* Universal Shift Register:

- A register capable of shifting in one direction only is a unidirectional shift register.
- One that can shift in both directions is a bidirectional shift register.
- If the register has both shifts and parallel load capabilities, it is referred to as a Universal shift register.
- So, a universal shift register is a bidirectional register, whose i/p can be either in serial form or in parallel form and whose o/p also can be either in serial form or in parallel form
- A universal shift register can be realized using multiplexers. It consists of 4 DFFs and 4 MUXs
- The MUX used here are 4:1 MUX, which have 4 o/ps & 2 selection lines (S_1, S_0).
- The selection inputs controls the mode of operation of the register according to the function entries.

Operation of Universal shift register:

- When $S_1S_0 = 0$, the present value of the register is applied to the D i/ps of the flip-flops.

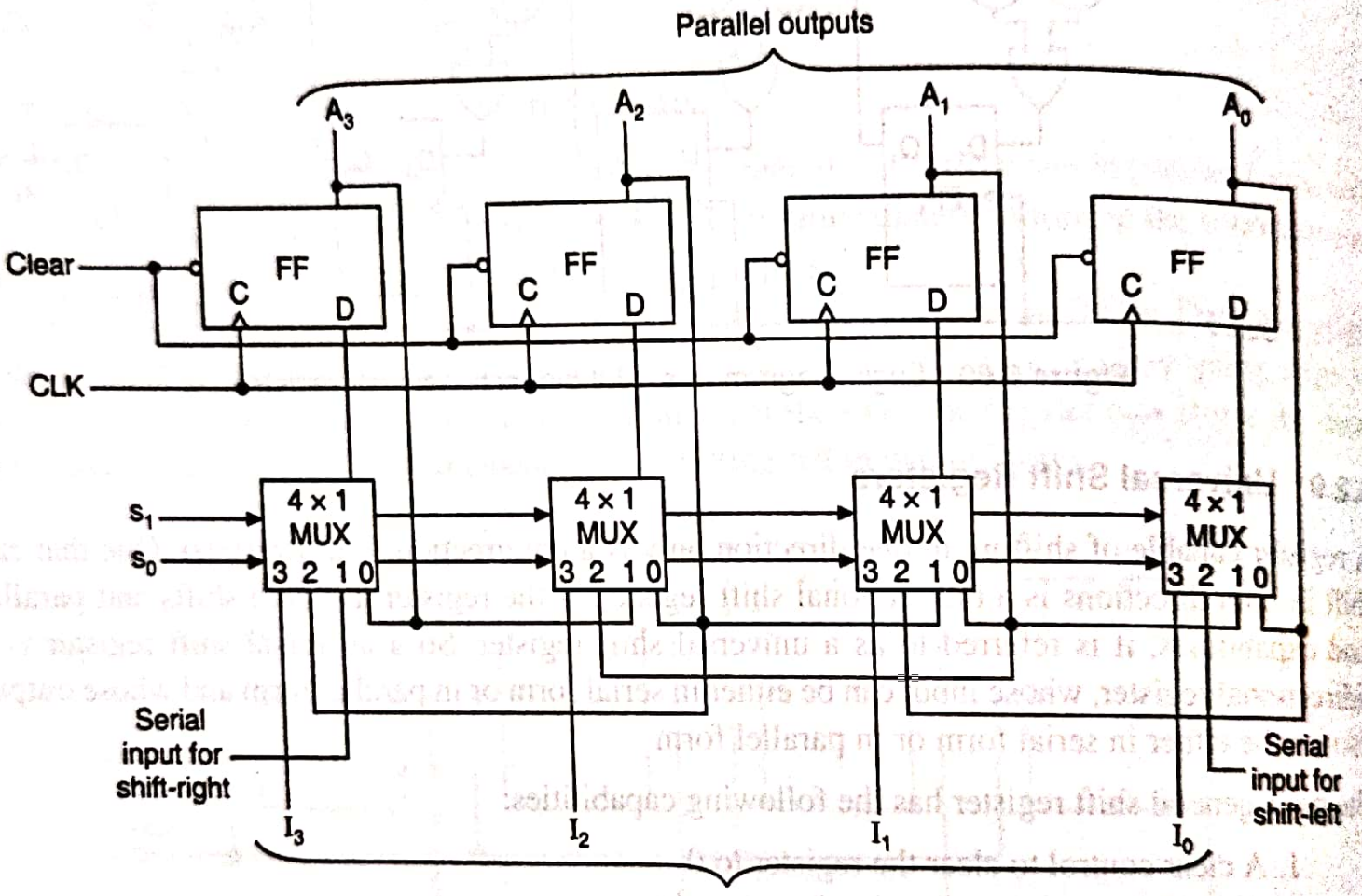


Figure 6.61 4-bit universal shift register.

Table 6.9 Function table for the register of Figure 6.61

Mode control		
S_1	S_0	Register operation
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

- This condition forms a path from the o/p of each flip-flop into the input of the same flipflop.
- When clock pulse is applied, no change of state occurs in all FFs o/ps.
- When $S_1S_0 = 01$, the terminal 1 of the multiplexer i/ps have a path to the D i/ps of the FFs.
- This causes a shift-right operation, with the Serial i/p transferred into FF3 ~~As~~.
- When $S_1S_0 = 10$, a shift-left operation results with the other serial i/p going into FF0 ~~As~~.
- When $S_1S_0 = 11$, the binary information on the parallel i/p lines is transferred into the register simultaneously ~~during the~~ when next clock pulse is applied.

* Applications of Shift Registers:

- Shift registers are very important digital building blocks.
- They have innumerable applications. They are used to provide time delays, for serial/parallel data conversion, ring counters etc.
- They also form the basis for some very important arithmetic operations.

6.2.11 Applications of Shift Registers

Time delays: In many digital systems, it is necessary to delay the transfer of data until such time as operations on other data have been completed, or to synchronize the arrival of data at a subsystem where it is processed with other data. A shift register can be used to delay the arrival of serial data by a specific number of clock pulses, since the number of stages corresponds to the number of clock pulses required to shift each bit completely through the register. The total time delay can be controlled by adjusting the clock frequency and by prescribing the number of stages in the register. In practice, the clock frequency is fixed and the total delay can be adjusted only by controlling the number of stages through which the data is passed. By using a serial-in, parallel-out register and by taking the serial output at any one of the intermediate stages, we have the flexibility to delay the output by any number of clock pulses equal to or less than the number of stages in the register. The arrangement shown in Figure 6.59 can be used to delay the data by 4 clock pulses.

Serial/Parallel data conversion: We know that data can be available either in serial form or in parallel form. Transfer of data in parallel form is much faster than that in serial form. Similarly, the processing of data is much faster when all the data bits are available simultaneously. For this reason, digital systems in which speed is an important consideration are designed to operate on data in parallel form. When large data is to be transmitted over long distances, transmitting data on parallel lines is costly and impracticable. It is convenient and economical to transmit data in serial form, since serial data transmission requires only one line. Shift registers are used for converting serial data to parallel form, so that a serial input can be processed by a parallel system and for converting parallel data to serial form, so that parallel data can be transmitted serially.

A serial-in, parallel-out, shift register can be used to perform serial-to-parallel conversion, and a parallel-in, serial-out, shift register can be used to perform parallel-to-serial conversion. A universal shift register can be used to perform both the serial-to-parallel and parallel-to-serial data conversions. A bidirectional shift register can be used to reverse the order of data. The arrangement shown in Figure 6.57 can be used for serial-to-parallel conversion of a 4-bit data. The arrangement shown in Figure 6.58 can be used for parallel-to-serial conversion of a 4-bit data.

Ring counters: Ring counters are constructed by modifying the serial-in, serial-out, shift registers. There are two types of ring counters—basic ring counter and Johnson counter. The basic ring counter can be obtained from a serial-in, serial-out, shift register by connecting the Q output of the last FF to the D input of the first FF. The Johnson counter can be obtained from a serial-in, serial-out, shift register by connecting the $\overline{Q}$ output of the last FF to the D input of the first FF. Ring counter outputs can be used as a sequence of synchronizing pulses. The ring counter is a decimal

counter. It is a divide-by-N counter, where N is the number of stages. The keyboard encoder is an example of the application of a shift register used as a ring counter in conjunction with other devices.

Ring counters are dealt with in detail later in this chapter only. Figures 6.108, and 6.109 (Section 6.3.13) illustrate the use of the shift register as a ring counter. Figures 6.112 and 6.113 (Section 6.3.13) illustrate the use of the shift register as a twisted ring counter.

...l asynchronous receiver transmitter (UART): C...

* Counters:

Introduction Of Counters:

- A Counter is a digital device used to count no. of pulses applied at the i/p.
- A Counter comprises of set of flip-flops, interconnected such that their combined state at any time is the binary equivalent of the total no. of pulses applied up to that time.
- Counters can Count in two ways
  ie) Up Count (0, 1, 2, ... N) and
  Down Count (N, N-1, ... 0)
- present Count of the Counter represents the state of Counter.

**
- N-bit Counter requires N flip flop and have 2^N states and divides the i/p frequency by 2^n. Hence, it is called as divide-by-2^n Counter or mod-2^n counter

Ex: → 2-bit Counter needs 2 FFs and it has $2^2 = 4$ states ie) 00, 01, 10, 11
Possible Count = 0, 1, 2, 3. called as Mod-4 counter

→ 4-bit Counter needs 4 FFs and it has $2^4 = 16$ states ie) from 0000 to 1111.
Possible Count = 0 to 15 Mod 16 Counter

Applications of Counters:

- They are used as frequency dividers
- They are used to perform the timing function as in digital watches, to create time delays, to produce non-sequential binary counts, to generate pulse trains, and to act as frequency Counters etc.

- Counters are of two types:
  - Asynchronous Counters and
  - Synchronous Counters.

→ What is the modulus of a Counter?

Ans: The no. of states through which the counter passes before returning to the starting state is called the modulus of the counter.

- Hence the modulus of a counter is equal to the total no. of distinct states (counts) including zero that a counter can store.
- In other words, the no. of i/p pulses that causes the Counter to reset to its initial count is called the modulus of the Counter.

* Shortened Modulus Counter:

A shortened modulus counter is a counter which does not utilize all the possible states. In this, some of the states are unutilized ie; invalid.

* Full Modulus Counter: A counter which goes through all the possible states before restarting is called full modulus counter. There are no invalid states.

* Variable Modulus counter: A counter in which the maximum no. of states can be changed is called the variable modulus counter

* Terminal Count: The final state of the counter sequence is called terminal count

* Comparison between Asynchronous Counters and Synchronous Counters:

Asynchronous Counter	Synchronous Counter
1. In this, type of Counter FFs are connected in such a way that the o/p of first FF drives the clock for the second FF, the o/p of the second the clock of the third and so on	1. In this type of Counter, there is no connection between the o/p of first FF and clock i/p of next FF and so on
2. All the FFs are not clocked simultaneously.	2. All the FFs are clocked simultaneously
3. Design and implementation is very simple even for more no. of states.	3. Design & implementation becomes tedious and complex as the no. of states increases.
4. Main drawback of these Counters is their low speed as the clock is propagated through a no. of FFs before it reaches the last FF.	4. Since clock is applied to all the FFs simultaneously the total propagation delay is equal to the propagation delay of only one FF Hence they are faster

* Design of Synchronous Counters:

Step 1: No. of Flip-flops: Based on the description of the problem, determine the required no. of FFs. and desired counting sequence

Step 2: State Diagram: Draw the state diagram showing all the possible states.

Definition of state diagram:

- Also called as transition diagram
- State diagram is a graphical means of depicting the sequence of states through which the counter progresses.

- In case the counter goes to a particular state from the invalid states on the next clock pulse, the same can also be included in the state diagram

Step 3: Choice of FF & excitation table: Select the type of flip flops to be used and write the excitation table

→ An excitation table is a table that lists the present state (PS), the next state (NS) and required excitations.

Step4: Minimal expressions for excitations:

Obtain the minimal expressions for the excitations of the FFs using K-maps drawn for the excitations of the FFs interms of present states & i/ps.

Step5: Logic diagram: Draw a logic diagram based on minimal expressions.

* Design of a synchronous 3-bit up-down Counter Using J-K FFs:

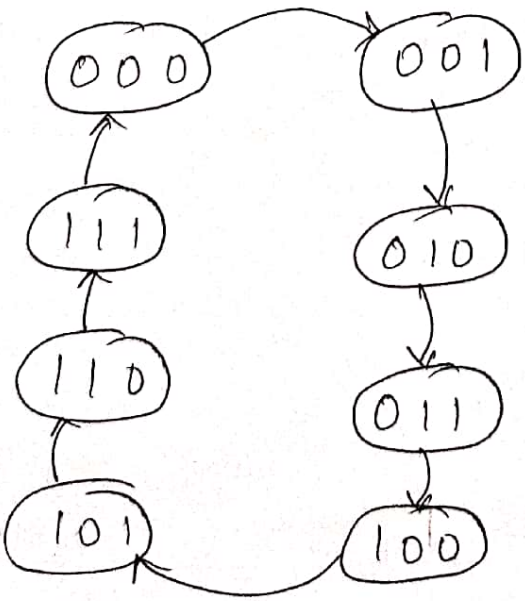
Up Counter design

Step1: No. of FFs = 3

No. of states = 8 (000, 001, 010, 011, 100, 101, 110, 111)

All states are valid ~~Hence~~ no don't cares.

Step2: State diagram



Step3: JK FFs are selected

Step 3:

PS Q_n	NS Q_{n+1}	Required i/ps J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

→ JK FF Excitation Table

PS Q_3 Q_2 Q_1	NS Q_3 Q_2 Q_1	Required excitations J_3 K_3	J_2 K_2	J_1 K_1
0 0 0	0 0 1	0 X	0 X	1 X
0 0 1	0 1 0	0 X	1 X	X 1
0 1 0	0 1 1	0 X	X 0	1 X
0 1 1	1 0 0	1 X	X 1	X 1
1 0 0	1 0 1	X 0	0 X	1 X
1 0 1	1 1 0	X 0	1 X	X 1
1 1 0	1 1 1	X 0	X 0	1 X
1 1 1	0 0 0	X 1	X 1	X 1

Step 4: From the excitation table, $J_1 = 1, K_1 = 1$ because all the entries for J_1 and K_1 are either X or 1.

K Map for J_3

Q_3 \ Q_2Q_1	00	01	11	10
0			1	
1	X	X	X	X

$J_3 = Q_2 Q_1$

K Map for K_3

Q_3 \ Q_2Q_1	00	01	11	10
0	X	X	X	X
1			1	

$K_3 = Q_2 Q_1$

KMap for J_2

Q_3 \ Q_2Q_1	00	01	11	10
0		1	X	X
1		1	X	X

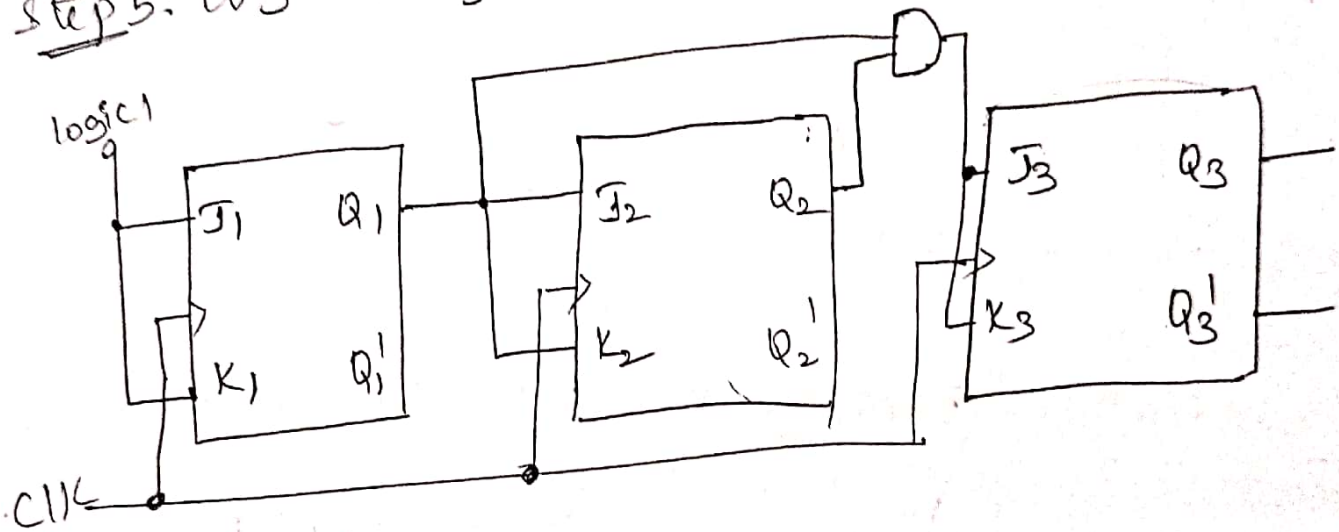
$J_2 = Q_1$

K Map for K_2

Q_3 \ Q_2Q_1	00	01	11	10
0	X	X	1	
1	X	X	1	

$K_2 = Q_1$

Step 5: logic diagram



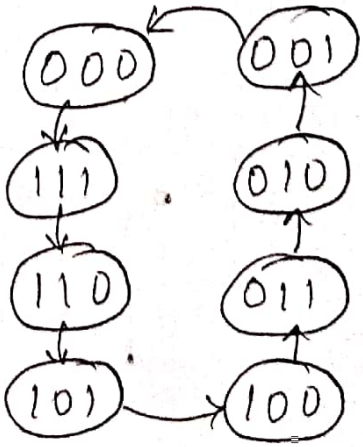
Down Counter Design:

Step 1: No. of FFs = 3

No. of states = 8 (000, 001, 010, 011, 100, 101, 110, 111)

All states are valid, so Hence no don't cares

Step 2: State diagram



Step 3: JK FFs are selected

PS			NS			Required Excitations					
Q_3	Q_2	Q_1	Q_3	Q_2	Q_1	J_3	K_3	J_2	K_2	J_1	K_1
1	1	1	1	1	0	X	0	X	0	X	1
1	1	0	1	0	1	X	0	X	1	1	X
1	0	1	1	0	0	X	0	0	X	X	1
1	0	0	0	1	1	X	1	1	X	1	X
0	1	1	0	1	0	0	X	X	0	X	1
0	1	0	0	0	1	0	X	X	1	1	X
0	0	1	0	0	0	0	X	0	X	X	1
0	0	0	1	1	1	1	X	1	X	1	X

PS Q_n	NS Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Excitation Table for JK

Step 4: From excitation table, $J_1 = K_1 = 1$ because all entries for J_1 & K_1 are either X or 1.

K Map for J_3

Q_3 \ Q_2Q_1	00	01	11	10
0	1			
1	X	X	X	X

$J_3 = Q_2' Q_1'$

K Map for K_3

Q_3 \ Q_2Q_1	00	01	11	10
0	X			
1	1	X	X	X

$K_3 = Q_2' Q_1'$

K Map for J_2

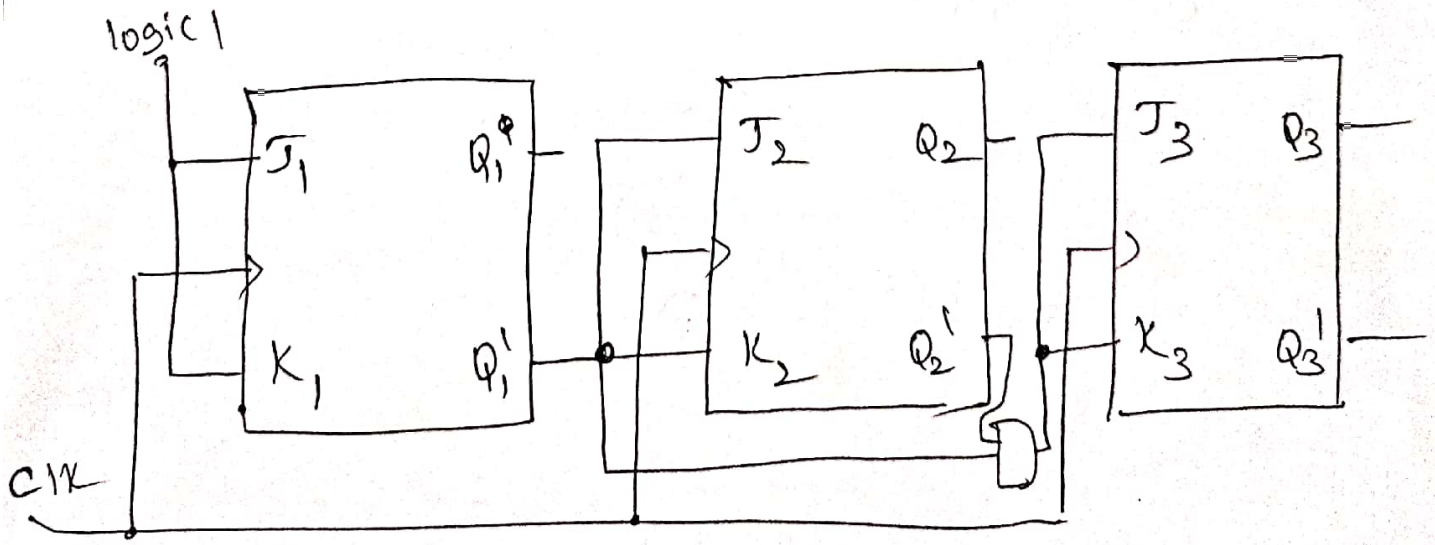
Q_3 \ Q_2Q_1	00	01	11	10
0	X	X		1
1	1		X	X

$J_2 = Q_1'$

K Map for K_2

Q_3 \ Q_2Q_1	00	01	11	10
0	X	X		1
1	X	X		1

$K_2 = Q_1'$



Up-down Counter:

Step 1: No. of FFs = 3

No. of states = 8 (000, 001, 010, 011, 100, 101, 110, 111)

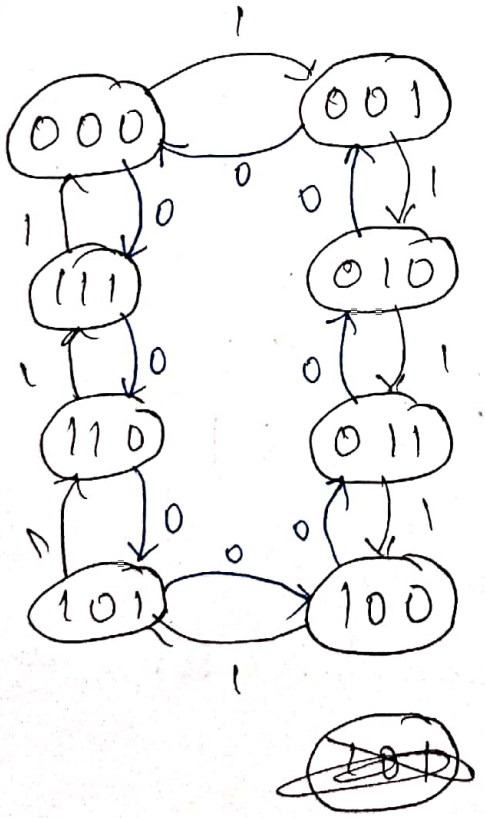
All states are valid → Hence no don't cares

Step 2:

→ For selecting up and down mode, a control or mode signal M is required

→ let's say, when M=1, Counter counts up and when M=0, Counter counts down

Step 2: state diagram



Step 3: JK FFs are selected.

Excitation Table for JK FF

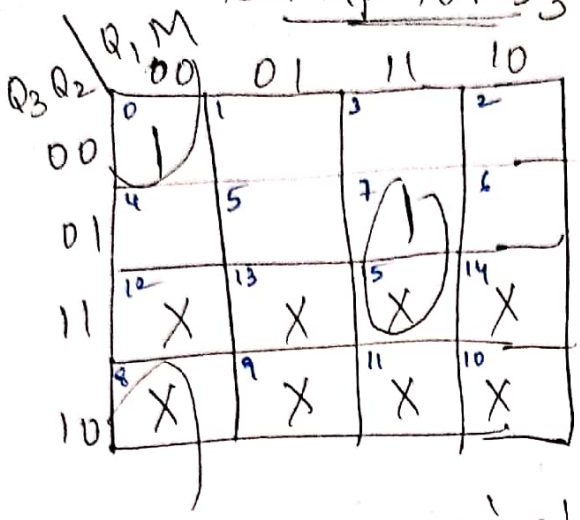
Ps Q_n	Ns Q_{n+1}	Reqd ips J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

	PS			Mode	NS			Required Excitations					
	Q_3	Q_2	Q_1	M	Q_3	Q_2	Q_1	J_3	K_3	J_2	K_2	J_1	K_1
0	0	0	0	0	1	1	1	1	X	1	X	1	X
1	0	0	0	1	0	0	1	0	X	0	X	1	X
2	0	0	1	0	0	0	0	0	X	0	X	X	1
3	0	0	1	1	0	1	0	0	X	1	X	X	1
4	0	1	0	0	0	0	1	0	X	X	1	1	X
5	0	1	0	1	0	1	1	0	X	X	0	X	1
6	0	1	1	0	0	1	0	0	X	X	0	X	1
7	0	1	1	1	1	0	0	1	X	X	1	1	X
8	1	0	0	0	0	1	1	X	1	1	X	1	X
9	1	0	0	1	1	0	1	X	0	0	X	X	1
10	1	0	1	0	1	0	0	X	0	0	X	X	1
11	1	0	1	1	1	1	0	X	0	1	X	1	X
12	1	1	0	0	1	0	1	X	0	X	1	1	X
13	1	1	0	1	1	1	1	X	0	X	0	X	1
14	1	1	1	0	1	1	0			X	0		
15	1	1	1	1	0	0	0	X	1	X	1	X	1

Step 4: From the excitation table,

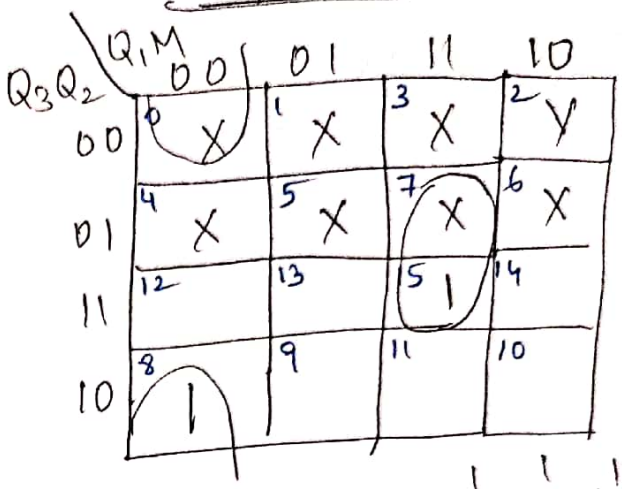
→ $J_1 = 1$, $K = 1$ because all the entries for J_1 and K, are either X or 1.

K-Map for J_3



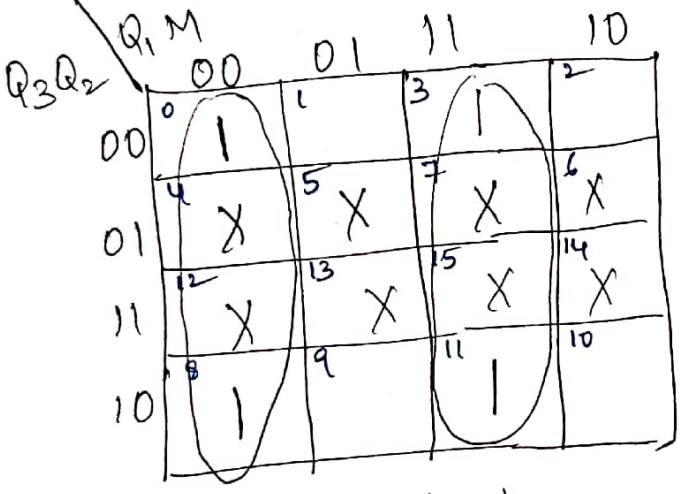
$$J_3 = Q_2 Q_1 M + Q_2' Q_1' M'$$

K-Map for K_3



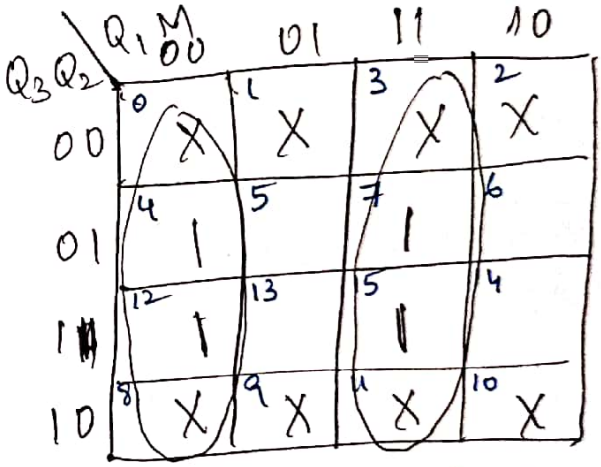
$$K_3 = Q_2 Q_1 M + Q_2' Q_1' M'$$

K-Map for J_2



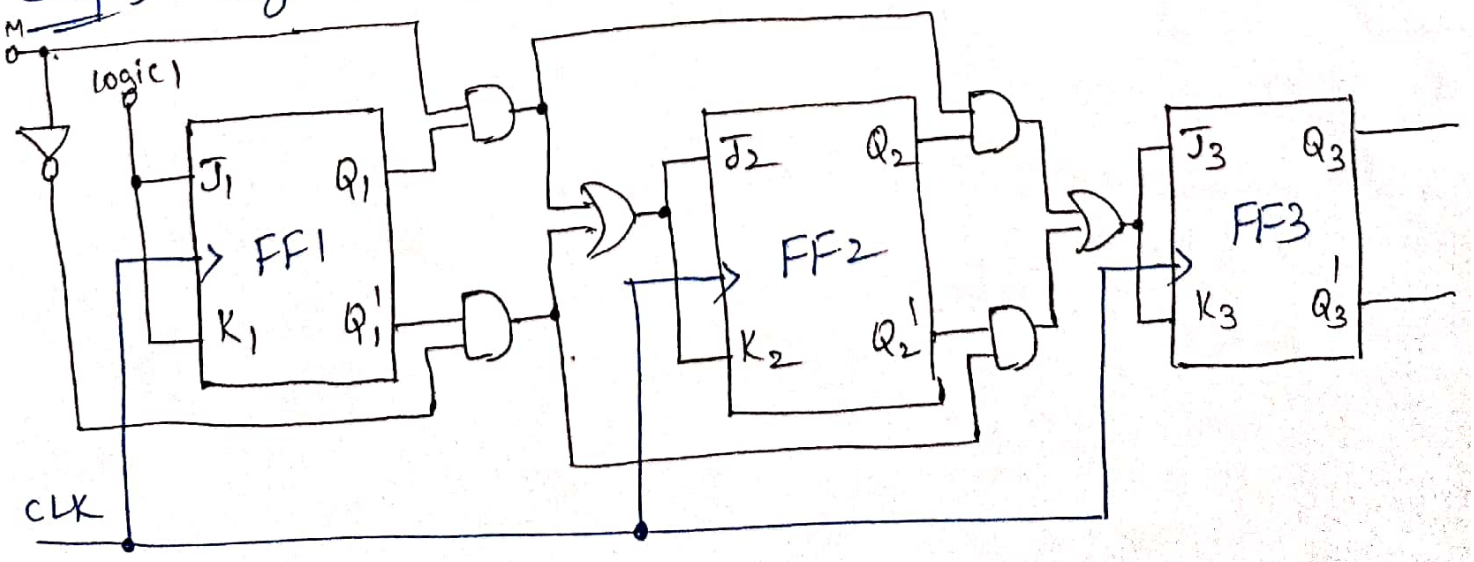
$$J_2 = Q_1 M + Q_1' M'$$

K-Map for K_2



$$K_2 = Q_1 M + Q_1' M'$$

Step 5: Logic diagram



* Design of a synchronous Modulo-6 Gray Code Counter:

Step 1: No. of FFs = 3

No. of states = 8 (000, 001, 010, 011, 100, 101, 110, 111)

→ Here 101 & 100 are invalid.

→ The entries for excitations corresponding to invalid states are don't cares.

Binary to Gray conversion

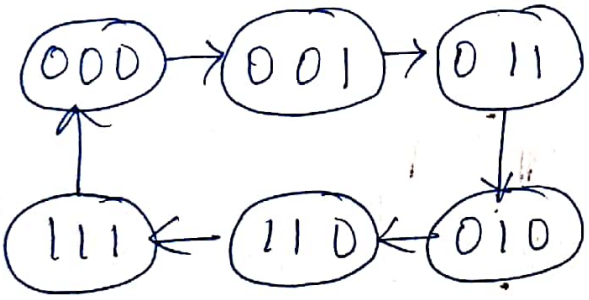
$B_2\ B_1\ B_0$	$G_2\ G_1\ G_0$
0 0 0	0 0 0
0 0 1	0 0 1
0 1 0	0 1 1
0 1 1	0 1 0
1 0 0	1 1 0
1 0 1	1 1 1
1 1 0	1 0 1
1 1 1	1 0 0

$G_2 = B_2$

$G_1 = B_2 \oplus B_1$

$G_0 = B_1 \oplus B_0$

Step 2: State diagram



Step 3: TFF are selected.

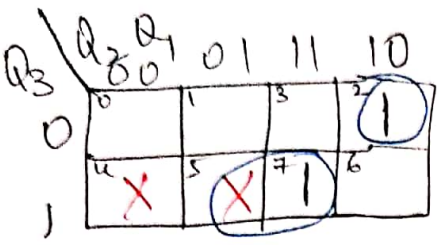
PS $Q_3\ Q_2\ Q_1$	NS $Q_3\ Q_2\ Q_1$	Required Excitation $T_3\ T_2\ T_1$
0 0 0 0	0 0 1	0 0 1
1 0 0 1	0 1 1	0 1 0
3 0 1 1	0 1 0	0 0 1
2 0 1 0	1 1 0	1 0 0
6 1 1 0	1 1 1	0 0 1
7 1 1 1	0 0 0	1 1 1

Excitation Table for TFF

PS Q_n	NS Q_{n+1}	Required i/p T
0	0	0
0	1	1
1	0	1
1	1	0

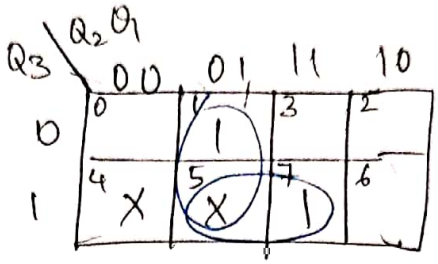
Step 4:

K-Map for T_3



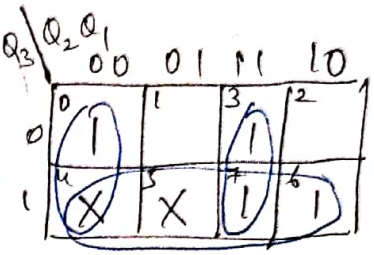
$T_3 = Q_3 Q_1 + Q_3' Q_2 Q_1'$

K-Map for T_2



$T_2 = Q_3 Q_1 + Q_2' Q_1$

K-Map for T_3



$T_1 = Q_3 + Q_2 Q_1 + Q_2' Q_1'$

Step 5: logic diagram