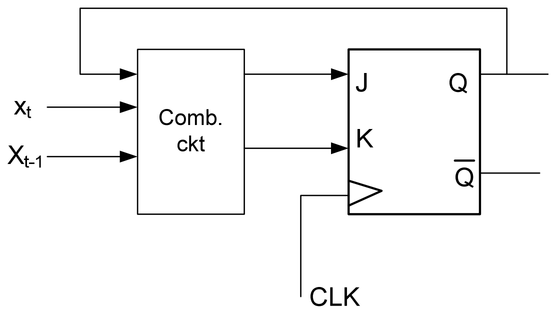
Tutorial PART 2 (SOLUTIONS)

NOTE THAT THERE ARE OTHER POSSIBLE SOLUTIONS TOO…

20. i) Design based on J-K FF



Next state table (from the problem specification):

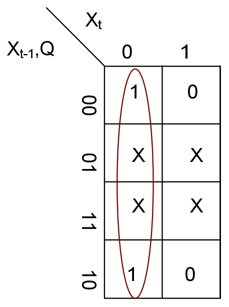
X(t)	x(t-1)	Q	Q+
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

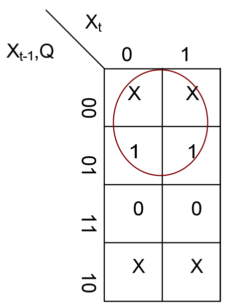
We need to use the excitation table of the J-K FF

Q	Q^+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

The truth table for the combination circuit is:

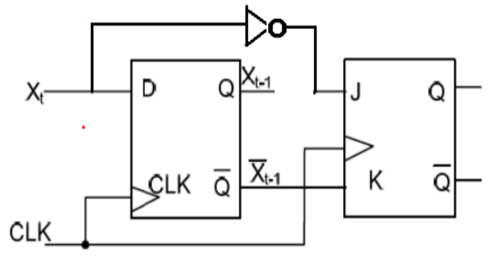
X(t)	x(t-1)	Q	Q+	J	K
0	0	0	1	1	x
0	0	1	0	x	1
0	1	0	1	1	x
0	1	1	1	x	0
1	0	0	0	0	x
1	0	1	0	x	1
1	1	0	0	0	x
1	1	1	1	x	0





$J = \overline{X}_t$ $K = \overline{X}_{t-1}$

Now, how to get X_{t-1} from X_t? Use D-FF.



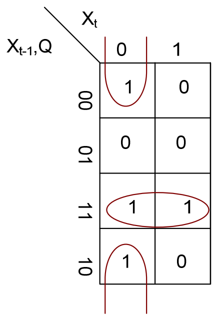
ii) Design based on D-FF

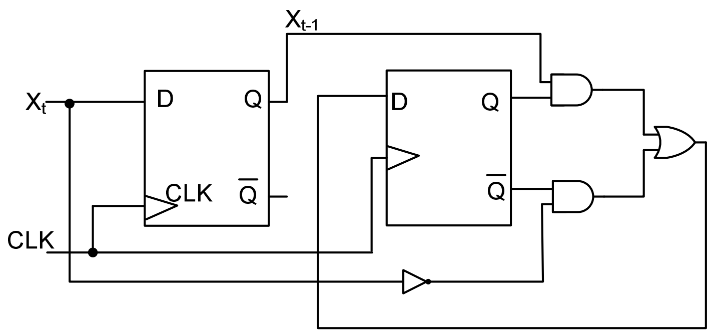
The next state table X_{t-1} , X_t , Q →Q+ remains same as before.

With D FF, the truth table for the combination circuit is:

X(t)	x(t-1)	Q	Q+ / D
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

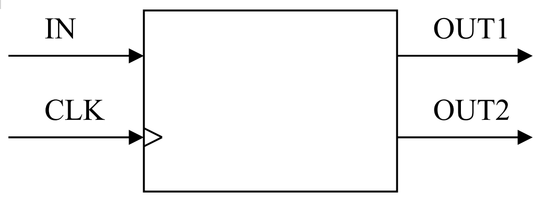
$$D = \overline{X}_t\overline{Q} + X_{t-1}Q$$

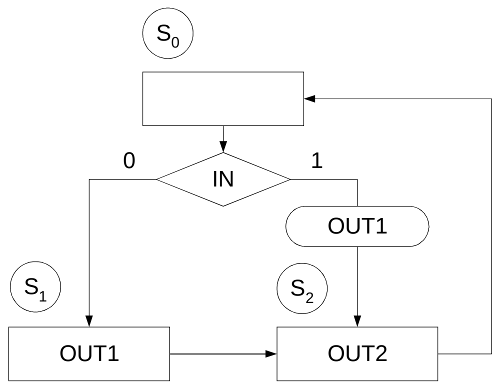




D-FF design needs more components than J-K FF design because D-FF is capable of only very simple behavior compared to J-K , i.e. D=Q+ vs. hold, set , reset and toggle.

21.





	IN	
	0	1
S_0	S_1	S_2
S_1	S_2	S_2
S_3	S_0	S_0
S_2	S_0	S_0

Next State Table for Min Risk Design

C_1C_0 \ IN	0	1
00	1	0
01	0	0
11	0	0
10	0	0

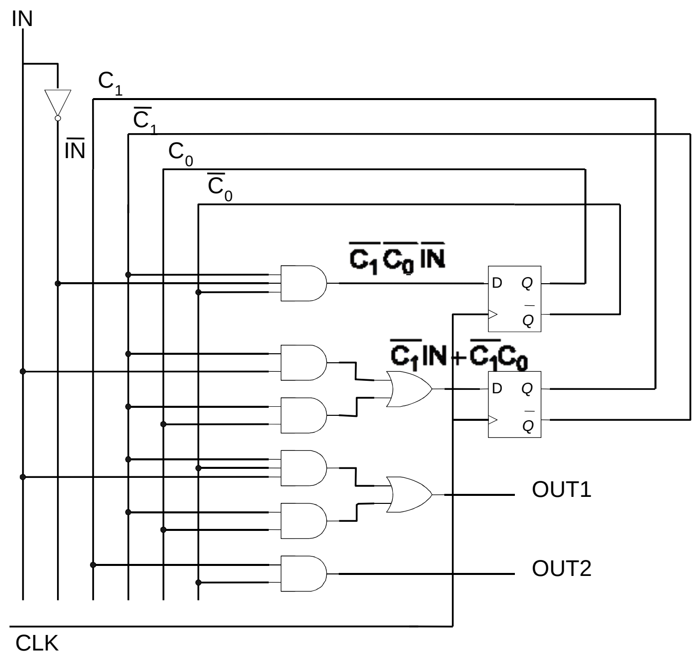
C_1C_0 \ IN	0	1
00	0	1
01	1	1
11	0	0
10	0	0

	C1	C0
S_0	0	0
S_1	0	1
S_2	1	0
S_3	1	1

State Assignment

$$C_0^+ = \overline{C_1}\ \overline{C_0}\ \overline{IN} \qquad C_1^+ = \overline{C_1}\,IN + \overline{C_1}C_0$$

$$OUT1 = S_0 IN + S_1 = \overline{C_1}\ \overline{C_0}\ IN + \overline{C_1}C_0 \qquad OUT2 = S_2 = C_1\overline{C_0}$$



<u>VHDL Code</u>

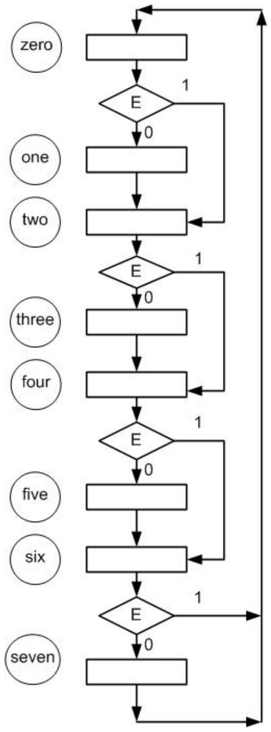
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ASMPROB2 is
    Port ( INP : in  STD_LOGIC;  --in is a reserved word
           CLOCK: in STD_LOGIC; OUT1: out STD_LOGIC; OUT2: out STD_LOGIC);
```

```
end ASMPROB2;

architecture Arch_ASMPROB2 of ASMPROB2 is
type states is (S0, S1, S2);
signal state:states;
begin

process(clock)
begin
if clock'event and clock = '1' then
      case state is
            when S0=>
                  OUT2 <= '0';
                  if INP = '0' then
                        state <= S1;
                        OUT1 <= '0';
                  else
                        state <= S2;
                        OUT1 <= '1';
                  end if;
            when S1 =>
                  state <= S2;
                  OUT1 <= '1';
            when S2 =>
                  state <= S0;
                  OUT1 <= '0';
                  OUT2 <= '1';
      end case;
end if;
end process;
end Arch_ASMPROB2;
```

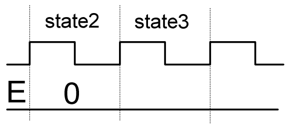
SS10

(a) Synchronous input → the input signal changes value only "at" the active clock edge, i.e the clock is one of the signals that is causing the input to change value. In practice, the input will not change value exactly at the clock edge, but after a small propagation delay t_{pd} (few nanoseconds).

b)

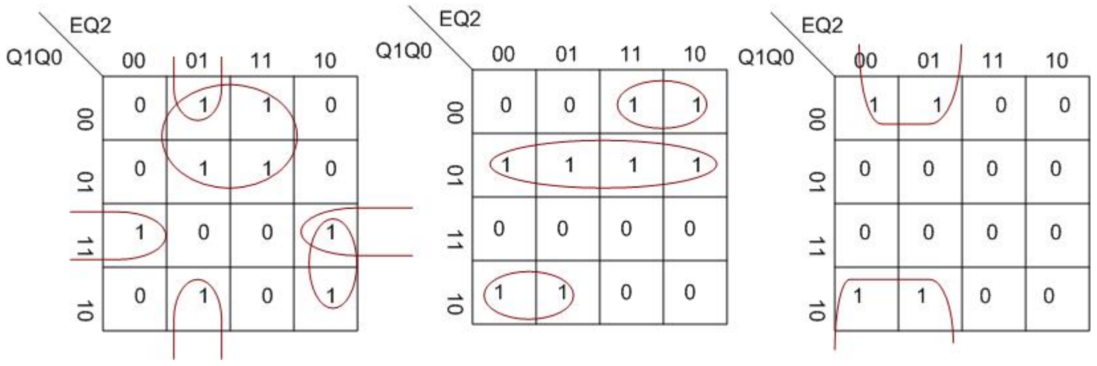


(c) In state 2, if E=0 the next state will be state 3. The value of E=0 in the clock cycle corresponding to state 2, just before the next clock edge comes will make the machine goto state 3. The machine will go to state 3 t_{pd} after the next edge. *(See figure below)*



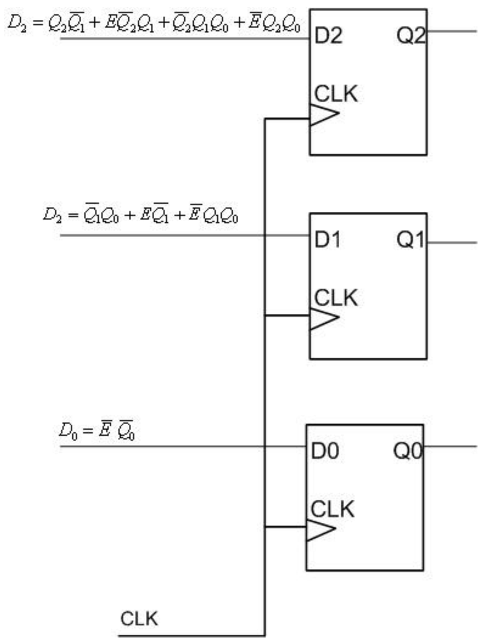
d)

E	Q2	Q1	Q0	Q2+ / D0	Q1+ / D1	Q0+ / D0
0	0	0	0	0	0	1
0	0	0	1	0	1	0
0	0	1	0	0	1	1
0	0	1	1	1	0	0
0	1	0	0	1	0	1
0	1	0	1	1	1	0
0	1	1	0	1	1	1
0	1	1	1	0	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	0	1	1	1	0	0
1	1	0	0	1	1	0
1	1	0	1	1	1	0
1	1	1	0	0	0	0
1	1	1	1	0	0	0



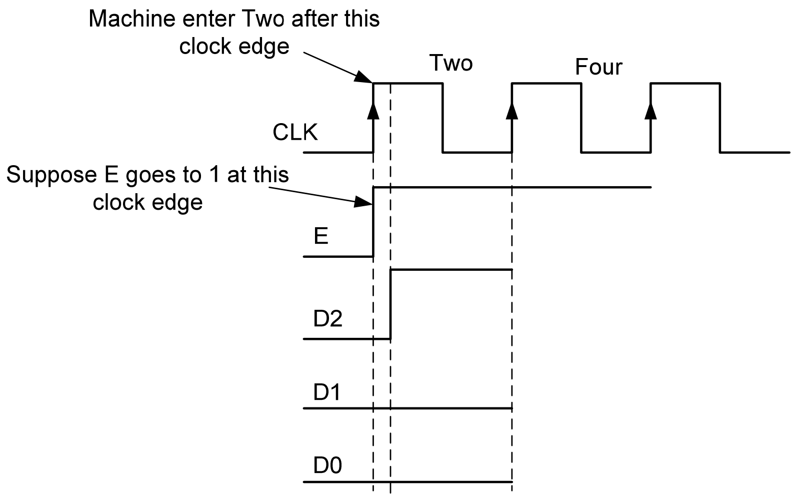
$$
\begin{array}{c|cccc}
Q1Q0 \backslash EQ2 & 00 & 01 & 11 & 10 \\
\hline
00 & 0 & 1 & 1 & 0 \\
01 & 0 & 1 & 1 & 0 \\
11 & 1 & 0 & 0 & 1 \\
10 & 0 & 1 & 0 & 1
\end{array}
\qquad
\begin{array}{c|cccc}
Q1Q0 \backslash EQ2 & 00 & 01 & 11 & 10 \\
\hline
00 & 0 & 0 & 1 & 1 \\
01 & 1 & 1 & 1 & 1 \\
11 & 0 & 0 & 0 & 0 \\
10 & 1 & 1 & 0 & 0
\end{array}
\qquad
\begin{array}{c|cccc}
Q1Q0 \backslash EQ2 & 00 & 01 & 11 & 10 \\
\hline
00 & 1 & 1 & 0 & 0 \\
01 & 0 & 0 & 0 & 0 \\
11 & 0 & 0 & 0 & 0 \\
10 & 1 & 1 & 0 & 0
\end{array}
$$

$$D_2 = Q_2\overline{Q}_1 + E\overline{Q}_2Q_1 + \overline{Q}_2Q_1Q_0 + \overline{E}Q_2Q_0$$
$$D_2 = \overline{Q}_1Q_0 + E\overline{Q}_1 + \overline{E}Q_1Q_0$$
$$D_0 = \overline{E}\ \overline{Q}_0$$



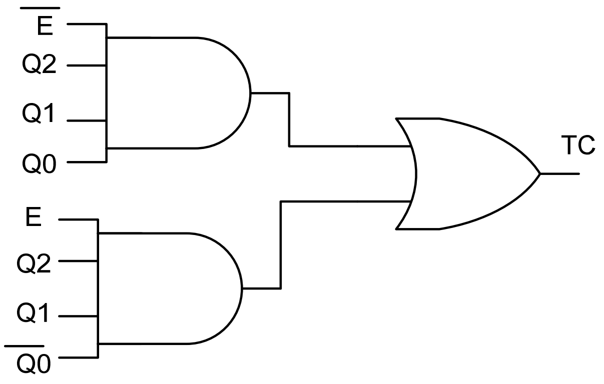
$D_2 = Q_2\overline{Q}_1 + E\overline{Q}_2Q_1 + \overline{Q}_2Q_1Q_0 + \overline{E}Q_2Q_0$
D2
Q2
CLK
$D_2 = \overline{Q}_1Q_0 + E\overline{Q}_1 + \overline{E}Q_1Q_0$
D1
Q1
CLK
$D_0 = \overline{E}\ \overline{Q}_0$
D0
Q0
CLK
CLK

e)

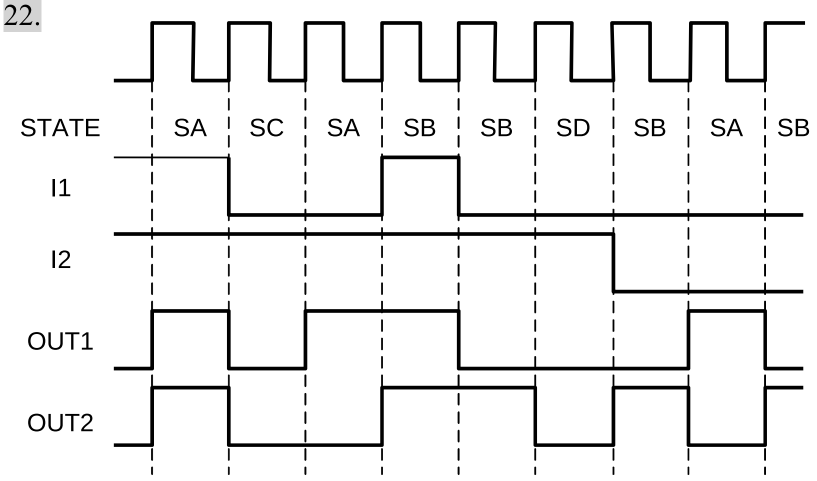

Machine enter Two after this clock edge
Two
Four
CLK
Suppose E goes to 1 at this clock edge
E
D2
D1
D0

For E=1, Q_2=0, Q_1=1 and Q_0=0, the combinational circuit will calculate D_2=1, D_1=0, D_0=0 after t_{pd} (as shown).
These signals are waiting at the respective FF inputs. When the next clock edge comes, the machine will goto Q_2^+=1, Q_1^+=0 and Q_0^+=0, i.e. state Four.

f) $TC = \overline{E}Q_2Q_1Q_0 + EQ_2Q_1\overline{Q_0}$



22.



	I_1, I_2			
	00	01	11	10
SA	SB	SB	SC	SC
SB	SA	SD	SB	SB
SC	SA	SA	SA	SA
SD	SB	SB	SB	SB

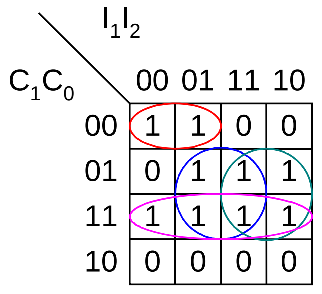
Next State Table

	C1	**C0**
SA	0	0
SB	0	1
SC	1	0
SD	1	1

State Assignment

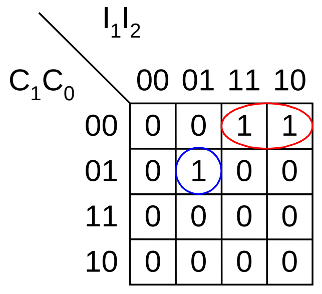
Traditional Method

Excitation Equations



C_1C_0 \ I_1I_2	00	01	11	10
00	1	1	0	0
01	0	1	1	1
11	1	1	1	1
10	0	0	0	0

$$C_0^+ = \overline{I_1}\,\overline{C_1}\,\overline{C_0} + C_1C_0 + C_0I_2 + I_1C_0$$



C_1C_0 \ I_1I_2	00	01	11	10
00	0	0	1	1
01	0	1	0	0
11	0	0	0	0
10	0	0	0	0

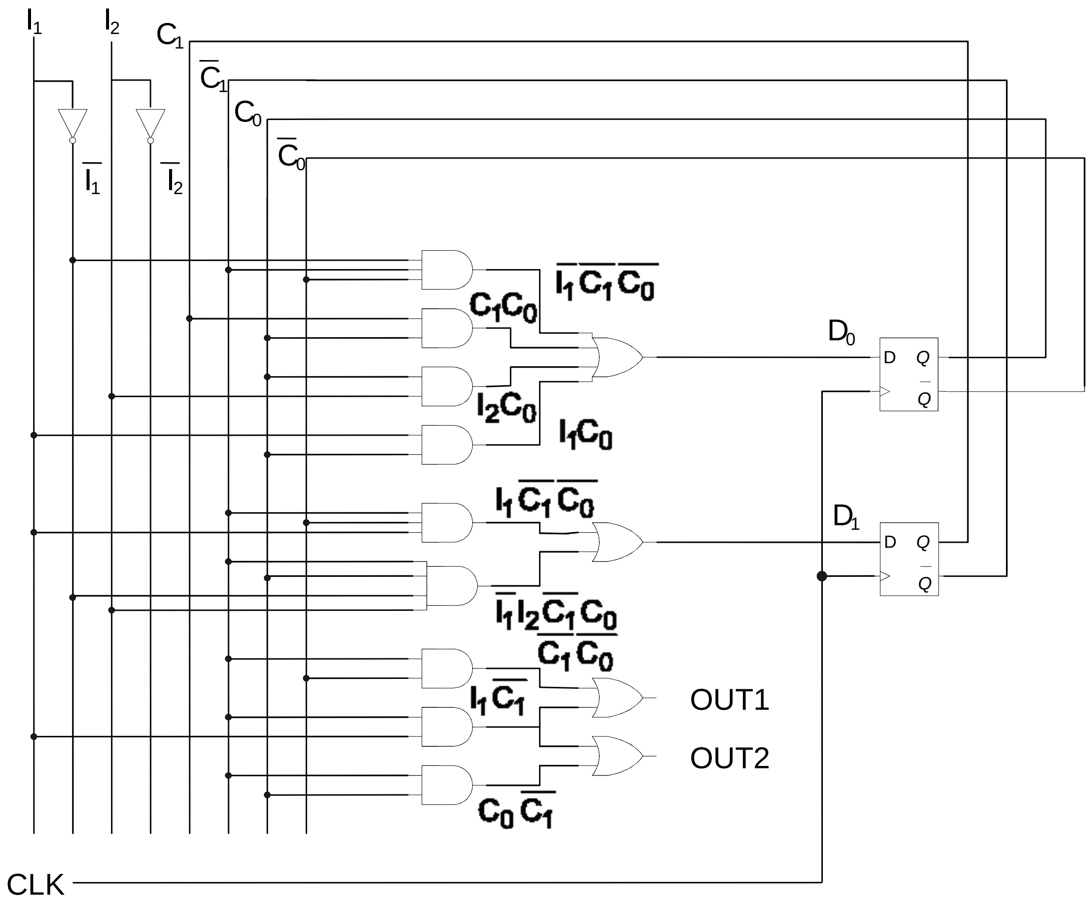
$$C_1^+ = \overline{C_1}\,\overline{C_0}\,I_1 + \overline{C_1}\,C_0\,\overline{I_1}\,I_2$$

Output Equations

From ASM chart:

$$\begin{aligned} OUT1 &= SA + SB.I_1 \\ &= \overline{C_1}\,\overline{C_0} + \overline{C_1}C_0I_1 \\ &= \overline{C_1}\,\overline{C_0} + \overline{C_1}I_1 \end{aligned}$$

$$\begin{aligned} OUT2 &= SA.I_1 + SB \\ &= \overline{C_1}\,\overline{C_0}\,I_1 + \overline{C_1}C_0 \\ &= \overline{C_1}C_0 + \overline{C_1}I_1 \end{aligned}$$



PLA Based Design

Given the 4-input, 4-output, 8 AND-term PLA, we can realize the combinational circuit required for the state generator with it, and realize the output logic separately. Or we can challenge ourselves to see if all of the combinational logic for the state generator as well as the output can be realized entirely with the PLA. If you count all the AND terms that will be required to implement the latter, you will find that you need 12 AND gates; however, the PLA has only 8 AND gates.

Below, it is shown, how we can group minterms selectively with Karnaugh maps to reduce the total number of AND terms required to less than 8. The strategy is to have as many common groupings as possible from the 4 maps.

C_1C_0 \ I_1I_2	00	01	11	10
00	1	1	0	0
01	0	1	1	1
11	1	1	1	1
10	0	0	0	0

$$C_0^+ = \overline{I_1}\,\overline{C_1}\,\overline{C_0} + C_1C_0 + \overline{C_1}C_0\overline{I_1}I_2 + I_1\overline{C_1}C_0$$

C_1C_0 \ I_1I_2	00	01	11	10
00	0	0	1	1
01	0	1	0	0
11	0	0	0	0
10	0	0	0	0

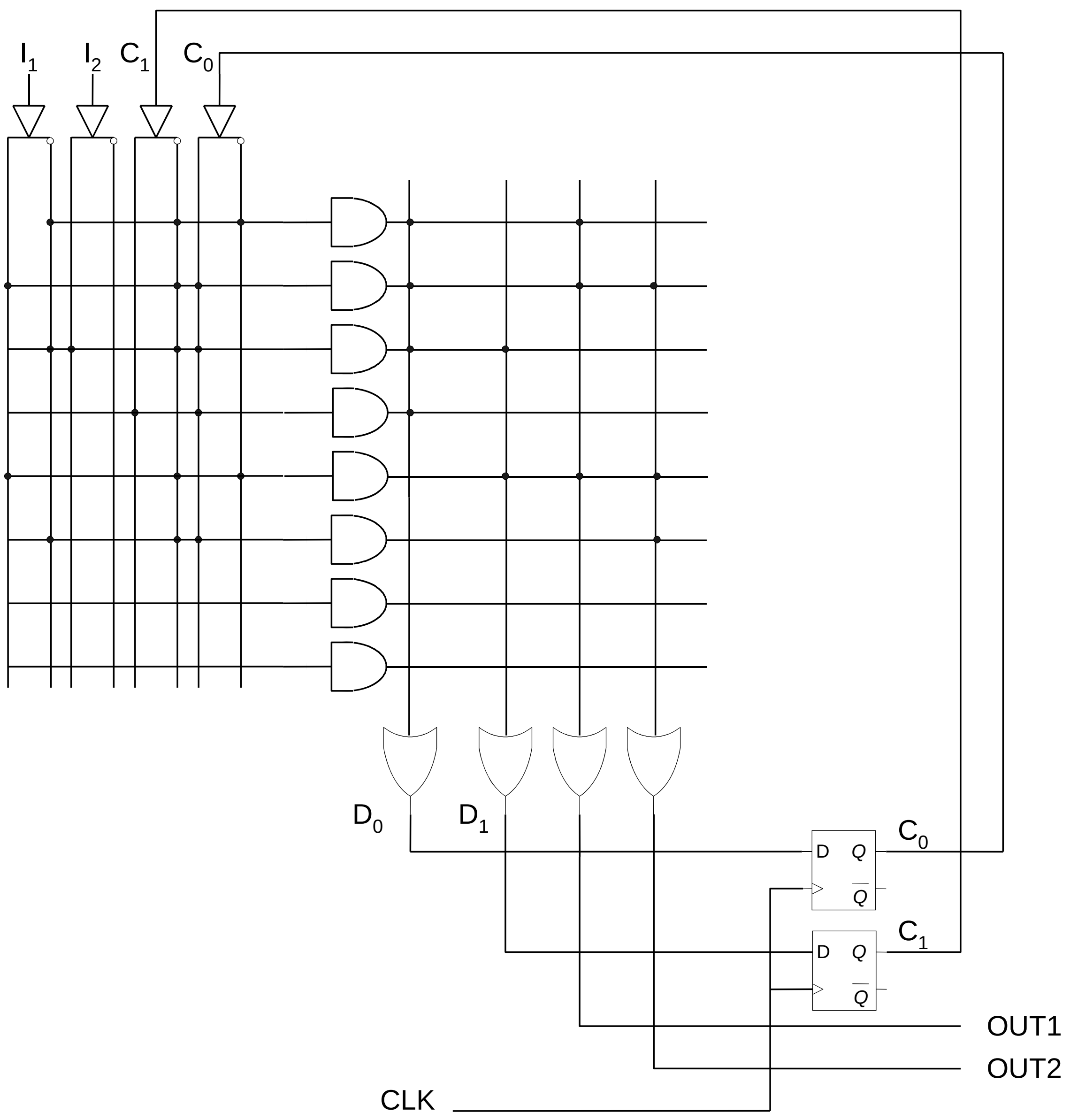
$$C_1^+ = \overline{C_1}\,\overline{C_0}I_1 + \overline{C_1}C_0\overline{I_1}I_2$$

C_1C_0 \ I_1I_2	00	01	11	10
00	1	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	0	0

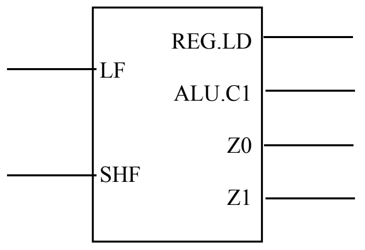
$$OUT1 = \overline{C_1}\,\overline{C_0}\,\overline{I_1} + \overline{C_1}\,\overline{C_0}\,I_1 + \overline{C_1}\,C_0 I_1$$

C_1C_0 \ I_1I_2	00	01	11	10
00	0	0	1	1
01	1	1	1	1
11	0	0	0	0
10	0	0	0	0

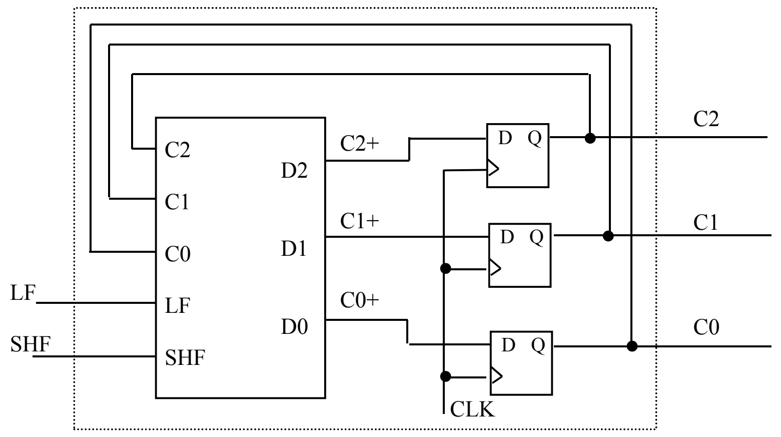
$$OUT2 = \overline{C_1}\,C_0\,\overline{I_1} + \overline{C_1}\,\overline{C_0}\,I_1 + \overline{C_1}\,C_0 I_1$$



23.



(a). State MC block diagram



(b). State generator block diagram

(c). Make the following state assignments (arbitrary)

S0: 000,   S1: 001,   S2: 010,   S3: 011,   S4:100

Then construct the next state table based on assigned states.

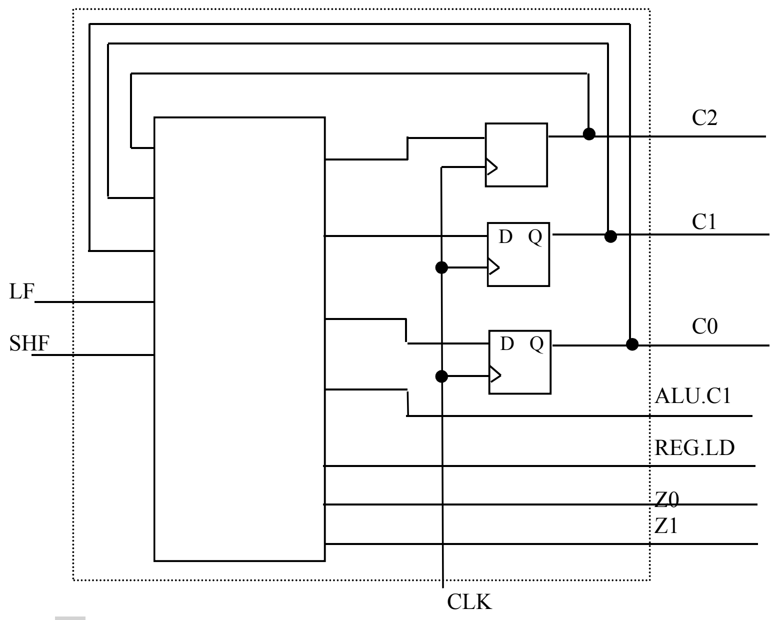
The idea is to use a ROM to realize the combinational circuit of the state generator as well as the output signals. Specifically, we will use the current state bits ($C_2C_1C_0$) and the inputs (LF and SHF) as the address signals of a ROM. Each memory location will have 7 bits of data, viz. C2+, C1+, C0+ (the next state bits) and REG.LD, ALU.C1, Z1 and Z0 (4 output signal bits). This will require us to use a ROM with 32 x 7 capacity. Construct the next state table as follows (by inspection of the ASM chart).

C2	C1	C0	LF	SHF	C2+ (D2)	C1+ (D1)	C0+ (D0)	REG.LD	ALU.C1	Z1	Z0
0	0	0	0	0	0	0	1	1	0	0	0
0	0	0	0	1	0	1	0	1	0	0	1
0	0	0	1	X	0	1	1	1	0	0	0
0	0	1	X	X	0	1	0	0	1	0	0
0	1	0	X	X	0	0	0	0	0	0	0
0	1	1	X	0	1	0	0	0	1	0	0
0	1	1	X	1	0	1	1	0	1	1	0
1	0	0	X	X	0	0	0	0	0	0	0

Now make the following correspondence between ROM address inputs and data outputs
Also decide that if controller ever gets into states 5,6 or 7 a transition to state S0 is made.

ROM Addr. Input	Signal Name
A4	C2
A3	C1
A2	C0
A1	LF
A0	SHF

Data Output	Signal Name
D6	C2+ (D2)
D5	C1+ (D1)
D4	C0+ (D0)
D3	REG.LD
D2	ALU.C1
D1	Z1
D0	Z0

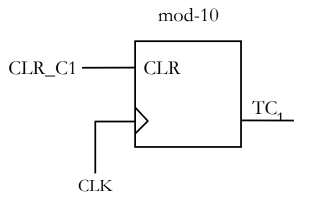


ROM Address	Contents
0	18H
1	29H
2,3	38H
4-7	24H
8-11	00H
12	44H
13	36H
14	44H
15	36H
16-19	00H
20-31	00H

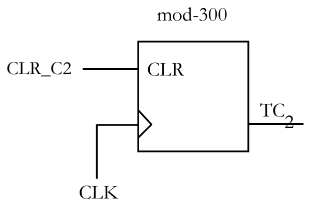
24.

For the architectural elements, we choose 2 counters: (a) A decade counter with synchronous clear and terminal count output. (b) A mod-300 counter with synchronous clear and a terminal count output.

Architectural components

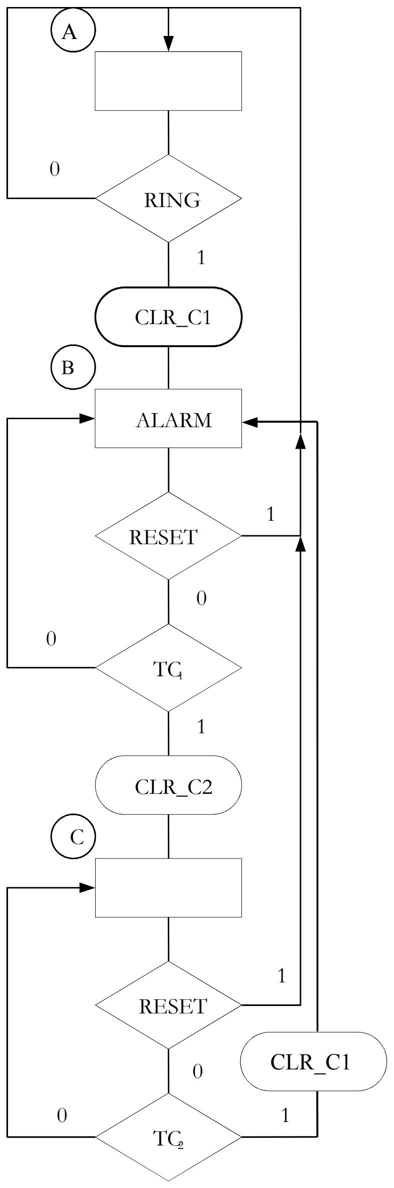


$TC_1 \rightarrow 1$ on count of 9



$TC_2 \rightarrow 1$ on count of 299

ASM Chart for hardware realization using architectural components.

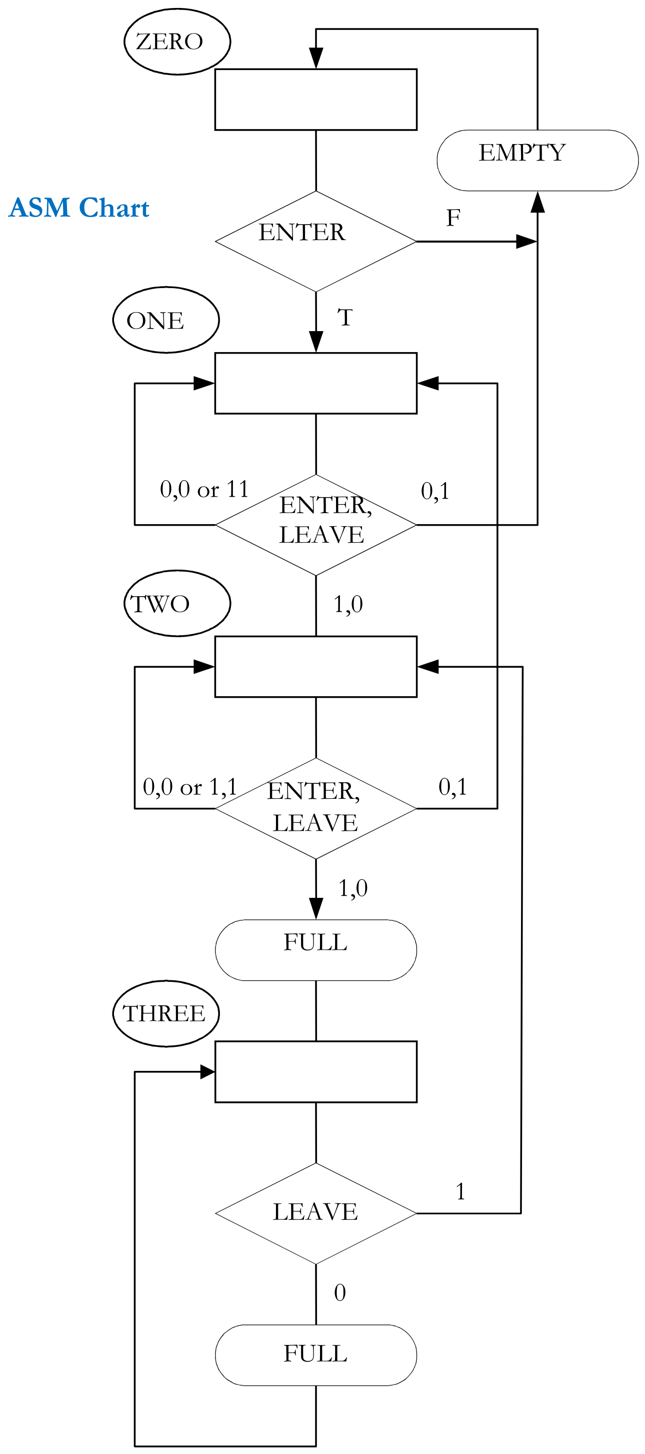


VHDL Code

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

-- this is a VHDL realization of the Alarm Clock state machine
-- ASM Chart for this realization can be easily inferred

entity ALARM_CLOCK is
    Port ( RING : in  STD_LOGIC;
                    CLOCK : in  STD_LOGIC;
           RESET : in  STD_LOGIC;
           ALARM : out  STD_LOGIC);
end ALARM_CLOCK;

architecture Arch_ALARM_CLOCK of ALARM_CLOCK is
type states is (IDLE, RING_ALARM, WAITING);
--WAIT is a VHDL reserved word and shouldn't be used as a state name
signal state:states;
begin

process(clock)  --synch state machine
variable counter1 : std_logic_vector(3 downto 0); -- 10 seconds
variable counter2 : std_logic_vector(8 downto 0); -- 300 seconds
begin
if clock'event and clock = '1' then
      case state is
            when IDLE =>
                  ALARM <= '0';
                  if RING = '1' then
                        state <= RING_ALARM;
                        counter1 := "0000";
                  end if;
            when RING_ALARM =>
                  ALARM <= '1';
                  if RESET = '1' then
                        state <= IDLE;
                  elsif counter1 = 9 then
                        state <= WAITING;
                        counter2 := (others=>'0');
                  --easier way to write counter2 <= "000000000";
                  else
                        counter1 := counter1+1;
                  end if;
            when WAITING =>
                  ALARM <= '0';
                  if RESET = '1' then
                        state <= IDLE;
                  elsif counter2 = 299 then
                        state <= RING_ALARM;
                        counter1 := "0000";
                  else
                        counter2 := counter2+1;
                  end if;
      end case;
end if;
end process;
end Arch_ALARM_CLOCK;
```

SS11 Write a *VHDL* program which realizes the circuit and verify its performance using available simulation tools.

25.



VHDL Code

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

entity PEOPLE_COUNTER is
port(CLOCK, RESET, ENTER, LEAVE :in std_logic;
      FULL, EMPTY: out std_logic);
end PEOPLE_COUNTER;

architecture Arch_PEOPLE_COUNTER of PEOPLE_COUNTER is
type states is (state0, state1, state2, state3);
signal state: states;
begin
process (CLOCK, RESET)
variable enterleave: std_logic_vector(1 downto 0) := "00";
begin
if RESET='1' then
      state <= state0; -- asynchronous reset
      FULL <= '0'; EMPTY <= '1';
elsif (CLOCK'event and CLOCK='1') then
enterleave(0):=LEAVE;
enterleave(1):=ENTER;
      case state is
      when state0 => -- zero person in room .. initial state
            if ENTER = '1' then
                  state <= state1;
                  FULL <= '0'; EMPTY <= '0';
            end if;
      when state1 => -- one person in room
            case enterleave is
            when "00" => state <= state1;
            when "10" => state <= state2;
            when "01" => state <= state0; EMPTY <= '1';
            when others => state <= state1;
            end case;
      when state2 => -- two persons in room
            case enterleave is
            when "00" => state <= state2;
            when "10" => state <= state3; FULL <= '1';
            when "01" => state <= state1;
            when others => state <= state2;
            end case;
      when state3 => -- three persons in room
            if LEAVE = '1' then
                  state <= state2;
                  FULL <= '0';
            end if;
      end case;
end if;
end process;
end Arch_PEOPLE_COUNTER;
-- Note: in ASM charts, output signals appear in states where they are to be asserted ("conditional" or "unconditional")
--        however, in the VHDL programs, these output signals remain at '1' or '0' until they are updated.
```