A, Q_B, Q_C, Q_D are corresponding parallel output. There are 2 seperate inputs for serial data for left and right shift. There are 2 mode control inputs which select the mode of operation for the universal shift register according to Table.

S_1	S_2		Next State $Q_{A,n+1}$	$Q_{B,n+1}$	$Q_{C,n+1}$	$Q_{D,n+1}$
0	0	Hold	$Q_{A,n}$	$Q_{B,n}$	$Q_{C,n}$	$Q_{D,n}$
0	1	Shift right	Din	$Q_{A,n}$	$Q_{B,n}$	$Q_{C,n}$
1	0	Shift left	$Q_{B,n}$	$Q_{C,n}$	$Q_{D,n}$	Din
1	1	Load	A	B	C	D

The input clear is active low and resets all the flip-flops asynchronously. Clock is positive edge triggered because of inversion.

Application Of Shift Registers:

Shift register can be used to count number of pulses entering into a system as ring counter / switch tail counter

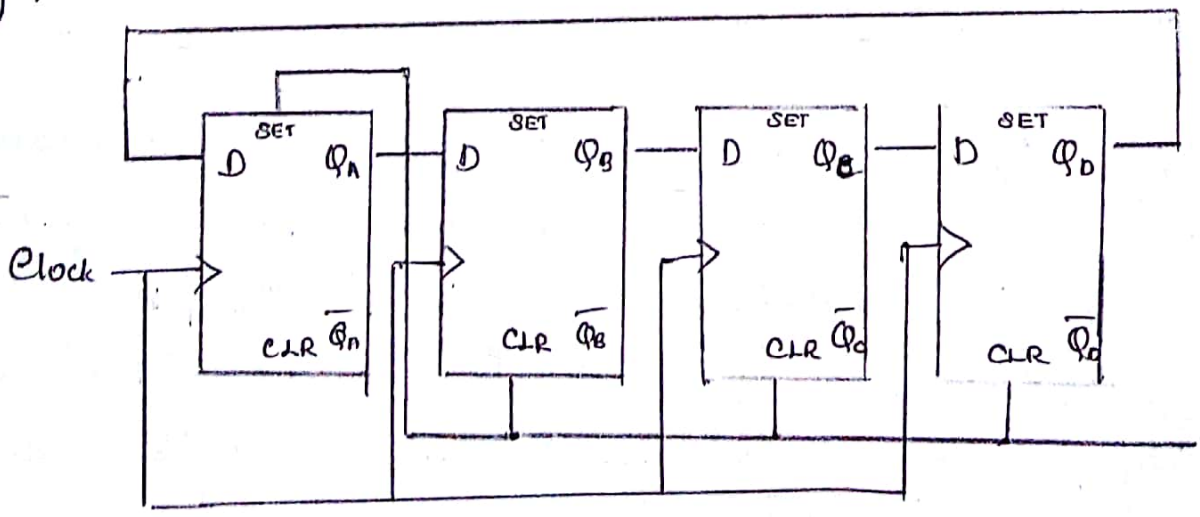
Shift register can generate a prescribed sequence repetitively or detect a particular sequence from data input.

It helps in reduction of hardware by converting parallel data fed to serial one. [eg: serial adder].

Ring Counter:

A ring counter is basically a circulating shift register in which the output of the most significant stage is fed back to the input of least significant stage.

The following is a 4 bit - ring counter constructed from D flip-flops. The output of each stage is shifted into the next stage on positive edge of a clock. If the CLEAR signal is high, all the flip-flop except the first one are reset to 0.

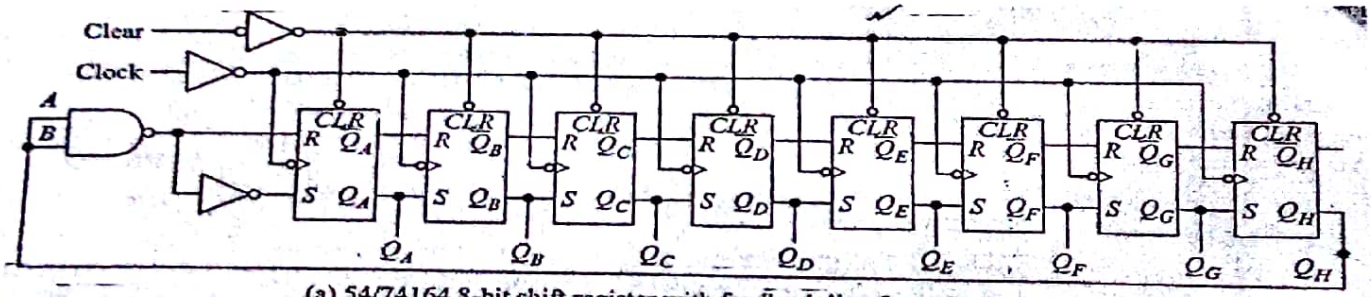


Advantage of ring counter over a binary counter is that it is self-decoding. i.e no extra decoding circuit is need to determine what state the counter is in.

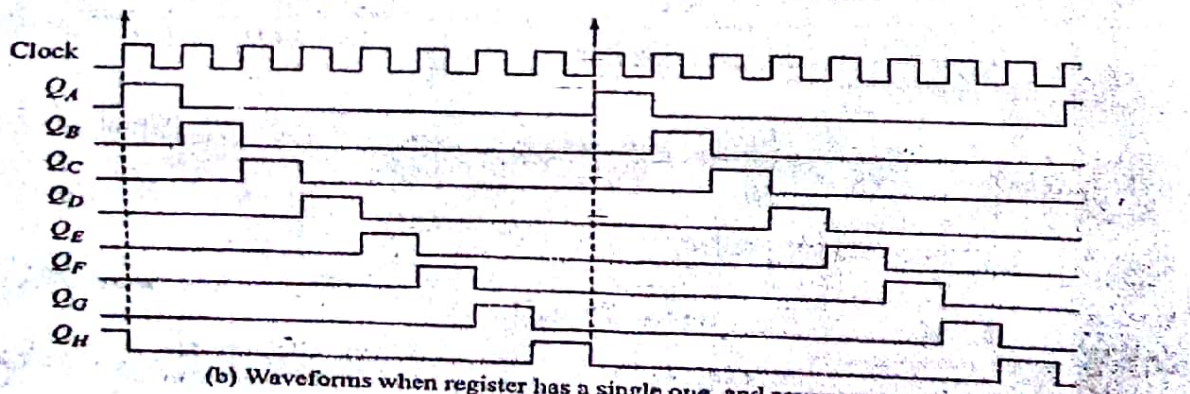
Since the counter (4 bit) has 4 distinct states, the

can be considered as a mod 4 counter. Only 4 of the max^m 16 states are used, making ring counters very inefficient:

54/74164 can be used as ring counter. If we connect the output of the last flip-flop Q_H back to the input of the 1st flip-flop, it will be converted into ring counter as show below fig.



(a) 54/74164 8-bit shift register with feedback line from Q_H to A-B



(b) Waveforms when register has a single one, and seven zeros

Suppose Q_A is high and all other flip-flops are low and then allow the clock to run. On first clock PT, the 1 in A will shift to B and A will be reset since the 0 in H will shift into A. All other flip-flops will still contain 0s. On 2nd clock PT, 1 in B will shift from B to C while B resets. Thus single 1 will shift down the register travelling from one flip-flop to the next flip-flop each time the clock goes high.

Consider the register as a tube full of ping-pong balls

seven white ones and one black one. The ping-pong balls simply circulate around the register in a clockwise direction, moving ahead one flip-flop with each clock P.T. This is referred as circulating register / a ring counter.

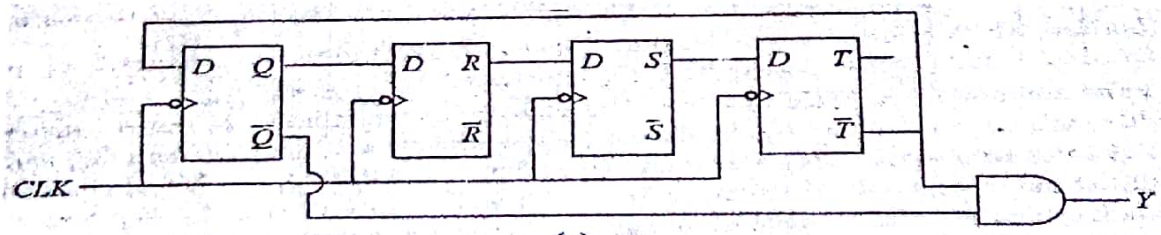
Clock Pulse	Q_A	Q_B	Q_C	Q_D
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	1	0	0	0

Truth table for 4 bit ring counter.

Repeats.

Switched tail Counter or Johnson Counter:

It is the variation of standard ring counters, with the inverted output of the last stage fed back to the input of the first stage



(a)

Clock	Serial in = T	Q	R	S	T	$Y = Q'T'$
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

(b)

(►) Fig. 9.22 (a) 4-bit switched tail counter, (b) Its state table

They are also known as twisted ring counters. An n stage Johnson counter yields a count sequence of length 2n so

It may be considered to be a mod-2n counter. The circuit shows a 4 bit Johnson counter. The state sequence for the counter is given in the state table

Maximum available states are not fully utilized (8 out of 16)

Both ring & Johnson counter must be initialized forcefully to a valid state.

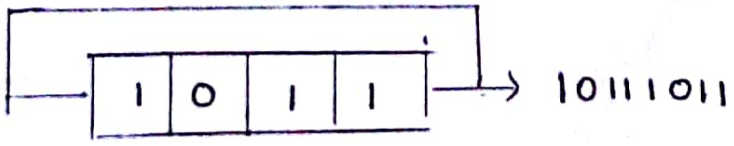
We can use decoding gates to buil

Sequence Generator and Sequence Detector

Sequence generator is useful in generating a sequence pattern repetitively. It may be the synchronizing bit pattern sent by a digital data transmitter or it may be a control word directing repetitive control task.

Sequence detector checks binary data stream and generate a signal when a particular sequence is detected.

The basic block diagram of a sequence generator is given below.



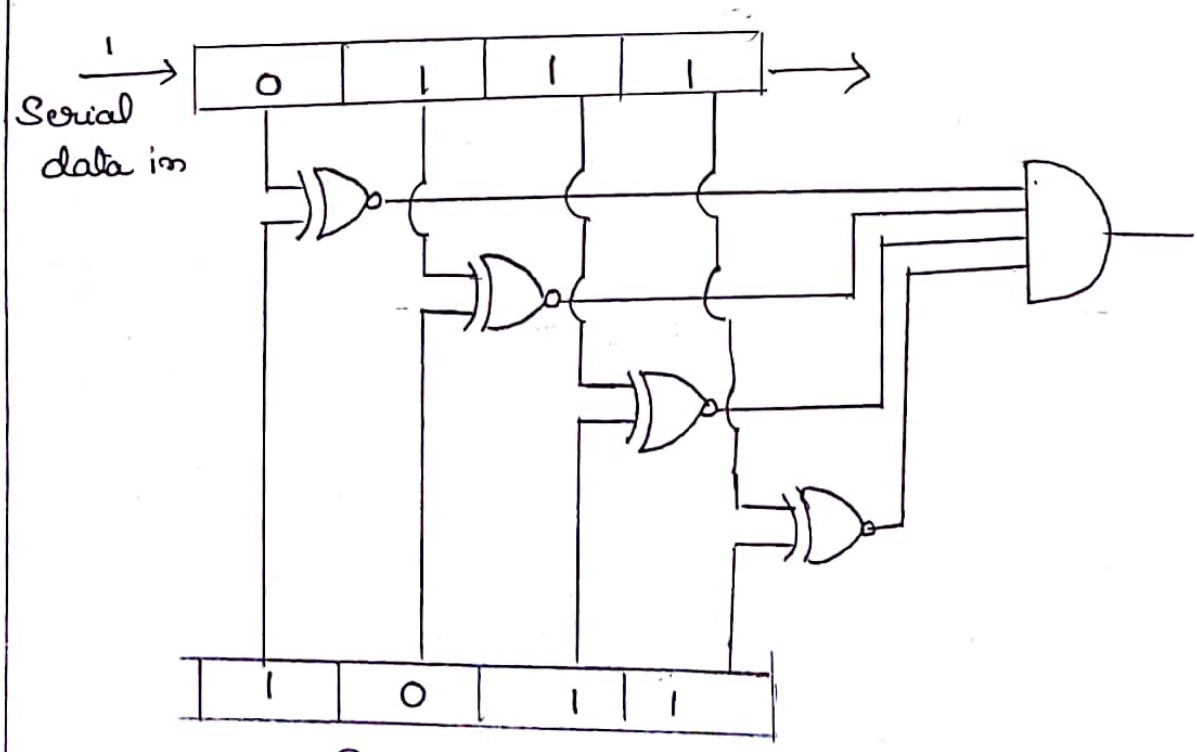
Here shift register is presented as pipe full of data and each flip-flop represents one compartment of it.

The leftmost flip-flop is connected to serial data in and

rightmost provides serial data out. The clock is implied and data transfer takes only when a clock bigger arrives. The shift register is connected like a ring counter and with clock trigger binary word stored in the register comes all serially and does not get lost as it is fed back as serial in to fill the register all over again. Above sequence generator generate binary word 1011.

If we want to generate n bit long sequence, we need to store the sequence in an n bit shift register.

The block diagram of 4bit sequence detector is shown below:

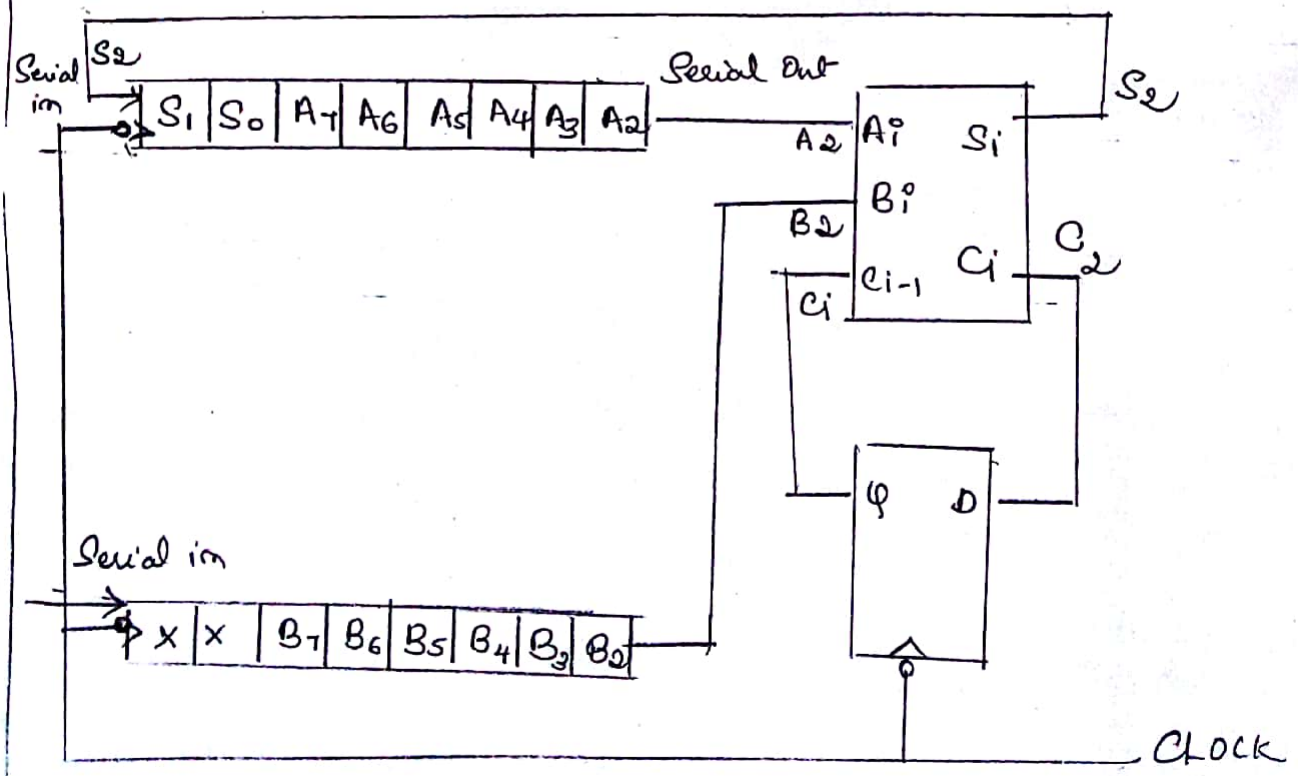


Sequence to be detected

Eg: 0111 should be matched with 1011. The first bits are unmatched and corresponding Ex-NOR o/p is 1 & also final o/p Y=1

One register is used to store the binary word we want to detect from the data stream. Input data stream enters one more 8 register as serial data in and leaves as serial out. At every clocking instant, bitwise comparisions of these 2 registers are done through Ex-NOR gate. When both the inputs are low & or high output of ExNOR will be high. i.e when both are equal. Final output is taken from A input AND gate which becomes 1 when all inputs are 1. i.e all bits are matched. If we want to change the binary word to be detected we can load that in bottom register Hence it is called programmable sequence detector. For fixed sequence detector, we can reduce h/w cost removing bottom register and directly connecting to +Vcc & GND depending 1 or 0 bit.

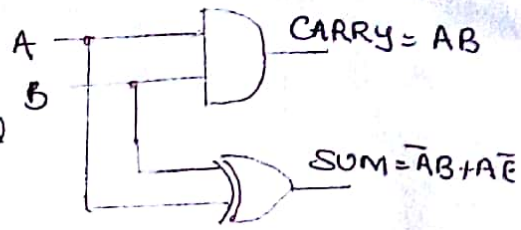
Serial Adder :-



Note:

Half Adder:

When we add binary number, we start with LSB column. Fig shows how to build a half adder.



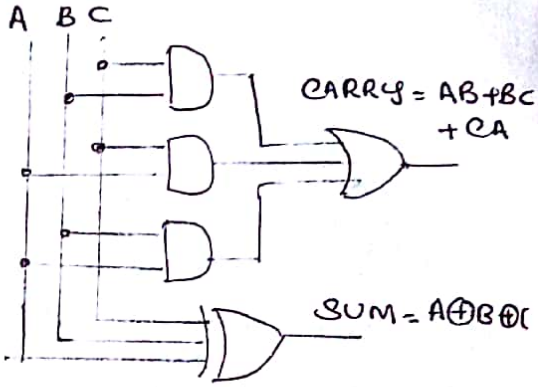
OUTPUT of XOR gate is SUM and AND gate is CARRY.

Full Adder:

Can add 3 bits at a time

Fig shows the circuit of Full adder.

CARRY = $AB + BC + AC$, SUM = $A \oplus B \oplus C$



General representation:

$C_i = A_i B_i + (A_i + B_i) C_{i-1}$, Sum = $A_i \oplus B_i \oplus C_i$

Two 8-bit numbers, to be added ($A_7 A_6 A_5 \cdots A_0$ and $B_7 B_6 B_5 \cdots B_0$) are loaded in 2 8bit shift registers A and B. LSB of each number appears in the rightmost position in 2 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.

Serial addition process:

At 1st clock cycle, the LSBs of 2 numbers (A_0 & B_0) appear at serial out of respective registers and added by FA and generate sum S_0 and C_0. S_0 is available at serial input of register A. and C_0 at input of D flip-flop

At NT of clock register shift its content to right by one unit. S_0 becomes MSB of A and C_0 appears at output of D flip-flop.

In second clock cycle FA is fed by 2nd bits (A_1 & B_1) of 2 numbers and previous carry C_0, S_1 and C_1 are generated and made available at serial data in of A register & input of D flip flop respectively. At NT of clock S_1 becomes MSB of A and S_0 occupies next position. At 3rd clock cycle A_2 & B_2 appear at data input of FA & C_1 at carry input and S_2 & C_2 are generated. This process goes on and is stopped by inhibiting the clock after 8 clock cycles. Now shift register A store sum S_7 is MSB and leftmost position and S_6 in rightmost position and Final carry at D flip-flop output.

Limitation: Final addition result is delayed by 8 clock cycles.

Register Implementation in HDL:

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

module shiftrightreg (D, Clock, T);
 input Clock D;
 output T;
 reg T;
 reg Q, R, S; // since are used inside always block
 always @ (negedge Clock)
```

```
begin
    Q <= D;
    R <= Q;
    S <= R;
    T <= S;
end
endmodule

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

module switchtailcount (Clock, reset, cnt);
input reset, Clock;
output [3:0] cnt;
reg [3:0] cnt = 4'b0;
always @ (posedge clock)
begin
    if(reset) cnt <= 4'b0;
    else
        begin cnt[3] <= ~cnt[0];
              cnt[2] <= cnt[3];
              cnt[1] <= cnt[2];
              cnt[0] <= cnt[1];
        end
end
endmodule
```

Design an 8 bit sequence generator that generates the sequence 11000100 repetitively using shift register. We load 8 bit ring counter as shown below fig with the given sequence and at the output the sequence will be generated repetitively.