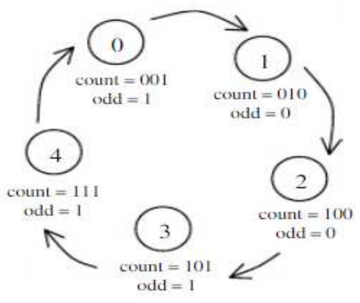
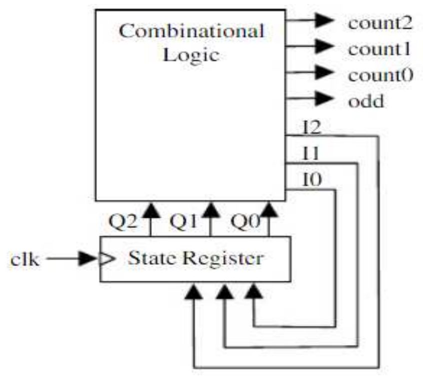
1. Design a 3-bit counts the following sequence: 1,2,4,5,7,1,2… etc. This counter has an output “odd” whose value is 1 when the current count value is odd. Use the sequential design technique of the chapter. Start from a state diagram, draw the state table, minimize the logic and draw the final circuit.

Solution:

Step 1: Design of state machine as per the problem

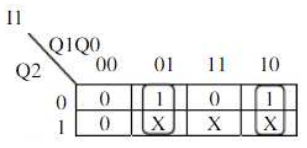




Step 2: Implementation of FSM into state table

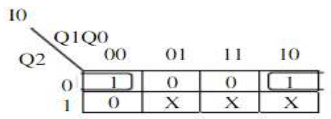
Inputs			Outputs						
Q2	Q1	Q0	I2	I1	I0	count2	count1	count0	add
0	0	0	0	0	1	0	0	1	1
0	0	1	0	1	0	0	1	0	0
0	1	0	0	1	1	1	0	0	0
0	1	1	1	0	0	1	0	1	1
1	0	0	0	0	0	1	1	1	1
1	0	1	x	x	x	x	x	x	x
1	1	0	x	x	x	x	x	x	x
1	1	1	x	x	x	x	x	x	x

Step 3: Minimization of logic from table above using K-map


I1

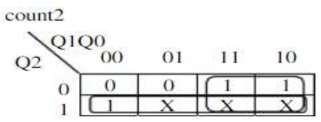
Q2 \ Q1Q0	00	01	11	10
0	0	1	0	1
1	0	X	X	X

I1 = Q1'Q0 + Q1Q0'


I0

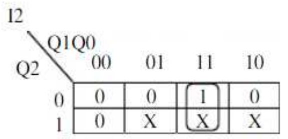
Q2 \ Q1Q0	00	01	11	10
0	1	0	0	1
1	0	X	X	X

I0 = Q2'Q0'


count2

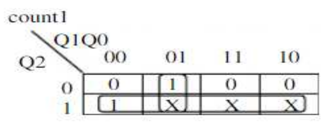
Q2 \ Q1Q0	00	01	11	10
0	0	0	1	1
1	1	X	X	X

count2 = Q2 + Q1


I2

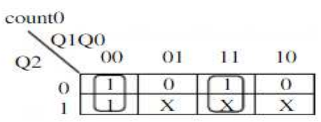
Q2 \ Q1Q0	00	01	11	10
0	0	0	1	0
1	0	X	X	X

I2 = Q1Q0


count1

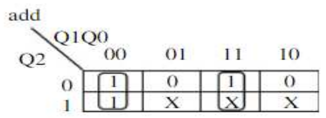
Q2 \ Q1Q0	00	01	11	10
0	0	1	0	0
1	1	X	X	X

count1 = Q2 + Q1'Q0


count0

Q2 \ Q1Q0	00	01	11	10
0	1	0	1	0
1	1	X	X	X

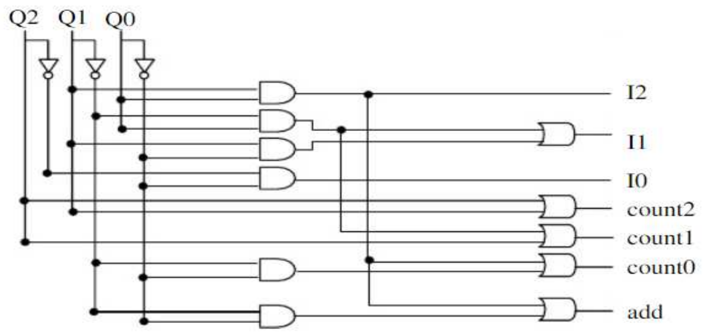
count0 = Q1'Q0' + Q1Q0


add

Q2 \ Q1Q0	00	01	11	10
0	1	0	1	0
1	1	X	X	X

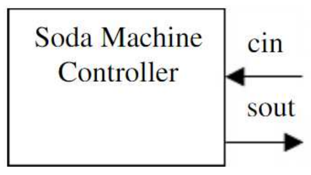
add = Q1'Q0' + Q1Q0

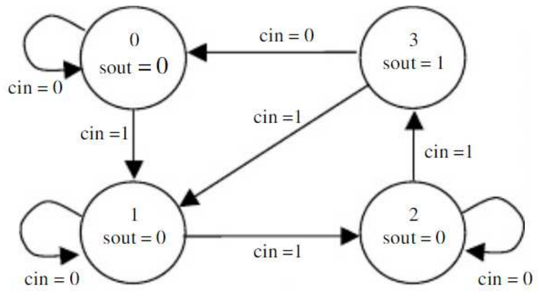
Step 4: Final Circuit realization from K-map



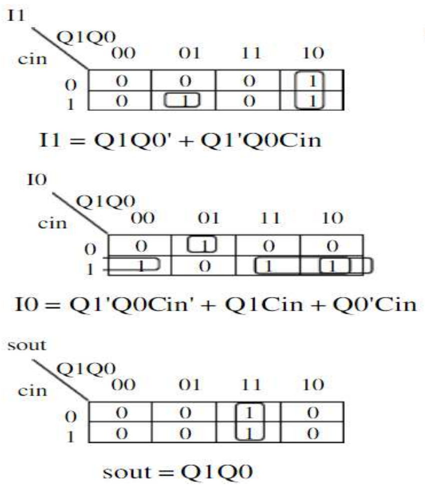
2. Design a soda machine controller, given that a soda cost 75 cents and your machine accepts quarters only. Draw a black box view, come up with a state diagram and state table, minimize the logic, and draw the final circuit?

Solution:
Step 1: The black box and state diagram as per the problem is shown below





Step 2: Repeating the above process and implementing the design in k-map and hardware gives us the following



I1

cin \ Q1Q0	00	01	11	10
0	0	0	0	1
1	0	1	0	1

I1 = Q1Q0' + Q1'Q0Cin

I0

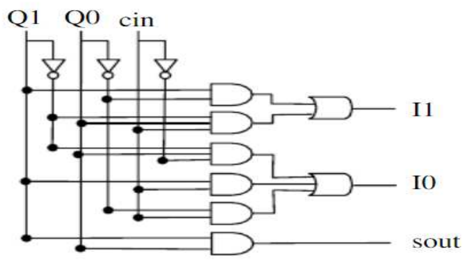
cin \ Q1Q0	00	01	11	10
0	0	1	0	0
1	1	0	1	1

I0 = Q1'Q0Cin' + Q1Cin + Q0'Cin

sout

cin \ Q1Q0	00	01	11	10
0	0	0	1	0
1	0	0	1	0

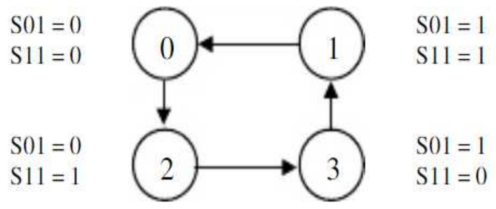
sout = Q1Q0



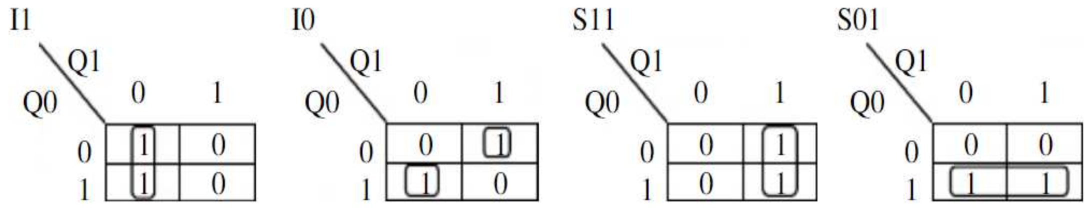
3. Four lights are connected to a decoder. Build a circuit that will blink the lights in the following order 0,2,1,3,0,2…Start from the state diagram, draw the state table minimize the logic and draw the final circuit.

Solution:

The state diagram and table are shown below

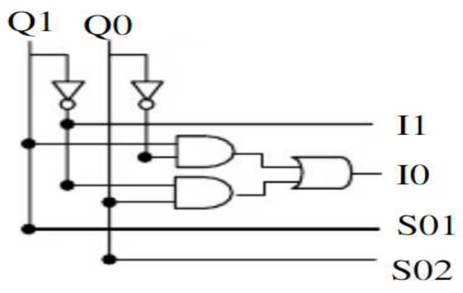


Q1	Q0	I1	I0	S11	S01
0	0	1	0	0	0
0	1	1	1	0	1
1	0	0	1	1	1
1	1	0	0	1	1

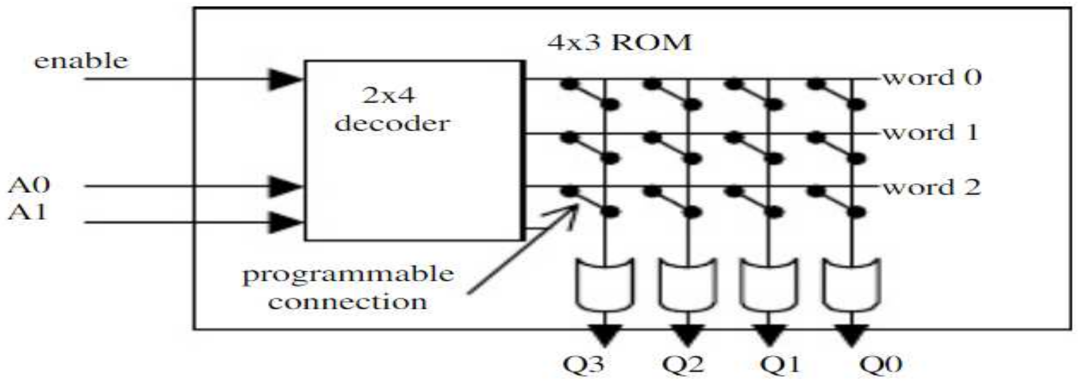


I1 = Q1'   I0 = Q1Q0' + Q1'Q0   S11 = Q1   S01 = Q0

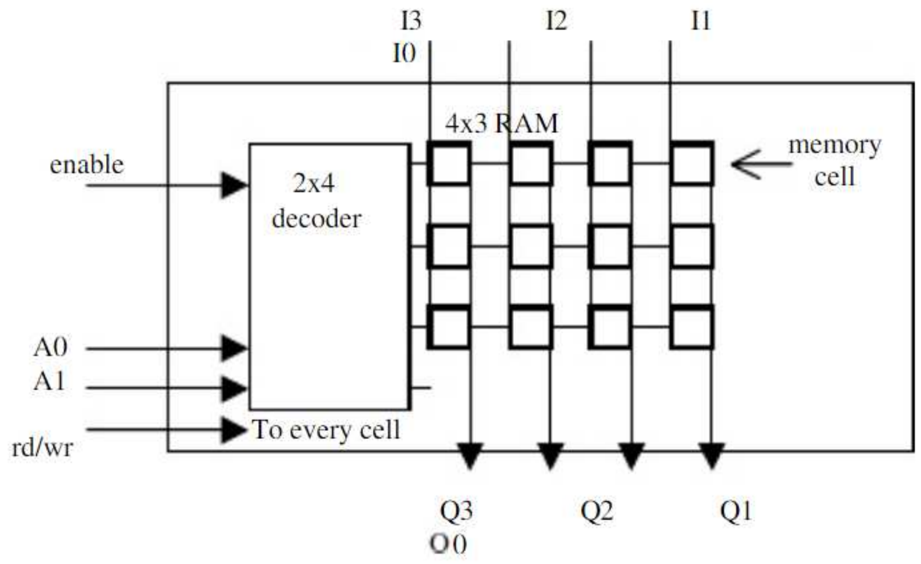
Final circuit implementation is shown below



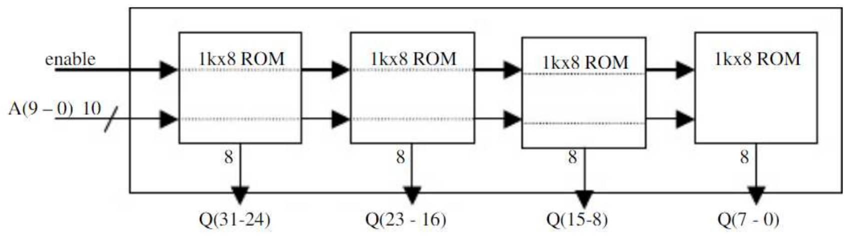
4. Sketch the internal design of 4×3 ROM



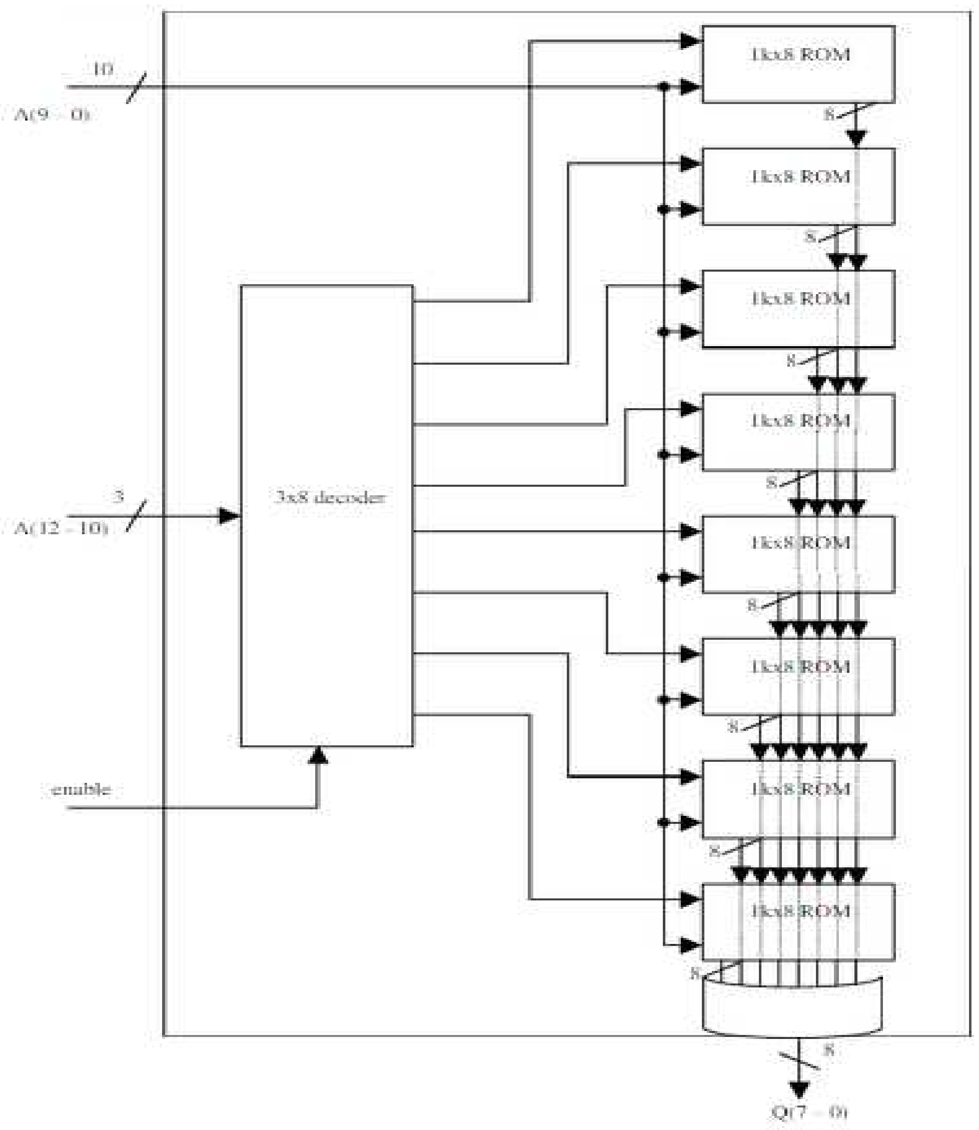
5. Sketch the internal design of 4×3 RAM



6. Compose 1K×8 ROMs into 1k×32 ROMs
Solution:



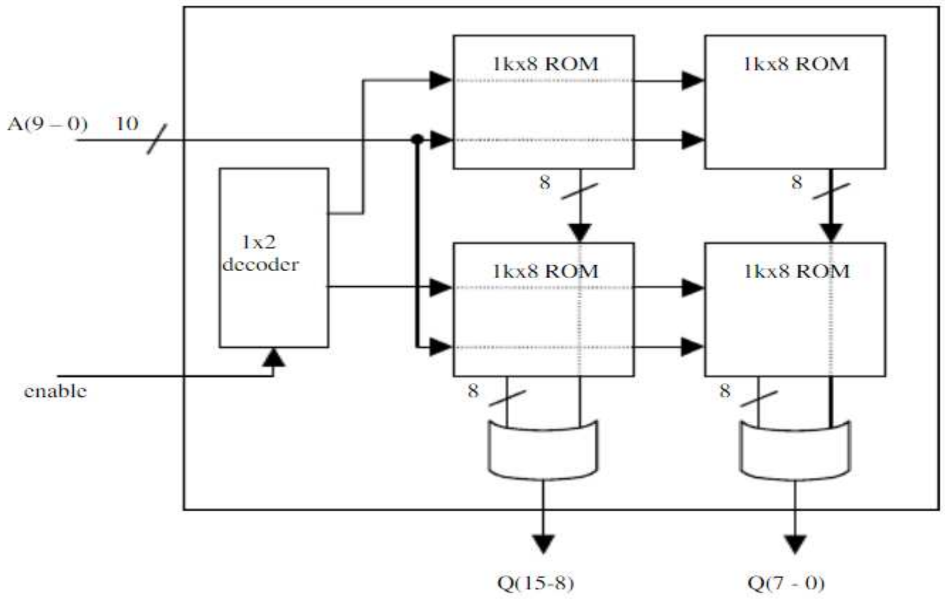
7. Compose 1k×8 ROMs into an 8k×8 ROMs
Refer to solution provided and carry out all the calculation yourself ;-)



8. Compose 1k×8ROMs into a 2K×16 ROM

Step1: Address Decoding

Here we are given 1k×8 ROMs to construct 2k×16 ROMs thus we have
$2^k=2^{10}$
Thus, k=10 hence we use these 10 address lines for address decoding.
Let us assume we use address line A0-A9 for address decoding.
Step 2: Selection of proper address decoding circuit.
Since we have to interface only two ROMs to meet the design specification 1×2 decoder will meet our demand.
Here when enable pin is high ROMs of upper part are selected and when enable pin is low lower part ROMs are selected. Thus upper significant bit appears at Q(15-8) and lower significant bits appears at Q(7-0) for particular address supplied in A(9-0). The requirement arrangement is shown below.



Step 3: Connection of proper control signal

Since this memory device is ROM (i.e. read only memory) we only have RD control signal which is not shown here but in exam you have to make it.

Step 4: Address decoding (most important part)

The address decoding of the above circuit is shown below(for upper part ROM)

A10(enable)	A9	A8	A7	A6	A5	A4	A3	A2	A1	AO
0	0	0	0	0	0	0	0	0	0	0
0	..	..	..	..	..	..	..	..	..	..
0	1	1	1	1	1	1	1	1	1	1

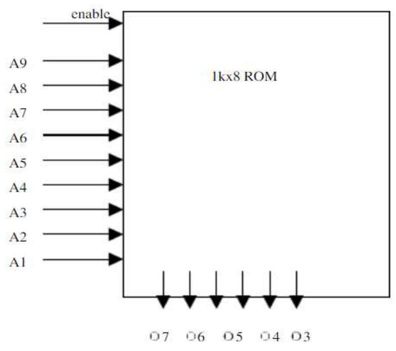
Thus usable address ranges from **00000000000-01111111111**

Similarly, for lower part ROM we have

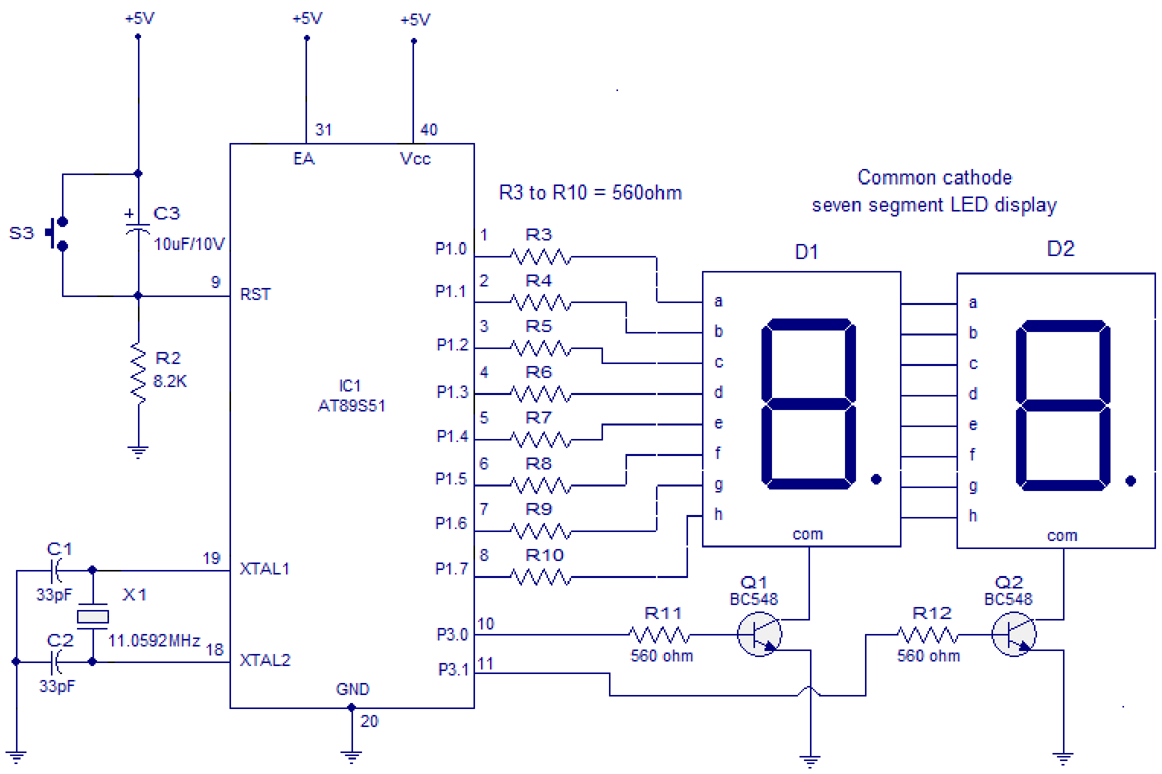
A10(enable)	A9	A8	A7	A6	A5	A4	A3	A2	A1	AO
1	0	0	0	0	0	0	0	0	0	0
1	..	..	..	..	..	..	..	..	..	..
1	1	1	1	1	1	1	1	1	1	1

Thus the usable address ranges from **10000000000-11111111111**

9. Show how to use 1k×8ROMs to implement a 512×6 ROMs
Since we don't need all the address lines and bits per words is also minimized the simple circuit shown here will fulfill our requirement.



10. Interfacing of two seven Segment display
    The following arrangement can be done to interface two seven segment display



About the circuit:
Above circuit shows one of the procedure of interfacing multiplexed seven segment display. Two seven segment display are connected to port P1 via suitable resistor to limit the current.
To select one of the display among two we have used pin p3.0 and pin p3.1 as shown in figure above. This pin is connected to the base of two npn transistor which acts as switch. When pin p3.0 is high transistor is on thus display D1 get connected to ground and what ever the data is supplied via port P1 gets displayed in display D1 in contrast when pin p3.0 is low and pin p3.1 is high transistor Q2 is on thus the data supplied by port P1 is displayed in Seven-segment D2.

Proper crystal and power supply are connected to 8051 for its operations as shown above. The code below display value from 00 to 99 on multiplexed Seven-segment display.

```
ORG 000H // initial starting address
MOV P1,#00000000B // clears port 1
MOV R6,#1H // stores "1"
MOV R7,#6H // stores "6"
MOV P3,#00000000B // clears port 3
MOV DPTR,#LABEL1 // loads the adress of line 29 to DPTR
MAIN: MOV A,R6 // "1" is moved to accumulator
SETB P3.0 // activates 1st display
ACALL DISPLAY // calls the display sub routine for getting the pattern for
"1"
MOV P1,A // moves the pattern for "1" into port 1
```

```
ACALL DELAY // calls the 1ms delay
CLR P3.0 // deactivates the 1st display
MOV A,R7 // "2" is moved to accumulator
SETB P3.1 // activates 2nd display
ACALL DISPLAY // calls the display sub routine for getting the pattern for
"2"
MOV P1,A // moves the pattern for "2" into port 1
ACALL DELAY // calls the 1ms delay
CLR P3.1 // deactivates the 2nd display
SJMP MAIN // jumps back to main and cycle is repeated
DELAY: MOV R3,#02H
DEL1: MOV R2,#0FAH
DEL2: DJNZ R2,DEL2
DJNZ R3,DEL1
RET
DISPLAY: MOVC A,@A+DPTR // adds the byte in A to the address in DPTR and
loads A with data present in the resultant address
RET
LABEL1:DB 3FH
DB 06H
DB 5BH
DB 4FH
DB 66H
DB 6DH
DB 7DH
DB 07H
DB 7FH
DB 6FH

END
```

11. Interfacing of counter, milk packet counter or people counter.

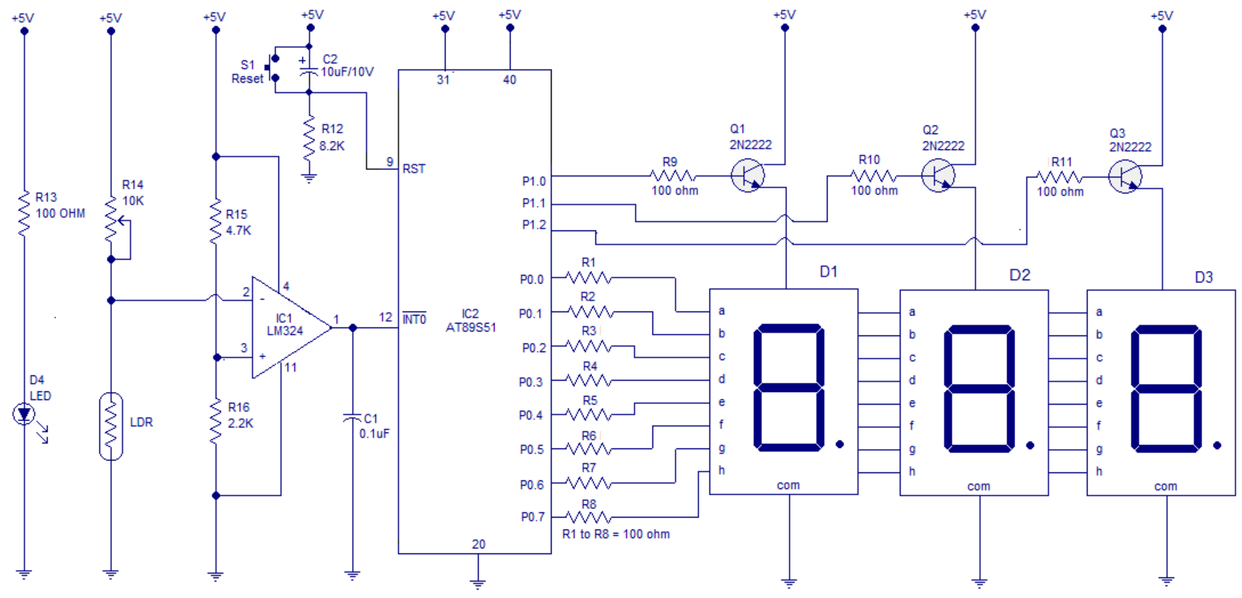


Figure above shown the simple event counter, here we have used INT0 pin for counting objects.

About Circuit:

Simple LDR circuit arrangement as shown in figure is used for counting objects. The led D4 is powered by 5V source which continuously falls on the LDR. The property of LDR is such that when light falls on this sensor its resistance decreases. We use this property of LDR to count events. A suitable voltage division is made as shown in circuit above. LM324 is a comparator ic that compares two input voltage levels one voltage is reference voltage which is fed to pin number 3 of the comparator and the next voltage level comes from LDR circuit. When light falls on LDR its resistance is low thus high voltage goes to pin 2 which does not trigger the Interrupt pin of 8051 but when some object block the light of the led falling to LDR then voltage input is less so low voltage flow from LM324 which triggers the interrupt pin and interrupt service routine is called which increment the count level.

```
ORG 000H
SJMP INIT
ORG 003H  // starting address of interrupt service routine (ISR)
ACALL ISR // calls interrupt service routine
RETI

INIT: MOV P0,#00000000B
      MOV P3,#11111111B
      MOV P1,#00000000B
      MOV R6,#00000000B
      MOV DPTR,#LUT
      SETB IP.0     // sets highest priority for the interrupt INT0
      SETB TCON.0   // interrupt generated by a falling edge signal at INT0
pin
      SETB IE.0     //enables the external interrupt
      SETB IE.7     //enables the global interrupt control

MAIN: MOV A,R6
CLR P1.2
      SJMP MAIN

ISR: INC R6     //interrupt service routine
     RET

DISPLAY: MOVC A,@A+DPTR // display sub routine
         CPL A
         MOV P0,A
         RET

DELAY: MOV R3,#255D  // 1mS delay
LABEL: DJNZ R3,LABEL
       RET
LUT: DB 3FH
     DB 06H
     DB 5BH
     DB 4FH
     DB 66H
     DB 6DH
     DB 7DH
     DB 07H
     DB 7FH
     DB 6FH
END
```

12. Octal to binary encoder

Solution:

```
library ieee;
use ieee.std_logic_1164.all;

entity encoder8x3 is
    port(
    x0,x1,x2,x3,x4,x5,x6,x7 : in std_logic;
    y0,y1,y2 : out std_logic
    );
end encoder8x3;

architecture behaviour of encoder8x3 is
signal y : std_logic_vector(2 downto 0);
begin

y <= "000" when x0&x1&x2&x3&x4&x5&x6&x7 = "10000000" else
     "001" when x0&x1&x2&x3&x4&x5&x6&x7 = "01000000" else
     "010" when x0&x1&x2&x3&x4&x5&x6&x7 = "00100000" else
     "011" when x0&x1&x2&x3&x4&x5&x6&x7 = "00010000" else
     "100" when x0&x1&x2&x3&x4&x5&x6&x7 = "00001000" else
     "101" when x0&x1&x2&x3&x4&x5&x6&x7 = "00000100" else
     "110" when x0&x1&x2&x3&x4&x5&x6&x7 = "00000010" else
     "111" when x0&x1&x2&x3&x4&x5&x6&x7 = "00000001" else
     "ZZZ";

    y0 <= y(0);
    y1 <= y(1);
    y2 <= y(2);

end behaviour;
```

Clock synchronization:

Clock synchronization aims to coordinate independent clocks. Even when initially set accurate, real clock will differ after some amount of time due to clock drift, caused by clocks counting time at slightly different rates. There are several problems that occur as a result of clock rate difference. In serial communication, clock synchronization refers to clock recovery which achieves frequency synchronization. Clock synchronization is used in telecommunications and other transmitting and receiving circuits to achieve tighter communication.
However, there are problems associated with clock skew on more complex distributed system in which many computers will need to realize the same global time. For instance in Linux the make command is used to compile new or modified code without the need to recompile unchanged code. The make command uses the clock of the machine to determine which files need to be recompiled. If the sources resides on a separate file server and the two machines have unsynchronized clocks the make program might not produce the correct results.

In a centralized system the solution is quiet easy as the centralized server will dictate the system time; however the solution is more complex in distributed system because global time is not easily known. The most used clock synchronization solution on the internet is Network Time Protocol.

Task Synchronization:

Synchronization is classified into two categories: *resource synchronization* and *activity synchronization* . Resource synchronization determines whether access to a shared resource is safe, and, if not, when it will be safe. Activity synchronization determines whether the execution of a multithreaded program has reached a certain state and, if it hasn't, how to wait for and be notified when this state is reached.

Resource Synchronization

Access by multiple tasks must be synchronized to maintain the integrity of a shared resource. This process is called *resource synchronization* , a term closely associated with critical sections and mutual exclusions.

> *Mutual exclusion* is a provision by which only one task at a time can access a shared resource. A *critical section* is the section of code from which the shared resource is accessed.

As an example, consider two tasks trying to access shared memory. One task (the sensor task) periodically receives data from a sensor and writes the data to shared memory. Meanwhile, a second task (the display task) periodically reads from shared memory and sends the data to a display. The common design pattern of using shared memory is illustrated in figure below

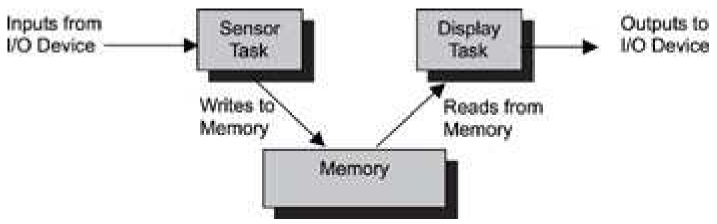


Figure: Multiple tasks accessing shared memory.

Problems arise if access to the shared memory is not exclusive, and multiple tasks can simultaneously access it. For example, if the sensor task has not completed writing data to the shared memory area before the display task tries to display the data, the display would contain a mixture of data extracted at different times, leading to erroneous data interpretation.

The section of code in the sensor task that writes input data to the shared memory is a critical section of the sensor task. The section of code in the display task that reads data from the shared memory is a critical section of the display task. These two critical sections are called *competing critical sections* because they access the same shared resource.

> A mutual exclusion algorithm ensures that one task's execution of a critical section is not interrupted by the competing critical sections of other concurrently executing tasks.

One way to synchronize access to shared resources is to use a client-server model, in which a central entity called a *resource server* is responsible for synchronization. Access requests are made to the resource server, which must grant permission to the requestor before the requestor can access the shared resource. The resource server determines the eligibility of the requestor based on pre-assigned rules or run-time heuristics.

While this model simplifies resource synchronization, the resource server is a bottleneck. Synchronization primitives, such as semaphores and mutexes, and other methods introduced in a later section of this chapter, allow developers to implement complex mutual exclusion algorithms. These algorithms in turn allow dynamic coordination among competing tasks without intervention from a third party.

Activity Synchronization

In general, a task must synchronize its activity with other tasks to execute a multithreaded program properly. *Activity synchronization* is also called *condition synchronization* or *sequence control* . Activity synchronization ensures that the correct execution order among cooperating tasks is used. Activity synchronization can be either synchronous or asynchronous.

One representative of activity synchronization methods is *barrier synchronization* . For example, in embedded control systems, a complex computation can be divided and distributed among multiple tasks. Some parts of this complex computation are I/O bound, other parts are CPU intensive, and still others are mainly floating-point operations that rely heavily on specialized floating-point coprocessor hardware. These partial results must be collected from the various tasks for the final calculation. The result determines what other partial computations each task is to perform next.

Cladding communication:

Cladding is one or more layers of materials of lower refractive index, in intimate contact with a core material of higher refractive index. It is used in optical fibers. The cladding causes light to be confined to the core of the fiber by total internal reflection at the boundary between the two. Light propagation in the cladding is suppressed in typical fiber. Some fibers can support cladding modes in which light propagates in the cladding as well as the core

Explain about optical fibre communication for cladding communication.

Wireless Communication Protocols:

Pg 174 book

Washing Machine as an embedded System

Washing machine supports three functional modes:

i) Fully Automatic Mode:

In fully automatic mode, once the system is started it perform independently without user interference and after the completion of work it should notify the user about the completion of work. This mode instantaneously sense cloth quality and requirement of water, water temperature, detergent, load, wash cycle time and perform operation accordingly.

ii) Semi Automatic Mode:
In this semiautomatic mode in which washing conditions are predefined. Once the predefined mode is started the system performs its job and after completion it informs the user about the completion of work.

iii) Manual Mode:
In this mode, user has to specify which operation he wants to do and has to provide related information to the control system. For example, if user wants to wash clothes only, he has to choose 'wash' option manually. Then the system asks the user to enter the wash time, amount of water and the load. After these data are entered, the user should start the machine. When the specified operation is completed system should inform the user.

Remember that Modes should be a selectable by a keypad.

A washing machine may have a System Controller (Brain of the System) which provides the power control for various monitors and pumps and even controls the display that tells us how the wash cycles are proceeding. A washing machine comprises several components as shown in Figure below.

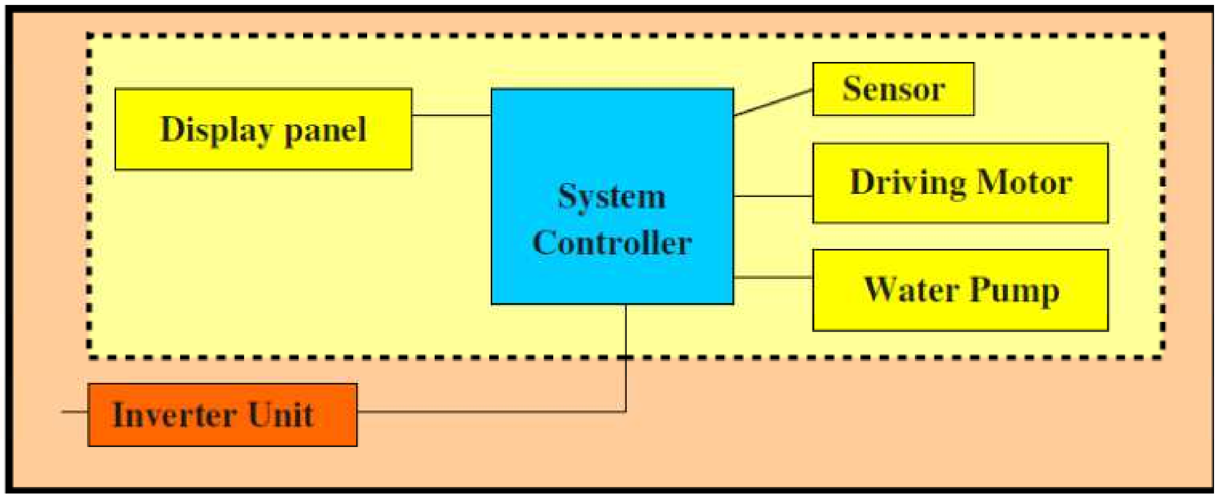


Figure: Block diagram of washing machine

The working of these components is as follows:

i) Display Panel: It is a touch panel screen to control all the operations of a machine

ii) Sensor: It measures the water level and appropriate amount of soap. Input devices for automatic washing machine are sensors for water flow, water level and temperature; door switch; selector knob or buttons for settings such as spin speed, temperature, load size and types of wash cycle required.

Water Level Sensor: It indicates beep sound when water level is low in washing tub. *Door Sensor*: It indicates beep sound when all clothes are washed that means now you can open the maching door and also you can move to your next phase. Next phase will be dry Phase. This phase also follows same concept for drying the clothes.

iii) Driving Motor: Motor can rotate in two directions either “reverse’ or ‘forward’. The forward direction drives the current in forward direction and motor rotates forward. The reverse direction driver does the opposite of it. A washing machine
can maintain single motor in fully automatic or double motor in semi automatic washing machine.

Sequence of washing the clothes with this can be explained in few steps as follows:
1) Put on your dirty clothes on to the wash tub for washing
2) Put the detergent Soap (of your choice like Surf n Excel etc.)
3) Put ON the tap, water rushes inside the tub.
4) If its electronic control , then by the press of the keys ,you could program , if its mechanical it shall something like an mechanical switches wherein you are allowed to operate for setting the wash time.
5) Now the wash motor rotates and washes the clothes and gives you a beep sound
6) Now your clothes are washed …remove it from the wash tub and put it on the spin tub and program it accordingly…after spinning clothes are dried and you are allowed to hang it for proper drying in sunlight.

The fully automatic also comes in two category front loading as well as top loading.

i) Front loading is the one wherein you are given an opening to put clothes in on the front side.

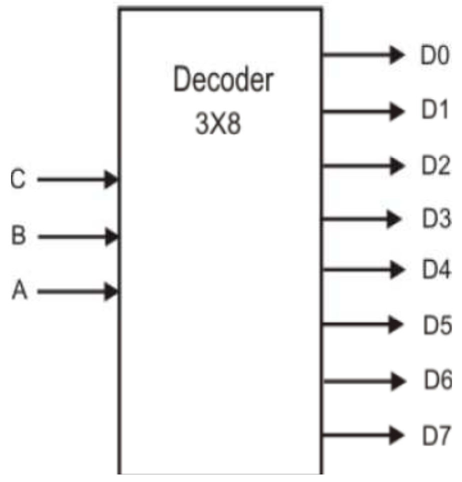
ii) Top loading is on the top.

iv) System Controller: Such Component is used to control the motor speed. Motor can move in forward direction as well as reverse direction. System Controller reads the speed of motor and controls the speed of motor in different phases such as in Washing, Cleaning Drying etc. All kinds of Sensors such as Door Sensor, Pressure Sensor and Keypad, Speed sensor are also maintained by this.

v) Water Pump: The water pump is used to recirculate water and drain out the dirty water. This pump actually contains two separate pumps inside one: The bottom half of the pump is hooked up to the drain line, while the top half recirculates the wash water. The motor that drives the pump can reverse direction. It spins one way when the washer is running a wash cycle and recirculates the water; and it spins the other way when the washer is doing a spin cycle and draining the water.

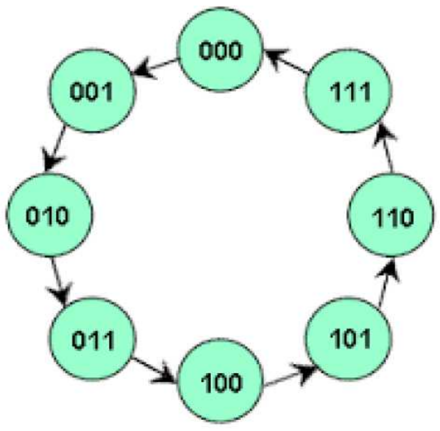
1. Implement 3-to-8 bit decoder in vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity decoder is
port ( sw : IN STD_LOGIC_VECTOR(2 downto 0);
led : OUT STD_LOGIC_VECTOR(7 downto 0));
end decoder;

architecture Behavioral of decoder is
begin
led <= "00000001" when sw = "000" else
"00000010" when sw = "001" else
"00000100" when sw = "010" else
"00001000" when sw = "011" else
"00010000" when sw = "100" else
"00100000" when sw = "101" else
"01000000" when sw = "110" else
"10000000" when sw = "111";
end Behavioral;
```



2. Implement counter as shown below in vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity counter is
Port ( CLK : in STD_LOGIC;
Count : out STD_LOGIC_VECTOR (2 downto 0));
end counter;
architecture Behavioral of counter is
signal cin : std_logic_vector(2 downto 0) :="000";
begin
process(CLK)
begin
if(rising_edge(CLK)) then
if(cin = "111") then
cin <= "000";
else
cin <= cin + 1;
end if;
end if;
Count <= cin;
end process;
end Behavioral;
```



3. Structure Model example

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity and_gate is
Port ( c : in STD_LOGIC;
d : in STD_LOGIC;
y2 : out STD_LOGIC);
end and_gate;

architecture Behavioral of and_gate is
begin
y2 <= c and d;
end Behavioral;

--Second Module: OR Gate
entity or_gate is
Port ( a : in STD_LOGIC;
b : in STD_LOGIC;
y1 : out STD_LOGIC);
end or_gate;
architecture Behavioral of or_gate is
begin
y1 <= a or b;
end Behavioral;

--Top Module:
entity top_level_model is
Port ( i1,i2,i3,i4: in STD_LOGIC;
o : out STD_LOGIC);
end top_level_model;
architecture Behavioral of top_level_model is
COMPONENT or_gate is
PORT(a,b: in STD_LOGIC;
y1: out STD_LOGIC);
END COMPONENT;
COMPONENT and_gate is
PORT(c,d: in STD_LOGIC;
y2: out STD_LOGIC);
END COMPONENT;
SIGNAL w1, w2: STD_LOGIC;
begin
--port mapping
component_1 : or_gate port map (a => i1, b => i2, y1 => w1);
component_2 : and_gate port map (c => i3 ,d => i4, y2 => w2);
component_3 : and_gate port map (c => w1, d => w2, y2 => o);
end Behavioral;
```