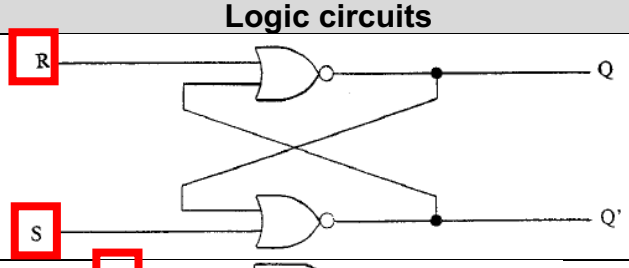
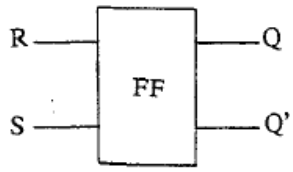
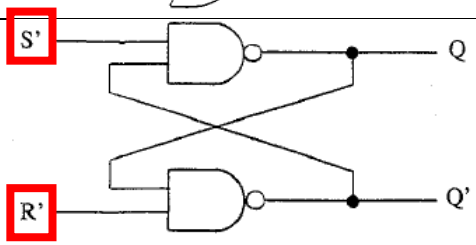
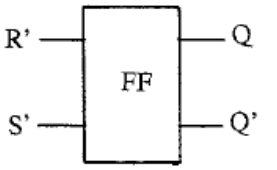
Sequential Logic Circuits

With contrary to combinational logic circuits, the output of sequential logic circuits not only depends on the present value of inputs, but also the past history of the system. Therefore, require a memory device. The most common memory device are flip flops.

Flip Flops

1. RS Flip Flop (RS FF)

Two ways to build a RS flip Flop:

Configurations	Logic circuits	Symbol
2 NOR gates		
2 NAND gates		

**Note: logical operators in diagram: $A' = \bar{A} = \neg A$

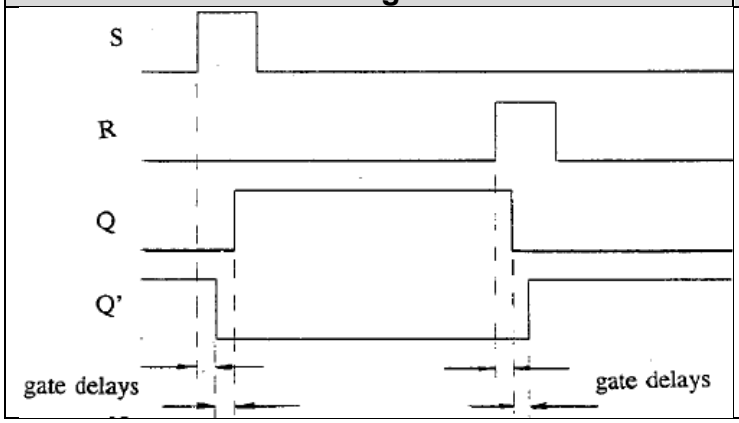
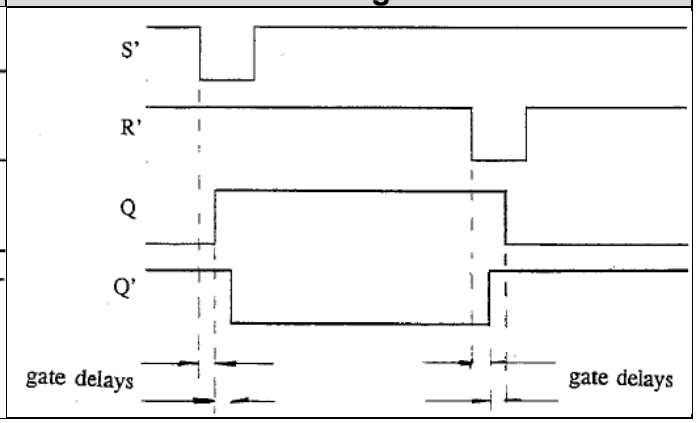
The truth tables for the two configurations above:

2 NOR gates: Inputs		2 NAND gates: Inputs		$Q(t)$	$Q(t+1)$	Remarks
$S(t)$	$R(t)$	$\overline{S(t)}$	$\overline{R(t)}$			
0	0	1	1	0	0	Unchanged
				1	1	
0	1	1	0	0	0	Reset
				1	0	
1	0	0	1	0	1	Set
				1	1	
1	1	0	0	0		Not Allowed
				1		

Which means that the two configurations will give the same results but the inputs of the NAND gates ones will be the complements of those inputs in the NOR ones:

2 NOR gates	2 NAND gates
$S(t) \neq R(t) \rightarrow Q(t+1) = S(t)$ $S(t) = R(t) = \begin{cases} 0 \rightarrow Q(t+1) = Q(t) \\ 1 \rightarrow NOT\ ALLOWED \end{cases}$	$\overline{S(t)} \neq \overline{R(t)} \rightarrow Q(t+1) = \overline{R(t)} = S(t)$ $\overline{S(t)} = \overline{R(t)} = \begin{cases} 0 \rightarrow NOT\ ALLOWED \\ 1 \rightarrow Q(t+1) = Q(t) \end{cases}$

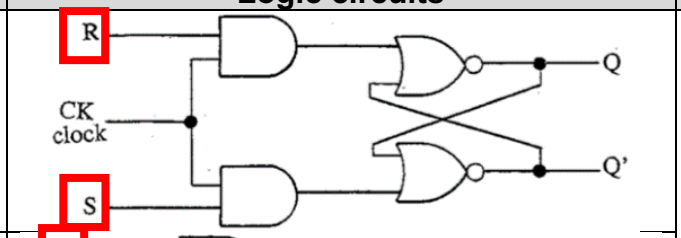
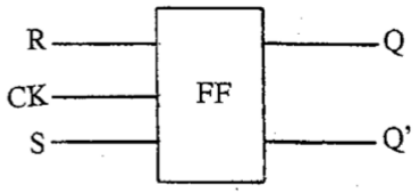
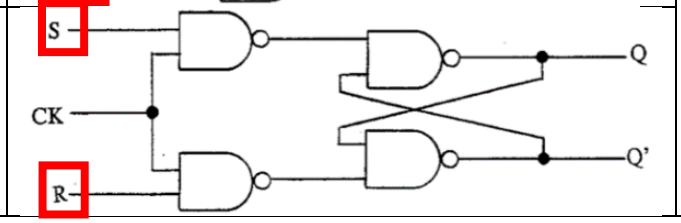
Timing diagrams:

2 NOR gates	2 NAND gates
	

- Logical gates take time to change state, the output Q and $\bar{Q}$ do not change instantly with inputs. Also, there exists a minimum input pulse width to ensure the flip flop (FF) function properly.
- Inputs:
  - Input S → Set: Q = 1 after S input is activated
  - Input R → Reset: Q = 0 after R input is activated

- S and R cannot be at logic 1 at the same time
  - If S = R = 1 (0 for NAND type), then Q = $\bar{Q}$ = 0 (1 for NAND type). This violate the definition of FF because Q and $\bar{Q}$ should be complements.
  - Furthermore, if S = R = 1 → 0 (0 → 1 for NAND type), then both outputs will try to = 1 (0 for NAND type). Since that is impossible, the output will oscillate and therefore unpredictable.
- The state of outputs remembers which of the inputs was last at logic 1.
- The NAND type inputs are low level active, because the output $Q(t+1) = 1$ when the Set input $\overline{S(t)} = 0$, and $Q(t+1) = 0$ when the Reset input $\overline{R(t)} = 0$.

2. Clocked RS Flip Flop

Configurations	Logic circuits	Symbol
2 NOR gates		
2 NAND gates		

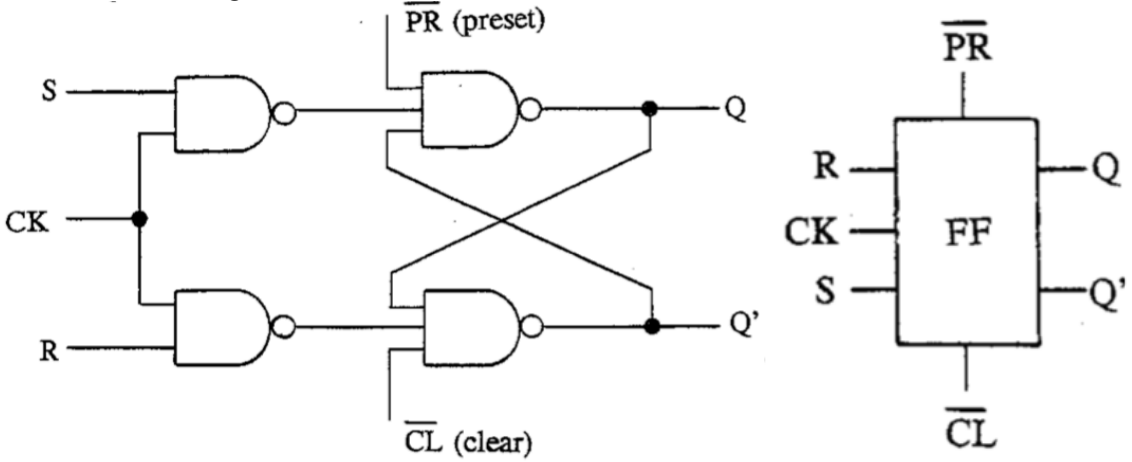
This type of FF will change states only at periodically spaced points in time, which can be achieved by synchronising all the state changes with a train of clock pulses:

Truth table for clocked RS FF, but in this case, $S(t)$, $R(t)$, $Q(t)$ refers to the values during the clock pulse, and $Q(t+1)$ is the value after the clock pulse has ended. Also, S and R cannot change state during a clock pulse. So the S and R signal must start before the Clock signal.

$S(t)$	$R(t)$	CK	$Q(t+1)$
0 **OR** 1	0 **OR** 1	0	Previous State: $Q(t)$
0	0	1	
0	1	1	0
1	0	1	1
1	1	1	Not allowed

Clocked RS FF with pre-set and clear

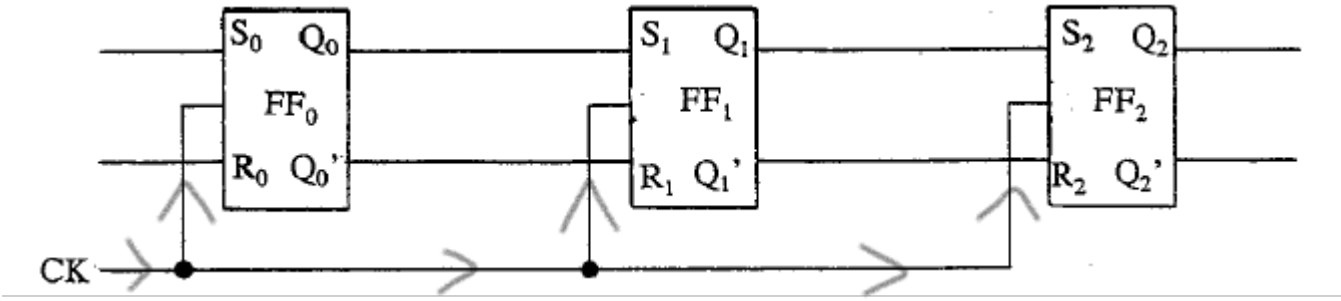
This type of FF can pre-set (set Q = 1) or clear (reset Q = 0) at any time point without waiting for the clock pulse. This can be achieved by providing a low level active pre-set input: $\overline{PR} = 0$ and clear input $\overline{CL} = 0$ in the NAND gate for Q and $\bar{Q}$:



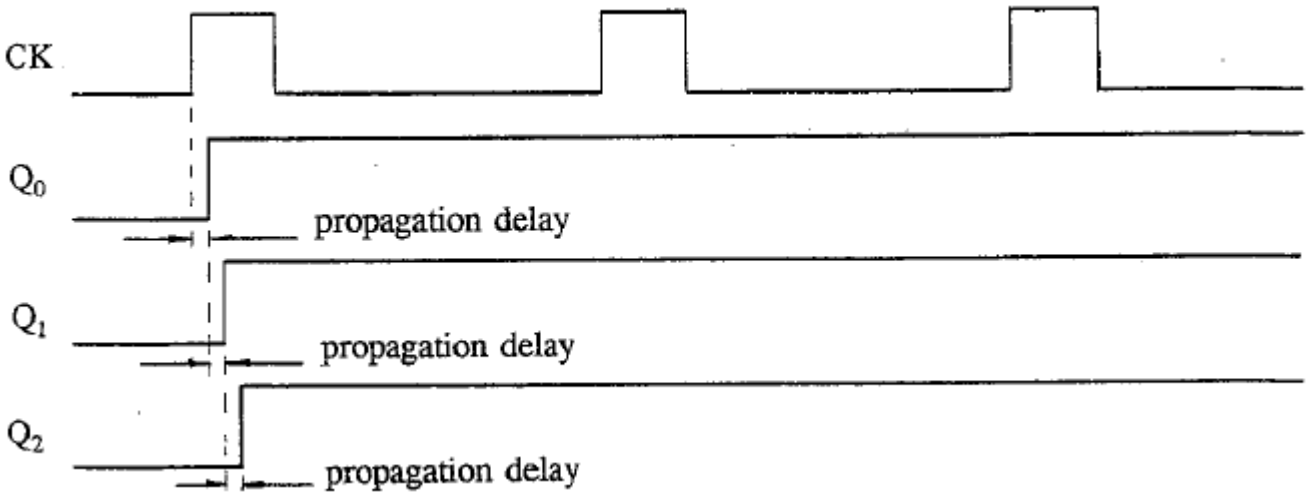
For the NOR type, $PR = CL = 1$

Race Problem

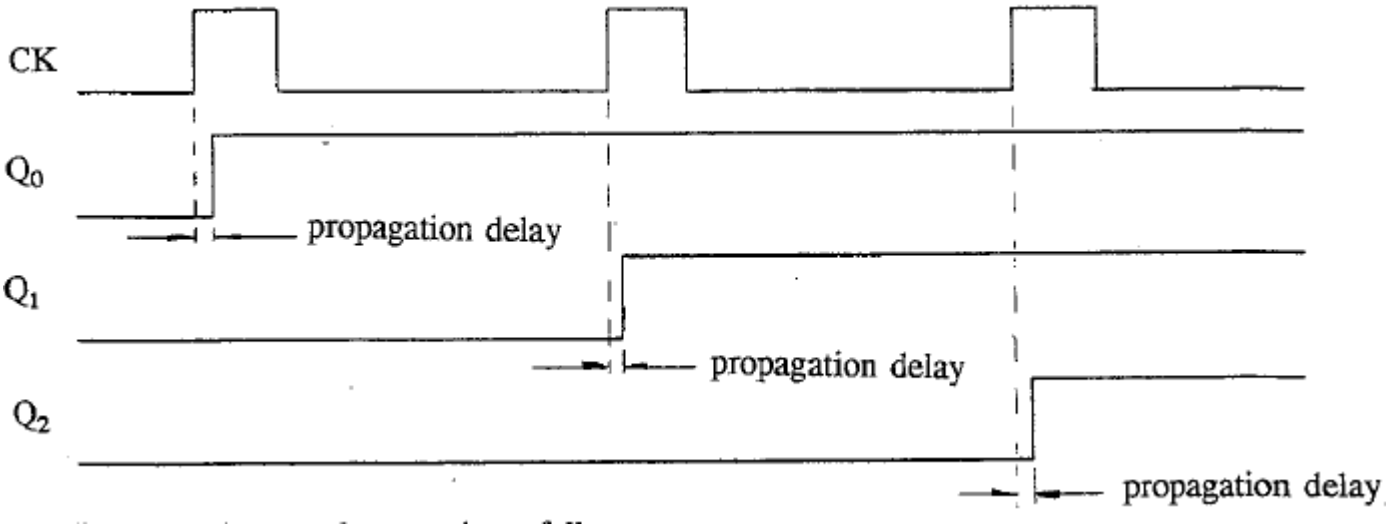
Consider the following circuit:



As long as the Clock pulse is long enough, the FFs will readily process their inputs S_X and R_X with their own propagation delay as soon as the inputs are ready, because when the Clock is active, it turns on all the FFs since all FFs receive logical signal 1. So, within one clock pulse, more than 1 state change have occurred. This happens because FF can change states at any time during the clock pulse. This is illustrated by the following time diagram:

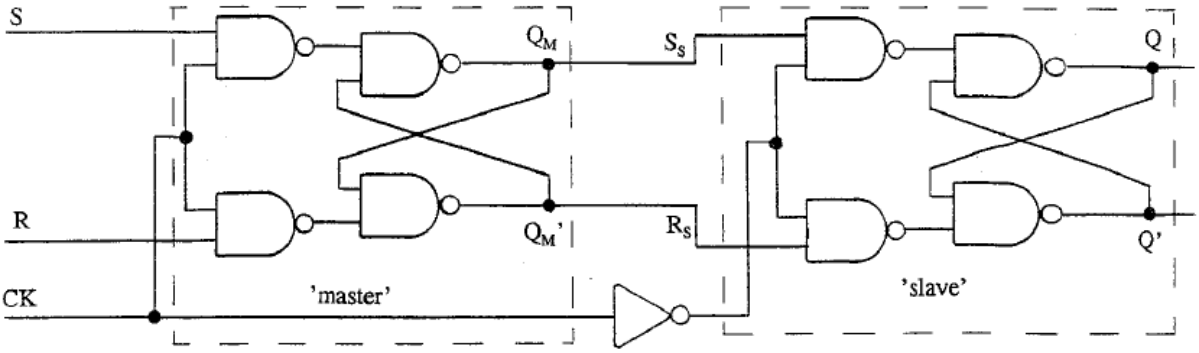


But when we have a circuit above, we rather want an output like this:



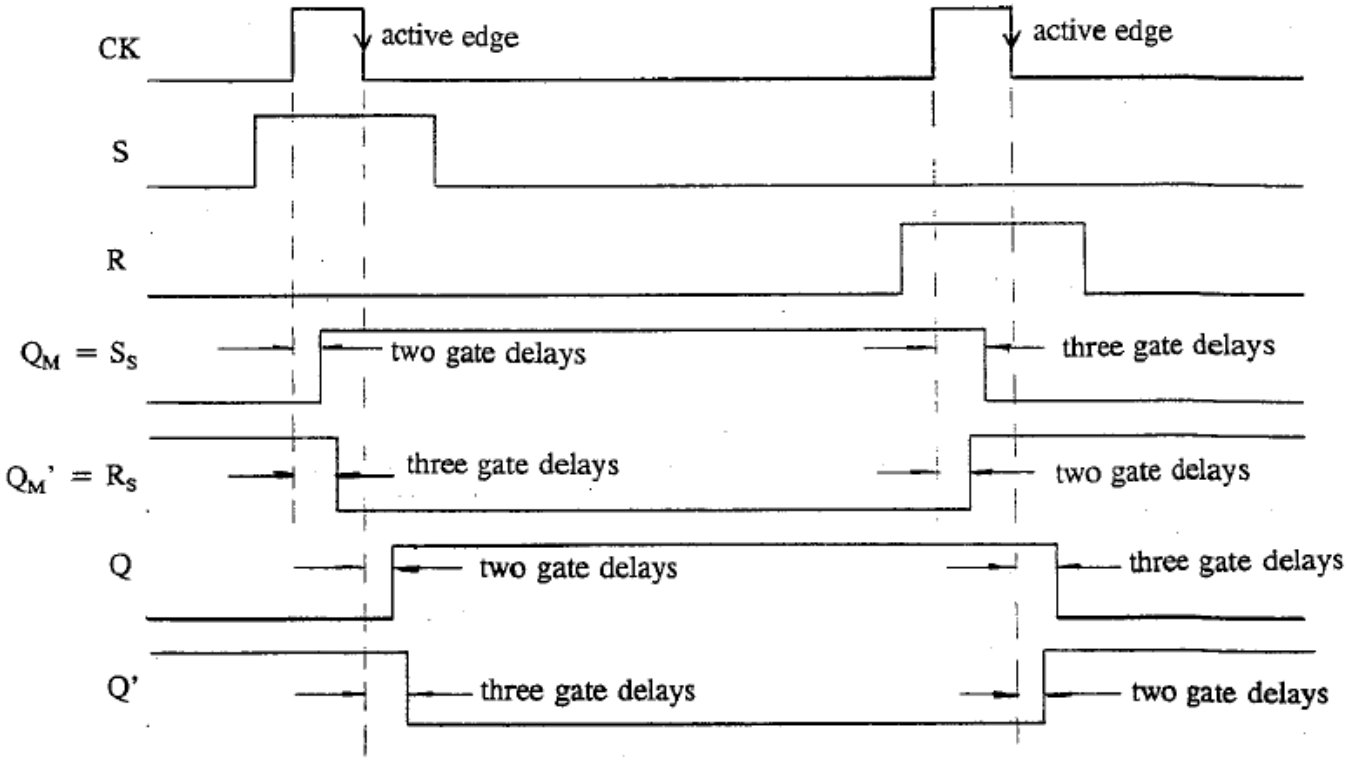
Because we want state changes at periodically spaced intervals of time, and only
1 state change/clock pulse. Another example is the input of the circuit arrives at different time.

3. Master Slave Flip Flop (MS FF)
   This type of FF is to tangle the Race problem, a NOT gate is added between two FFs. So, state changes can only occur at edges of the clock pulses:



Falling edge type MS FF

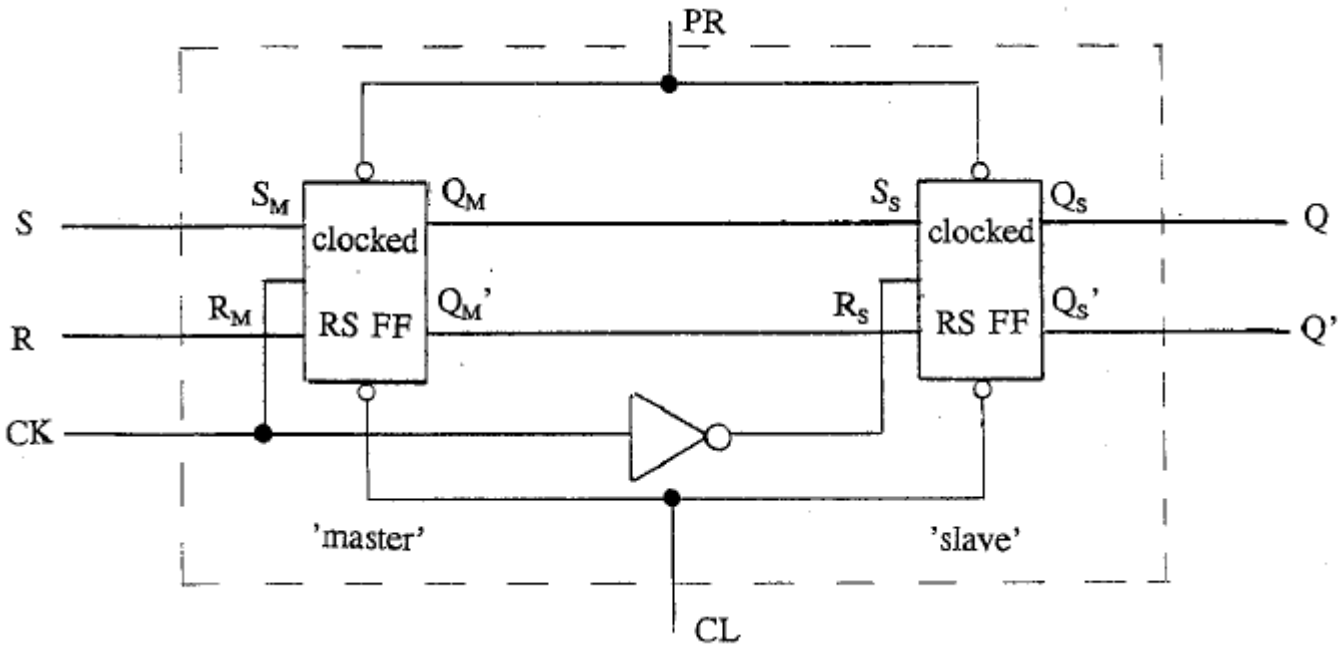
The NOT gate feeds the complement of the Clock signal to the slave. Therefore, when the Clock signal change from 0 to 1, the master FF will transform the inputs since it receives a logical signal 1 from the clock terminal, meanwhile the slave will not do anything because the clock signal received is a logical signal 0.
Then, as the clock signal change from 1 to 0, the master FF will not do anything and the slave FF will now transform the input signals because it receives a logical signal 1 from the clock terminal.



With this, the racing problem in Clocked RS FF is eliminated.

The truth table for MS FF is the same as that for RS FF, but in this case, $S(t)$, $R(t)$, $Q(t)$ refers to the values JUST before the active clock (rising / falling) edge, and $Q(t+1)$ is the value after that edge.

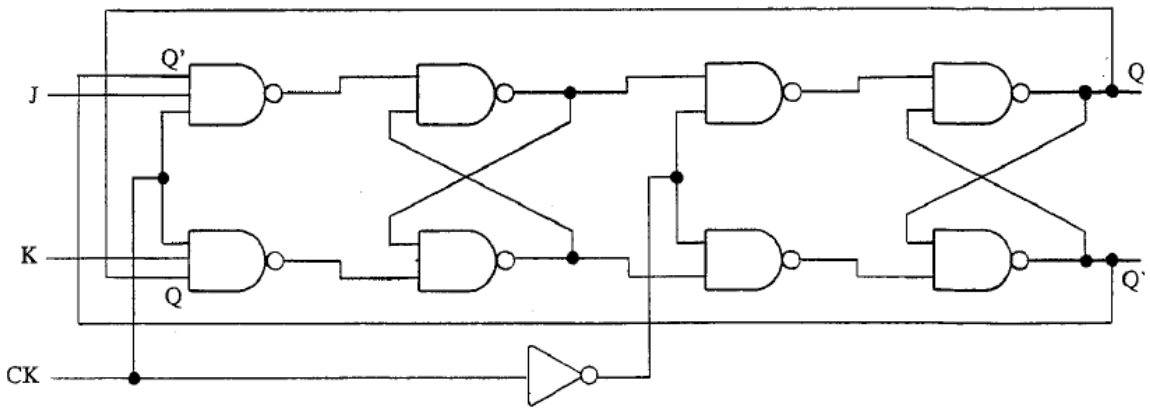
There also exist MS FF with pre-set and clear:



So, it can also pre-set (set Q = 1) or clear (reset Q = 0) without waiting for the clock pulse.

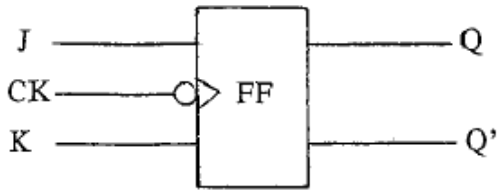
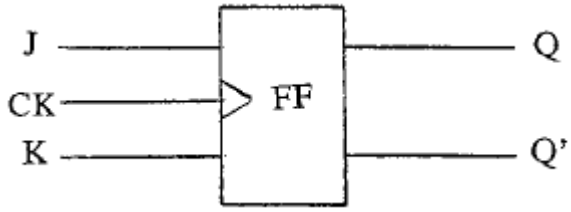
4. JK Flip Flop (JK FF)

This type of FF unlocks the restriction of RS FF that the input S and R are not allowed at logical signal 1 at the same time. This is achieved by modifying the MS RS FF such that the output Q and $\bar{Q}$ are fed back to the inputs:



Falling edge type JK FF

Symbols for JK FF:

Falling edge type	Rising edge type
	

The truth table for a JK FF is:

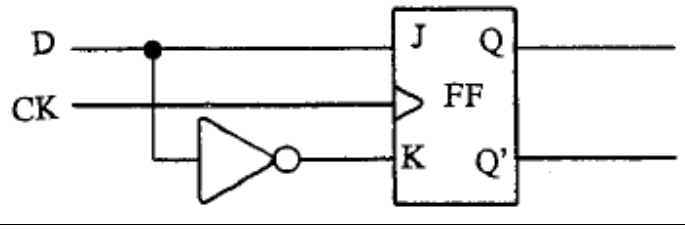
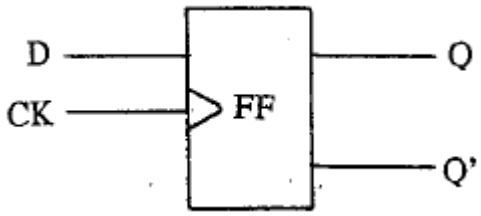
$J(t)$	$K(t)$	$Q(t)$	$Q(t+1)$	Remarks
0	0	0	0	Unchanged
		1	1	
0	1	0	0	Reset
		1	0	
1	0	0	1	Set
		1	1	
1	1	0	1	Toggle
		1	0	

Which means that:

$$J(t) \neq K(t) \rightarrow Q(t+1) = J(t)$$
$$J(t) = K(t) = \begin{cases} 0 \rightarrow Q(t+1) = Q(t) \\ 1 \rightarrow Q(t+1) = \overline{Q(t)} \end{cases}$$

In a JK FF, when both inputs are at logical signal 1, the output will change to the complements of the previous logical signal (toggle) at each clock pulse.

5. D Flip Flop (Delay FF)
   This type of FF accepts any input given and makes it available at the output after 1 clock pulse delay. Constructed using a JK FF:

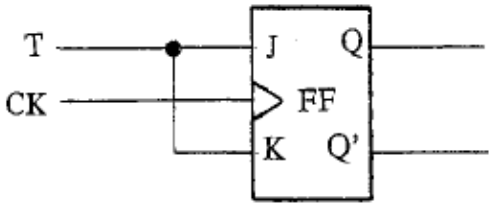
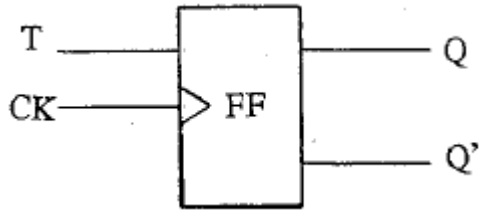
Logical circuit (Rising Edge)	Symbol
	

Since the output is the same as the input logically, the truth table is therefore:

$D(t)$	$Q(t+1)$
0	0
1	1

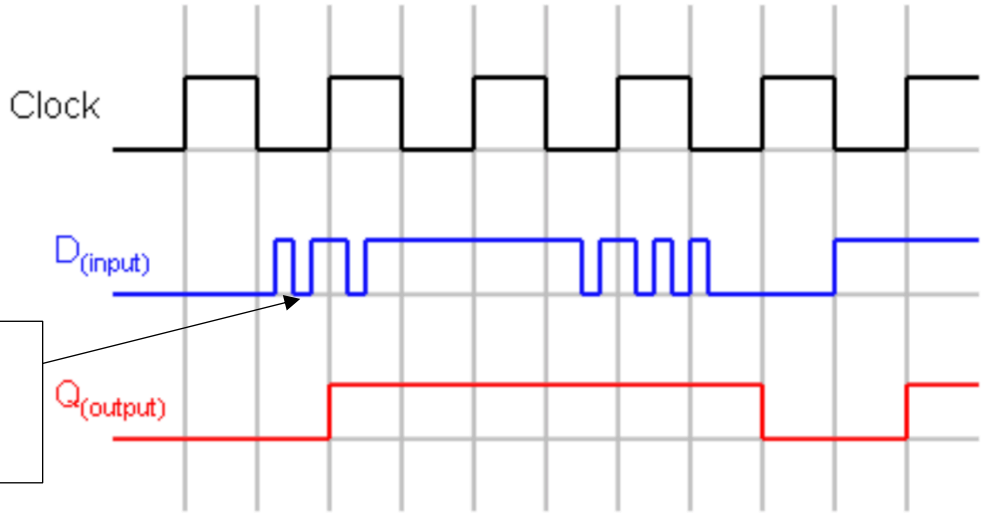
We can see that $Q(t+1)$ memorize $D(t)$.

6. T Flip Flop (Toggle FF)
   This type if FF will have its output toggle if the input T = 1, and remain unchanged if T = 0. In toggle mode, the output toggle once for each arrival of an active clock edge. Constructed using a JK FF:

Logical circuit (Rising Edge)	Symbol
	

The truth table is therefore:

$T(t)$	$Q(t+1)$	Remarks
0	$Q(t)$	No change
1	$\overline{Q(t)}$	Toggle



Only the signal just before the active edge matters.

Shift Registers

A register is a set of Flip Flops (FF) used to:

1. Store binary data in digital systems, until it is called for a later time. (D-FF)
2. Data transmission
3. Arithmetic manipulations

- **Data transmission**
  Data stored in one register can transfer to another register through a data line.

  A bit is the smallest data size:
  - 8 bits → 1 byte
  - 2 bytes → 1 word

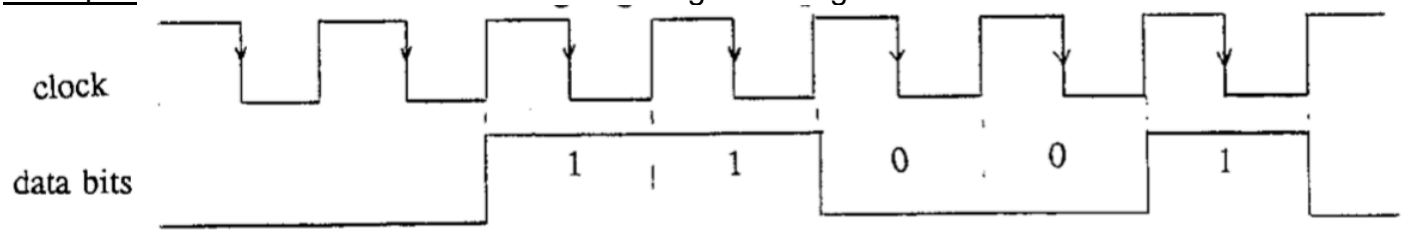
  Parity check:
  - Even parity: The total number of '1's in a binary number is even.
  - Odd parity: The total number of '1's in a binary number is odd.

  ASCII code only have 7 bits, so the 8th bit is used make up the parity of the binary number.
  Example:
  For "A", the binary number is $(100\ 0001)_{BIN}$:
  In odd parity circuit, "A" should be: 1100 0001.
  In even parity circuit, "A" should be: 0100 0001.

  Registers sued for data transmission have some / all of the following capabilities:

Serial	Parallel
• Accept data coming on a data line. • Output data on a data line, each bit in succession, synchronised with the system clock.	• Accept parallel data: at the instance of a clock pulse, all the data bits of the data word get loaded simultaneously into the register. • Output data in parallel form.

  Example: Serial data transmission of 10011 using Least Significant Bit first:

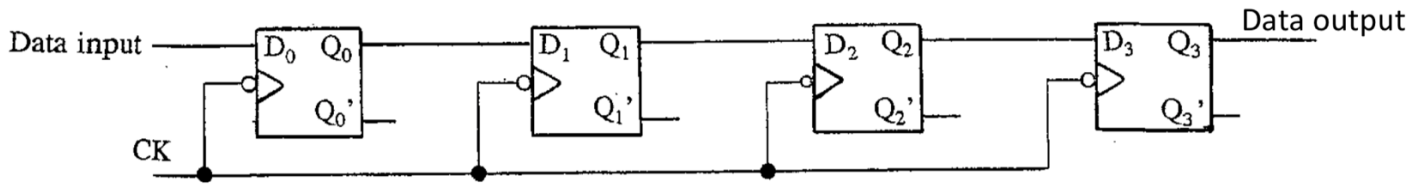


- **Binary multiplication and division**
  Multiplication:
  When a binary number is multiplied by 2, all the bits will shift to the left and the empty space created on the right is filled by a '0'

  Division:
  When a binary number is divided by 2, all the bits will shift to the right. Any remainder will be lost.

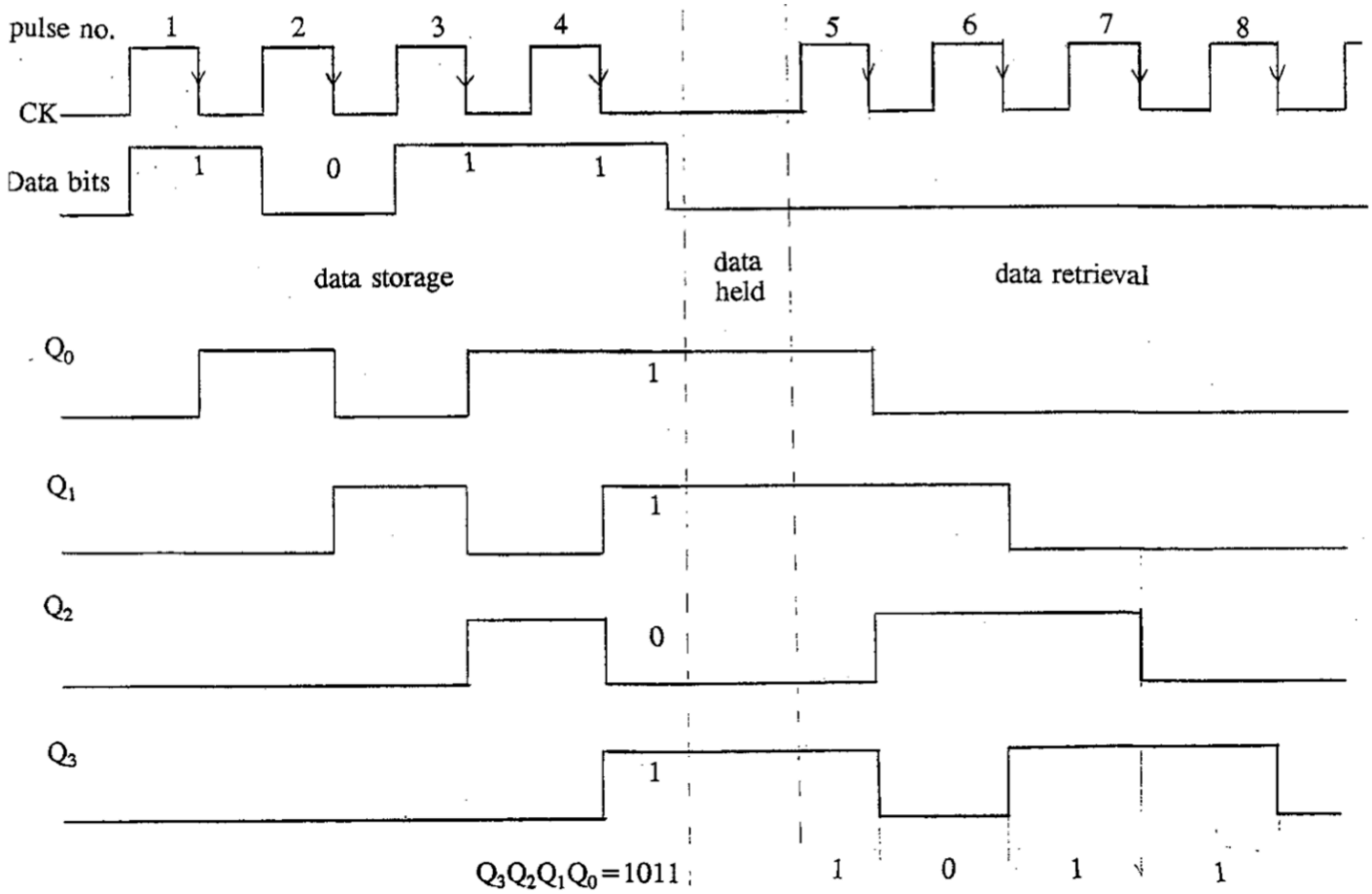
  Shift registers are capable to perform the above tasks.

- **Serial in serial out shift register**
  Four D-FF are connected as below to construct a 4 bit serial in serial out shift register:

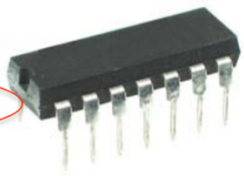


Function:

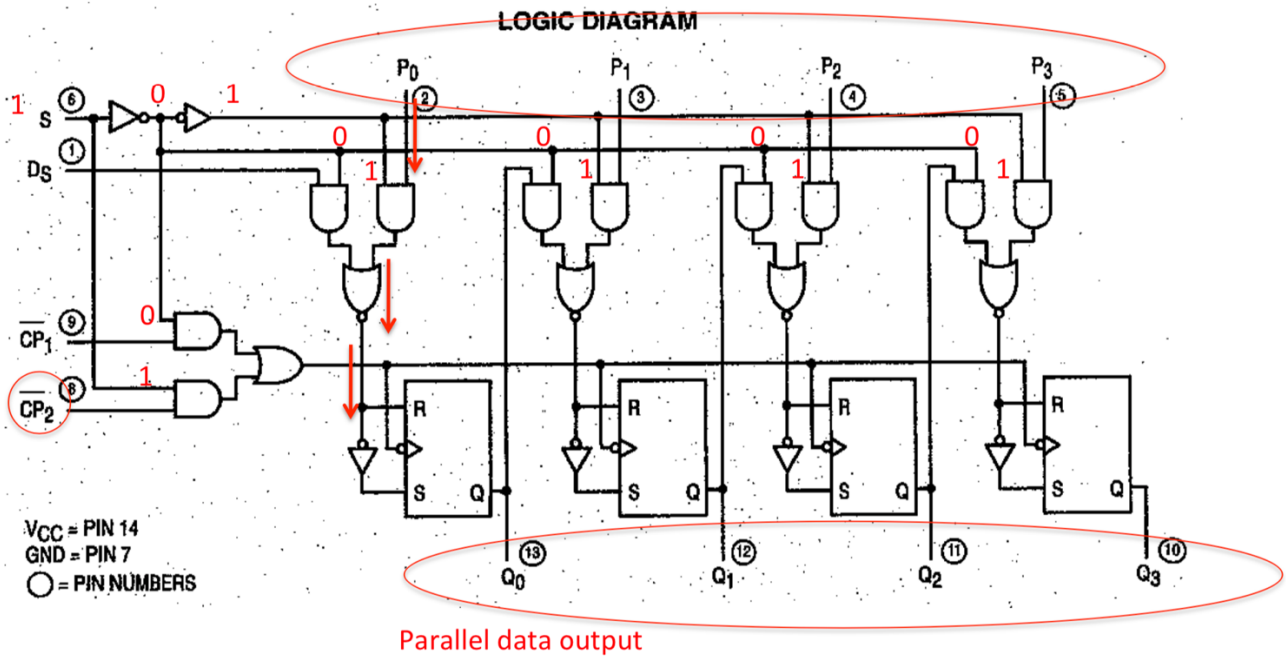
1. Read the serial data input into 4-bit register (4 clock cycles)
2. Store the 4-bit data in the register for a certain time (no clock cycles)
3. When required, send the stored 4-bit data at the Data Output (Q3) one by one (4 clock cycles)



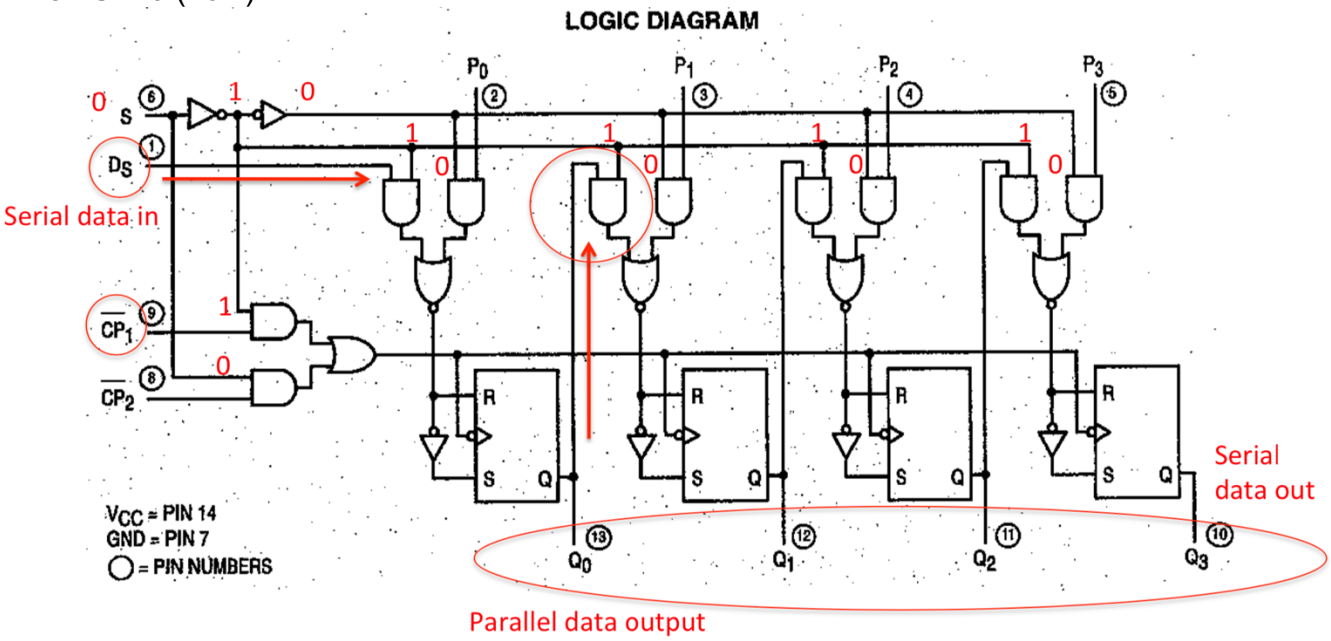
- **More versatile shift register**
  1. SN54/74LS95B 4 bit shift register



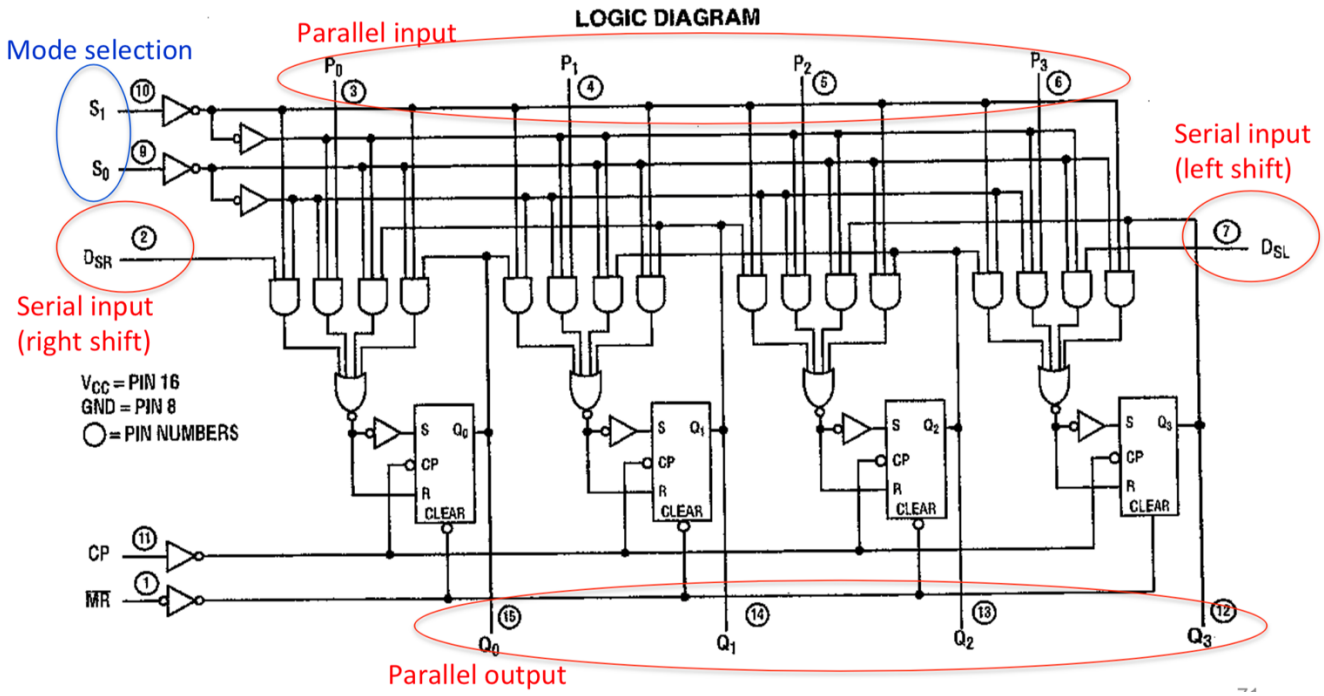
When S = 1 (HIGH):



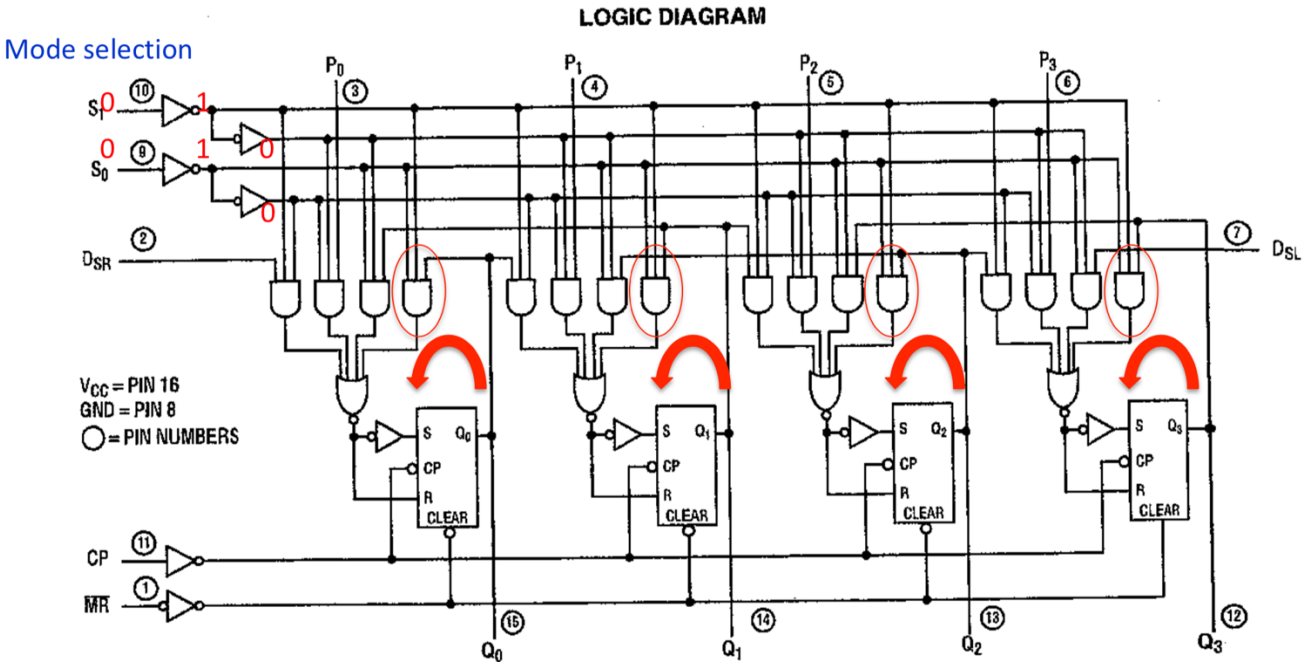
When S = 0 (Low):


LOGIC DIAGRAM
P0
P1
P2
P3
S
DS
Serial data in
CP1
CP2
VCC = PIN 14
GND = PIN 7
= PIN NUMBERS
R
S
Q
Q0
Q1
Q2
Q3
Serial data out
Parallel data output

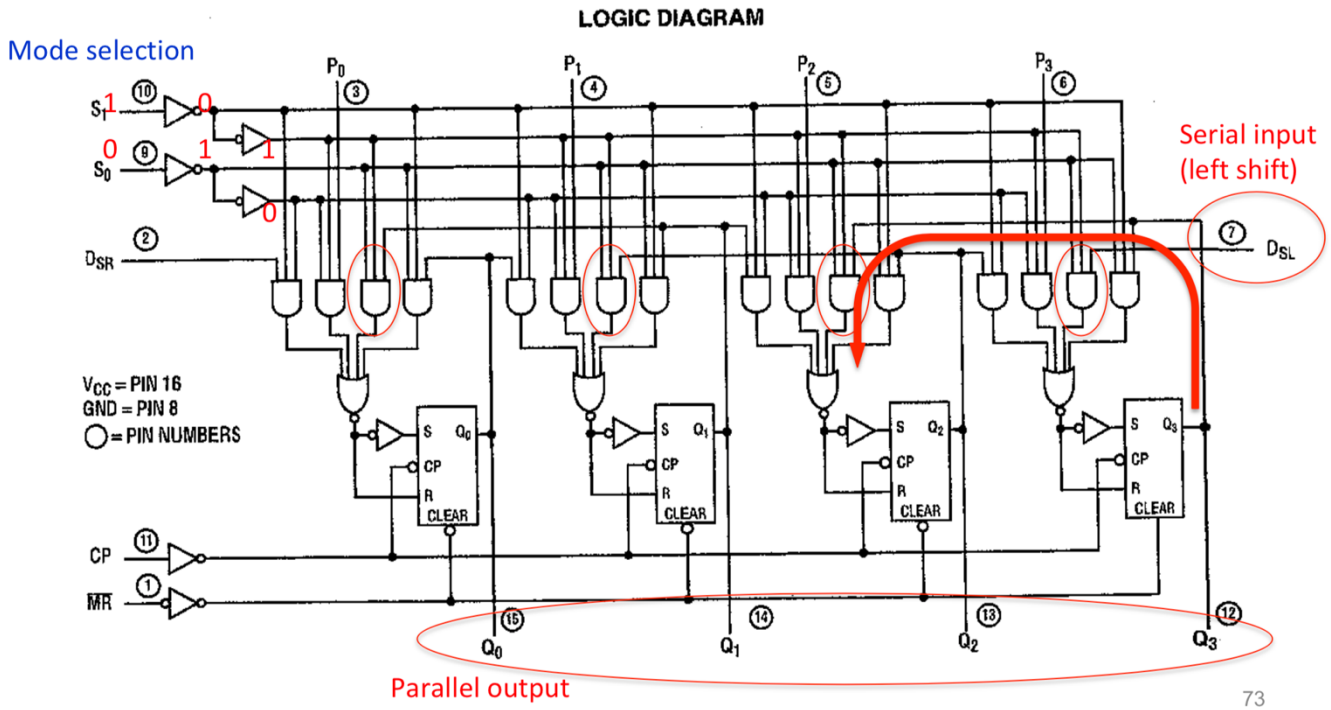
2. SN74LS194A 4-bit bidirectional universal shift register



When $S_1 = S_0 = 0$ → Hold (memory):

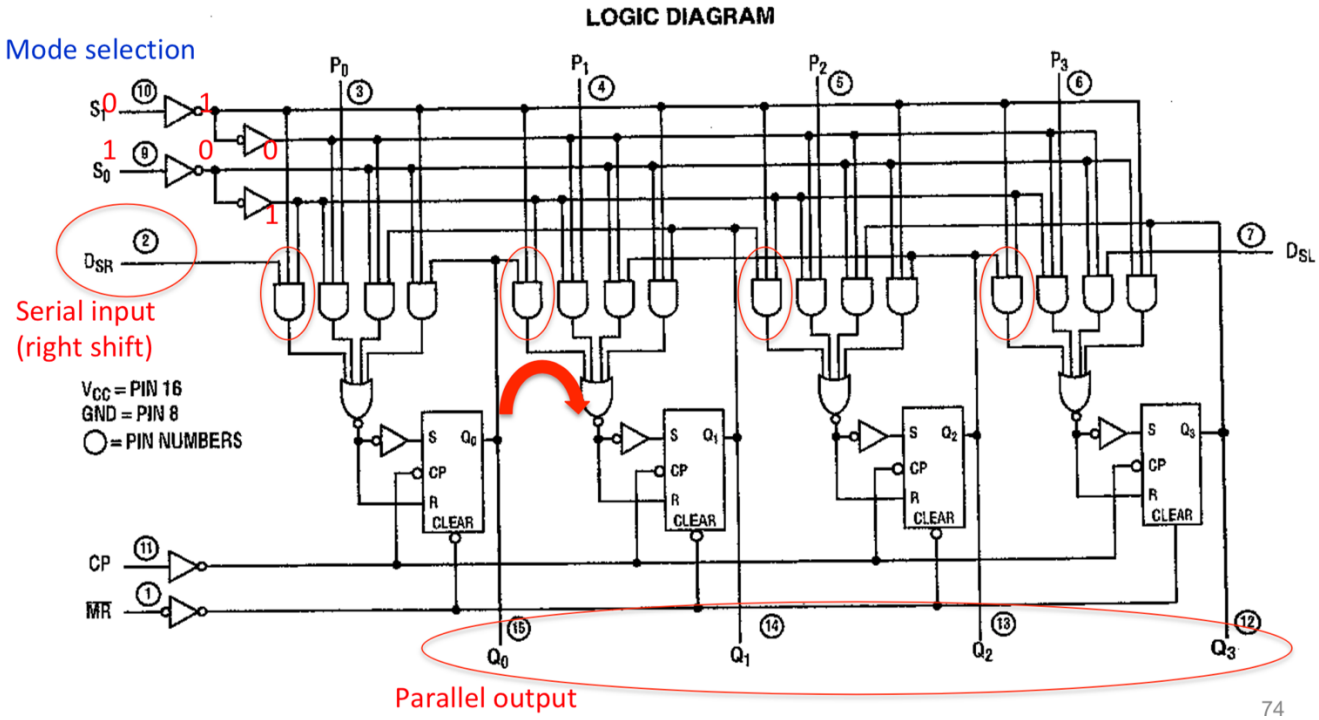


When $S_1 = 1$, $S_0 = 0$ → Serial in parallel out (Left shift):



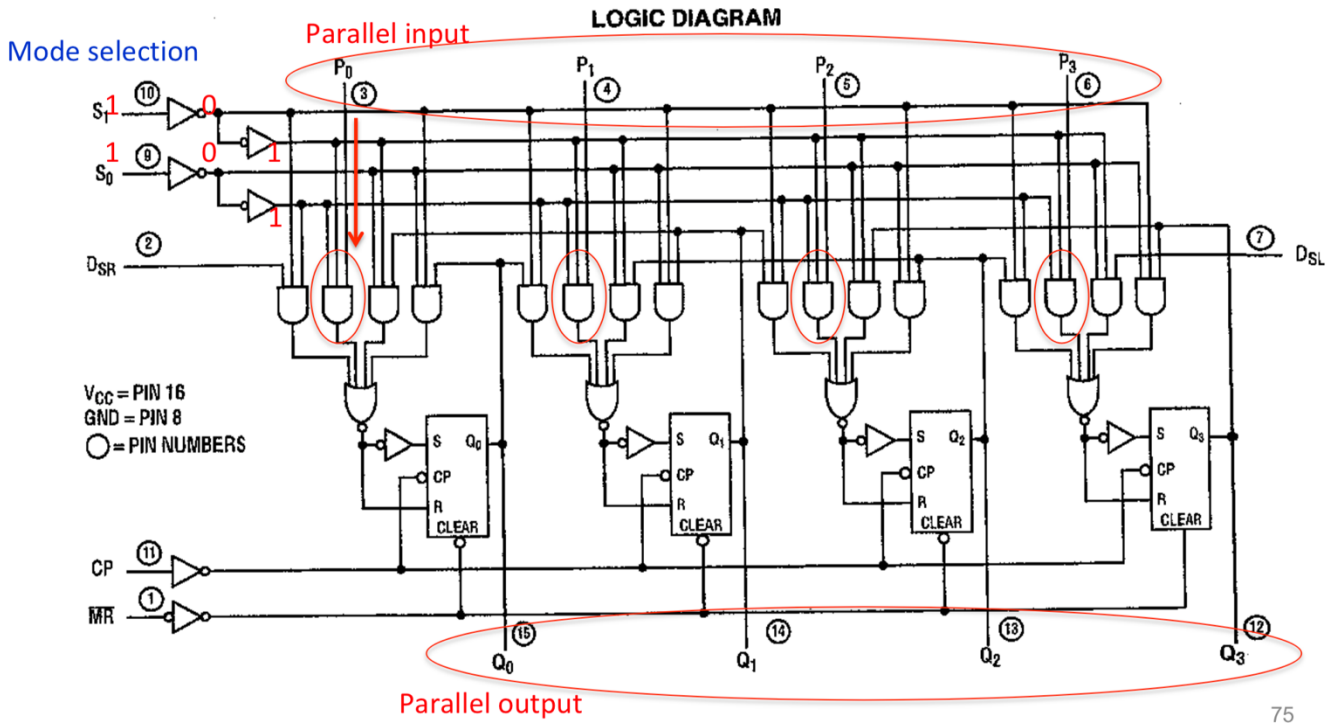
73

When $S_1 = 0$, $S_0 = 1$ → Serial in parallel out (Right shift):



74

When $S_1 = S_0 = 1$ → Parallel in parallel out:



Overall truth table:

MODE SELECT — TRUTH TABLE

OPERATING MODE	INPUTS						OUTPUTS			
	$\overline{MR}$	S_1	S_0	D_{SR}	D_{SL}	P_n	Q_0	Q_1	Q_2	Q_3
Reset	L	X	X	X	X	X	L	L	L	L
Hold	H	l	l	X	X	X	q_0	q_1	q_2	q_3
Shift Left	H	h	l	X	l	X	q_1	q_2	q_3	L
	H	h	l	X	h	X	q_1	q_2	q_3	H
Shift Right	H	l	h	l	X	X	L	q_0	q_1	q_2
	H	l	h	h	X	X	H	q_0	q_1	q_2
Parallel Load	H	h	h	X	X	P_n	P_0	P_1	P_2	P_3

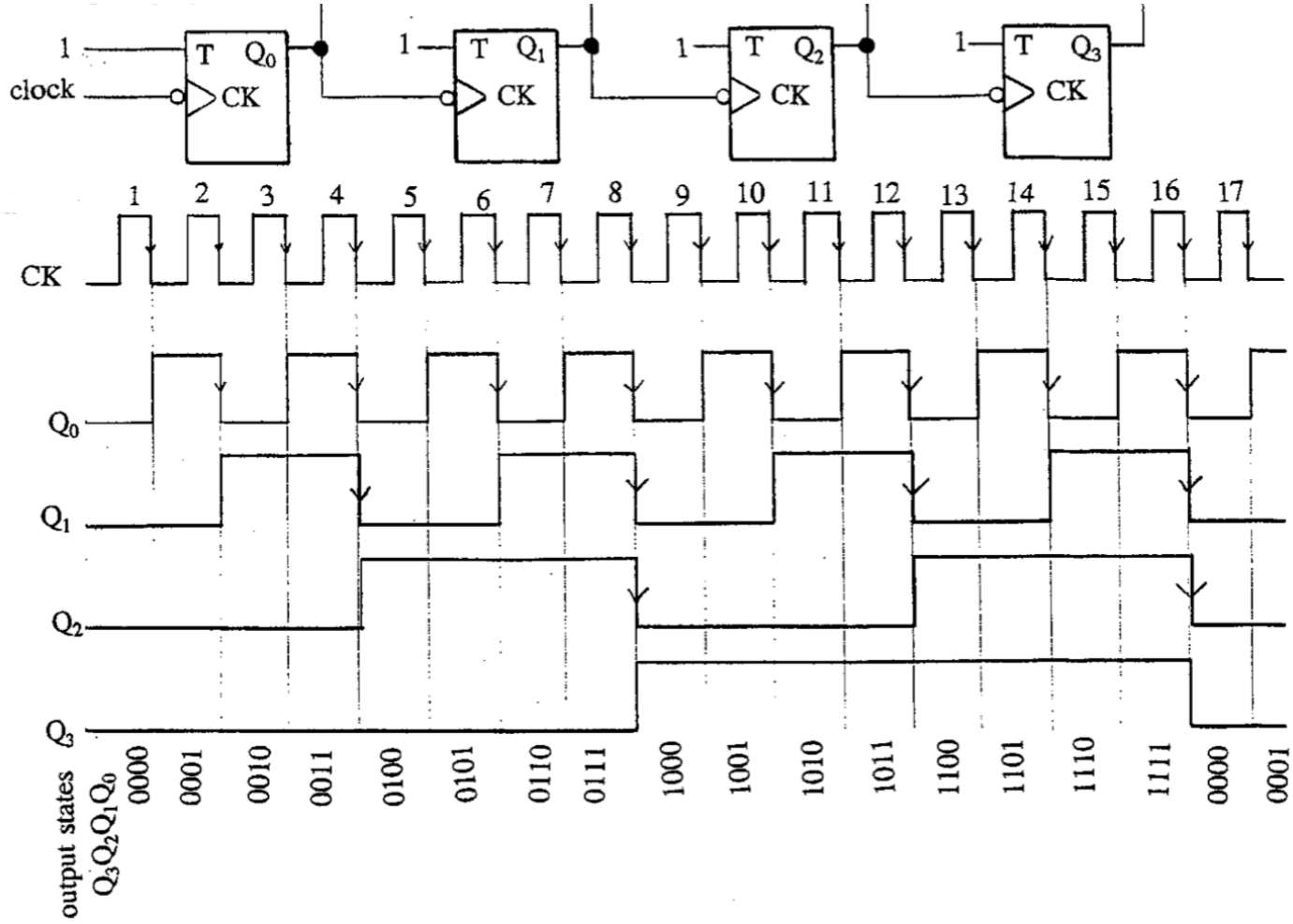
L = LOW Voltage Level
H = HIGH Voltage Level
X = Don't Care
l = LOW voltage level one set-up time prior to the LOW to HIGH clock transition
h = HIGH voltage level one set-up time prior to the LOW to HIGH clock transition
p_n (q_n) = Lower case letters indicate the state of the referenced input (or output) one set-up time prior to the LOW to HIGH clock transition.

Counters

In a counter, the clock pulses causes the output pattern of the FFs to count over certain sequence.
Usage:

- Medium and large scale integration (MSI, LSI)
- Digital measuring instruments
- Industrial controls

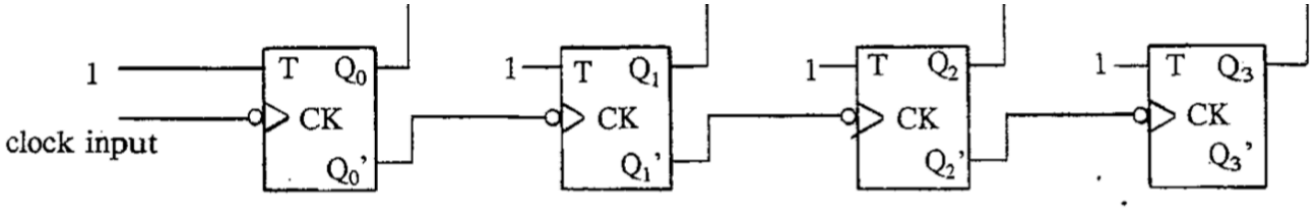
- **Ripple counter**
  Also known as asynchronous counters, constructed using a chain of T-FF, the following is a 4 bit counter:



The parallel output states goes through the binary sequences from 0000 to 1111.
If Q_n is given a weight of 2^n, then this is a counter counting from 0 to 15 repeatly.

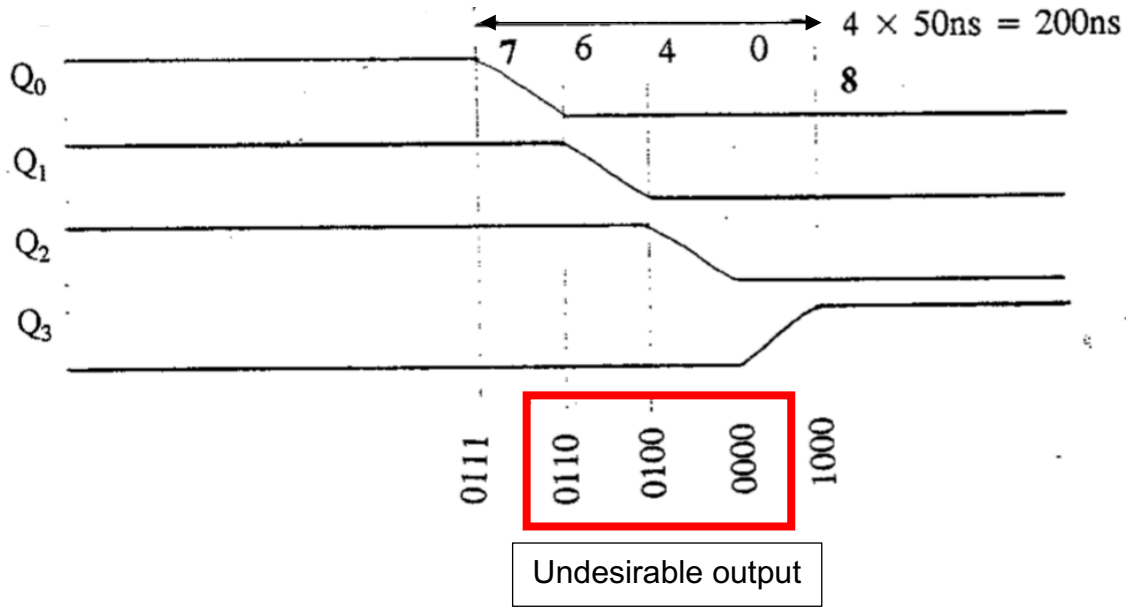
Note:
In this case, Q_3 is called the Most Significant Bit (MSB) and Q_0 is called the Least Significant Bit (LSB). The LSB changes most often and MSB changes least often.

- **Down counter**
  This counter will repeatly count 1111 to 0000 (15 to 0):



- **Disadvantages of ripple counters**
  1. Propagation delay
     Since FFs usually have propagation delays, in a long chain of FFs, the last FF changes its state considerably later than the first FF as the propagation delay accumulate.

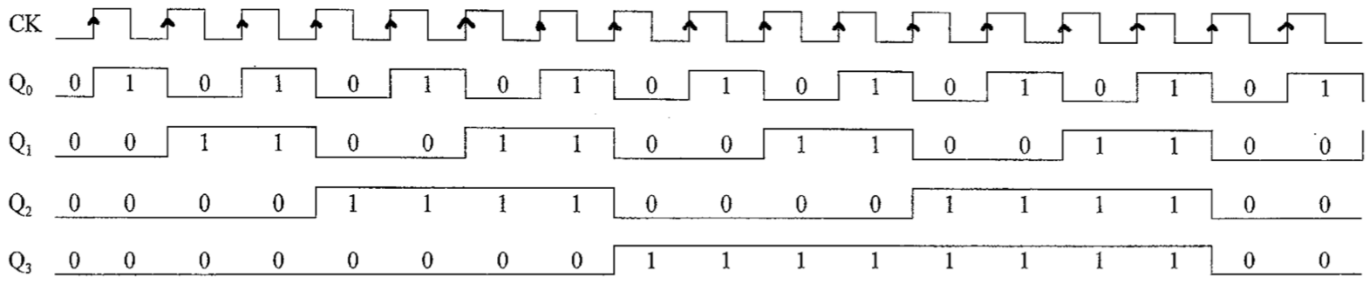
  2. Intermediate states
     This also arise from the propagation delay. When one FF change state, an undesired number, which is out of the sequence, will appear, until the last:



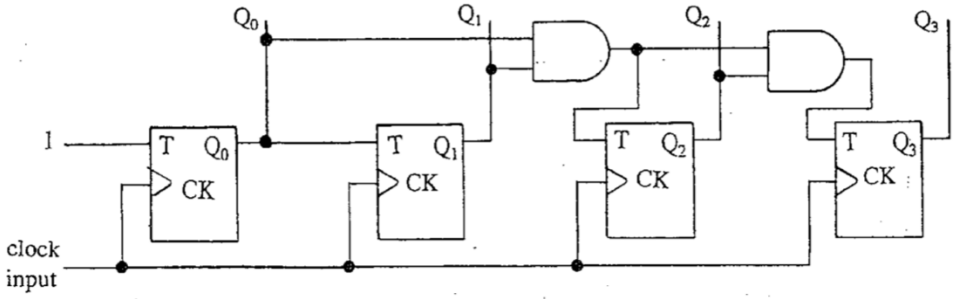
If each FF have a propagation delay of 50ns, the desired out 1000 will appear only after 200ns.

If this is connected to other logic circuit which can respond to those short intermediate states, serious errors would occur.

- **Synchronous counter**
  By synchronising all the state changes, the above problems can be overcome.
  The following is a 4 bit binary synchronous counter circuit and its outputs:



Count	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1



Note:

- All the CK inputs of the FFs are connected to the same train of pulses, now there is no accumulation of propagation delays and intermediate states will not occur. The transitions of all stages occur simultaneously.
- Logic gates are used in synchronous counters to empower them to count any sequences. In the counter above:
  - Q_2 is high only when Q_0, Q_1 are high (number 3 and 11)
  - Q_3 is high only when Q_0, Q_1, Q_2 are high (number 7)

Formal design of synchronous counter

1. Determine the number of FF required. If there are m output states and n FFs are used, then:

$$m \leq 2^n$$

   If there are 10 states in a counter, then 4 FFs will be required.
2. Write out the state transition table which shows the desired next state for each current state.
3. From the states transition table, find out the truth table for the FF inputs. That is to determine the input to the FFs to enable the required state change.
4. For each FF input, minimize the logic function and design the gate circuit to implement the logic function.

If the number of used states exceeds the maximum number of possible states, it is advisable to force those unused states to the initial output.

Example:
Present a formal design for a modulo 6 up counter using JK FF.

A modulo 6 counter has 6 states, therefore 3 JK FF are required because $6 \leq 2^3$. Since 3 JK FF give 8 outputs, therefore 2 unused states are forced to 0.

We know that the truth table of JK FF is:

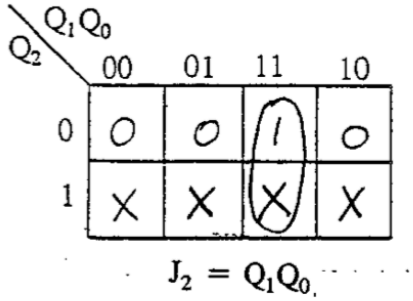
$J(t)$	$K(t)$	$Q(t)$	$Q(t+1)$	Remarks
0	0	0	0	Unchanged
		1	1	
0	1	0	0	Reset
		1	0	
1	0	0	1	Set
		1	1	
1	1	0	1	Toggle
		1	0	

The we can work out the state transition table and the required inputs for the FFs from 000 to 111 in the form of $Q_2Q_1Q_0$:

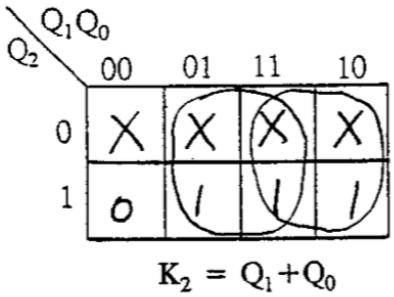
No.	$Q_2(t)$	$Q_2(t+1)$	J_2	K_2	$Q_1(t)$	$Q_1(t+1)$	J_1	K_1	$Q_0(t)$	$Q_0(t+1)$	J_0	K_0
0 → 1	0	0	0	X	0	0	0	X	0	1	1	X
1 → 2	0	0	0	X	0	1	1	X	1	0	X	1
2 → 3	0	0	0	X	1	1	X	0	0	1	1	X
3 → 4	0	1	1	X	1	0	X	1	1	0	X	1
4 → 5	1	1	X	0	0	0	0	X	0	1	1	X
5 → 6	1	0	X	1	0	0	0	X	1	0	X	1
6 → 7	1	0	X	1	1	0	X	1	0	0	0	X
7 → 0	1	0	X	1	1	0	X	1	1	0	X	1

**X means either 0 or 1

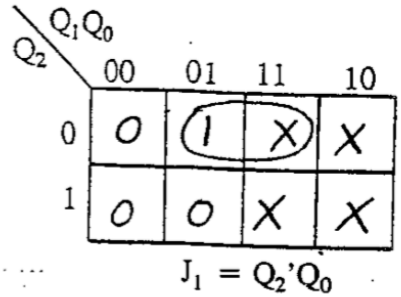
Then use K maps to get the simplified logic functions for JK FF inputs:



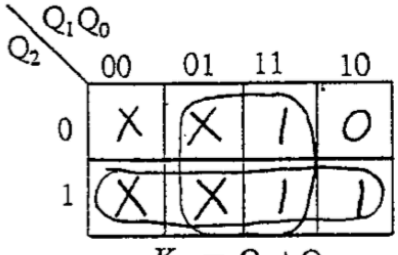
$J_2 = Q_1Q_0$



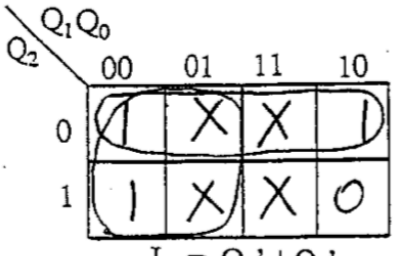
$K_2 = Q_1 + Q_0$



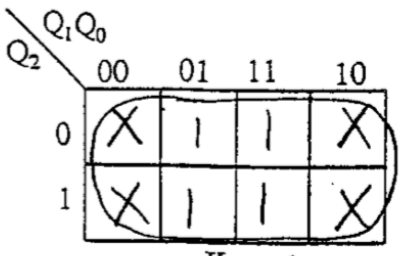
$J_1 = Q_2'Q_0$



$K_1 = Q_2 + Q_0$

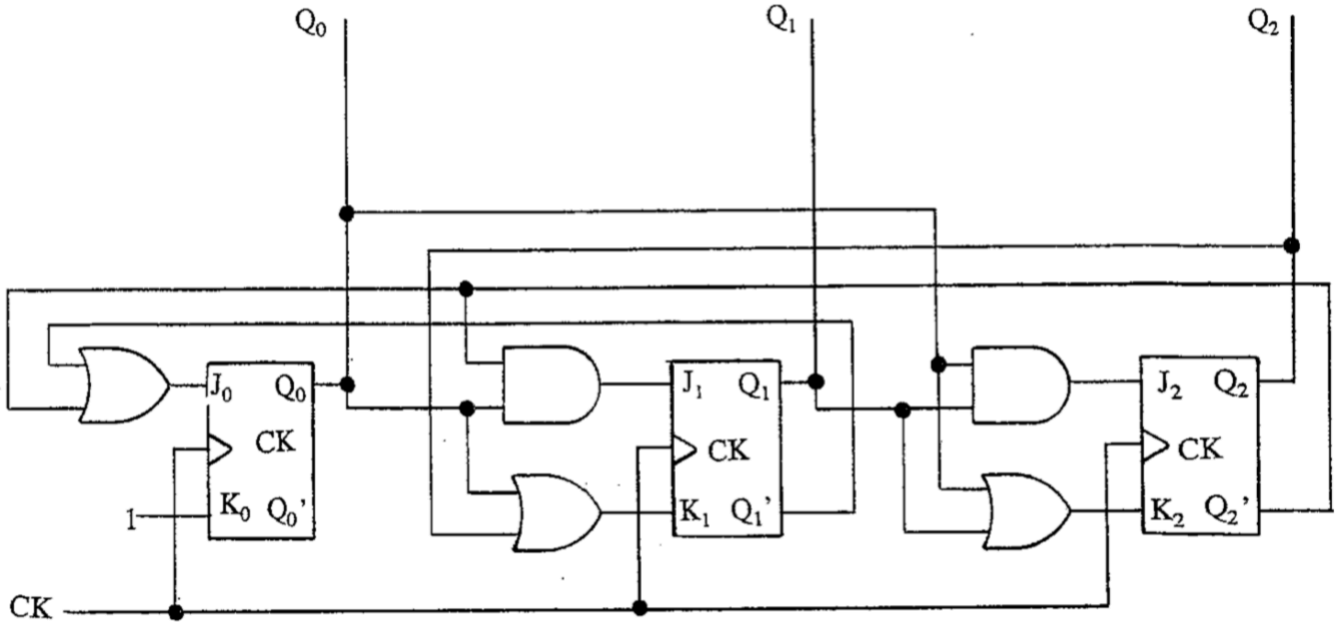


$J_0 = Q_2' + Q_1'$



$K_0 = 1$

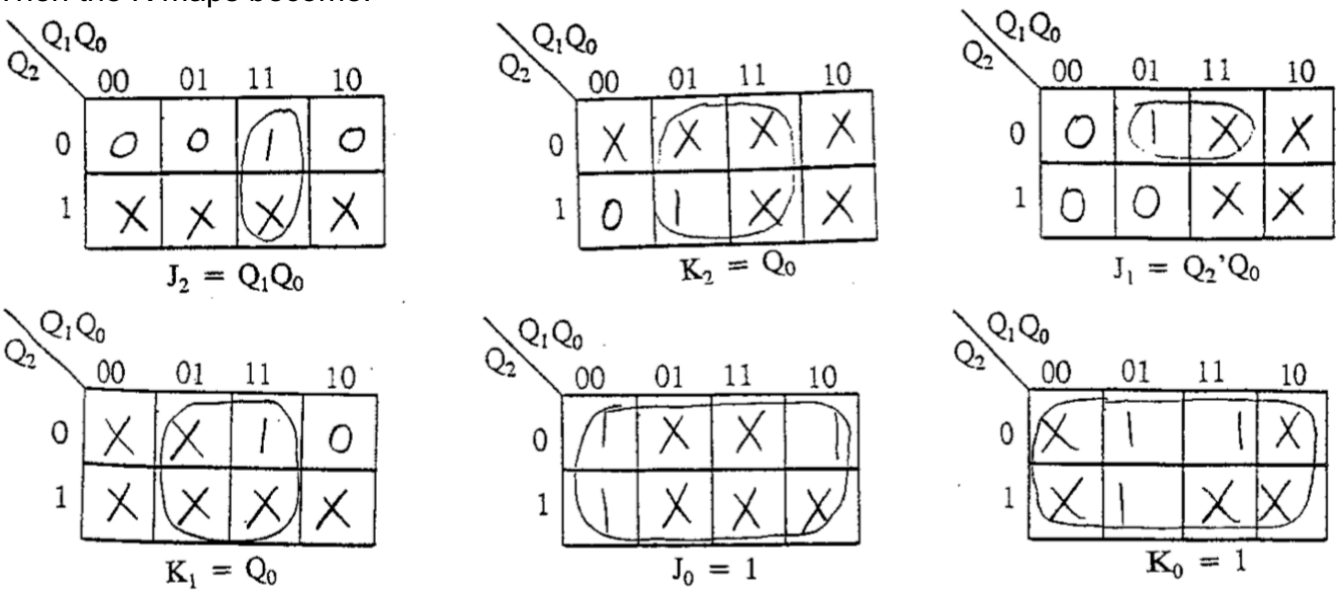
Which gives the logic circuit below:



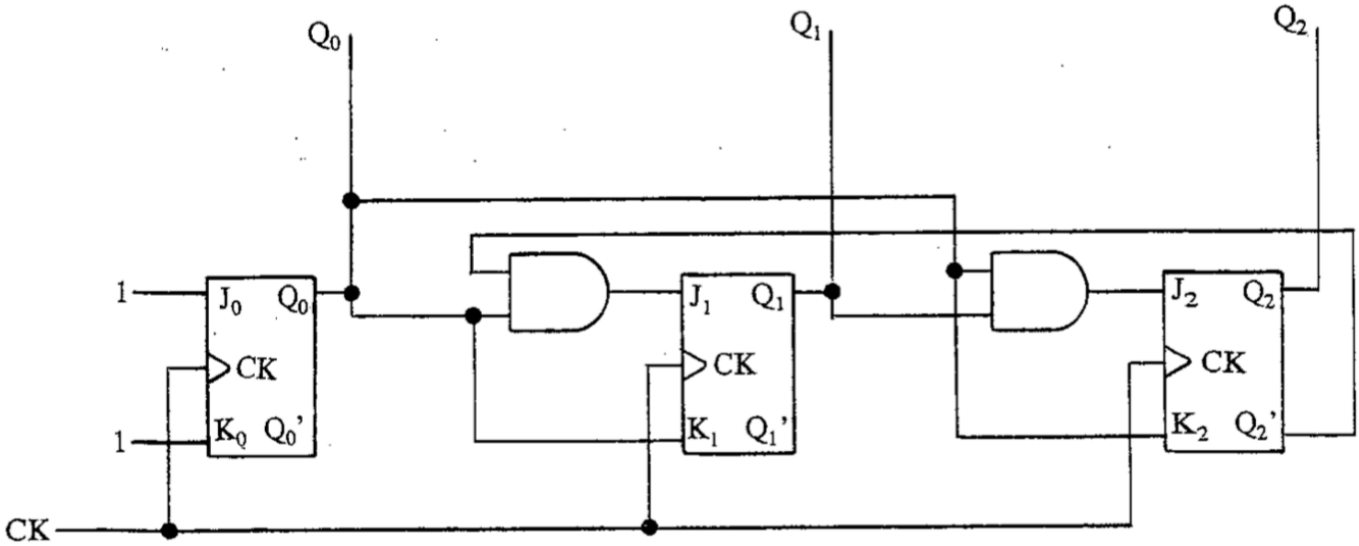
If we do not care the unused states, then the logic functions for the FF inputs will be simpler:

No.	$Q_2(t)$	$Q_2(t+1)$	J_2	K_2	$Q_1(t)$	$Q_1(t+1)$	J_1	K_1	$Q_0(t)$	$Q_0(t+1)$	J_0	K_0
0 → 1	0	0	0	X	0	0	0	X	0	1	1	X
1 → 2	0	0	0	X	0	1	1	X	1	0	X	1
2 → 3	0	0	0	X	1	1	X	0	0	1	1	X
3 → 4	0	1	1	X	1	0	X	1	1	0	X	1
4 → 5	1	1	X	0	0	0	0	X	0	1	1	X
5 → 6	1	0	X	1	0	0	0	X	1	0	X	1
6 → 7	1	X	X	X	1	X	X	X	0	X	X	X
7 → 0	1	X	X	X	1	X	X	X	1	X	X	X

Then the K maps become:



This gives the circuit below:



However, if the initial output of the counter is not one of the 6 desired outputs when first start up, then the counter has to go through several stages before getting to 000.

Example:

The initial output is 110 (decimal 6), then it has to sequence through 111 before going to 000 to start the cycle.

If the intermediate output cause errors in other parts of the logic circuit, then the first design must be used to eliminate the intermediate states. Or we can change 5 → 6 to 5 → 0 in the state transition table.

Tutorial Questions

Present a formal design of a synchronous counter using JK flip-flops to count the following sequences:

- 2,3,5,7 All unused states are forced to the initial count.
- 6,5,4,3,2 Unused states can be treated as don't care.
- 1,6,2,3,7 All unused states are forced to the initial count.
- 7,4,1,5,3 Unused states can be treated as don't care.