# MODULE-V:
# Sequential Logic Circuits - II

## Sequential Circuit Design

**Steps in the design process for sequential circuits**
**State Diagrams and State Tables**
**Examples**

## Steps in Design of a Sequential Circuit

1. Specification – A description of the sequential circuit. Should include a detailing of the inputs, the outputs, and the operation. Possibly assumes that you have knowledge of digital system basics.
2. Formulation: Generate a state diagram and/or a state table from the statement of the problem.
3. State Assignment: From a state table assign binary codes to the states.
4. Flip-flop Input Equation Generation: Select the type of flip-flop for the circuit and generate the needed input for the required state transitions
5. Output Equation Generation: Derive output logic equations for generation of the output from the inputs and current state.
6. Optimization: Optimize the input and output equations. Today, CAD systems are typically used for this in real systems.
7. Technology Mapping: Generate a logic diagram of the circuit using ANDs, ORs, Inverters, and F/Fs.
8. Verification: Use a HDL to verify the design.

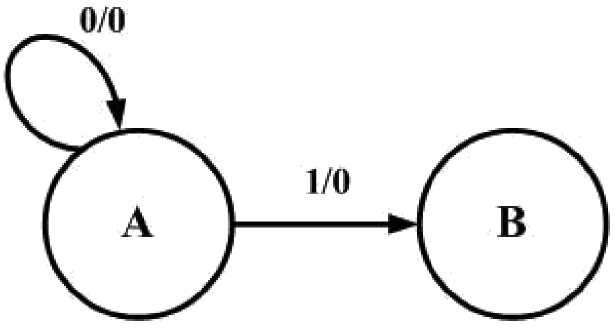
Sequential machines are typically classified as either a Mealy machine or a Moore machine implementation.

Moore machine: The outputs of the circuit depend only upon the current state of the circuit.

Mealy machine: The outputs of the circuit depend upon both the current state of the circuit and the inputs.

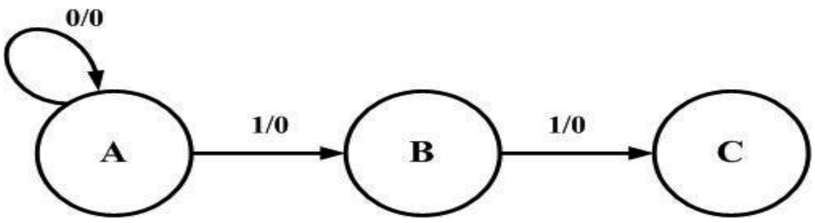
An example to go through the steps
The specification: The circuit will have one input, X, and one output, Z. The output Z will be 0 except when the input sequence 1101 are the last 4 inputs received on X. In that case it will be a 1
Generation of a state diagram
Create states and meaning for them.

State A – the last input was a 0 and previous inputs unknown. Can also be the reset state. State B – the last input was a 1 and the previous input was a 0. The start of a new sequence possibly.
Capture this in a state diagram



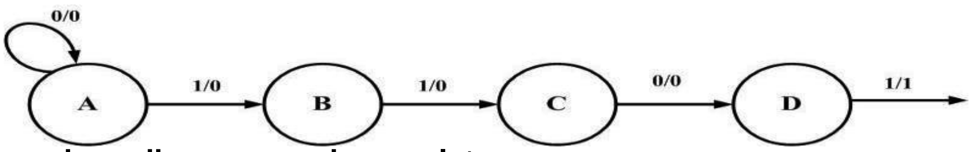
- Capture this in a state diagram

Circles represent the states
Lines and arcs represent the transition between states.
The notation Input/output on the line or arc specifies the input that causes this transition and the output for this change of state.
Add a state C – Have detected the input sequence 11 which is the start of the sequence

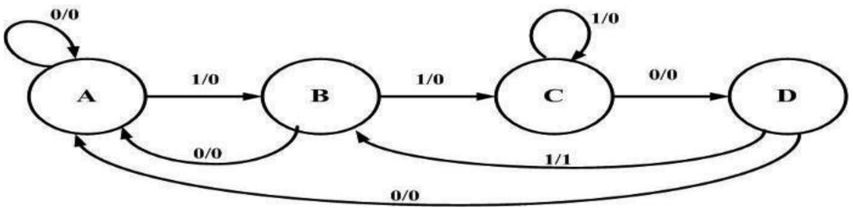


- Add a state D

State D – have detected the 3rd input in the start of a sequence, a 0, now having 110. From State D, if the next input is a 1 the sequence has been detected and a 1 is output.



- The previous diagram was incomplete.
- In each state the next input could be a 0 or a 1. This must be included



**The state table**

**This can be done directly from the state diagram**

| Prresent State | Next State X =0 | Next State X=1 | Output X=0 | Output X=1 |
|---|---|---|---|---|
| A | A | B | 0 | 0 |
| B | A | C | 0 | 0 |
| C | D | C | 0 | 0 |
| D | A | B | 0 | 1 |

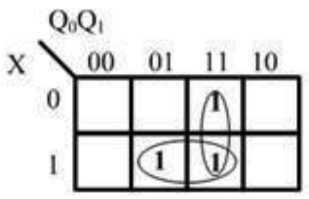
**Now need to do a state assignment**

**Will select a gray encoding**
**For this state A will be encoded 00, state B 01, state C 11 and state D 10**

| Prresent State | Next State X =0 | Next State X=1 | Output X=0 | Output X=1 |
|---|---|---|---|---|
| 00 | 00 | 01 | 0 | 0 |
| 01 | 00 | 11 | 0 | 0 |
| 11 | 10 | 11 | 0 | 0 |
| 10 | 00 | 01 | 0 | 1 |

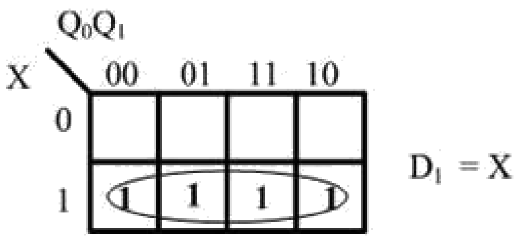
**Flip-flop input equations**

**Generate the equations for the flip-flop inputs Generate the $D_0$ equation**

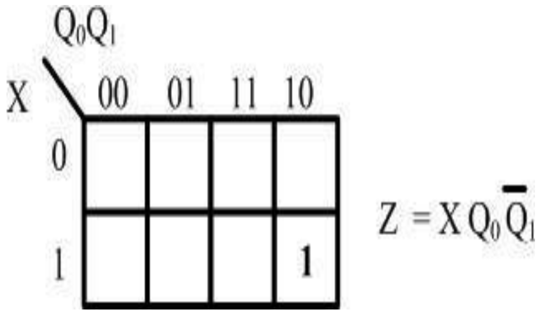


$$D_0 = Q_0 Q_1 + X Q_1$$

**Generate the $D_1$ equation**

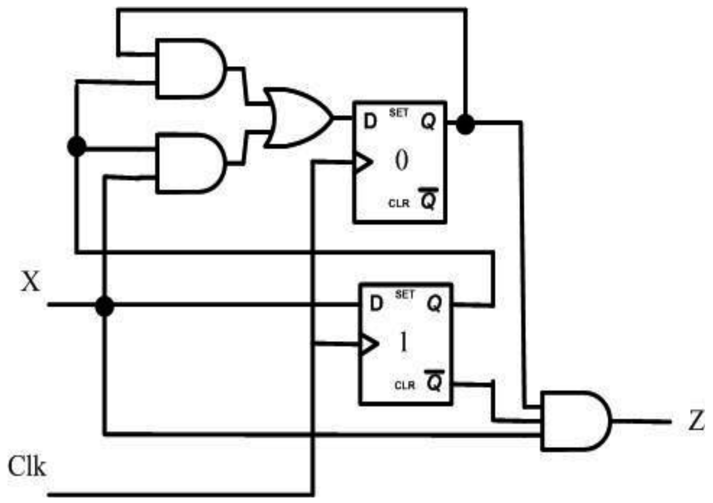


**The output equation**
**The next step is to generate the equation for the output Z and what is needed to generate it.**
**Create a K-map from the truth table.**



**Now map to a circuit**

**The circuit has 2 D type F/Fs**



**Shift registers:**

**In digital circuits, a shift register is a cascade of flip-flops sharing the same clock, in which the output of each flip-flop is connected to the "data" input of the next flip-flop in the chain, resulting in a circuit that shifts by one position the "bit array" stored in it, shifting in the data present at its input and shifting out the last bit in the array, at each transition of the clock input. More generally, a shift register may be multidimensional, such that its "data in" and stage outputs are themselves bit arrays: this is implemented simply by running several shift registers of the same bit-length in parallel.**

**Shift registers can have both parallel and serial inputs and outputs. These are often configured as serial-in, parallel-out (SIPO) or as parallel-in, serial-out (PISO). There are also types that have both serial and parallel input and types with serial and parallel output. There are also bi-directional shift registers which allow shifting in both directions: L→R or R→L. The serial input and last output of a shift register can also be connected to create a circular shift register**

**Shift registers are a type of logic circuits closely related to counters. They are basically for the storage and transfer of digital data.**

**Buffer register:**

**The buffer register is the simple set of registers. It is simply stores the binary word. The buffer may be controlled buffer. Most of the buffer registers used D Flip-flops.**

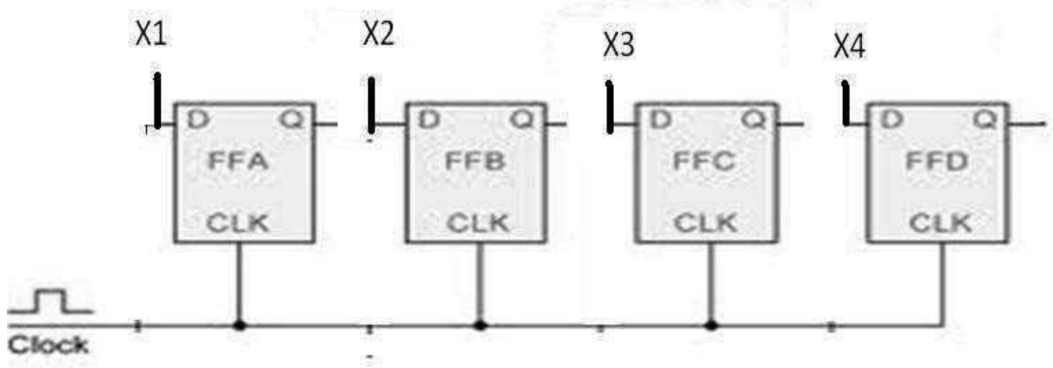


**Figure: logic diagram of 4-bit buffer register**

**The figure shows a 4-bit buffer register. The binary word to be stored is applied to the data terminals. On the application of clock pulse, the output word becomes the same as the word applied at the terminals. i.e., the input word is loaded into the register by the application of clock pulse.**

**When the positive clock edge arrives, the stored word becomes:**

$Q_4Q_3Q_2Q_1=X_4X_3X_2X_1$

**Q=X**

**Controlled buffer register:**

**If goes LOW, all the FFs are RESET and the output becomes, Q=0000.**

**When is HIGH, the register is ready for action. LOAD is the control input.**

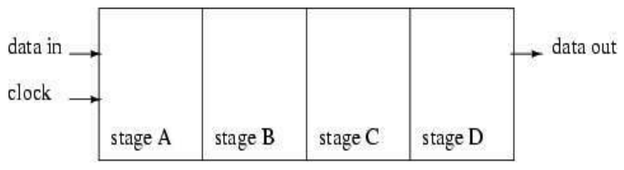
**When LOAD is HIGH, the data bits X can reach the D inputs of FF's.**

$Q_4Q_3Q_2Q_1=X_4X_3X_2X_1$

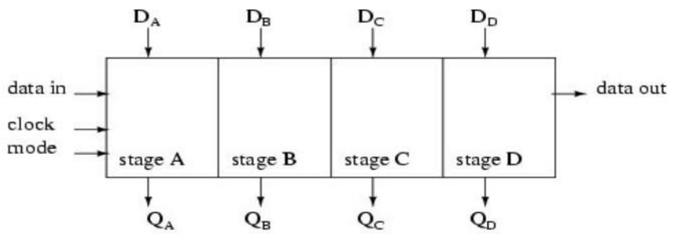
**Q=X**

**When load is low, the X bits cannot reach the FF's.**

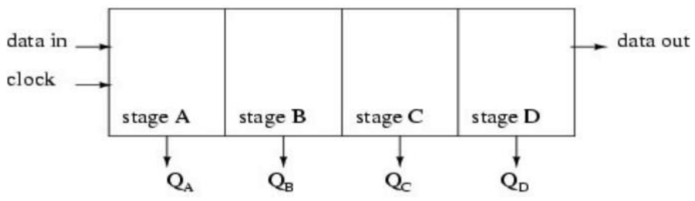
**Data transmission in shift registers:**



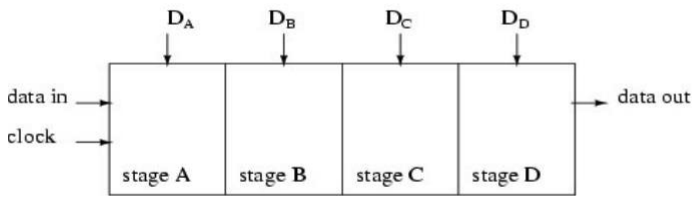
Serial-in, serial-out shift register with 4-stages



Parallel-in, parallel-out shift register with 4-stages



Serial-in, parallel-out shift register with 4-stages



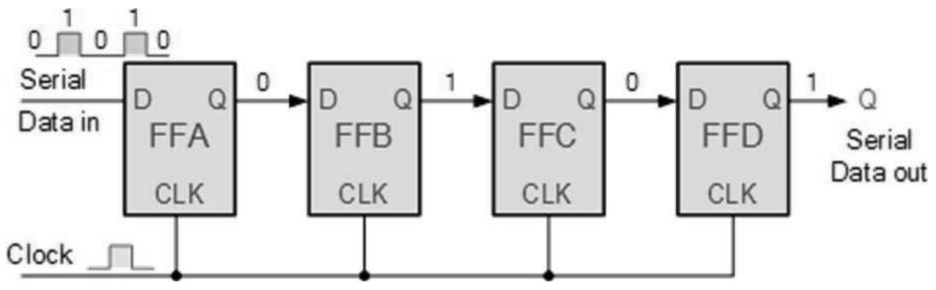
Parallel-in, serial-out shift register with 4-stages

**A number of ff's connected together such that data may be shifted into and shifted out of them is called shift register. data may be shifted into or out of the register in serial form or in parallel form. There are four basic types of shift registers.**

1. **Serial in, serial out, shift right, shift registers**
2. **Serial in, serial out, shift left, shift registers**
3. **Parallel in, serial out shift registers**
4. **Parallel in, parallel out shift registers**

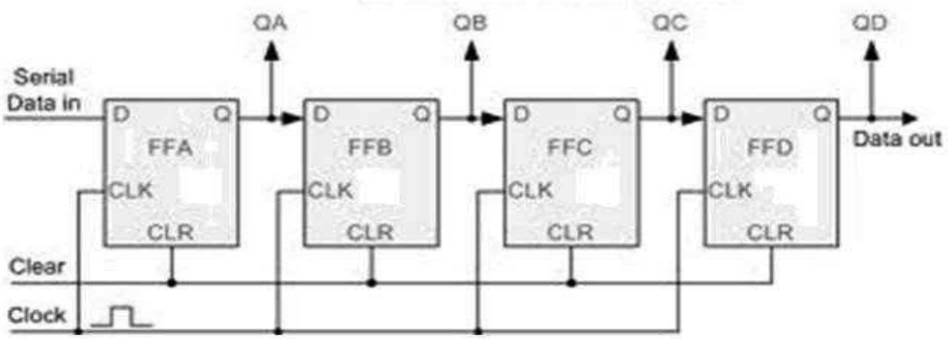
**Serial IN, serial OUT, shift right, shift left register:**

**The logic diagram of 4-bit serial in serial out, right shift register with four stages. The register can store four bits of data. Serial data is applied at the input D of the first FF. the Q output of the first FF is connected to the D input of another FF. the data is outputted from the Q terminal of the last FF.**



**When serial data is transferred into a register, each new bit is clocked into the first FF at the positive going edge of each clock pulse. The bit that was previously stored by the first FF is transferred to the second FF. the bit that was stored by the Second FF is transferred to the third FF.**

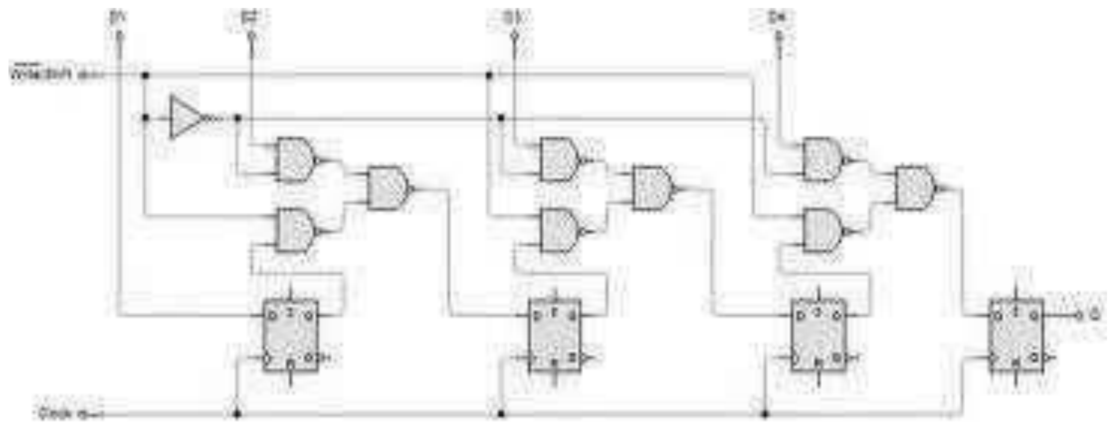
**Serial-in, parallel-out, shift register:**



**In this type of register, the data bits are entered into the register serially, but the data stored in the register is shifted out in parallel form.**

**Once the data bits are stored, each bit appears on its respective output line and all bits are available simultaneously, rather than on a bit-by-bit basis with the serial output. The serial-in, parallel out, shift register can be used as serial-in, serial out, shift register if the output is taken from the Q terminal of the last FF.**

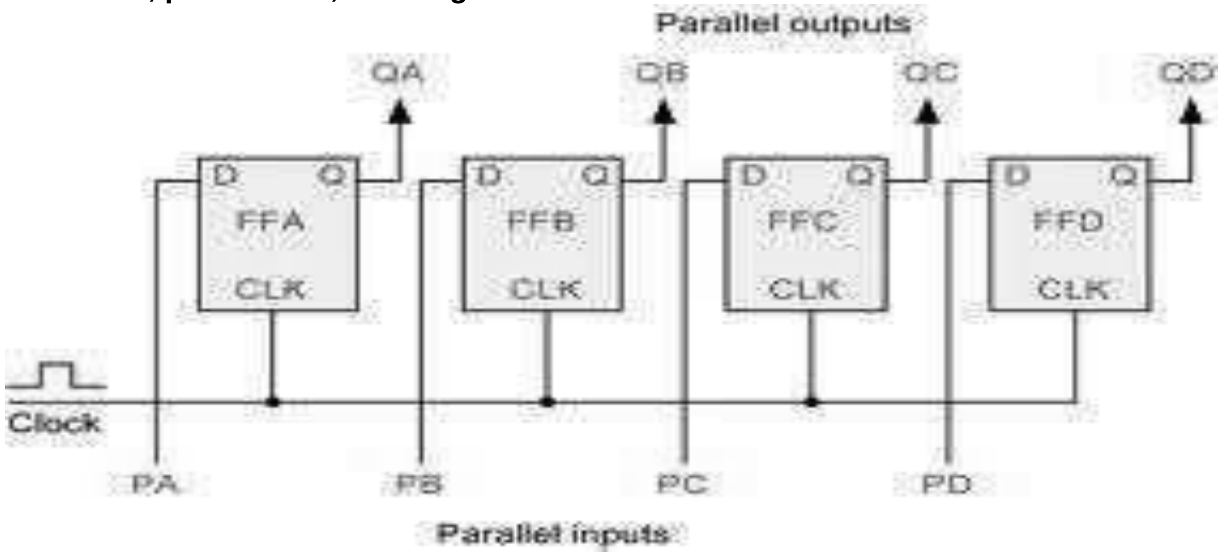
**Parallel-in, serial-out, shift register:**



**For a parallel-in, serial out, shift register, the data bits are entered simultaneously into their respective stages on parallel lines, rather than on a bit-by-bit basis on one line as with serial data bits are transferred out of the register serially. On a bit-by-bit basis over a single line.**

**There are four data lines A,B,C,D through which the data is entered into the register in parallel form. The signal shift/ load allows the data to be entered in parallel form into the register and the data is shifted out serially from terminalQ4**

**Parallel-in, parallel-out, shift register**



**In a parallel-in, parallel-out shift register, the data is entered into the register in parallel form, and also the data is taken out of the register in parallel form. Data is applied to the D input terminals of the FF‘s. When a clock pulse is applied, at the positive going edge of the pulse, the D inputs are shifted into the Q outputs of the FFs. The register now stores the data. The stored data is available instantaneously for shifting out in parallel form.**

**Bidirectional shift register:**

**A bidirectional shift register is one which the data bits can be shifted from left to right or from right to left. A fig shows the logic diagram of a 4-bit serial-in, serial out, bidirectional shift register. Right/left is the mode signal, when right /left is a 1, the logic circuit works as a shift-register.the bidirectional operation is achieved by using the mode signal and two NAND gates and one OR gate for each stage.**

**A HIGH on the right/left control input enables the AND gates G1, G2, G3 and G4 and disables the AND gates G5,G6,G7 and G8, and the state of Q output of each FF is passed through the gate to the D input of the following FF. when a clock pulse occurs, the data bits are then effectively shifted one place to the right. A LOW on the right/left control inputs enables the AND gates G5, G6, G7 and G8 and disables the And gates G1, G2, G3 and G4 and the Q output of each FF is passed to the D input of the preceding FF. when a clock pulse occurs, the data bits are then effectively shifted one place to the left. Hence, the circuit works as a bidirectional shift register**

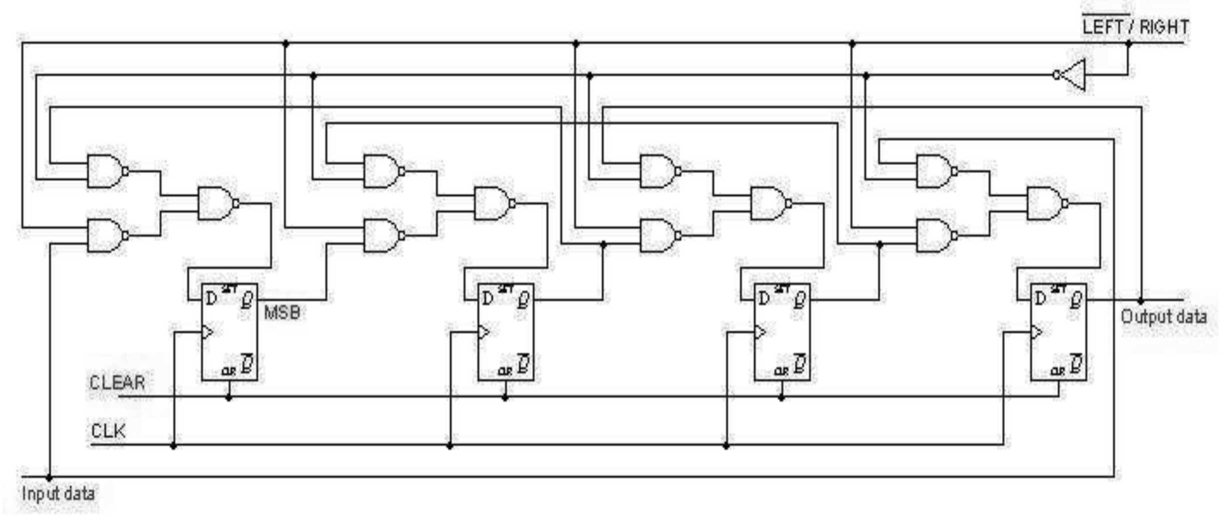


**Figure: logic diagram of a 4-bit bidirectional shift register**

**Universal shift register:**

**A register is capable of shifting in one direction only is a unidirectional shift register. One that can shift 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 registers. Universal shift register is a bidirectional register, whose input can be either in serial form or in parallel form and whose output also can be in serial form or I parallel form. The most general shift register has the following capabilities.**

1. **A clear control to clear the register to 0**
2. **A clock input to synchronize the operations**
3. **A shift-right control to enable the shift-right operation and serial input and output lines associated with the shift-right**

4. **A shift-left control to enable the shift-left operation and serial input and output lines associated with the shift-left**
5. **A parallel loads control to enable a parallel transfer and the n input lines associated with the parallel transfer**
6. **N parallel output lines**
7. **A control state that leaves the information in the register unchanged in the presence of the clock.**

**A universal shift register can be realized using multiplexers. The below fig shows the logic diagram of a 4-bit universal shift register that has all capabilities. It consists of 4 D flip-flops and four multiplexers. The four multiplexers have two common selection inputs s1 and s0. Input 0 in each multiplexer is selected when S1S0=00, input 1 is selected when S1S0=01 and input 2 is selected when S1S0=10 and input 4 is selected when S1S0=11. The selection inputs control the mode of operation of the register according to the functions entries. When S1S0=0, the present value of the register is applied to the D inputs of flip-flops. The condition forms a path from the output of each flip-flop into the input of the same flip-flop. The next clock edge transfers into each flip-flop the binary value it held previously, and no change of state occurs. When S1S0=01, terminal 1 of the multiplexer inputs have a path to the D inputs of the flip-flop. This causes a shift-right operation, with serial input transferred into flip-flopA4. When S1S0=10, a shift left operation results with the other serial input going into flip-flop A1. Finally when S1S0=11, the binary information on the parallel input lines is transferred into the register simultaneously during the next clock cycle**

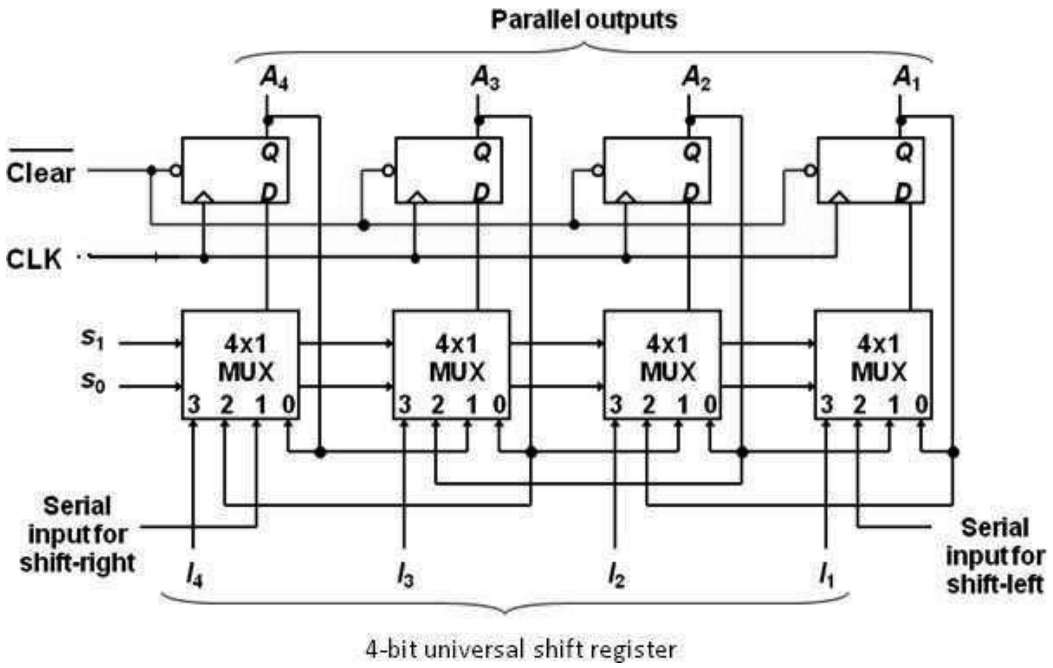

Parallel outputs
A4
A3
A2
A1
Q
D
Clear
CLK
s1
s0
4x1 MUX
3 2 1 0
Serial input for shift-right
I4
I3
I2
I1
Serial input for shift-left
4-bit universal shift register

**Figure: logic diagram 4-bit universal shift register**

**Function table for theregister**

| | | mode control |
|---|---|---|
| S0 | S1 | register operation |
| | | |
| 0 | 0 | No change |
| 0 | 1 | Shift Right |
| 1 | 0 | Shift left |
| 1 | 1 | Parallel load |

**Counters:**

**Counter is a device which stores (and sometimes displays) the number of times particular event or process has occurred, often in relationship to a clock signal. A Digital counter is a set of flip flops whose state change in response to pulses applied at the input to the counter. Counters may be asynchronous counters or synchronous counters. Asynchronous counters are also called ripple counters**

**In electronics counters can be implemented quite easily using register-type circuits such as the flip-flops and a wide variety of classifications exist:**

**Asynchronous (ripple) counter – changing state bits are used as clocks to subsequent state flip-flops**
**Synchronous counter – all state bits change under control of a singleclock Decade counter – counts through ten states per stage**
**Up/down counter – counts both up and down, under command of a control input Ring counter – formed by a shift register with feedback connection in a ring**
**Johnson counter – a twisted ring counter**

- **Cascaded counter**
- **Modulus counter.**

**Each is useful for different applications. Usually, counter circuits are digital in nature, and count in natural binary Many types of counter circuits are available as digital building blocks, for example a number of chips in the 4000 series implement different counters.**

**Occasionally there are advantages to using a counting sequence other than the natural binary sequence such as the binary coded decimal counter, a linear feed-back shift register counter, or a gray-code counter.**

**Counters are useful for digital clocks and timers, and in oven timers, VCR clocks, etc.**

## Asynchronous counters:

**An asynchronous (ripple) counter is a single JK-type flip-flop, with its J (data) input fed from its own inverted output. This circuit can store one bit, and hence can count from zero to one before it overflows (starts over from 0). This counter will increment once for every clock cycle and takes two clock cycles to overflow, so every cycle it will alternate between a transition from 0 to 1 and a transition from 1 to 0. Notice that this creates a new clock with a 50% duty cycle at exactly half the frequency of the input clock. If this output is then used as the clock signal for a similarly arranged D flip-flop (remembering to invert the output to the input), one will get another 1 bit counter that counts half as fast. Putting them together yields a two-bit counter:**

## Two-bit ripple up-counter using negative edge triggered flip flop:

**Two bit ripple counter used two flip-flops. There are four possible states from 2 – bit up-counting I.e. 00, 01, 10 and 11.**

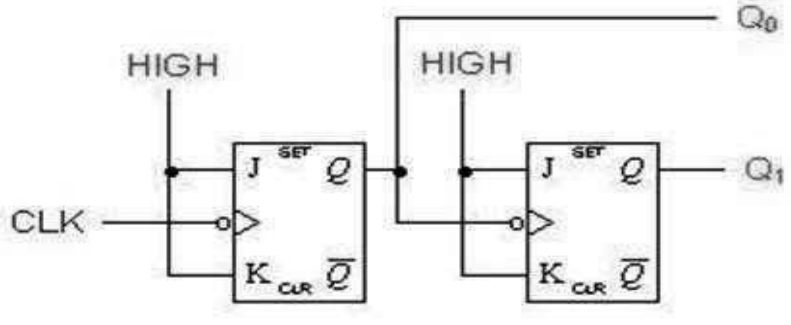
- **The counter is initially assumed to be at a state 00 where the outputs of the tow flip-flops are noted as $Q_1Q_0$. Where $Q_1$ forms the MSB and $Q_0$ forms the LSB.**

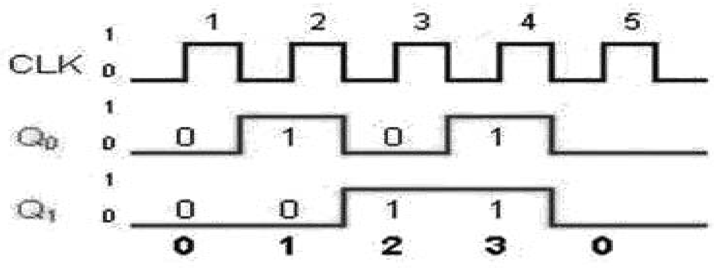
- **For the negative edge of the first clock pulse, output of the first flip-flop $FF_1$ toggles its state. Thus $Q_1$ remains at 0 and $Q_0$ toggles to 1 and the counter state are now read as 01.**

- **During the next negative edge of the input clock pulse $FF_1$ toggles and $Q_0 = 0$. The output Q0 being a clock signal for the second flip-flop $FF_2$ and the present transition acts as a negative edge for $FF_2$ thus toggles its state $Q_1 = 1$. The counter state is now read as 10.**

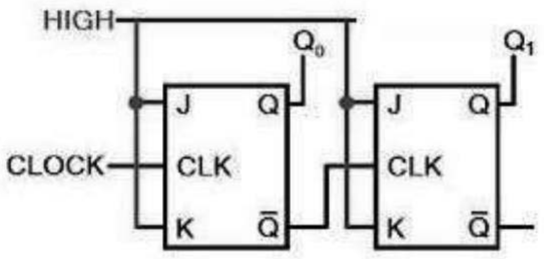
- **For the next negative edge of the input clock to $FF_1$ output Q0 toggles to 1. But this transition from 0 to 1 being a positive edge for $FF_2$ output $Q_1$ remains at 1. The counter state is now read as 11.**

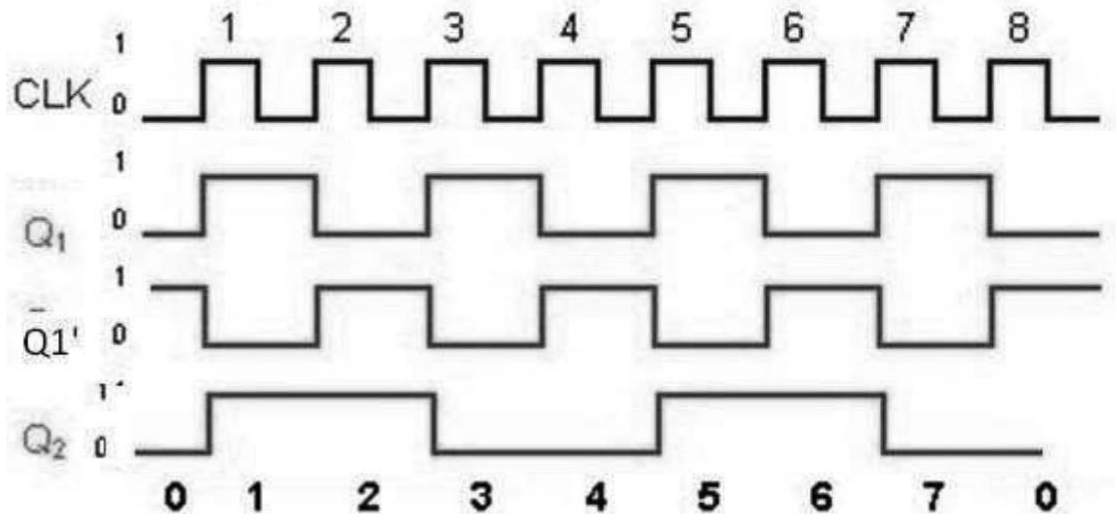
- **For the next negative edge of the input clock, $Q_0$ toggles to 0. This transition from 1 to 0 acts as a negative edge clock for $FF_2$ and its output $Q_1$ toggles to 0. Thus the starting state 00 is attained. Figure shown below**





**Two-bit ripple down-counter using negative edge triggered flip flop:**





**A 2-bit down-counter counts in the order 0,3,2,1,0,1…….,i.e, 00,11,10,01,00,11 …..,etc. the above fig. shows ripple down counter, using negative edge triggered J-K FFs and its timing diagram.**
**For down counting, Q1‘ of FF1 is connected to the clock of Ff2. Let initially all the FF1 toggles, so, Q1 goes from a 0 to a 1 and Q1‘ goes from a 1 to a 0.**

**The negative-going signal at Q1‘ is applied to the clock input of FF2, toggles Ff2 and, therefore, Q2 goes from a 0 to a 1.so, after one clock pulse Q2=1 and Q1=1, I.e., the state of the counter is 11.**

**At the negative-going edge of the second clock pulse, Q1 changes from a 1 to a 0 and Q1‘ from a 0 to a 1.**

**This positive-going signal at Q1‘ does not affect FF2 and, therefore, Q2 remains at a 1. Hence , the state of the counter after second clock pulse is 10**

**At the negative going edge of the third clock pulse, FF1 toggles. So Q1, goes from a 0 to a 1 and Q1‘ from 1 to 0. This negative going signal at Q1‘ toggles FF2 and, so, Q2 changes from 1 to 0, hence, the state of the counter after the third clock pulse is 01.**

**At the negative going edge of the fourth clock pulse, FF1 toggles. So Q1, goes from a 1 to a 0 and Q1‘ from 0 to 1. . This positive going signal at Q1‘ does not affect FF2 and, so, Q2 remains at 0, hence, the state of the counter after the fourth clock pulse is 00.**

**Two-bit ripple up-down counter using negative edge triggered flip flop:**

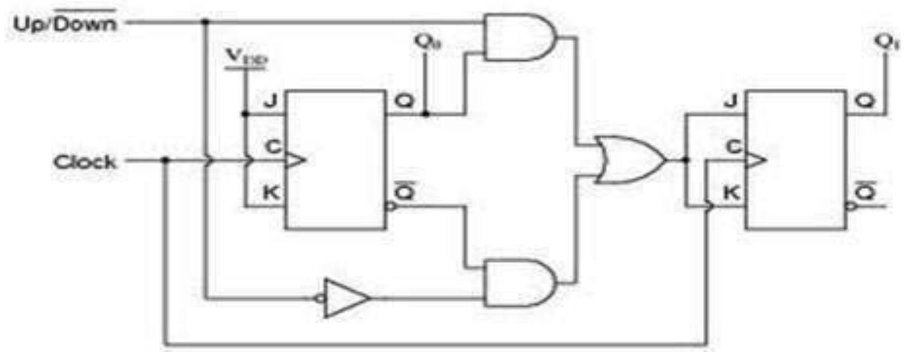


**Figure: asynchronous 2-bit ripple up-down counter using negative edge triggered flip flop:**

**As the name indicates an up-down counter is a counter which can count both in upward and downward directions. An up-down counter is also called a forward/backward counter or a bidirectional counter. So, a control signal or a mode signal M is required to choose the direction of count. When M=1 for up counting, Q1 is transmitted to clock of FF2 and when M=0 for down counting, Q1‘ is transmitted to clock of FF2. This is achieved by using two AND gates and one OR gates. The external clock signal is applied to FF1.**

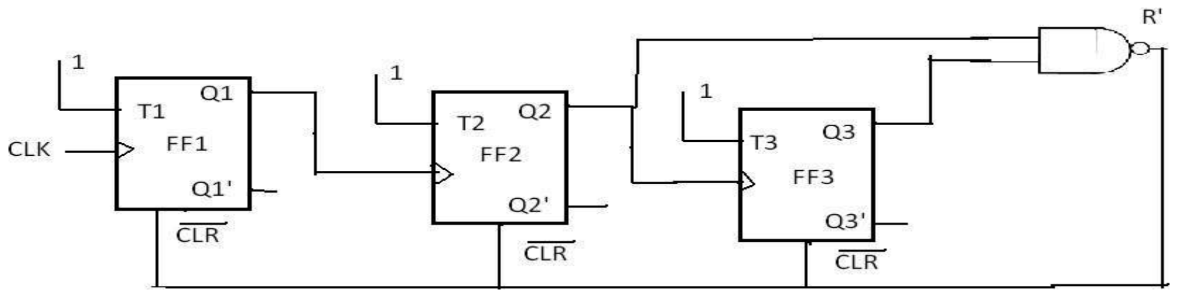
**Clock signal to FF2= (Q1.Up)+(Q1‘. Down)= Q1m+Q1‘M‘**

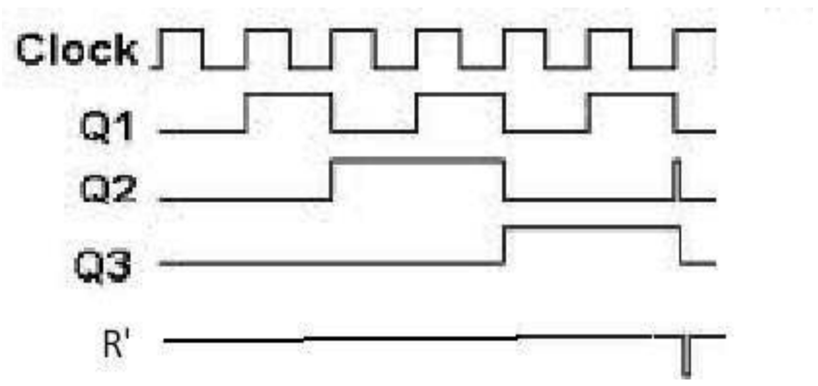
**Design of Asynchronous counters:**

**To design a asynchronous counter, first we write the sequence , then tabulate the values of reset signal R for various states of the counter and obtain the minimal expression for R and R‘ using K-Map or any other method. Provide a feedback such that R and R‘ resets all the FF‘s after the desired count**

## Design of a Mod-6 asynchronous counter using T FFs:

**A mod-6 counter has six stable states 000, 001, 010, 011, 100, and 101. When the sixth clock pulse is applied, the counter temporarily goes to 110 state, but immediately resets to 000 because of the feedback provided. it is —divide by-6-counter‖, in the sense that it divides the input clock frequency by 6.it requires three FFs, because the smallest value of n satisfying the conditionN≤$2^n$ is n=3; three FFs can have 8 possible states, out of which only six are utilized and the remaining two states 110and 111, are invalid. If initially the counter is in 000 state, then after the sixth clock pulse, it goes to 001, after the second clock pulse, it goes to 010, and so on.**





**After sixth clock pulse it goes to 000. For the design, write the truth table with present state outputs Q3, Q2 and Q1 as the variables, and reset R as the output and obtain an expression for R in terms of Q3, Q2, and Q1that decides the feedback into be provided. From the truth table, R=Q3Q2. For active-low Reset, R‘ is used. The reset pulse is of very short duration, of the order of nanoseconds and it is equal to the propagation delay time of the NAND gate used. The expression for R can also be determined as follows.**

**R=0 for 000 to 101, R=1 for 110, and R=X=for111**
**Therefore,**
**R=Q3Q2Q1‘+Q3Q2Q1=Q3Q2**

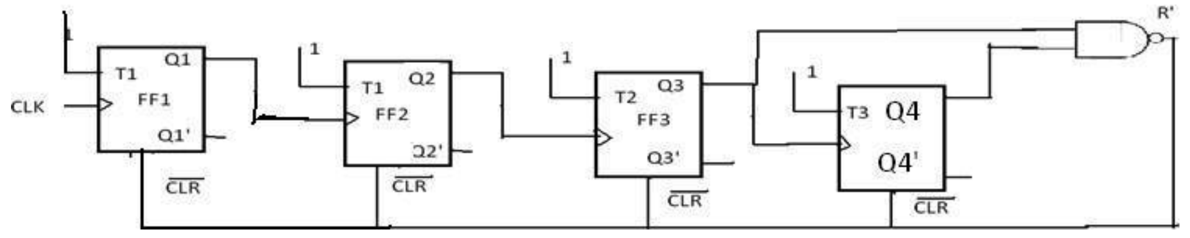
**The logic diagram and timing diagram of Mod-6 counter is shown in the above fig.**

**The truth table is as shown in below.**

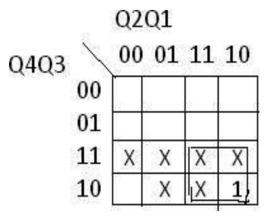
| After | States | | | |
|---|---|---|---|---|
| pulses | Q3 | Q2 | Q1 | R |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 1 | 0 |
| 4 | 1 | 0 | 0 | 0 |
| 5 | 1 | 0 | 1 | 0 |
| 6 | 1 | 1 | 0 | 1 |
| | ↓ | ↓ | ↓ | |
| | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 |

## Design of a mod-10 asynchronous counter using T-flip-flops:

**A mod-10 counter is a decade counter. It also called a BCD counter or a divide-by-10 counter. It requires four flip-flops (condition 10 ≤$2^n$ is n=4). So, there are 16 possible states, out of which ten are valid and remaining six are invalid. The counter has ten stable state, 0000 through 1001, i.e., it counts from 0 to 9. The initial state is 0000 and after nine clock pulses it goes to 1001. When the tenth clock pulse is applied, the counter goes to state 1010 temporarily, but because of the feedback provided, it resets to initial state 0000. So, there will be a glitch in the waveform of Q2. The state 1010 is a temporary state for which the reset signal R=1, R=0 for 0000 to 1001, and R=C for 1011 to 1111.**



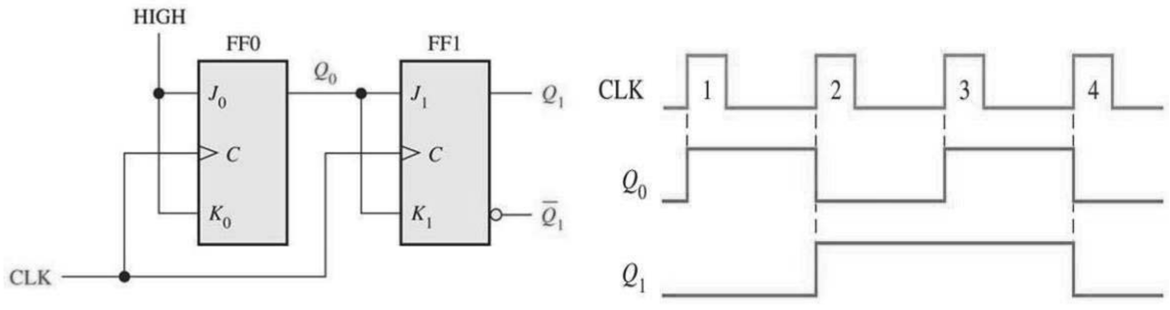
**The count table and the K-Map for reset are shown in fig. from the K-Map R=Q4Q2. So, feedback is provided from second and fourth FFs. For active –HIGH reset, Q4Q2 is applied to the clear terminal. For active-LOW reset 4 2 is connected isof all Flip=flops.**



| After pulses | Count Q4 | Q3 | Q2 | Q1 |
|---|---|---|---|---|
| 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 | 0 | 0 | 1 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 0 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 |
| 9 | 0 | 1 | 0 | 1 |
| 10 | 0 | 0 | 0 | 0 |

## Synchronous counters:

**Asynchronous counters are serial counters. They are slow because each FF can change state only if all the preceding FFs have changed their state. if the clock frequency is very high, the asynchronous counter may skip some of the states. This problem is overcome in synchronous counters or parallel counters. Synchronous counters are counters in which all the flip flops are triggered simultaneously by the clock pulses Synchronous counters have a common clock pulse applied simultaneously to all flip flops. □ A 2-Bit Synchronous Binary Counter**



## Design of synchronous counters:

**For a systematic design of synchronous counters. The following procedure is used.**

**Step 1:State Diagram: draw the state diagram showing all the possible states state diagram which also be called nth transition diagrams, is a graphical means of depicting the sequence of states through which the counter progresses.**

**Step2: number of flip-flops: based on the description of the problem, determine the required number n of the flip-flops- the smallest value of n is such that the number of states N≤$2^n$--- and the desired counting sequence.**

**Step3: choice of flip-flops excitation table: select the type of flip-flop 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 FF using K-maps drawn for the excitation of the flip-flops in terms of the present states and inputs.**

**Step5: logic diagram: draw a logic diagram based on the minimal expressions**

**Design of a synchronous 3-bit up-down counter using JK flip-flops:**

**Step1: determine the number of flip-flops required. A 3-bit counter requires three FFs. It has 8 states (000,001,010,011,101,110,111) and all the states are valid. Hence no don‘t cares. For selecting up and down modes, a control or mode signal M is required. When the mode signal M=1 and counts down when M=0. The clock signal is applied to all the FFs simultaneously.**

**Step2: draw the state diagrams: the state diagram of the 3-bit up-down counter is drawn as**

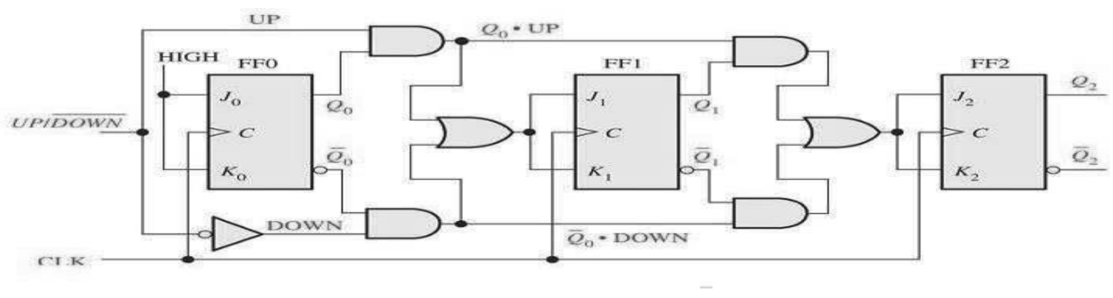
**Step3: select the type of flip flop and draw the excitation table: JK flip-flops are selected and the excitation table of a 3-bit up-down counter using JK flip-flops is drawn as shown in fig.**

| PS | | | mode | NS | | | required excitations | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Q3 | Q2 | Q1 | M | Q3 | Q2 | Q1 | J3 | K3 | J2 | K2 | J1 | K1 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | x | 1 | x | 1 | x |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | x | 0 | x | 1 | x |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | x | 0 | x | x | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | x | 1 | x | x | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | x | x | 1 | 1 | x |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | x | x | 0 | 1 | x |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | x | x | 0 | x | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | x | x | 1 | x | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | x | 1 | 1 | x | 1 | x |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | x | 0 | 0 | x | 1 | x |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | x | 0 | 0 | x | x | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | x | 0 | 1 | x | x | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | x | 0 | x | 1 | 1 | x |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | x | 0 | x | 0 | 1 | x |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 | x | 0 | x | 0 | x | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | x | 1 | x | 1 | x | 1 |

**Step4: obtain the minimal expressions: From the excitation table we can conclude that J1=1 and K1=1, because all the entries for J1and K1 are either X or 1. The K-maps for J3, K3,J2 and K2 based on the excitation table and the minimal expression obtained from them are shown in fig.**

| Q3Q2 \ Q1M | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| | 1 | | | |
| | | | 1 | |
| | X | X | X | X |
| | X | X | X | X |

**Step5: draw the logic diagram: a logic diagram using those minimal expressions can be drawn as shown in fig.**



**Design of a synchronous modulo-6 gray cod counter:**

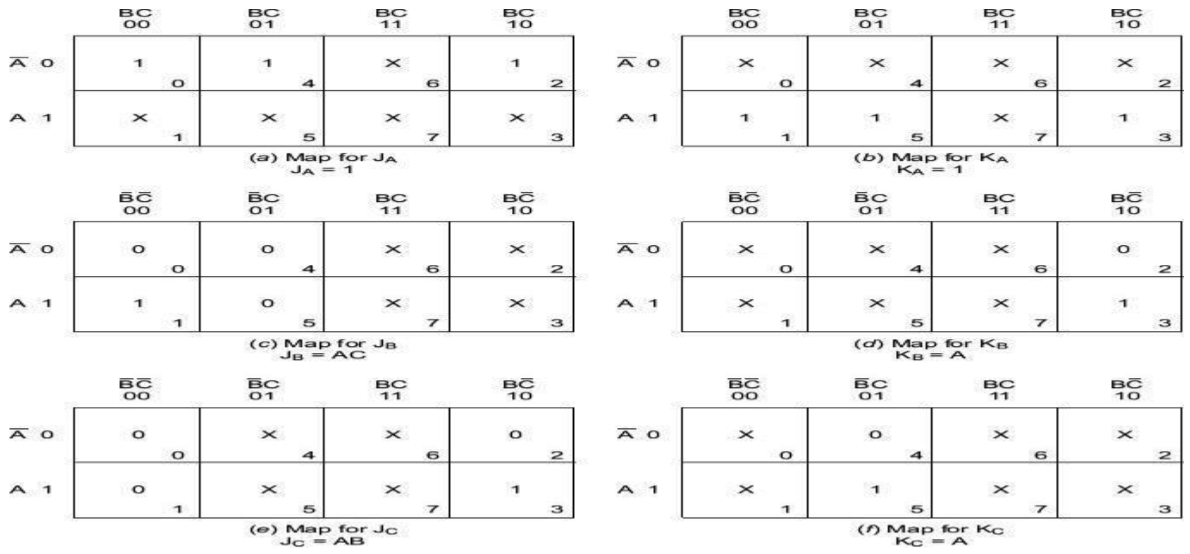
**Step 1: the number of flip-flops: we know that the counting sequence for a modulo-6 gray code counter is 000, 001, 011, 010, 110, and 111. It requires n=3FFs ($N \le 2^n$, i.e., $6 \le 2^3$). 3 FFs can have 8 states. So the remaining two states 101 and 100 are invalid. The entries for excitation corresponding to invalid states are don't cares.**
**Step2: the state diagram: the state diagram of the mod-6 gray code converter is drawn as shown in fig.**

$$\begin{tikzcd}
000 \arrow[r] \arrow[d] & 001 \arrow[r] & 011 \arrow[d] \\
111 \arrow[r] & 110 \arrow[r] & 010
\end{tikzcd}$$

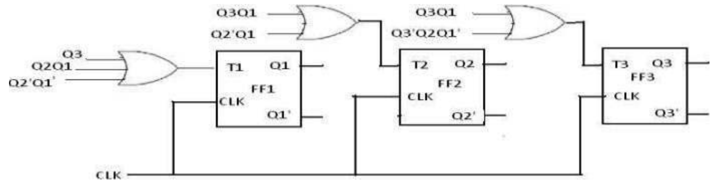
**Step3: type of flip-flop and the excitation table: T flip-flops are selected and the excitation table of the mod-6 gray code counter using T-flip-flops is written as shown in fig.**

| PS | | | NS | | | required excitations | | |
|---|---|---|---|---|---|---|---|---|
| Q3 | Q2 | Q1 | Q3 | Q2 | Q1 | T3 | T2 | T1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |

**Step4: The minimal expressions: the K-maps for excitations of FFs T3,T2,and T1 in terms of outputs of FFs Q3,Q2, and Q1, their minimization and the minimal expressions for excitations obtained from them are shown if fig**



**Step5: the logic diagram: the logic diagram based on those minimal expressions is drawn as shown in fig.**



**Design of a synchronous BCD Up-Down counter using FFs:**

**Step1: the number of flip-flops: a BCD counter is a mod-10 counter has 10 states (0000 through 1001) and so it requires n=4FFs($N\leq 2^n$,, i.e., $10\leq 2^4$). 4 FFS can have 16 states. So out of 16 states, six states (1010 through 1111) are invalid. For selecting up and down mode, a control or mode signal M is required. , it counts up when M=1 and counts down when M=0. The clock signal is applied to all FFs.**

**Step2: the state diagram: The state diagram of the mod-10 up-down counter is drawn as shown in fig.**

**Step3: types of flip-flops and excitation table: T flip-flops are selected and the excitation table of the modulo-10 up down counter using T flip-flops is drawn as shown in fig.**

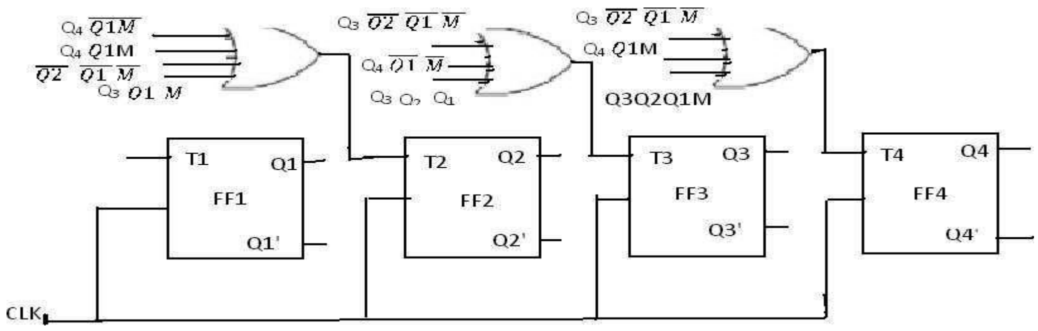
**The remaining minterms are don‘t cares(∑d(20,21,22,23,24,25,26,37,28,29,30,31)) from the excitation table we can see that T1=1 and the expression for T4,T3,T2 are asfollows.**

**T4=∑m(0,15,16,19)+d(20,21,22,23,24,25,26,27,28,29,30,31)**
**T3=∑m(7,15,16,8)+d(20,21,22,23,24,25,26,27,28,29,30,31)**
**T2=∑m(3,4,7,8,11,12,15,16)+d(20,21,22,23,24,25,26,27,28,29,30,31)**

| PS | | | | mode | NS | | | | required excitations | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Q4 | Q3 | Q2 | Q1 | M | Q4 | Q3 | Q2 | Q1 | T4 | T3 | T2 | T1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |

**Step4: The minimal expression: since there are 4 state variables and a mode signal, we require 5 variable kmaps. 20 conditions of Q4Q3Q2Q1M are valid and the remaining 12 combinations are invalid. So the entries for excitations corresponding to those invalid combinations are don‘t cares. Minimizing K-maps for T2 we get**
**T 2= Q4Q1‘M+Q4‘Q1M+Q2Q1‘M‘+Q3Q1‘M‘**

**Step5: the logic diagram: the logic diagram based on the above equation is shown in fig.**


$Q_4\overline{Q1M}$
$Q_4 Q1M$
$\overline{Q2}\ \overline{Q1}\ \overline{M}$
$Q_3\ Q1\ M$
$Q_3\ \overline{Q2}\ \overline{Q1}\ \overline{M}$
$Q_4\ \overline{Q1}\ \overline{M}$
$Q_3\ Q_2\ Q_1$
$Q_3\ \overline{Q2}\ \overline{Q1}\ \overline{M}$
$Q_4\ Q1M$
Q3Q2Q1M
T1
Q1
FF1
Q1'
T2
Q2
FF2
Q2'
T3
Q3
FF3
Q3'
T4
Q4
FF4
Q4'
CLK

**Shift register counters:**
**One of the applications of shift register is that they can be arranged to form several types of counters. The most widely used shift register counter is ring counter as well as the twisted ring counter.**

**Ring counter: this is the simplest shift register counter. The basic ring counter using D flip-flops is shown in fig. the realization of this counter using JK FFs. The Q output of each stage is connected to the D flip-flop connected back to the ring counter.**

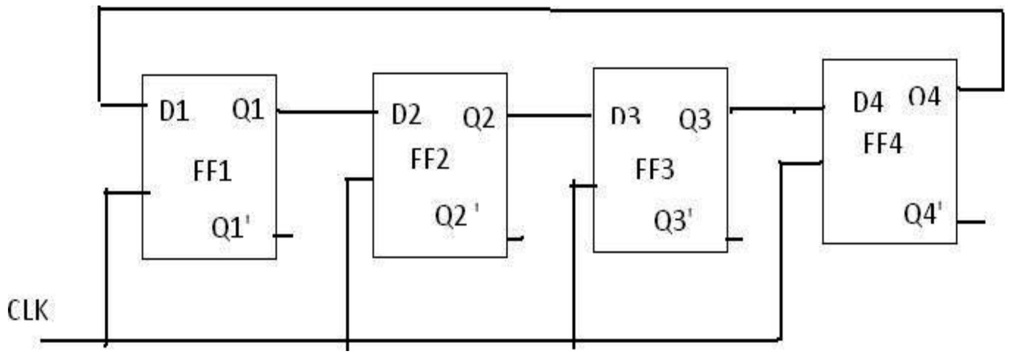
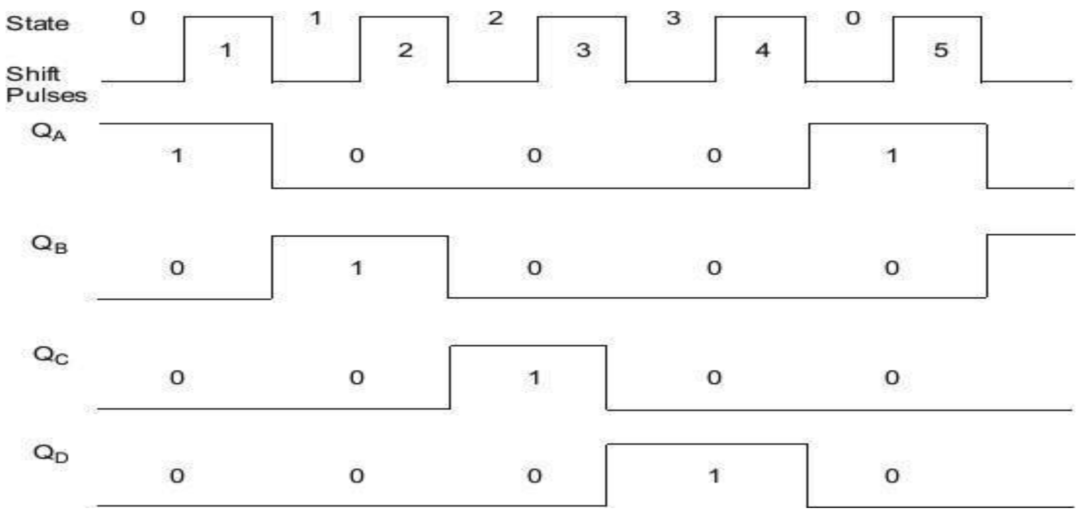


**FIGURE: logic diagram of 4-bit ring counter using D flip-flops**

**Only a single 1 is in the register and is made to circulate around the register as long as clock pulses are applied. Initially the first FF is present to a 1. So, the initial state is 1000, i.e., Q1=1, Q2=0,Q3=0,Q4=0. After each clock pulse, the contents of the register are shifted to the right by one bit and Q4 is shifted back to Q1. The sequence repeats after four clock pulses. The number**

**of distinct states in the ring counter, i.e., the mod of the ring counter is equal to number of FFs used in the counter. An n-bit ring counter can count only n bits, where as n-bit ripple counter can count $2^n$ bits. So, the ring counter is uneconomical compared to a ripple counter but has advantage of requiring no decoder, since we can read the count by simply noting which FF is set. Since it is entirely a synchronous operation and requires no gates external FFs, it has the further advantage of being very fast.**

**Timing diagram:**



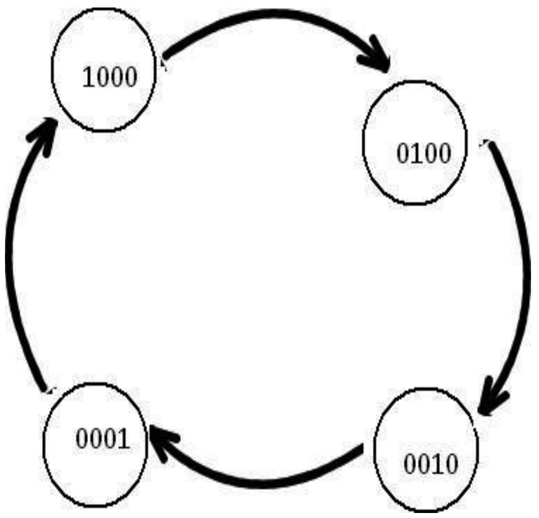


**Figure: state diagram**

**Twisted Ring counter (Johnson counter):**

**This counter is obtained from a serial-in, serial-out shift register by providing feedback from the inverted output of the last FF to the D input of the first FF. the Q output of each is connected to the D input of the next stage, but the Q' output of the last stage is connected to the D input of the first stage, therefore, the name twisted ring counter. This feedback arrangement produces a unique sequence of states.**

**The logic diagram of a 4-bit Johnson counter using D FF is shown in fig. the realization of the same using J-K FFs is shown in fig.. The state diagram and the sequence table are shown in figure. The timing diagram of a Johnson counter is shown in figure.**

**Let initially all the FFs be reset, i.e., the state of the counter be 0000. After each clock pulse, the level of Q1 is shifted to Q2, the level of Q2to Q3, Q3 to Q4 and the level of Q4'to Q1 and the sequences given in fig.**

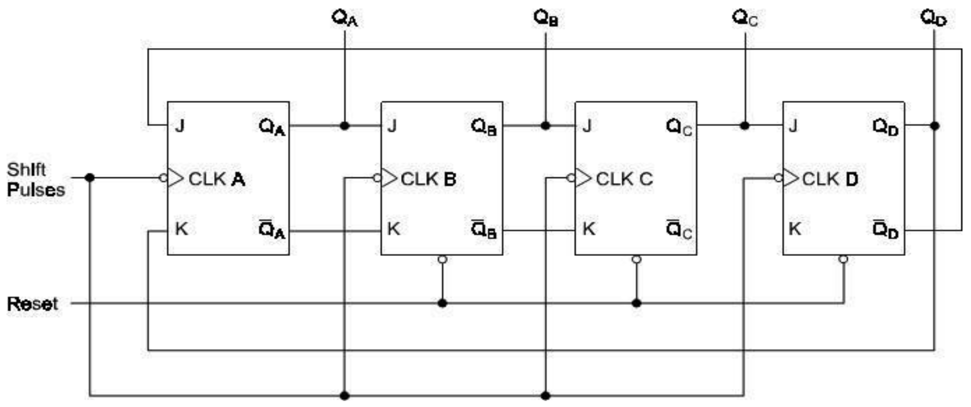


**Figure: Johnson counter with JK flip-flops**

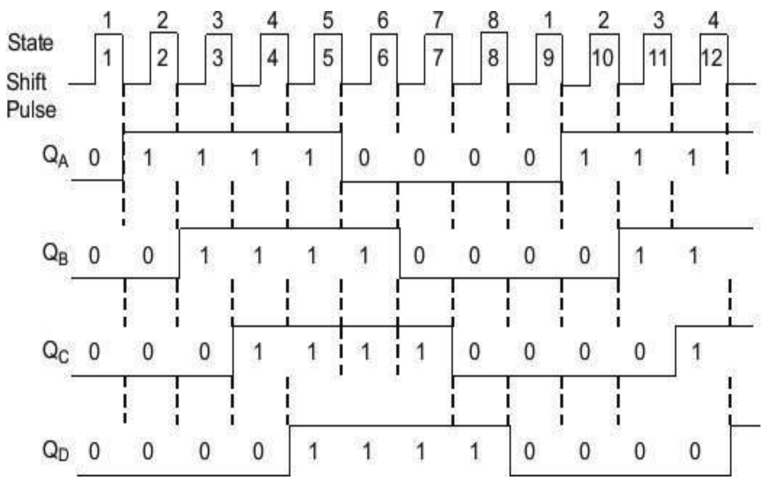
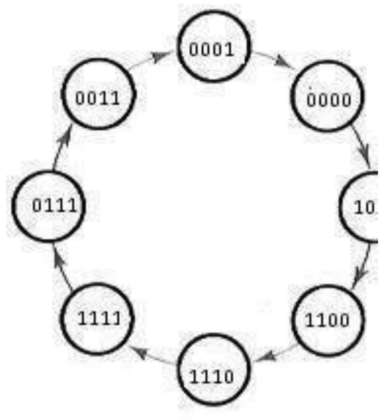


**Figure: timing diagram**

**State diagram:**



| Q1 | Q2 | Q3 | Q4 | after clock pulse |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 2 |
| 1 | 1 | 1 | 0 | 3 |
| 1 | 1 | 1 | 1 | 4 |
| 0 | 1 | 1 | 1 | 5 |
| 0 | 0 | 1 | 1 | 6 |
| 0 | 0 | 0 | 1 | 7 |
| 0 | 0 | 0 | 0 | 8 |
| 1 | 0 | 0 | 0 | 9 |

**Excitation table**

**Synthesis of sequential circuits:**

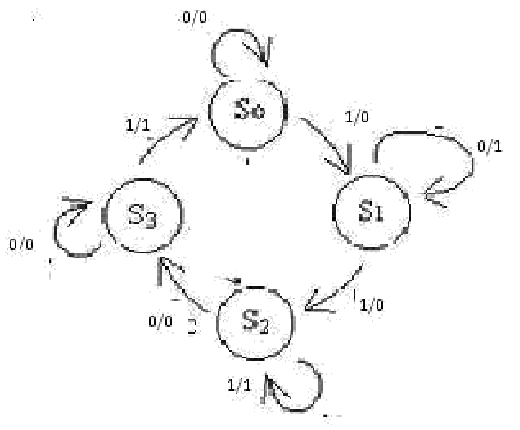
**The synchronous or clocked sequential circuits are represented by two models.**

1. **Moore circuit: in this model, the output depends only on the present state of the flip-flops**
2. **Meelay circuit: in this model, the output depends on both present state of the flip-flop. And the inputs.**

**Sequential circuits are also called finite state machines (FSMs). This name is due to the fast that the functional behavior of these circuits can be represented using a finite number of states.**

**State diagram: the state diagram or state graph is a pictorial representation of the relationships between the present state, the input, the next state, and the output of a sequential circuit. The state diagram is a pictorial representation of the behavior of a sequential circuit.**

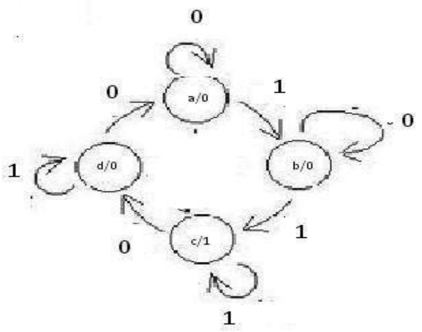
**The state represented by a circle also called the node or vertex and the transition between states is indicated by directed lines connecting circle. a directed line connecting a circle with itself indicates that the next state is the same as the present state. The binary number inside each circle identifies the state represented by the circle. The direct lines are labeled with two binary numbers separated by a symbol. The input value is applied during the present state is labeled after the symbol.**



|  | NS,O/P INPUT X |  |
|---|---|---|
| PS | X=0 | X=1 |
| a | a,0 | b,0 |
| b | b,1 | c,0 |
| c | d,0 | c,1 |
| d | d,0 | a,1 |

**Fig :a) state diagram (meelay circuit)** **fig: b) state table**

**In case of moore circuit ,the directed lines are labeled with only one binary number representing the input that causes the state transition. The output is indicated with in the circle below the present state, because the output depends only on the present state and not on the input.**



|  | NS INPUT X |  |  |
|---|---|---|---|
| PS | X=0 | X=1 | O/P |
| a | a | b | 0 |
| b | b | c | 0 |
| c | d | c | 1 |
| d | a | d | 0 |

**Fig: a) state diagram (moore circuit)** **fig:b) state table**

## Serial binary adder:

**Step1: word statement of the problem: the block diagram of a serial binary adder is shown in fig. it is a synchronous circuit with two input terminals designated X1and X2 which carry the two binary numbers to be added and one output terminal Z which represents the sum. The inputs and outputs consist of fixed-length sequences 0s and 1s.the output of the serial $Z_i$ at time $t_i$is a function of the inputs $X_1(t_i)$ and $X_2(t_i)$ at that time $t_{i-1}$ and of carry which had been generated at $t_{i-1}$. The carry which represent the past history of the serial adder may be a 0 or 1. The circuit has two states. If one state indicates that carry from the previous addition is a 0, the other state indicates that the carry from the previous addition is a 1**

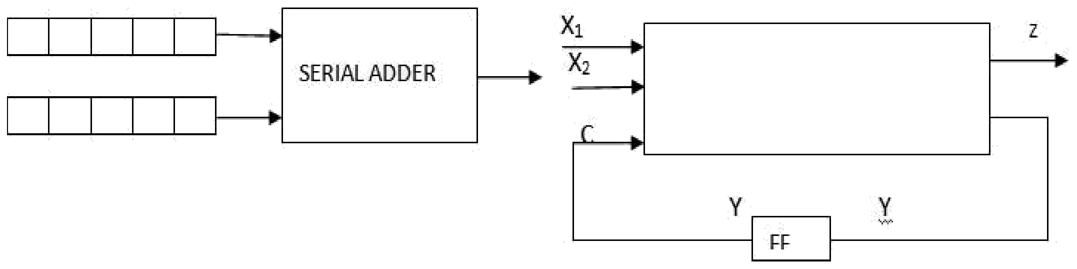
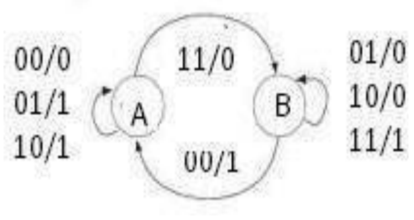


**Figure: block diagram of serial binary adder**

**Step2 and 3: state diagram and state table: let a designate the state of the serial adder at $t_i$ if a carry 0 was generated at $t_{i-1}$, and let b designate the state of the serial adder at $t_i$ if carry 1 was generated at $t_{i-1}$ .the state of the adder at that time when the present inputs are applied is referred to as the present state(PS) and the state to which the adder goes as a result of the new carry value is referred to as next state(NS).**
**The behavior of serial adder may be described by the state diagram and state table.**



| PS | NS ,O/P X1 X2 | | | |
|---|---|---|---|---|
| | 0 0 | 0 1 | 1 0 | 1 1 |
| A | A,0 | B,0 | B,1 | B,0 |
| B | A,1 | B,0 | B,0 | B,1 |

**Figures: serial adder state diagram and state table**

**If the machine is in state B, i.e., carry from the previous addition is a 1, inputs $X_1$=0 and $X_2$=1 gives sum, 0 and carry 1. So the machine remains in state B and outputs a 0. Inputs $X_1$=1 and $X_2$=0 gives sum, 0 and carry 1. So the machine remains in state B and outputs a 0. Inputs $X_1$=1 and $X_2$=1 gives sum, 1 and carry 0. So the machine remains in state B and outputs a 1. Inputs $X_1$=0 and $X_2$=0 gives sum, 1 and carry 0. So the machine goes to state A and outputs a 1. The state table also gives the same information.**

**Setp4: reduced standard from state table: the machine is already in this form. So no need to do anything**

**Step5: state assignment and transition and output table:**
**The states, A=0 and B=1 have already been assigned. So, the transition and output table is as shown.**

| PS | NS | | | | O/P | | | |
|---|---|---|---|---|---|---|---|---|
| | 0<br>0 | 0<br>1 | 1<br>0 | 1<br>1 | 0<br>0 | 0<br>1 | 1<br>0 | 1<br>1 |
| 0<br>1 | 0<br>0 | 0<br>1 | 0<br>1 | 1<br>1 | 0<br>1 | 1<br>0 | 1<br>0 | 1<br>1 |

**STEP6: choose type of FF and excitation table: to write table, select the memory element the excitation table is as shown in fig.**

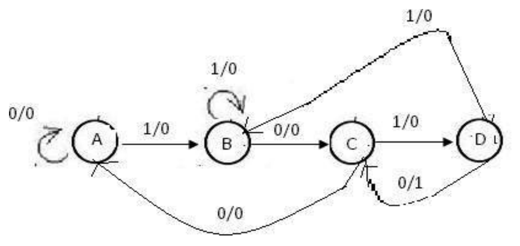
| PS | I/P | | NS | I/P-FF | O/P |
|---|---|---|---|---|---|
| y | x1 | x2 | Y | D | Z |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 |

**Sequence detector:**

**Step1: word statement of the problem: a sequence detector is a sequential machine which produces an output 1 every time the desired sequence is detected and an output 0 at all other times**

**Suppose we want to design a sequence detector to detect the sequence 1010 and say that overlapping is permitted i.e., for example, if the input sequence is 01101010 the corresponding output sequence is 00000101.**

**Step2 and 3: state diagram and state table: the state diagram and the state table of the sequence detector. At the time $t_1$, the machine is assumed to be in the initial state designed arbitrarily as A. while in this state, the machine can receive first bit input, either a 0 o r a 1. If the input bit is 0, the machine does not start the detection process because the first bit in the desired sequence is a 1. If the input bit is a 1 the detection process starts.**



| PS | NS,Z | |
|---|---|---|
| | X=0 | X=1 |
| A | A,0 | B,0 |
| B | C,0 | B,0 |
| C | A,0 | D,0 |
| D | C,1 | B,0 |

**Figure: state diagram and state table of sequence detector**

**So, the machine goes to state B and outputs a 0. While in state B, the machinery may receive 0 or 1 bit. If the bit is 0, the machine goes to the next state, say state c, because the previous two bits are 10 which are a part of the valid sequence, and outputs 0.. if the bit is a 1, the two bits become 11 and this not a part of the valid sequence**

**Step4: reduced standard form state table: the machine is already in this form. So no need to do anything.**

**Step5: state assignment and transition and output table: there are four states therefore two states variables are required. Two state variables can have a maximum of four states, so, all states are utilized and thus there are no invalid states. Hence, there are no don‘t cares. Let a=00, B=01, C=10 and D=11 be the state assignment.**

| | NS(Y1Y2) | | | O/P(z) | |
|---|---|---|---|---|---|
| **PS(y1y2** | **X=0** | **X=1** | | **X=0** | **X=1** |
| A=0 0 | 0 | 0 0 | 1 0 | 0 | |
| B=0 1 | 1 | 0 0 | 1 0 | 0 | |
| C=1 0 | 0 | 0 1 | 1 0 | 0 | |
| D=1 1 | 1 | 1 0 | 1 1 | | 0 |

**Step6: choose type of flip-flops and form the excitation table: select the D flip-flops as memory elements and draw the excitation table.**

| PS | | I/P | NS | | INPUTS - FFS | | O/P |
|---|---|---|---|---|---|---|---|
| **y1** | **Y2** | **X** | **Y1** | **Y2** | **D1** | **D2** | **Z** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |

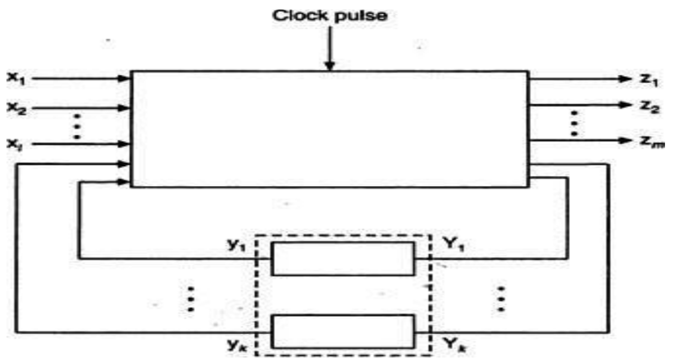
**Step7: K-maps and minimal functions: based on the contents of the excitation table , draw the k-map and simplify them to obtain the minimal expressions for D1 and D2 in terms of y1, y2 and x as shown in fig. The expression for z (z=y1,y2) can be obtained directly from table**

**Step8: implementation: the logic diagram based on these minimal expressions**

## Finite State Machine:

**Finite state machine can be defined as a type of machine whose past histories can affect its future behavior in a finite number of ways. To clarify, consider for example of binary full adder. Its output depends on the present input and the carry generated from the previous input. It may have a large number of previous input histories but they can be divided into two types: (i) Input**

**The most general model of a sequential circuit has inputs, outputs and internal states. A sequential circuit is referred to as a finite state machine (FSM). A finite state machine is abstract model that describes the synchronous sequential machine. The fig. shows the block diagram of a finite state model. $X_1$, X2,….., $X_l$, are inputs. $Z_1$, Z2,….,$Z_m$ are outputs. $Y_1$,$Y_2$,….$Y_k$ are state variables, and $Y_1$,$Y_2$,….$Y_k$ represent the next state.**



## Capabilities and limitations of finite-state machine

**Let a finite state machine have n states. Let a long sequence of input be given to the machine. The machine will progress starting from its beginning state to the next states according to the state transitions. However, after some time the input string may be longer than n, the number of states. As there are only n states in the machine, it must come to a state it was previously been in and from this phase if the input remains the same the machine will function in a periodically repeating fashion. From here a conclusion that ‗for a n state machine the output will become periodic after a number of clock pulses less than equal to n can be drawn. States are memory elements. As for a finite state machine the number of states is finite, so finite number of memory elements are required to design a finite state machine.**

## Limitations:

1. **Periodic sequence and limitations of finite states: with n-state machines, we can generate periodic sequences of n states are smaller than n states. For example, in a 6-state machine, we can have a maximum periodic sequence as 0,1,2,3,4,5,0,1….**

2. **No infinite sequence: consider an infinite sequence such that the output is 1 when and only when the number of inputs received so far is equal to P(P+1)/2 for P=1,2,3….,i.e., the desired input-output sequence has the following form:**

**Input: x x x x x x x x x x x x xx x x xx x xx x**
**Output: 1 0 1 0 0 1 0 0 0 01 0 0 0 0 1 0 0 00 01**

**Such an infinite sequence cannot be produced by a finite state machine.**

3. **Limited memory: the finite state machine has a limited memory and due to limited memory it cannot produce certain outputs. Consider a binary multiplier circuit for multiplying two arbitrarily large binary numbers. The memory is not sufficient to store arbitrarily large partial products resulted duringmultiplication.**

**Finite state machines are two types. They differ in the way the output is generate they are:**

1. **Mealy type model: in this model, the output is a function of the present state and the present input.**
2. **Moore type model: in this model, the output is a function of the present state only.**

**Mathematical representation of synchronous sequential machine:**

**The relation between the present state S(t), present input X(t), and next state s(t+1) can be given as**
**S(t+1)= f{S(t),X(t)}**
**The value of output Z(t) can be given as**
**Z(t)= g{S(t),X(t)} for mealy model**
**Z(t)= G{S(t)} for Moore model**
**Because, in a mealy machine, the output depends on the present state and input, where as in a Moore machine, the output depends only on the present state.**

**Comparison between the Moore machine and mealy machine:**

| Moore machine | mealy machine |
|---|---|
| 1. its output is a function of present state only Z(t)= g{S(t)} | 1. its output is a function of present state as well as present input Z(t)=g{S(t),X(t)} |
| 2. input changes do not affect the output | 2. input changes may affect the output of the circuit |
| 3. it requires more number of states for implementing same function | 3. it requires less number of states for implementing same function |

**Mealy model:**

**When the output of the sequential circuit depends on the both the present state of the flip-flops and on the inputs, the sequential circuit is referred to as mealy circuit or mealy machine. The fig. shows the logic diagram of the mealy model. Notice that the output depends up on the present state as well as the present inputs. We can easily realize that changes in the input during the clock pulse cannot affect the state of the flip-flop. They can affect the output of the circuit. If the input variations are not synchronized with a clock, he derived output will also not be synchronized with the clock and we get false output. The false outputs can be eliminated by allowing input to change only at the active transition of the clock.**

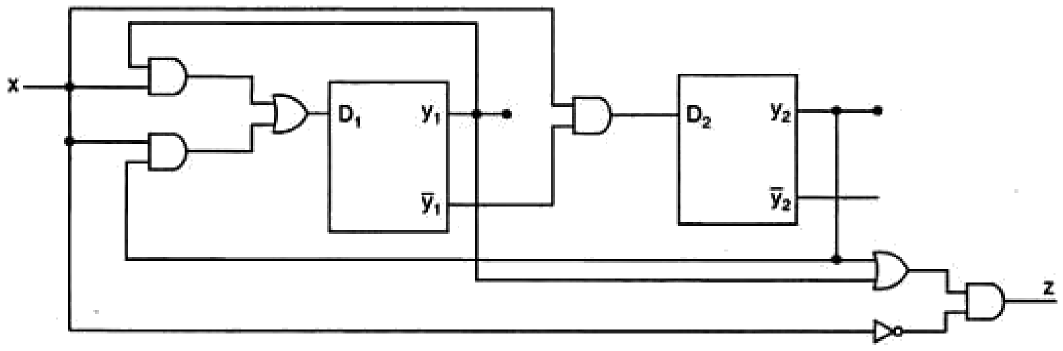


**Fig: logic diagram of a mealy model**

**The behavior of a clocked sequential circuit can be described algebraically by means of state equations. A state equation specifies the next state as a function of the present state and inputs. The mealy model shown in fig. consists of two D flip-flops, an input x and an output z. since the D input of a flip-flop determines the value of the next state, the state equations for the model can be written as**

**$Y_1$ (t+1)=$y_1$(t)x(t)+$y_2$(t)x(t)**

**$Y_2$(t+1)= 1(t)x(t)**

**And the output equation is**

**Z(t)={ $y_1$(t)+$y_2$(t)} X'(t)**

**Where y(t+1) is the next state of the flip-flop one clock edge later, x(t) is the present input, and z(t) is the present output. If y1(t+1) are represented by y1(t) and y2(t) , in more compact form, the equations are**

**Y1(t+1)=y1=y1x+y2x**

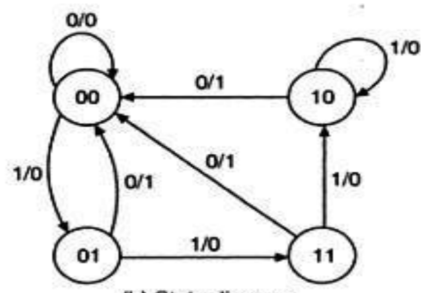
**Y2(t+1)=y2=y1'x**

**Z=(y1+y2)x'**

**The stable table of the mealy model based on the above state equations and output equation is shown in fig. the state diagram based on the state table is shown in fig.**

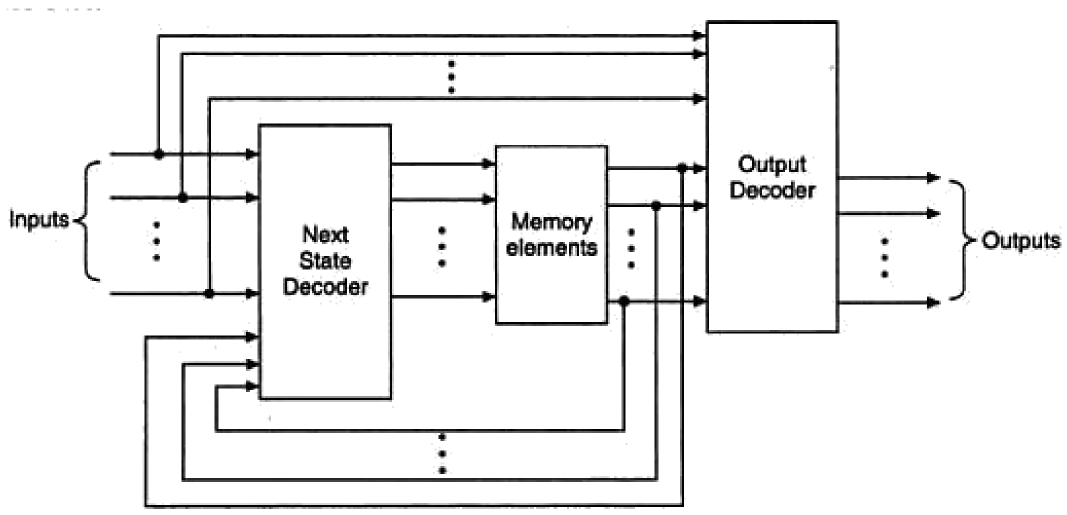
| PS | | NS | | | | O/P | |
|---|---|---|---|---|---|---|---|
| | | x = 0 | | x = 1 | | x = 0 | x = 1 |
| $y_1$ | $y_2$ | $Y_1$ | $Y_2$ | $Y_1$ | $Y_2$ | z | z |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |

(a) State table



(b) State diagram

**In general form, the mealy circuit can be represented with its block schematic as shown in below fig.**



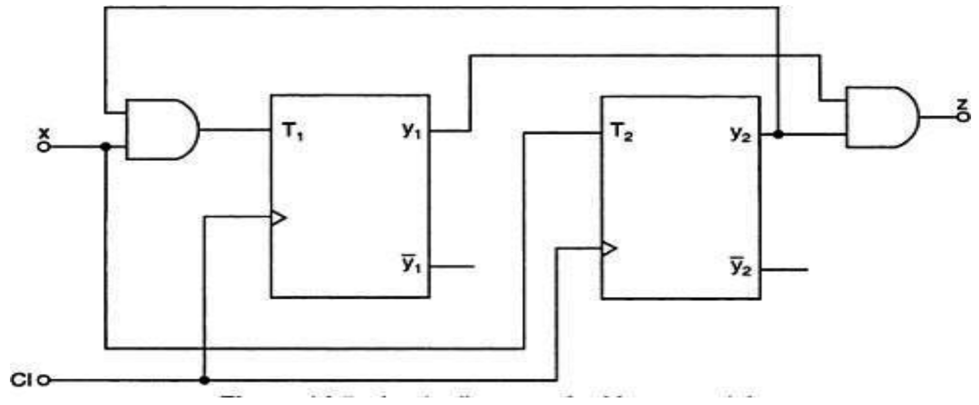
**Moore model: when the output of the sequential circuit depends up only on the present state of the flip-flop, the sequential circuit is referred as to as the Moore circuit or the Moore machine.**

**Notice that the output depend only on the present state. It does not depend upon the input at all. The input is used only to determine the inputs of flip-flops. It is not used to determine the output. The circuit shown has two T flip-flops, one input x, and one output z. it can be described algebraically by two input equations an output equation.**

**$T_1=y_2x$**

**$T_2=x$**

**$Z=y_1y_2$**



**n of a T-flip-flop is Q(t+1)=TQ'+T'Q**

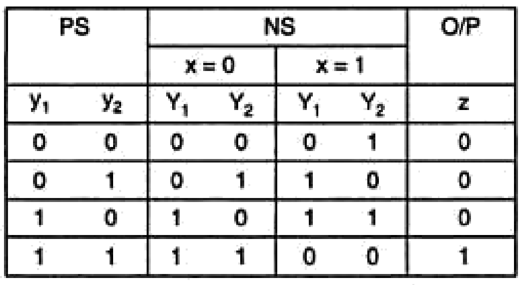
**The values for the next state can be derived from the state equations by substituting $T_1$ and $T_2$ in the characteristic equation yielding**

**$Y_1(t+1)=Y_1=(y_2x) \oplus$ =( 2 )y1+(y2x) 1**

**= y1 2+ y1 + 1y2x**

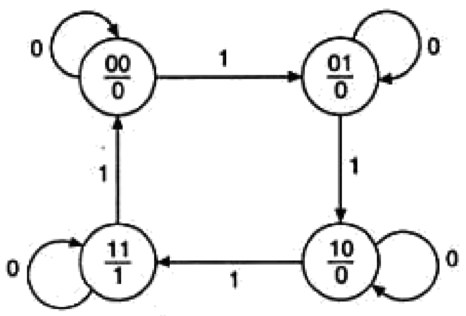
**= y2 (t+1)= x $\oplus$y2= x 2+ y2**

**The state table of the Moore model based on the above state equations and output equation is shown in fig.**



| PS | | NS | | | | O/P |
|---|---|---|---|---|---|---|
| | | x = 0 | | x = 1 | | |
| $y_1$ | $y_2$ | $Y_1$ | $Y_2$ | $Y_1$ | $Y_2$ | z |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 1 |

(a) State table



(b) State diagram

**In general form , the Moore circuit can be represented with its block schematic as shown in below fig.**

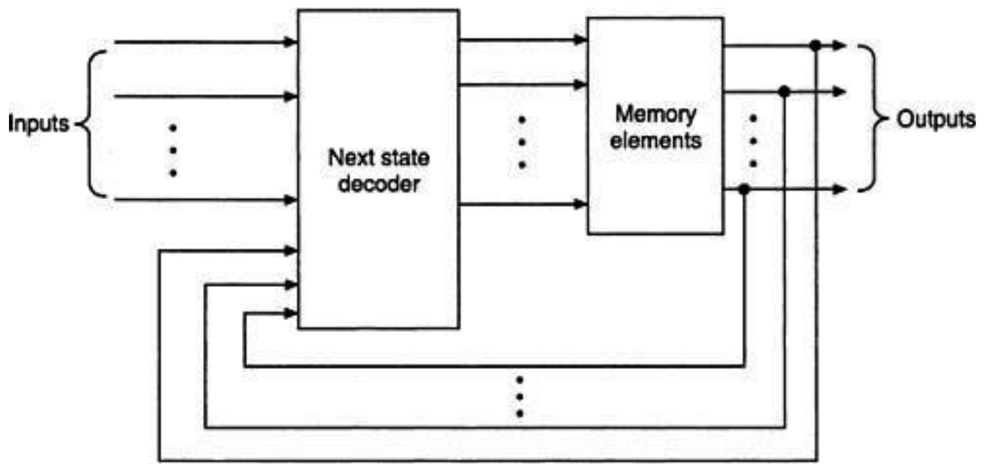


**Figure: moore circuit model:**

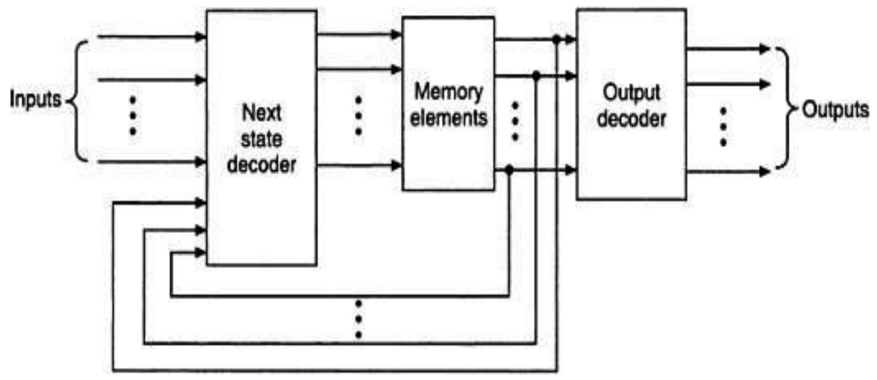
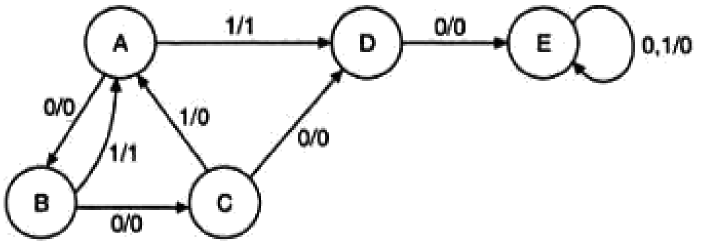


**Figure: moore circuit model with an output decoder**

**Important definitions and theorems:**
**A). Finite state machine-definitions:**
**Consider the state diagram of a finite state machine shown in fig. it is five-state machine with one input variable and one output variable.**



**Successor: looking at the state diagram when present state is A and input is 1, the next state is D. this condition is specified as D is the successor of A. similarly we can say that A is the 1 successor of B, and C,D is the 11 successor of B and C, C is the 00 successor of A and D, D is the 000 successor of A,E, is the 10 successor of A or 0000 successor of A and so on.**

**Terminal state: looking at the state diagram , we observe that no such input sequence exists which can take the sequential machine out of state E and thus state E is said to be a terminal state.**

**Strongly-connected machine: in sequential machines many times certain subsets of states may not be reachable from other subsets of states. Even if the machine does not contain any terminal state. If for every pair of states $s_i$, $s_j$, of a sequential machine there exists an input sequence which takes the machine M from $s_i$ to sj, then the sequential machine is said to be strongly connected.**

**B). state equivalence and machine minimization:**

**In realizing the logic diagram from a stat table or state diagram many times we come across redundant states. Redundant states are states whose functions can be accomplished by other states. The elimination of redundant states reduces the total number of states of the machines which in turn results in reduction of the number of flip-flops and logic gates, reducing the cost of the final circuit.**

**Two states are said to be equivalent. When two states are equivalent, one of them can be removed without altering the input output relationship.**

**State equivalence theorem: it states that two states $s_1$, and $s_2$ are equivalent if for every possible input sequence applied. The machine goes to the same next state and generates the same output. That is**
**If $S_1(t+1)= s_2(t+1)$ and $z_1=z_2$, then $s_1=s_2$**

**C). distinguishable states and distinguishing sequences:**

**Two states $s_a$, and $s_b$ of a sequential machine are distinguishable, if and only if there exists at least one finite input sequence which when applied to the sequential machine causes different outputs sequences depending on weather $s_a$ or $s_b$ is the initial state.**

**Consider states A and B in the state table, when input X=0, their outputs are 0 and 1 respectively and therefore, states A and B are called 1-distinguishabke. Now consider states A and E . the output sequence is as follows.**

**X=0 A C,0 and E D, 0 ; outputs are the same**

**C → E,0** **and D→ b,1 ; outputs are different**

**Here the outputs are different after 2-state transition and hence states A and E are 2-distungishable. Again consider states A and C . the output sequence is as follows:**

**X=0 A→ C,0 and C → E, 0; outputs are the same**

**C → E,0 and E → D,0 ; outputs are the**

**same E → D,0 – and D B,1 ; outputs are**

**different**

**Here the outputs are different after 3- transition and hence states A and B are 3-distuingshable. the concept of K- distuingshable leads directly to the definition of K-equivalence. States that are not K-distinguishable are said to be K-equivalent.**

**Truth table for Distunigshable states:**

| PS | NS,Z | |
|---|---|---|
| | X=0 | X=1 |
| A | C,0 | F,0 |
| B | D,1 | F,0 |
| C | E,0 | B,0 |
| D | B,1 | E,0 |
| E | D,0 | B,0 |
| F | D,1 | B,0 |

**Merger Chart Methods:**

**Merger graphs:**

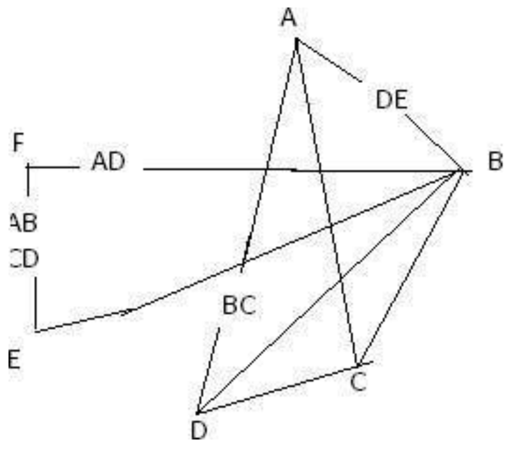
**The merger graph is a state reducing tool used to reduce states in the incompletely specified machine. The merger graph is defined as follows.**

1. **Each state in the state table is represented by a vertex in the merger graph. So it contains the same number of vertices as the state table contains states.**
2. **Each compatible state pair is indicated by an unbroken line draw between the two state vertices**
3. **Every potentially compatible state pair with non-conflicting outputs but with different next states is connected by a broken line. The implied states are written in theline break between the two potentially compatible states.**
4. **If two states are incompatible no connecting line is drawn.**

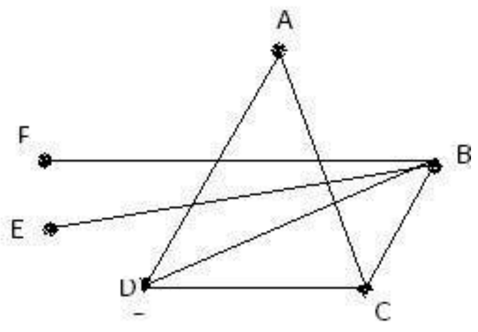
**Consider a state table of an incompletely specified machine shown in fig. the corresponding merger graph shown in fig.**

**State table:**

| PS | NS,Z | | | |
|---|---|---|---|---|
| | I1 | I2 | I3 | I4 |
| A | … | E,1 | B,1 | …. |
| B | … | D,1 | … | F,1 |
| C | F,1 | … | … | … |
| D | … | … | C,1 | … |
| E | C,0 | … | A,0 | F,1 |
| F | D,0 | A,1 | B,0 | … |



**a) Merger graph**



**b) simplified merger graph**

**States A and B have non-conflicting outputs, but the successor under input $I_2$are compatible only if implied states D and E are compatible. So, draw a broken line from A to B with DE written in between states A and C are compatible because the next states and output entries of states A and C are not conflicting. Therefore, a line is drawn between nodes A and C. states A and D have non-conflicting outputs but the successor under input $I_3$ are B and C. hence join A and D by a broken line with BC entered In between.**

**Two states are said to be incompatible if no line is drawn between them. If implied states are incompatible, they are crossed and the corresponding line is ignored. Like, implied states D and E are incompatible, so states A and B are also incompatible. Next, it is necessary to check whether the incompatibility of A and B does not invalidate any other broken line. Observe that states E and F also become incompatible because the implied pair AB is incompatible. The broken lines which remain in the graph after all the implied pairs have been verified to be compatible are regarded as complete lines. After checking all possibilities of incompatibility, the merger graph gives the following seven compatible pairs.**

**(A, C) (A, D) (B, C) (B, D) (C, D) (B, E) (B, F)**

**These compatible pairs are further checked for further compatibility. For example, pairs (B,C)(B,D)(C,D) are compatible. So (B, C, D) is also compatible. Also pairs (A,c)(A,D)(C,D) are compatible. So (A,C,D) is also compatible. . In this way the entire set of compatibles of sequential machine can be generated from its compatible pairs.**

**To find the minimal set of compatibles for state reduction, it is useful to find what are called the maximal compatibles. A set of compatibles state pairs is said to be maximal, if it is not completely covered by any other set of compatible state pairs. The maximum compatible can be found by looking at the merger graph for polygons which are not contained within any higher order complete polygons. For example only triangles (A, C,D) and (B,C,D) are of higher order. The set of maximal compatibles for this sequential machine given as**

(A, C, D) (B, C, D) (B, E) (B, F)

**Example:**

Draw the merger graph and obtain the set of maximal compatibles for the incompletely specified sequential machine whose state table is given in Table 7.24.

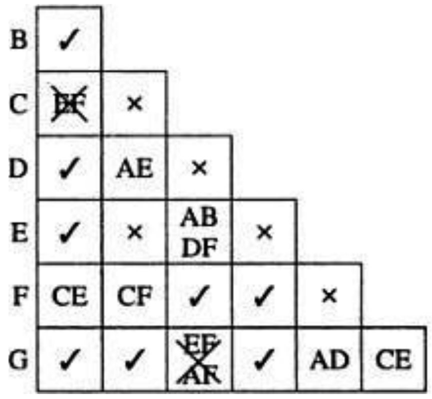
**Table 7.24** Example 7.9: State table

| PS | NS, Z | |
|---|---|---|
| | $I_1$ | $I_2$ |
| A | E, 0 | B, 0 |
| B | F, 0 | A, 0 |
| C | E, – | C, 0 |
| D | F, 1 | D, 0 |
| E | C, 1 | C, 0 |
| F | D, – | B, 0 |

mark × in the corresponding cell. For example, states B and C are incompatible because their outputs are conflicting and hence the cell corresponding to them contains a cross mark ×. Similarly states B, E; D, E; E, F are incompatible. Hence put a × mark in the corresponding cells. On the other hand, states A and B are compatible and hence the cell corresponding to them contains the check mark ✓. Similarly, cells corresponding to states A, D; A ,E; A, G; B, G; C, F; D, F ; D, G are also compatible. So a check mark is put in those cells also. The implied pairs or pairs corresponding to the state pair are written within the cell as shown in Table 7.26. For example, states A and C are compatible only when implied states E and F are compatible. Therefore, EF is written in the cell corresponding to states A and C. States C and E are compatible only when implied states A and B, and D and F are compatible. So AB and DF are written in the cell corresponding to states C and E. In a similar way, the entire merger table is written. Now it is necessary to check whether the implied pairs are compatible or not by observing the merger table. The implied states are incompatible if the corresponding cell contains a ×. For example, implied pair E, F is incompatible because cell EF contains a ×. Similarly, implied pairs EF, AF are incompatible because EF contains a ×. It is indicated by a ×.

| PS | NS, Z | | | |
|---|---|---|---|---|
| | 00 | 01 | 11 | 10 |
| A | E, 0 | – | – | – |
| B | – | F, 1 | E, 1 | A, 1 |
| C | F, 0 | – | A, 0 | F, 1 |
| D | – | – | A, 1 | – |
| E | – | C, 0 | B, 0 | D, 1 |
| F | C, 0 | C, 1 | – | – |
| G | E, 0 | – | – | A, 1 |

**Figure: state table**



**State Minimization:**
**Completely Specified Machines**

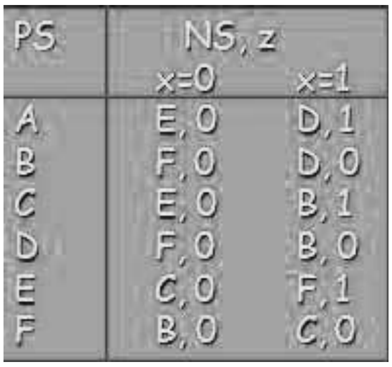
**Two states, $s_i$ and $s_j$ of machine M are distinguishable if and only if there exists a finite input sequence which when applied to M causes different output sequences depending on whether M started in $s_i$ or $s_j$.**
**Such a sequence is called a distinguishing sequence for ($s_i$, $s_j$).**
**If there exists a distinguishing sequence of length k for ($s_i$, $s_j$), they are said to be k-distinguishable.**

**EXAMPLE:**

| PS | NS, z | |
|---|---|---|
| | x=0 | x=1 |
| A | E, 0 | D, 1 |
| B | F, 0 | D, 0 |
| C | E, 0 | B, 1 |
| D | F, 0 | B, 0 |
| E | C, 0 | F, 1 |
| F | B, 0 | C, 0 |

- **states A and B are 1-distinguishable, since a 1 input applied to A yields an output 1, versus an output 0 from B.**
- **states A and E are 3-distinguishable, since input sequence 111 applied to A yields output 100, versus an output 101 from E.**
- **States $s_i$ and $s_j$ ($s_i \sim s_j$) are said to be equivalent iff no distinguishing sequence exists for ($s_i$, $s_j$).**
- **If $s_i \sim s_j$ and $s_j \sim s_k$, then $s_i \sim s_k$. So state equivalence is an equivalence relation (i.e. it is a reflexive, symmetric and transitive relation).**
- **An equivalence relation partitions the elements of a set into equivalence classes.**
- **Property: If $s_i \sim s_j$, their corresponding X-successors, for all inputs X, are also equivalent.**
- **Procedure: Group states of M so that two states are in the same group iff they are equivalent (forms a partition of the states).**

**Completely Specified Machines**



| PS | NS, z | |
|---|---|---|
| | x=0 | x=1 |
| A | E, 0 | D, 1 |
| B | F, 0 | D, 0 |
| C | E, 0 | B, 1 |
| D | F, 0 | B, 0 |
| E | C, 0 | F, 1 |
| F | B, 0 | C, 0 |

**$P_i$ : partition using distinguishing sequences of length i.**

| Partition: | Distinguishing Sequence: |
|---|---|
| $P_0$=(ABCDEF) | |
| $P_1$= (A C E)(BD F) | x =1 |
| $P_2$= (A C E)(B D)(F) | x =1; x =1 |
| $P_3$= (A C)(E)(B D)(F) | x =1; x =1; x =1 |

- **All states equivalent to each other form an equivalence class. These may becombined into one state in the reduced (quotient) machine.**
- **Start an initial partition of a single block. Iteratively refine this partition by separating the 1-distinguishable states, 2-distinguishable states and so on.**
- **To obtain $P_{k+1}$, for each block $B_i$ of $P_k$, create one block of states that not 1-distinguishable within $B_i$ , and create different blocks states that are 1-distinguishable**

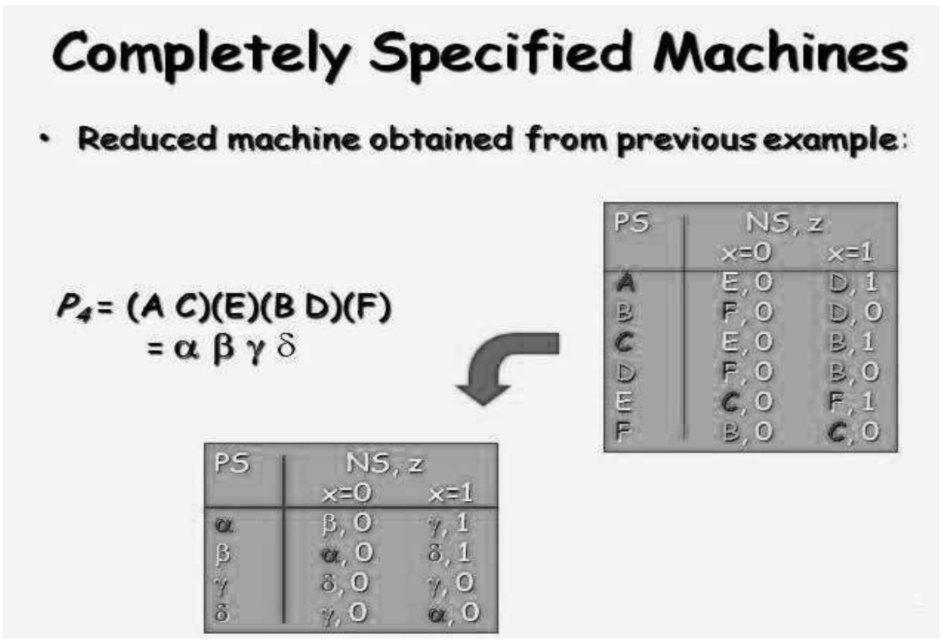
**within $B_i$ .**

**Theorem: The equivalence partition is unique.**

**Theorem: If two states, $s_i$ and $s_j$, of machine M are distinguishable, then they are (n-1 )-distinguishable, where n is the number of states in M.**

**Definition: Two machines, $M_1$ and $M_2$, are equivalent ($M_1 \sim M_2$ ) if, for every state in $M_1$ there is a corresponding equivalent state in $M_2$ and vice versa.**

**Theorem. For every machine M there is a minimum machine $M_{red}$ ~ M. $M_{red}$ is unique up to isomorphism.**



## Completely Specified Machines

- Reduced machine obtained from previous example:

| PS | NS, z | |
|---|---|---|
| | x=0 | x=1 |
| A | E, 0 | D, 1 |
| B | F, 0 | D, 0 |
| C | E, 0 | B, 1 |
| D | F, 0 | B, 0 |
| E | C, 0 | F, 1 |
| F | B, 0 | C, 0 |

$P_4$ = (A C)(E)(B D)(F)
= α β γ δ

| PS | NS, z | |
|---|---|---|
| | x=0 | x=1 |
| α | β, 0 | γ, 1 |
| β | α, 0 | δ, 1 |
| γ | δ, 0 | γ, 0 |
| δ | γ, 0 | α, 0 |

**State Minimization of CSMs: Complexity**
**Algorithm DFA ~ $DFA_{min}$**
**Input: A finite automaton M = (Q, , , q $_0$, F ) with no unreachable states.**
**Output: A minimum finite automaton M' = (Q', , ', q '$_0$, F' ).**
**Method:**

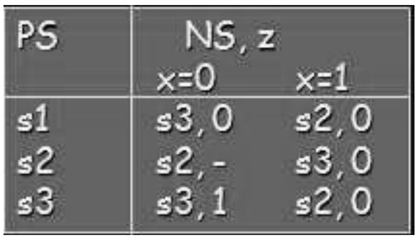
```
    1. t :=2; $Q_0$:= { undefined }; $Q_1$:=F; $Q_2$:= Q\F.
    2. while there is 0 < i  t, awith  ($Q_i$,a)   $Q_j$, for all j  t
do (a) Choose such an i, a , and jt with   ($Q_i$,a)    $Q_j$     .
        (b) $Q_{t+1}$ := {q $Q_i$ | (q,a) $Q_j$ }; $Q_i$
                := $Q_i$ \ $Q_{t+1}$;
t := t +1.
end.
    3. (* Denote [q ] the equivalence class of state q , and {$Q_i$ } the set of all
    equivalence classes. *)

    q '$_0$ := [$q_0$].

' ( [q], a) := [ (q,a)] for all q   Q, a.
```

**Standard implementation: O (kn $^2$), where n =|Q| and k = |  |**
**Modification of the body of the while loop:**

1. **Choose such an i, a, and choose $j_1$,$j_2$   t with   $j_1$   $j_2$,  ($Q_i$,a)   $Q_{j1}$, and ($Q_i$,a)   $Q_{j2}$     .**
2. **If |{q   $Q_i$ |  (q,a)   $Q_{j1}$}|  |{q   $Q_i$ |  (q,a)   $Q_{j2}$}|**

**then $Q_{t+1}$ := {q $Q_i$ | (q,a) $Q_{j1}$ else $Q_{t+1}$ := {q }**
**$Q_i$ | (q,a) $Q_{j2}$ } fl;**
**$Q_i$ := $Q_i$ \ $Q_{t+1}$;**
**t := t +1.**
**(i.e. put smallest set in t +1 )**
**Note: |$Q_{t+1}$| 1/2|$Q_i$|. Therefore, for all q Q, the name of the class which contains a given state q changes at most log(n ) times.**
**Goal: Develop an implementation such that all computations can be assigned to transitions containing a state for which the name of the corresponding class is changed.**
**Suitable data structures achieve an O (kn log n) implementation.**
**State Minimization:**
**Incompletely Specified Machines**
**Statement of the problem: given an incompletely specified machine M, find a machine M' such that:**
**– on any input sequence, M' produces the same outputs as M, whenever M is specified.**
**– there does not exist a machine M'' with fewer states than M' which has the same property**

**Machine M:**

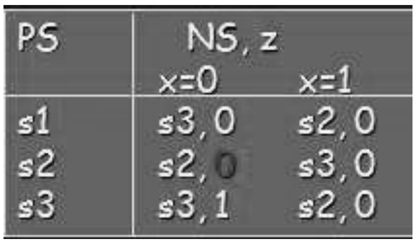


| PS | NS, z | |
|---|---|---|
| | x=0 | x=1 |
| s1 | s3, 0 | s2, 0 |
| s2 | s2, - | s3, 0 |
| s3 | s3, 1 | s2, 0 |

**Attempt to reduce this case to usual state minimization of completely specified machines.**

**Brute Force Method: Force the don't cares to all their possible values and choose the smallest of the completely specified machines soobtained.**
**In this example, it means to state minimize two completely specified machines obtained from M, by setting the don't care to either 0 and 1.**

**Suppose that the - is set to be a 0.**

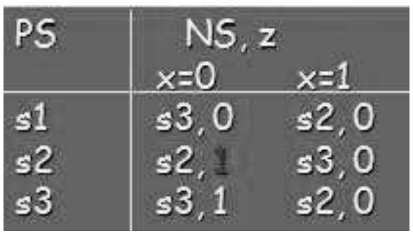


| PS | NS, z | |
|---|---|---|
| | x=0 | x=1 |
| s1 | s3, 0 | s2, 0 |
| s2 | s2, 0 | s3, 0 |
| s3 | s3, 1 | s2, 0 |

**States s1 and s2 are equivalent if s3 and s2 are equivalent, but s3 and s2 assert different outputs under input 0, so s1 and s2 are not equivalent.**
**States s1 and s3 are not equivalent either.**

**So this completely specified machine cannot be reduced further (3 states is the minimum).**

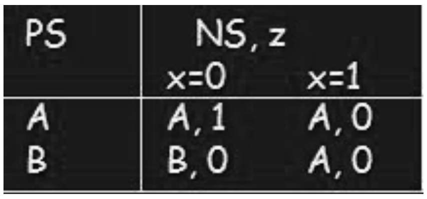
**Suppose that the - is set to be a 1.**



| PS | NS, z | |
|---|---|---|
| | x=0 | x=1 |
| s1 | s3, 0 | s2, 0 |
| s2 | s2, 1 | s3, 0 |
| s3 | s3, 1 | s2, 0 |

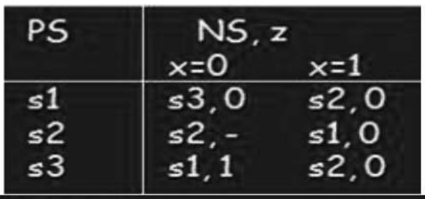
**States s1 is incompatible with both s2 and s3.**
**States s3 and s2 are equivalent.**
**So number of states is reduced from 3 to 2.**

**Machine M''$_{red}$ :**

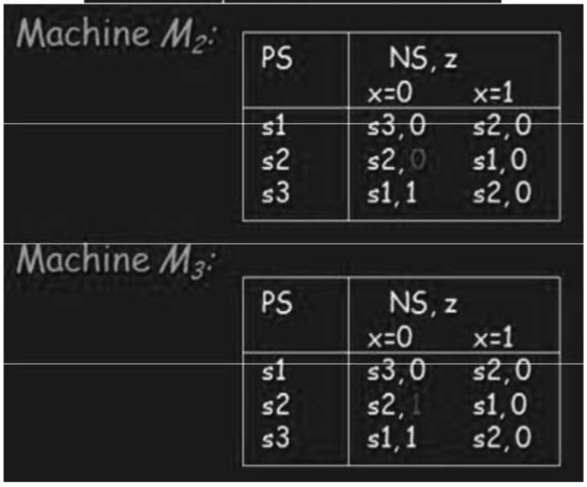


| PS | NS, z | |
|---|---|---|
| | x=0 | x=1 |
| A | A, 1 | A, 0 |
| B | B, 0 | A, 0 |

**Can this always be done?**
**Machine M:**



| PS | NS, z | |
|---|---|---|
| | x=0 | x=1 |
| s1 | s3, 0 | s2, 0 |
| s2 | s2, - | s1, 0 |
| s3 | s1, 1 | s2, 0 |



Machine $M_2$:

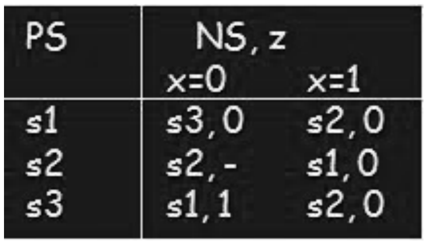
| PS | NS, z | |
|---|---|---|
| | x=0 | x=1 |
| s1 | s3, 0 | s2, 0 |
| s2 | s2, 0 | s1, 0 |
| s3 | s1, 1 | s2, 0 |

Machine $M_3$:

| PS | NS, z | |
|---|---|---|
| | x=0 | x=1 |
| s1 | s3, 0 | s2, 0 |
| s2 | s2, 1 | s1, 0 |
| s3 | s1, 1 | s2, 0 |

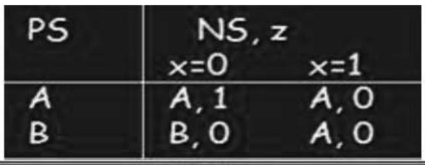
**Machine M$_2$ and M$_3$ are formed by filling in the unspecified entry in M with 0 and 1, respectively.**

**Both machines $M_2$ and $M_3$ cannot be reduced.**
**Conclusion?: M cannot be minimized further!**
**But is it a correct conclusion?**
**Note: that we want to ‗merge' two states when, for any input sequence, they generate the same output sequence, but only where both outputs are specified.**
**Definition: A set of states is compatible if they agree on the outputs where they are all specified.**
**Machine M'' :**

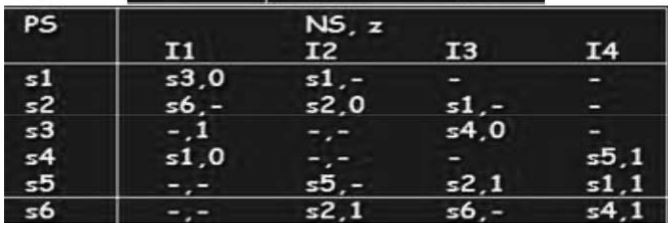


| PS | NS, z x=0 | x=1 |
|---|---|---|
| s1 | s3, 0 | s2, 0 |
| s2 | s2, - | s1, 0 |
| s3 | s1, 1 | s2, 0 |

**In this case we have two compatible sets: A = (s1, s2) and B = (s3, s2). A reduced machine $M_{red}$ can be built as follows.**
**Machine $M_{red}$**

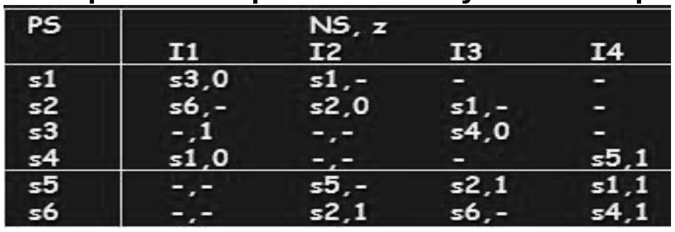


| PS | NS, z x=0 | x=1 |
|---|---|---|
| A | A, 1 | A, 0 |
| B | B, 0 | A, 0 |



| PS | NS, z I1 | I2 | I3 | I4 |
|---|---|---|---|---|
| s1 | s3,0 | s1,- | - | - |
| s2 | s6,- | s2,0 | s1,- | - |
| s3 | -,1 | -,- | s4,0 | - |
| s4 | s1,0 | -,- | - | s5,1 |
| s5 | -,- | s5,- | s2,1 | s1,1 |
| s6 | -,- | s2,1 | s6,- | s4,1 |

**A set of compatibles that cover all states is: (s3s6), (s4s6), (s1s6), (s4s5), (s2s5).**
**But (s3s6) requires (s4s6),**
**(s4s6) requires(s4s5),** **(s4s5) requires (s1s5),**
**(s1s6) requires (s1s2),** **(s1s2) requires (s3s6),**
**(s2s5) requires (s1s2).**
**So, this selection of compatibles requires too many other compatibles...**



| PS | NS, z I1 | I2 | I3 | I4 |
|---|---|---|---|---|
| s1 | s3,0 | s1,- | - | - |
| s2 | s6,- | s2,0 | s1,- | - |
| s3 | -,1 | -,- | s4,0 | - |
| s4 | s1,0 | -,- | - | s5,1 |
| s5 | -,- | s5,- | s2,1 | s1,1 |
| s6 | -,- | s2,1 | s6,- | s4,1 |

**Another set of compatibles that covers all states is (s1s2s5), (s3s6), (s4s5). But (s1s2s5) requires (s3s6) (s3s6) requires (s4s6)**
**(s4s6) requires (s4s5)** **(s4s5) requires (s1s5).**
**So must select also (s4s6) and (s1s5).**
**Selection of minimum set is a binate covering problem**

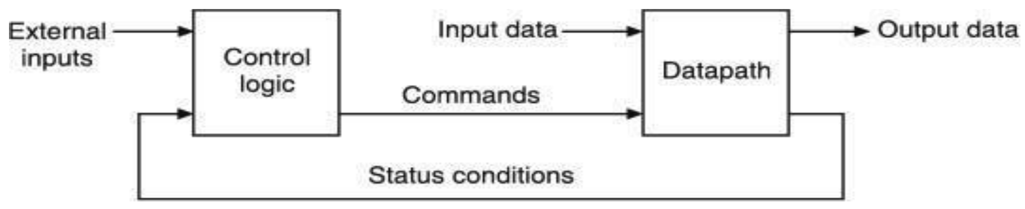
**When a next state is unspecified, the future behavior of the machine is unpredictable. This suggests the definition of admissible input sequence.**
**Definition. An input sequence is admissible, for a starting state of a machine if no unspecified next state is encountered, except possibly at the final step.**
**Definition. State $s_i$ of machine $M_1$ is said to cover, or contain, state $s_j$ of $M_2$ provided**

1. **every input sequence admissible to $s_j$ is also admissible to $s_i$ , and**
2. **its application to both $M_1$ and $M_2$ (initially is $s_i$ and $s_j$, respectively) results in identical output sequences whenever the outputs of $M_2$ are specified.**

**Definition. Machine $M_1$ is said to cover machine $M_2$ if for every state $s_j$ in $M_2$, there is a corresponding state $s_i$ in $M_1$ such that $s_i$ covers $s_j$.**

**Algorithmic State Machines:**

**The binary information stored in the digital system can be classified as either data or control information.**
**The data information is manipulated by performing arithmetic, logic, shift and other data processing tasks.**
**The control information provides the command signals that controls the various operations on the data in order to accomplish the desired data processing task.**
**Design a digital system we have to design two subsystems data path subsystem and control subsystem.**



Interaction between control logic and datapath.

**ASM CHART:**

**A special flow chart that has been developed specifically to define digital hardware algorithms is called ASM chart.**
**A hardware algorithm is a step by step procedure to implement the desire task.**

**Difference b/n conventional flow chart and ASM chart:**

**conventional flow chart describes the sequence of procedural steps and decision paths for an algorithm without concern for their time relationship**
**An ASM chart describes the sequence of events as well as the timing relationship b/n the states of sequential controller and the events that occur while going from one state to the next**

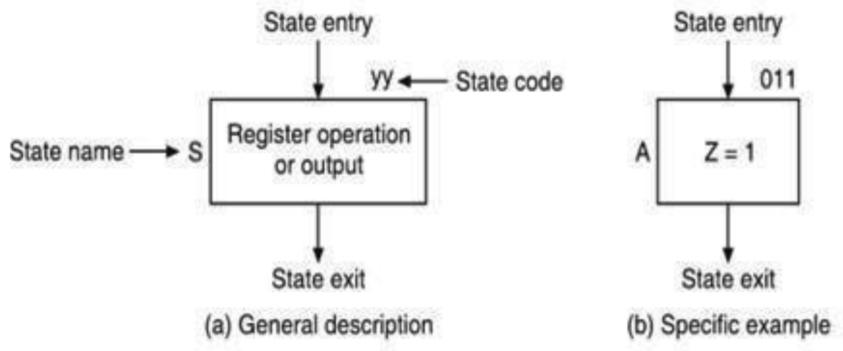
**1. State box:** A state of a clocked sequential circuit is represented by a rectangle called *state box*. It is equivalent to a node in the state diagram or a row in the state table. The name of the state is written to the left of the box. The binary code assigned to the state is indicated outside on the top right-side of the box. A list of unconditional outputs if any associated with the state are written within the box.

**2. Decision box:** The decision box or condition box is represented by a diamond-shaped symbol with one input and two or more output paths. The output branches are true and false branches. The decision box describes the effect of an input on the control subsystem. A Boolean variable or input or expression written inside the diamond indicates a condition which is evaluated to determine which branch to take.

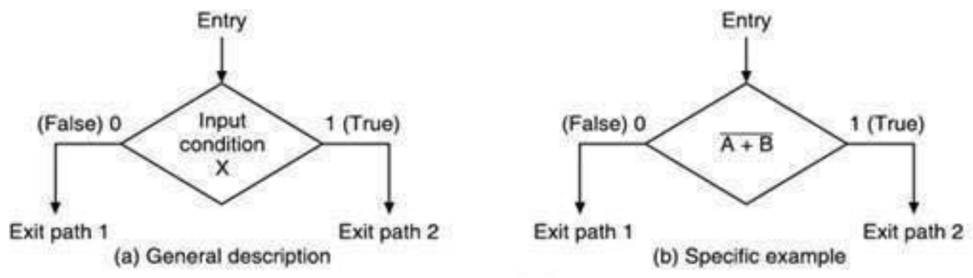
**ASM consists of**

1. **State box**
2. **Decision box**
3. **Conditional box**

**State box**

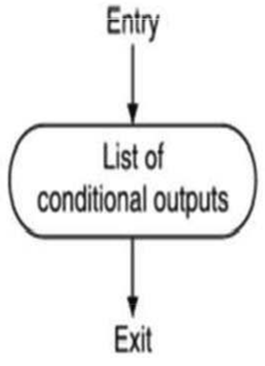


**Decision box**



Decision box.

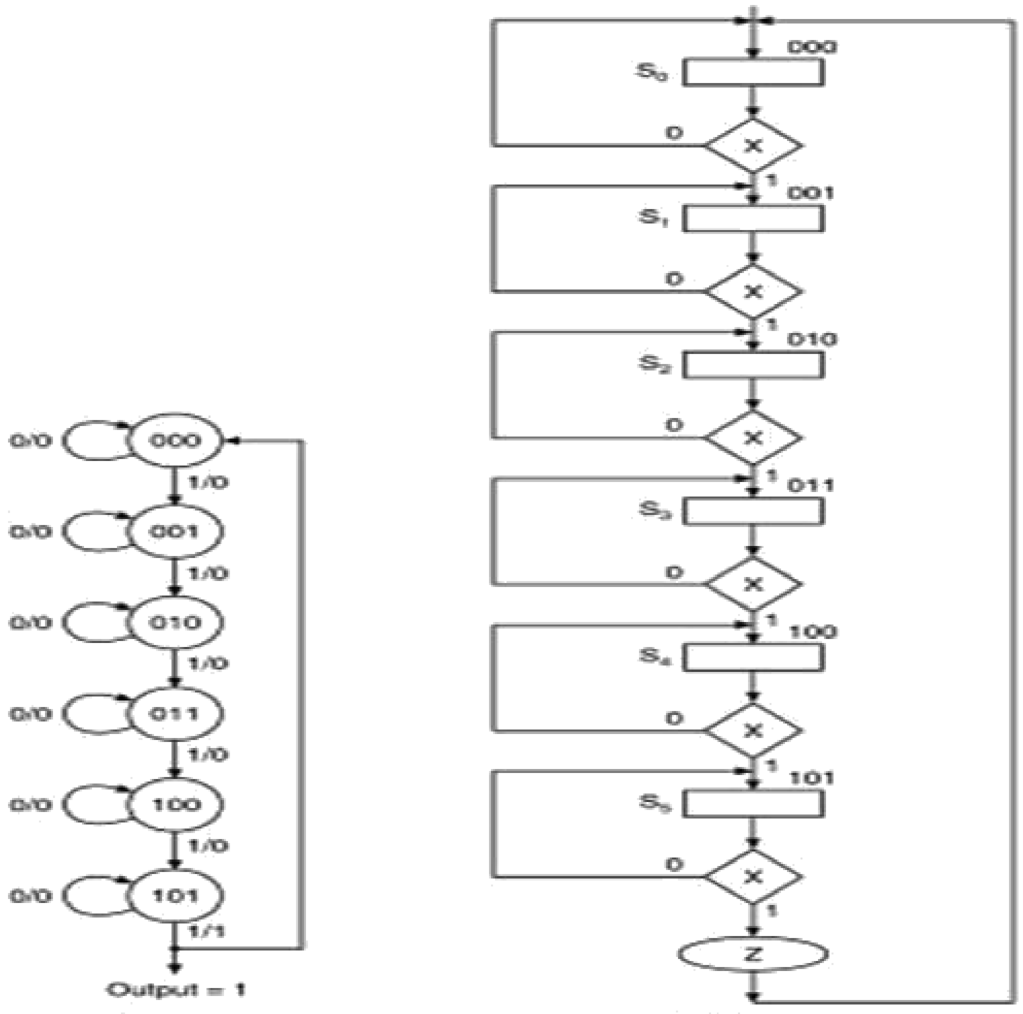
**3. Conditional output box:** The conditional output box is represented by a rectangle with rounded corners or by an oval with one input line and one output line. The outputs that depend on both the state of the system and the inputs are indicated inside the box.



Conditional output box.

**SALIENT FEATURES OF ASM CHARTS**

1. An ASM chart describes the sequence of events as well as the timing relationship between the states of a sequential controller and the events that occur while going from one state to the next.
2. An ASM chart contains one or more interconnected ASM blocks.
3. Each ASM block contains exactly one state box together with the decision boxes and conditional output boxes associated with that state.
4. Every block in an ASM chart specifies the operations that are to be performed during one common clock pulse.
5. An ASM block has exactly one entrance path and one or more exit paths represented by the structure of the decision boxes.
6. A path through an ASM block from entrance to exit is referred to as a link path.
7. The operations specified within the state and conditional output boxes in the block are performed in the datapath subsystem.
8. Internal feedback within an ASM block is not permitted. Even so, following a decision box or conditional output boxes, the machine may reenter the same state.
9. Each block in the ASM chart describes the state of the system during one clock pulse interval. When a digital system enters the state associated with a given ASM block, the outputs indicated within the state box become true. The conditions associated with the decision boxes are evaluated to determine which path or paths to be followed to enter the next ASM block.



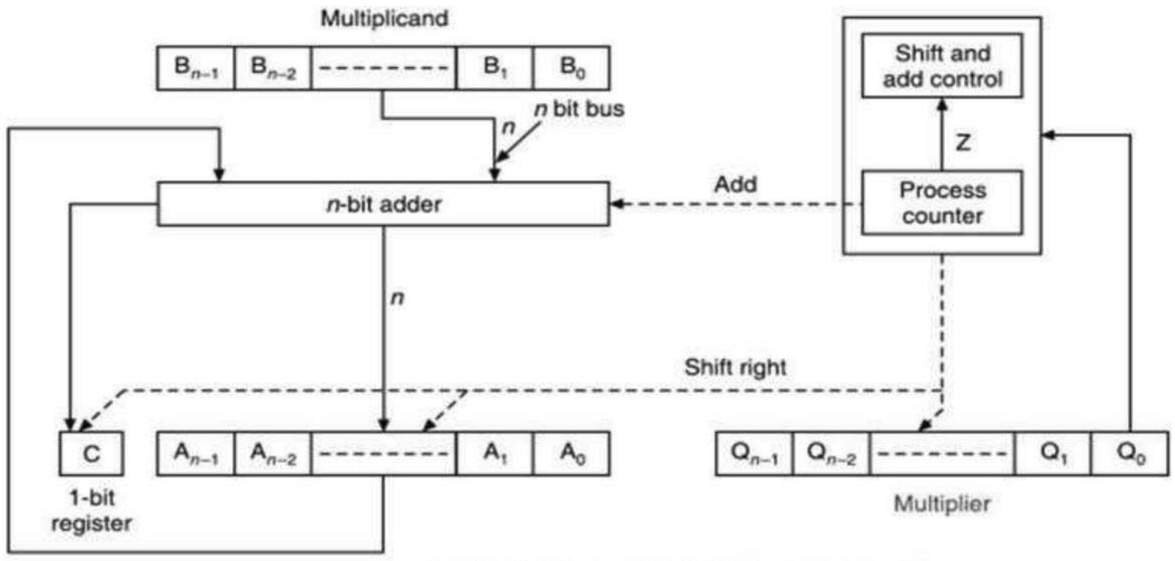
(a) State diagram (b) ASM chart

State diagram and ASM chart for mod-6 counter.

**BINARY MULTIPLIER**

$$\begin{array}{rcl}
1\,1\,0\,1 & \leftarrow & 13_{10} \ldots \text{Multiplicand} \\
1\,0\,1\,0 & \leftarrow & 10_{10} \ldots \text{Multiplier} \\
\hline
0\,0\,0\,0 & \leftarrow & \text{Partial product 1} \\
1\,1\,0\,1\phantom{\,0} & \leftarrow & \text{Partial product 2} \\
0\,0\,0\,0\phantom{\,0\,0} & \leftarrow & \text{Partial product 3} \\
1\,1\,0\,1\phantom{\,0\,0\,0} & \leftarrow & \text{Partial product 4} \\
\hline
1\,0\,0\,0\,0\,0\,1\,0 & \leftarrow & 130_{10} \ldots \text{Product} \\
\hline
\end{array}$$

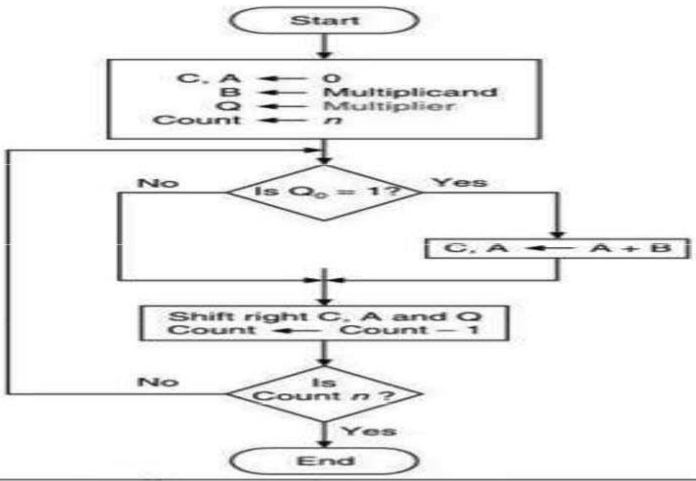
**Data path subsystem for binary multiplier**



Datapath subsystem for binary multiplier.

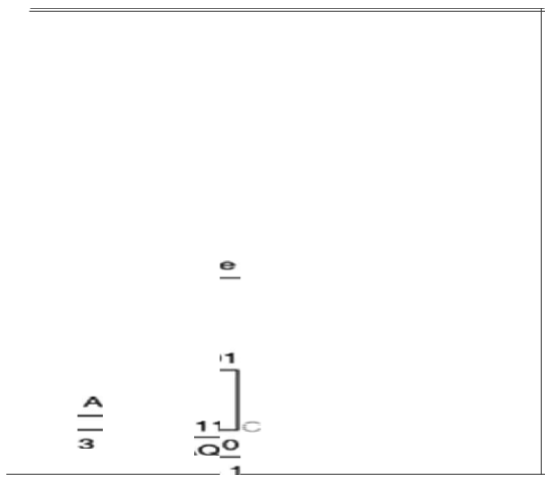
**Multiplication Operation Steps**

1. Bit 0 of multiplier operand ($Q_0$ of Q register) is checked.
2. If bit 0 ($Q_0$) is one then multiplicand and partial product are added and all bits of C, A and Q registers are shifted to the right one bit, so that the C bit goes into $A_{n-1}$, $A_0$ goes into $Q_{n-1}$, and $Q_0$ is lost. If bit 0 ($Q_0$) is 0, then no addition is performed, only shift operation is carried out.
3. Steps 1 and 2 are repeated n times to get the desired result in the A and Q registers.



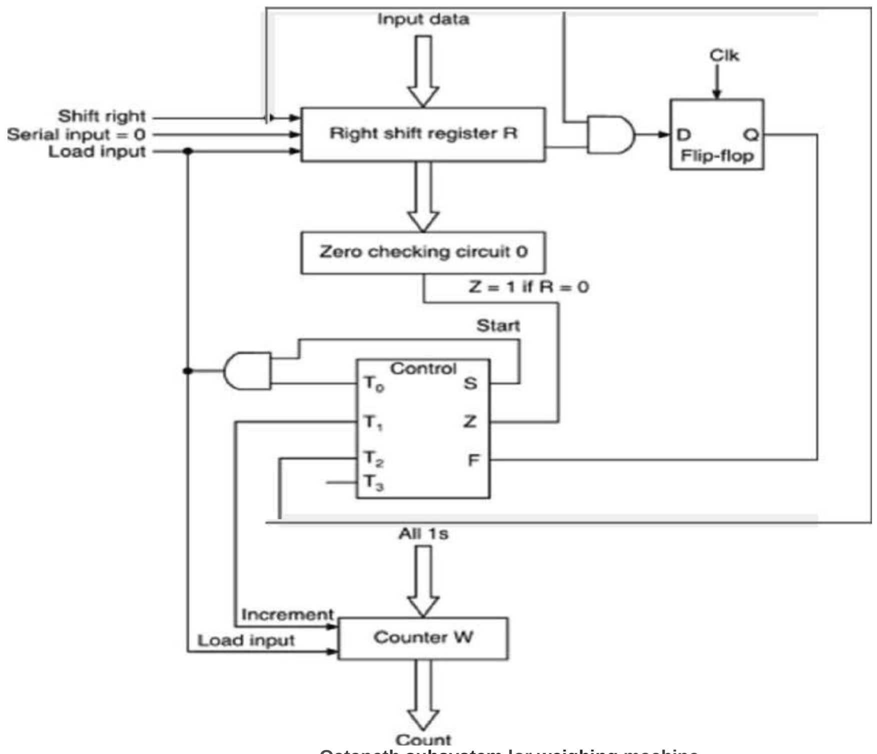
| B | C | A | Q | Components | Count P |
|---|---|---|---|---|---|
| 1101 | O | O OOO | IOIO | B ← Multiplicand<br>I OO t4 Q ← Multiplier<br>A ← 0, C ← 0, P ← $n$ | |
| 1IO1 | O | O OOO | IOIO | 1 | |
| | O | OOOO 0 | I O I | C A Q shifted right | |
| | | | | O 1 O i 21 | |
| JIOI | O | OIIO | IOIO | $Q_0 = 1$, A ← A + B<br>C A P ← P − 1 right<br>„=O. | 001 (1) |
| | t› | ‹› ‹› I i u | i t› i | C A Q shifted right | |
| 1 1 0 1 | 1 | 0 0 0 0 0 | 1 0 I | P ← P − 1 | |
| | 0 | 1 0 0 0 0 | 0 1 0 | C A @ .shi ftoct right | |

flow cf-iart f<>r mwltipficati<>n in a corripwter.

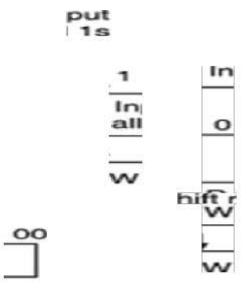


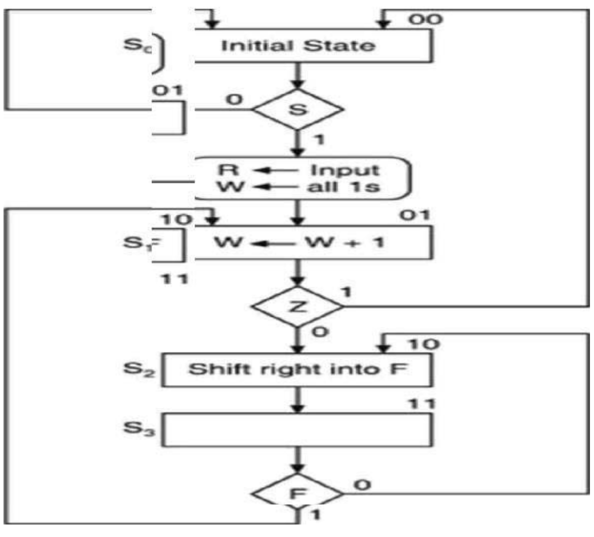
## ASM FOR WEIGHING MACHINE

**In tte a[$oriOin for tabolir minimization of Boolean expressions, we have to arrange be ininnnns in thc ascending order of their weights. This is only one of the iniuiy situations when we have tn examine the 1» of a given binary word. The wsight of a binary number is defined as the number of 5 }7f0 S9ltt 1f1 lls blf1d£\ fe]7fed nldtlOn.**



Oatapeth **subsystem** lor **weighing** machine.





6r / S . Ln*ti atJy lite weighing stack ine i s in state Sq. one weighing process stacts wficn suu-t (S } signal becomes I . While in state fi if *> i s I , the clock pulse cauws three jobs to be done sinTu ltaneousfy."

1. Bi n ay n u m ber is loader4 i nEo xug isCer R.
2. W regJ ster is set to all G s.
3. The xtachine is rransFerred to sta re S , .

*State $S_1$:* While in state $S_1$, the clock pulse causes two jobs to be done simultaneously:

. COuntcr W in income rated by' 1{in thJz First councJ, all I s b+>c:Om &J1 US).

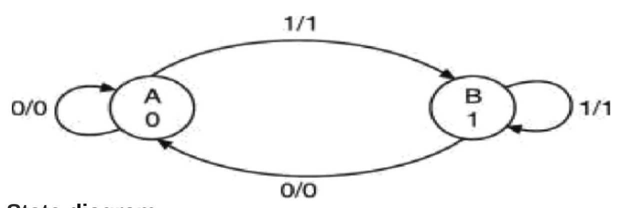
2. IT Z i s 0. the in ash ine gc• s la Lhc scale fi : i I Z i s I . tMe machinc gc•es to state fi .

crore $ $_2$• In this state. regiMer R ix lifted righthyl hitin thaE L SC y ‹ws int‹t F and M $ B is 1naded with 0

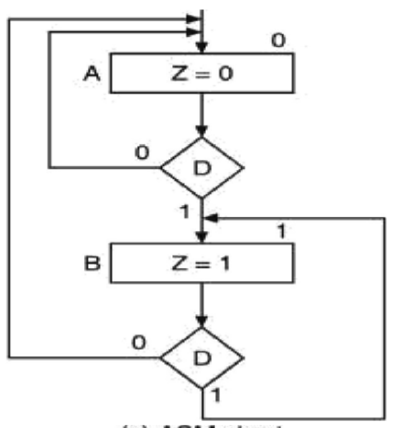
fiinm Sy: In iii is siate, rhe value of F is chicken. Jf iz is 0, the machine is rransf'erred to the state S,, othew'ise the machine i3 transferred to state S,. Thus. when F - I. W is incremented.

All the operations occur in coincidence with the clock pulse while in the corresponding state.

Also nokcc ‹w the rugi star k should eventually contain all 0s when the last I is stiir in into .

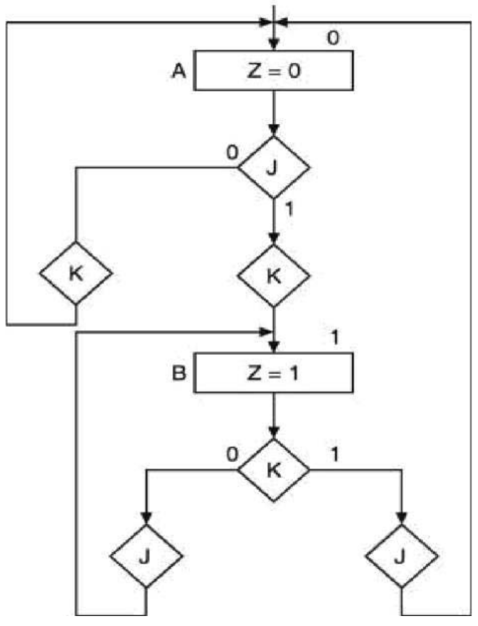


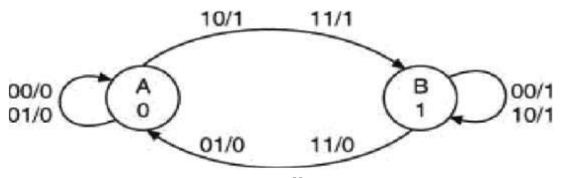
(a) State diagram



| PS | NS, O/P | |
|---|---|---|
| | Input0 | |
| | D - D | D - 1 |
| A | A, D | B, 1 |
| B | A, D | B, 1 |

(D) State table (c) ASM chart





(a) State diagram

| PS | NS,OGP | | | |
|---|---|---|---|---|
| | Input J-K | | | |
| | OD | D1 | 10 | 11 |
| A | A, O | A, O | B, 1 | B, 1 |
| B | B, 1 | A, O | B, 1 | A, 0 |

‹b› s‹ata lable (c) ADM chart