BILKENT UNIVERSITY
Department of Electrical and Electronics Engineering
EEE102 Introduction to Digital Circuit Design
Midterm Exam II SOLUTION

14-12-2007

Duration 120 minutes

Surname: _____

Name: _____

ID-Number: _____

Signature: _____

There are 5 questions. Solve all.
Do not detach pages. Show all your work.

Q1	
Q2	
Q3	
Q4	
Q5	
Total	

Q1. (20 points)

Use one 74x163 binary up counter and minimal combinational logic to design a counter which has the following properties:

i) The count becomes 0 at Power ON.

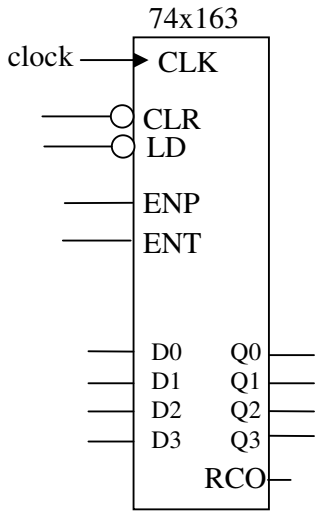
ii) It counts as 0,1,2,3,4,5 and stops at 5.

iii) While the count is 5 if the signal RESTART is 1 then it goes to 0 and continues to count up to 5 again.

iv) It is not self-correcting and if an undesired count is encountered then the signal FAULT becomes 1, else it is 0.

v) However while it is in an undesired count if RESTART is 1, then it goes to 0 and counts up to five again.

(You can assume that at Power ON the 74x163 outputs are 0. Also remember that CLR has precedence over LD in 74x163)



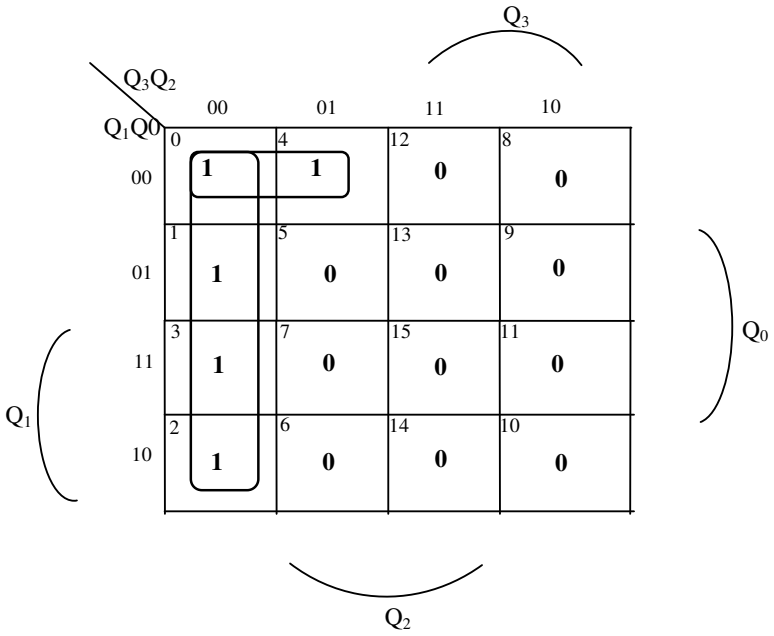
SOLUTION:

Since it is not self-correcting let us assume that if it goes to an undesired state it remains there. Thus if it is at state 5 or at an undesired state then we disable the counter.

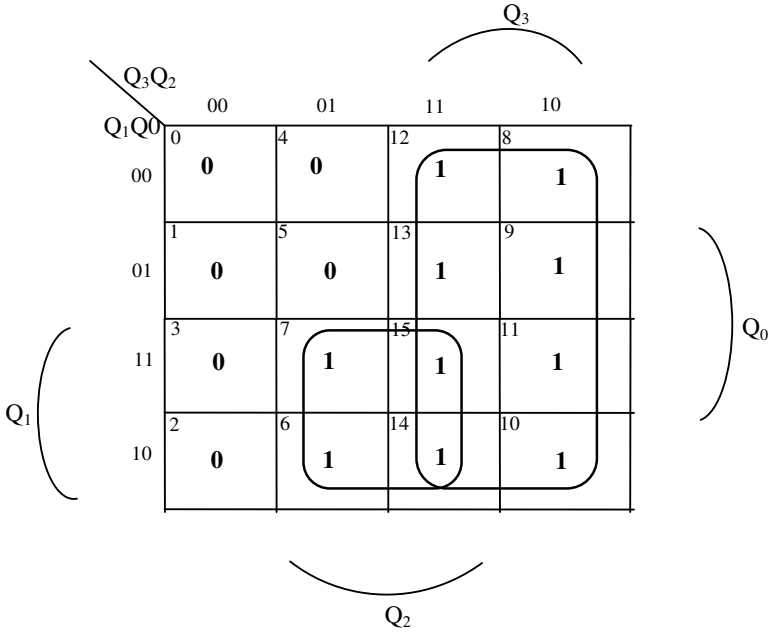
Thus

Q3	Q2	Q1	Q0	EN	FAULT
0	0	0	0	1	0
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	1	0
0	1	0	0	1	0
0	1	0	1	0	0
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	0	1
1	0	0	1	0	1
1	0	1	0	0	1

1	**0**	**1**	**1**	**0**	**1**
1	**1**	**0**	**0**	**0**	**1**
1	**1**	**0**	**1**	**0**	**1**
1	**1**	**1**	**0**	**0**	**1**
1	**1**	**1**	**1**	**0**	**1**

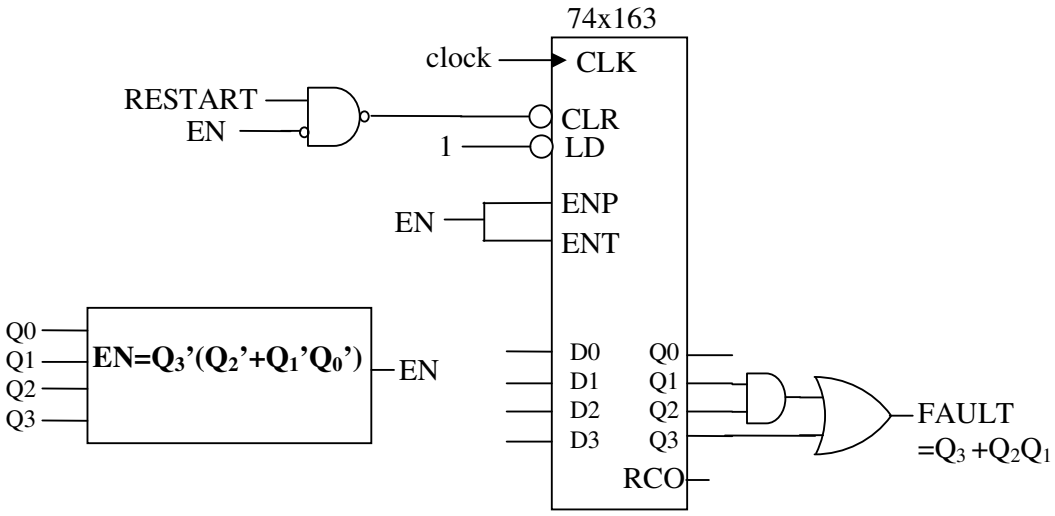


EN = $Q_3'Q_2'+ Q_3'Q_1'Q_0' = Q_3'(Q_2'+ Q_1'Q_0')$



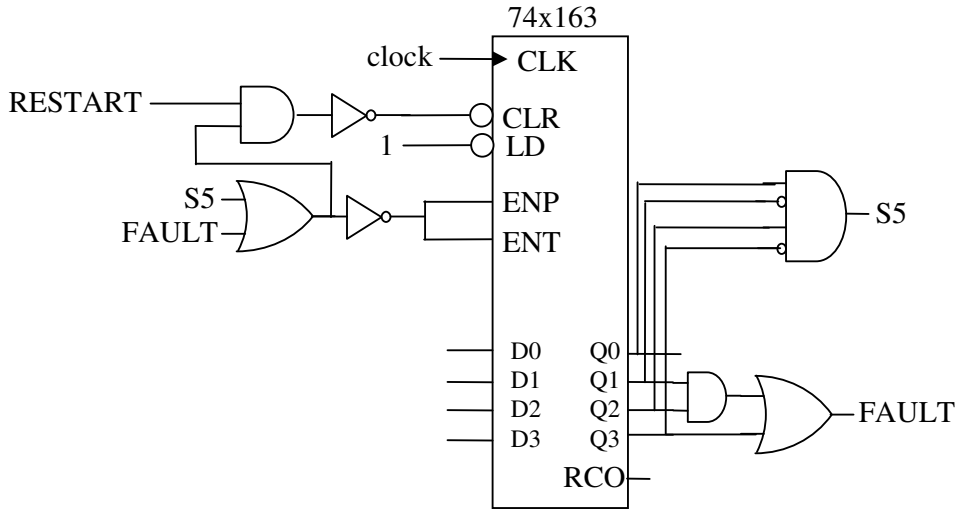
FAULT = $Q_3 + Q_2Q_1$

However if we are at state 5 or at an undesired state and also if RESTART is 1 then we go to 0. Thus CLR_L = (EN'. RESTART)'



Grading:
5 points FAULT
5 points Counts up to 5 and stops
4 points Is not self-correcting
6 points RESTART (is effective when in state5 3 points, is effective when in undesired state 3 points)

Another solution:



In this solution S5 = $Q_3'Q_2Q_1'Q_0$, however one can also have S5 = Q_2Q_0.

Q2. (20 points)
For the counter described in Question 1 write VHDL code. Look at the appendix for VHDL templates.

SOLUTION:

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

entity MT2_Q2_V1 is
    Port ( CLK : in  STD_LOGIC;
           RESTART : in  STD_LOGIC;
           FAULT : out  STD_LOGIC);
end MT2_Q2_V1;

architecture Behavioral of  MT2_Q2_V1 is
signal Q:std_logic_vector(2 downto 0) :="000";
signal D:std_logic_vector(2 downto 0);
begin
--state memory
process(CLK)
begin
if CLK'event and CLK='1' then Q<=D;end if;
end process;
--next state logic
process(Q,RESTART)
begin
case Q is
      when "000"=> D<="001";
      when "001"=> D<="010";
      when "010"=> D<="011";
      when "011"=> D<="100";
      when "100"=> D<="101";
      when "101"=> D<="101";
      when "110"=> D<="110";
      when "111"=> D<="111";
      when others => D<="000";
end case;
if (Q="101" or Q="110" or Q="111") and RESTART='1' then D<="000"; end if;
end process;
--output logic
FAULT<='1' when (Q="110" or Q="111") else '0';
end Behavioral;
```

3 points entity (-1 for each unnecessary signal except Q)
3 points initialization of Q
4 points Counts up to 5 with CLK and stops
4 points RESTART (works at 5 2 points, works at 6,7 2 points)

4 points FAULT
2 points Is not self correcting

-2 points for each simple VHDL mistake
-3, -4, -5 points for each less simple and major VHDL mistakes

Another solution

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

entity MT2_Q2_V2 is
    Port ( CLK : in  STD_LOGIC;
           RESTART : in  STD_LOGIC;
           FAULT : out  STD_LOGIC);
end MT2_Q2_V2;

architecture Behavioral of  MT2_Q2_V2 is
signal Q:std_logic_vector(2 downto 0) :="000";
signal D:std_logic_vector(2 downto 0);
begin
--state memory
process(CLK)
begin
if CLK'event and CLK='1' then Q<=D;end if;
end process;
--next state logic
process(Q,RESTART)
begin
if (Q<"101") then D<=Q+1; else D<=Q; end if;
if (Q="101" or Q="110" or Q="111") and RESTART='1' then D<="000"; end if;
end process;
--output logic
FAULT<='1' when (Q="110" or Q="111") else '0';
end Behavioral;
```

Another solution

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

entity MT2_Q2_V4 is
    Port ( CLK : in  STD_LOGIC;
           RESTART : in  STD_LOGIC;
           FAULT : out  STD_LOGIC);
```

```
end MT2_Q2_V4;

architecture Behavioral of MT2_Q2_V4 is
signal Q:std_logic_vector(2 downto 0) :="000";
begin
process(CLK)
begin
if CLK'event and CLK='1' then
if (Q<"101") then Q<=Q+1; else Q<=Q; end if;
if (Q="101" or Q="110" or Q="111") and RESTART='1' then Q<="000"; end if;
end if;
end process;
FAULT<='1' when (Q="110" or Q="111") else '0';
end Behavioral;
```

Another solution (RESTART is asynchronous)

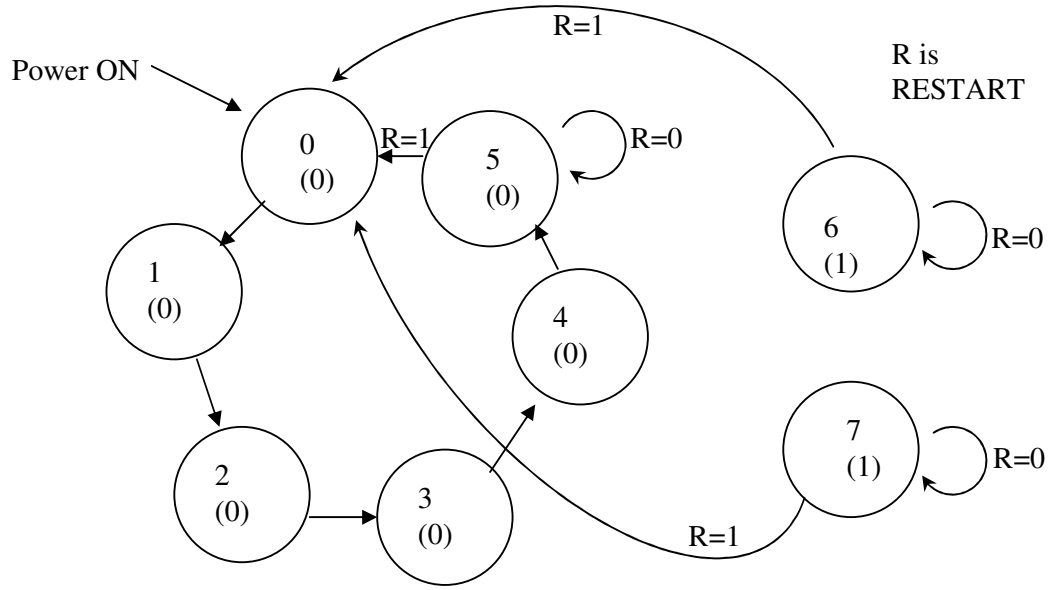
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity MT2_Q2_V3 is
    Port ( CLK : in  STD_LOGIC;
           RESTART : in  STD_LOGIC;
           FAULT : out  STD_LOGIC);
end MT2_Q2_V3;

architecture Behavioral of  MT2_Q2_V3 is
signal Q:std_logic_vector(2 downto 0) :="000";
signal D:std_logic_vector(2 downto 0);
begin
--state memory
process(CLK,RESTART)
begin
if CLK'event and CLK='1' then Q<=D;end if;
if (Q="101" or Q="110" or Q="111") and RESTART='1' then Q<="000"; end if;
end process;
--next state logic
process(Q,RESTART)
begin
if (Q<"101") then D<=Q+1; else D<=Q; end if;
if (Q="101" or Q="110" or Q="111") and RESTART='1' then D<="000"; end if;
end process;
--output logic
FAULT<='1' when (Q="110" or Q="111") else '0';
end Behavioral;
```

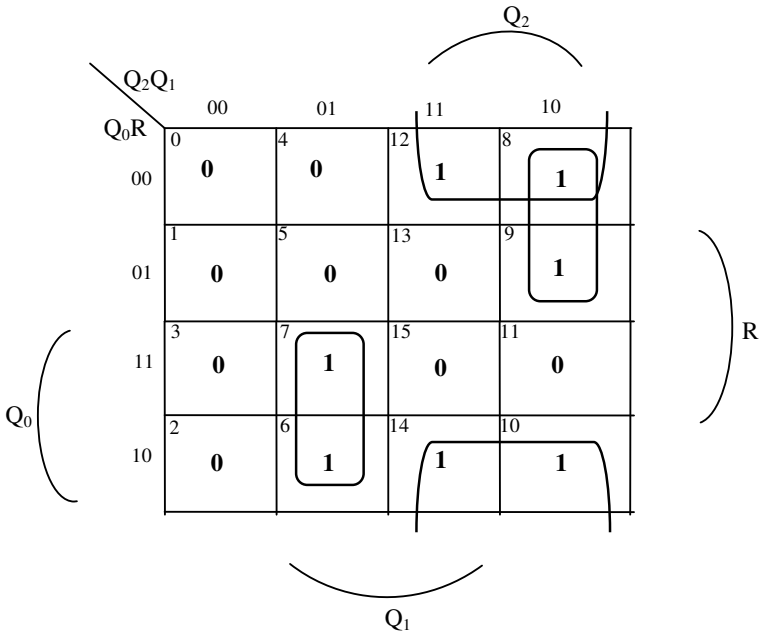
Q3. (Total of 20 points)
Design and draw the counter described in Question 1 using D flip flops with active low PR and CLR controls. Show all steps of the FSM design procedure except the timing diagrams (optional). Look at the appendix for FSM design procedure.

SOLUTION:
There are 6 states and therefore we need 3 ffs, say Q2, Q1, and Q0.
State-Output diagram:

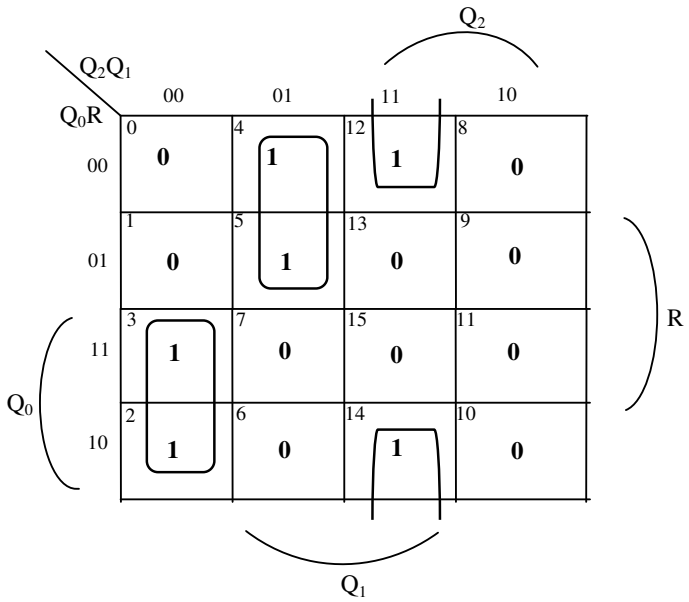


Next state table

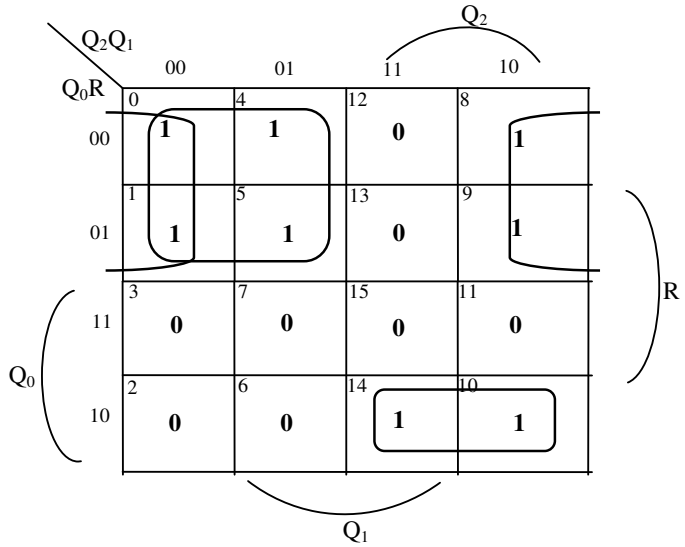
Q2	Q1	Q0	R	Q2*	Q1*	Q0*
0	0	0	0	0	0	1
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	0	1	1
0	1	0	1	0	1	1
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	1	0	1
1	0	0	1	1	0	1
1	0	1	0	1	0	1
1	0	1	1	0	0	0
1	1	0	0	1	1	0
1	1	0	1	0	0	0
1	1	1	0	1	1	1
1	1	1	1	0	0	0



$D_2 = Q_2R' + Q_2Q_1'Q_0' + Q_2'Q_1Q_0$



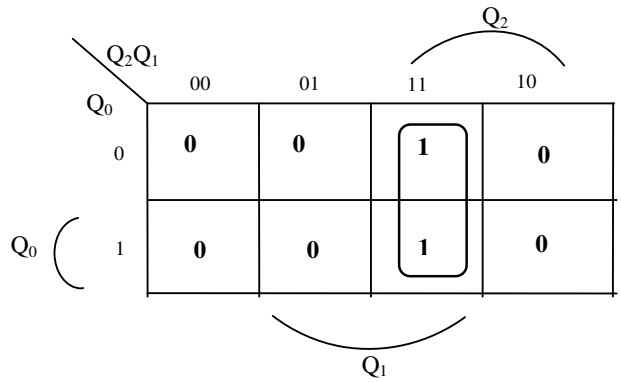
$D_1 = Q_2'Q_1'Q_0 + Q_2'Q_1Q_0' + Q_2Q_1R'$



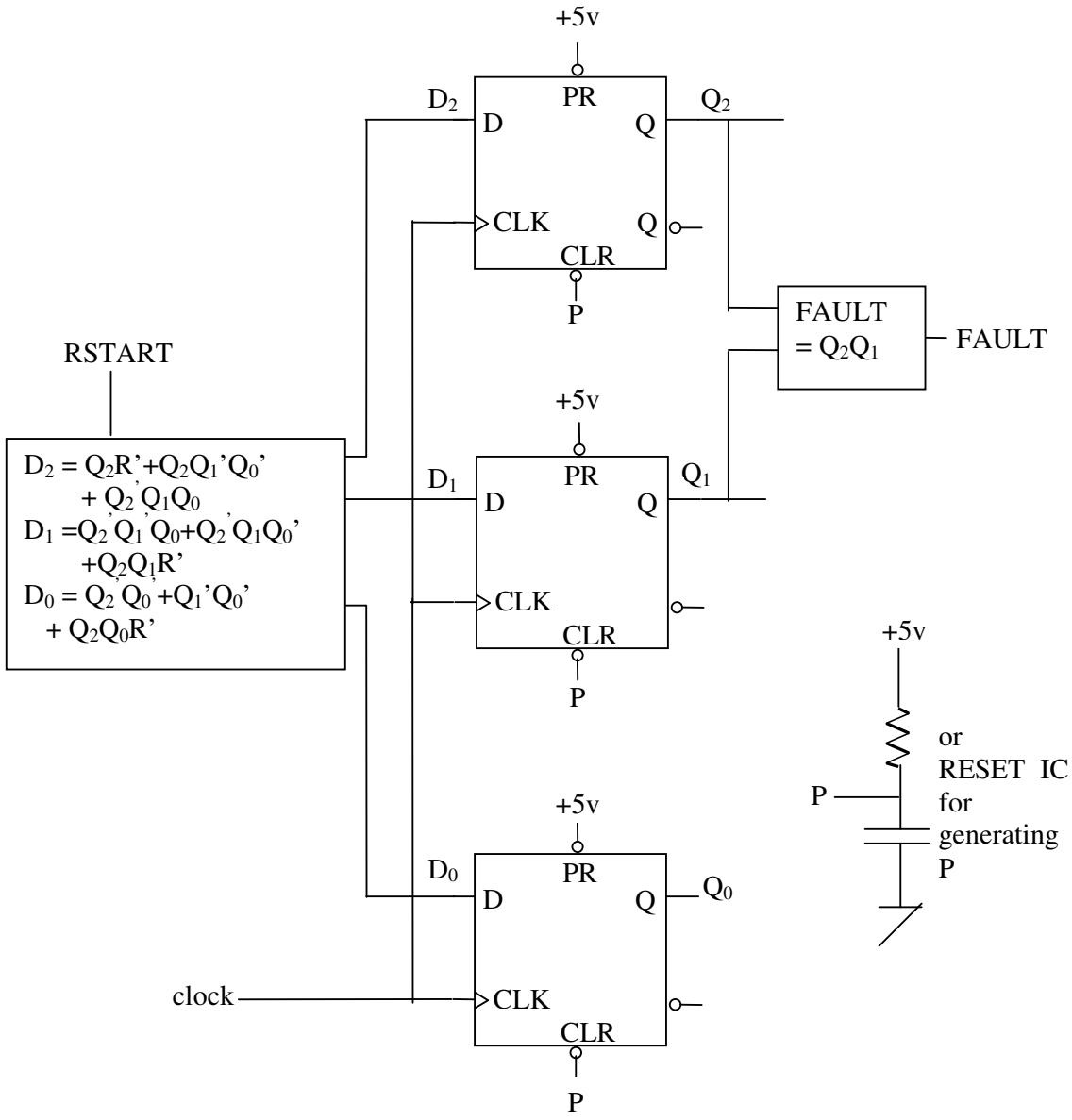
$$D_0 = Q_2'Q_0' + Q_1'Q_0' + Q_2Q_0R'$$

Output table

Q2	Q1	Q0	FAULT
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



$$FAULT = Q_2Q_1$$

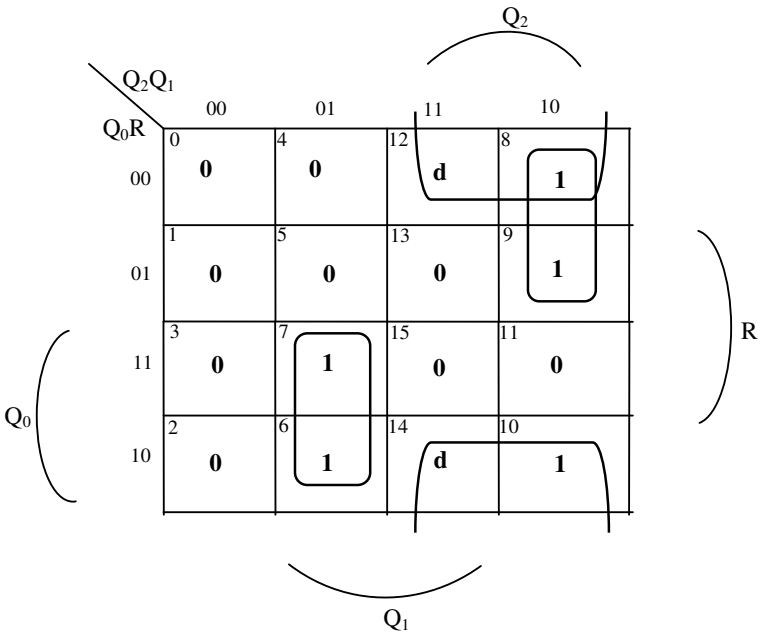


5 points State/output diagram and number of ffs (-1 for outputs, -1 for n of ffs, -1 for unused states, -1 for Power ON)
1 points State encoding
3 points Next state table and excitation table (-2 for don't cares)
3 points Minimized next state logic
2 points Output table
2 points Minimized output logic
4 points Circuit (2 points) with initialization (2 points)

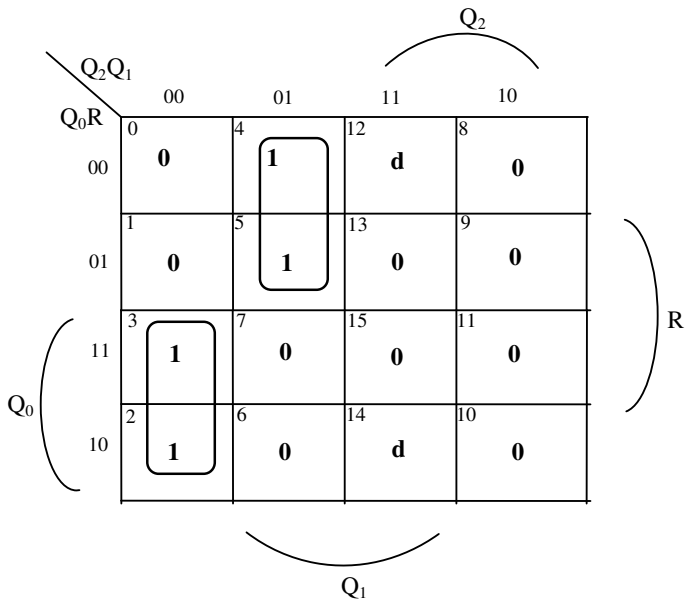
Let us see if the FSM becomes self correcting if for the unused states we don't care what the next state is for when RESTART = 0:

Next state table

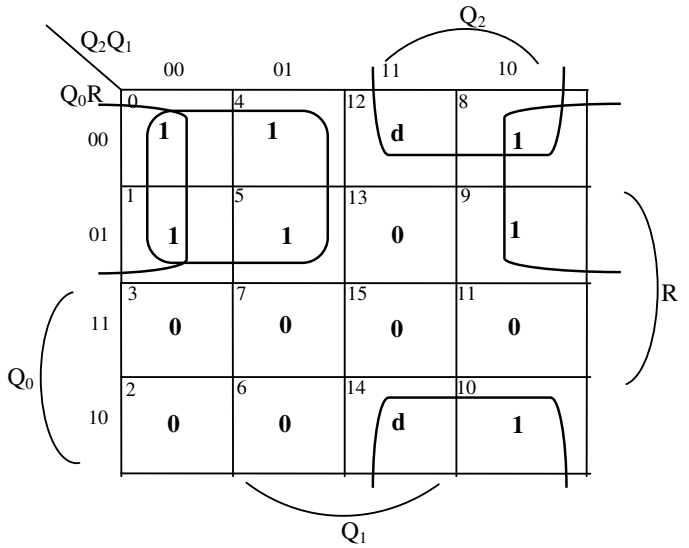
Q2	Q1	Q0	R	Q2*	Q1*	Q0*
0	0	0	0	0	0	1
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	0	1	1
0	1	0	1	0	1	1
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	1	0	1
1	0	0	1	1	0	1
1	0	1	0	1	0	1
1	0	1	1	0	0	0
1	1	0	0	d	d	d
1	1	0	1	0	0	0
1	1	1	0	d	d	d
1	1	1	1	0	0	0



$$D_2 = Q_2R' + Q_2Q_1'Q_0' + Q_2'Q_1Q_0$$



$$D_1 = Q_2'Q_1'Q_0 + Q_2'Q_1Q_0'$$



$$D_0 = Q_2'Q_0' + Q_2R' + Q_1'Q_0'$$

Next state table

Q2	Q1	Q0	R	Q2*	Q1*	Q0*
0	0	0	0	0	0	1
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	0	1	1
0	1	0	1	0	1	1
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	1	0	1
1	0	0	1	1	0	1
1	0	1	0	1	0	1
1	0	1	1	0	0	0
1	1	0	0	1	0	1
1	1	0	1	0	0	0
1	1	1	0	1	0	1
1	1	1	1	0	0	0

The FSM is now self correcting which is not desired.

The above designs are Moore designs. However from the word description one may also interpret that the RESTART signal is effective immediately without wating for a clock tick. This means a Mealy FSM. Such a solution is also acceptable.

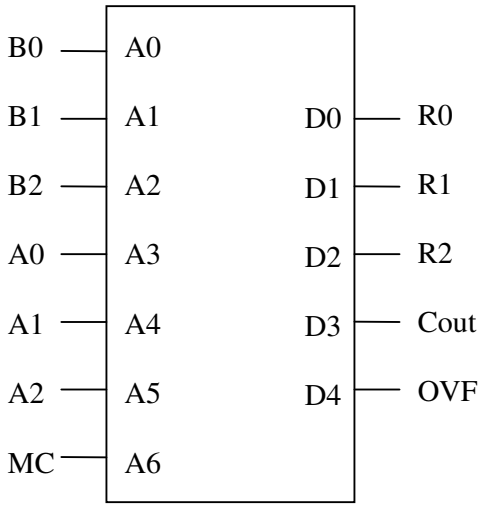
Q4. (Total of 20 points)

Consider the implementation of a 3-bit two's complement adder-subtractor with mode control (mode control = 0 means A+B, mode control = 1 means A-B), carry output, and two's complement overflow output, using a single Nx5 ROM.

a. **(3 points) Determine N.**
b. **(5 points) Label the IO pins of the ROM appropriately and make your signal connections (Draw your circuit).**
c. **(6 points) With reference to your IO labeling, list all rows of your ROM table that corresponds to the case when the first operand A is "101". In your list, indicate the data value and the ROM address for each of your entries.**
d. **(6 points) With reference to your IO labeling, list all rows of your ROM table that corresponds to the case when the second operand B is "011". In your list, indicate the data value and the ROM address for each of your entries.**

SOLUTION:

a. **N = 2^7=128 (Two 3-bit numbers, and one mode control input)**
b.



-2 for not labeling the internal signals

c.

		A6	A5	A4	A3	A2	A1	A0	D4	D3	D2	D1	D0		
A	**B**	**MC**	**A2**	**A1**	**A0**	**B2**	**B1**	**B0**	**OVF**	**CO**	**R2**	**R1**	**R0**	**R**	**RC**
-3	0	0	1	0	1	0	0	0	0	0	1	0	1	-3	-3
-3	1	0	1	0	1	0	0	1	0	0	1	1	0	-2	-2
-3	2	0	1	0	1	0	1	0	0	0	1	1	1	-1	-1
-3	3	0	1	0	1	0	1	1	0	1	0	0	0	0	0
-3	-4	0	1	0	1	1	0	0	1	1	0	0	1	1	-7
-3	-3	0	1	0	1	1	0	1	1	1	0	1	0	2	-6
-3	-2	0	1	0	1	1	1	0	1	1	0	1	1	3	-5
-3	-1	0	1	0	1	1	1	1	0	1	1	0	0	-4	-4
-3	0	1	1	0	1	0	0	0	0	1	1	0	1	-3	-3
-3	1	1	1	0	1	0	0	1	0	1	1	0	0	-4	-4
-3	2	1	1	0	1	0	1	0	1	1	0	1	1	3	-5
-3	3	1	1	0	1	0	1	1	1	1	0	1	0	2	-6
-3	-4	1	1	0	1	1	0	0	0	1	0	0	1	1	1
-3	-3	1	1	0	1	1	0	1	0	1	0	0	0	0	0
-3	-2	1	1	0	1	1	1	0	0	0	1	1	1	-1	-1
-3	-1	1	1	0	1	1	1	1	0	0	1	1	0	-2	-2

Addresses are 40, 41, ... , 47, 104, 105. ... , 111

d.

		A6	A5	A4	A3	A2	A1	A0	D4	D3	D2	D1	D0		
A	**B**	**MC**	**A2**	**A1**	**A0**	**B2**	**B1**	**B0**	**OVF**	**CO**	**R2**	**R1**	**R0**	**R**	**RC**
0	3	0	0	0	0	0	1	1	0	0	0	1	1	3	3
1	3	0	0	0	1	0	1	1	1	0	1	0	0	-4	4
2	3	0	0	1	0	0	1	1	1	0	1	0	1	-3	5
3	3	0	0	1	1	0	1	1	1	0	1	1	0	-2	6
-4	3	0	1	0	0	0	1	1	0	0	1	1	1	-1	-1
-3	3	0	1	0	1	0	1	1	0	1	0	0	0	0	0
-2	3	0	1	1	0	0	1	1	0	1	0	0	1	1	1
-1	3	0	1	1	1	0	1	1	0	1	0	1	0	2	2
0	3	1	0	0	0	0	1	1	0	0	1	0	1	-3	-3
1	3	1	0	0	1	0	1	1	0	0	1	1	0	-2	-2
2	3	1	0	1	0	0	1	1	0	0	1	1	1	-1	-1
3	3	1	0	1	1	0	1	1	0	1	0	0	0	0	0
-4	3	1	1	0	0	0	1	1	1	1	0	0	1	1	-7
-3	3	1	1	0	1	0	1	1	1	1	0	1	0	2	-6
-2	3	1	1	1	0	0	1	1	1	1	0	1	1	3	-5
-1	3	1	1	1	1	0	1	1	0	1	1	0	0	-4	-4

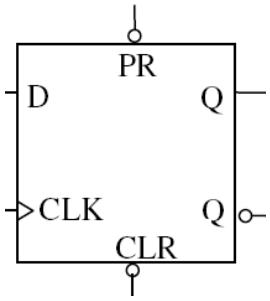
Addresses are 3, 11, 19, ... , 51, 59, 67, 75, ..., 123

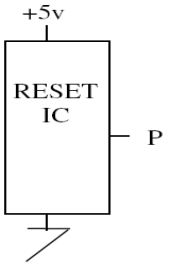
+1 point for knowing two's complement binary addition

+1 point for knowing two's complement binary subtraction

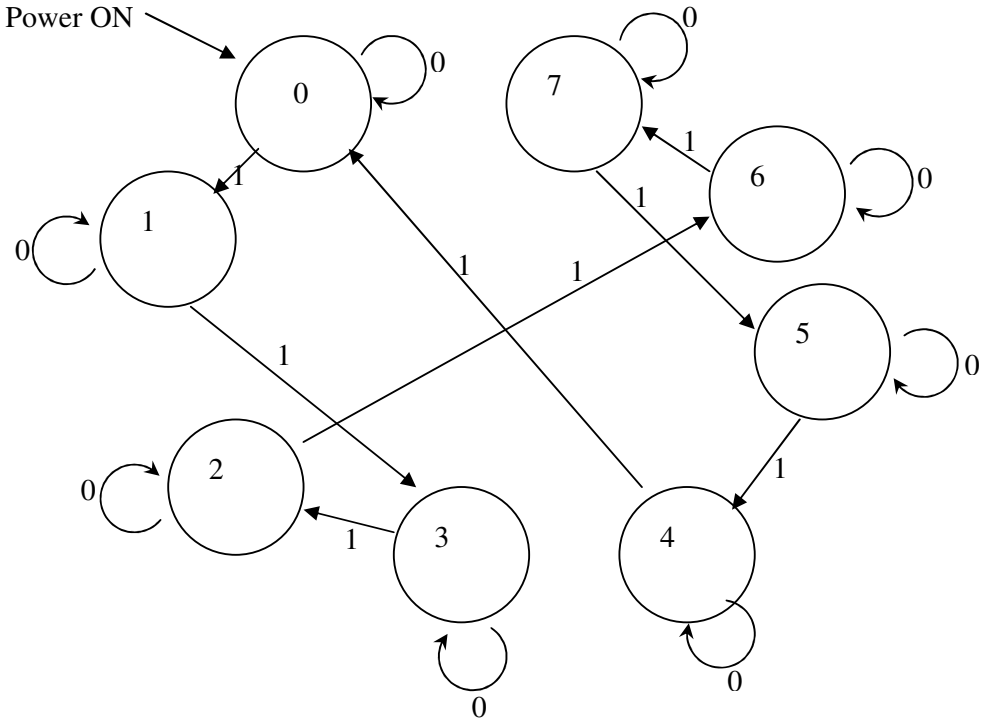
Q5. (20 points)

Using combinational logic and D flip-flops as in the first figure below, design a counter that has an active-high enable control, and counts in the following order when enable is asserted: 000, 001, 011, 010, 110, 111, 101, 100, 000, ... If enable is unasserted then the counter stops counting and stays in the present count. Make sure that Power-ON state is 000. Assume the availability of a reset IC as in the second figure below.



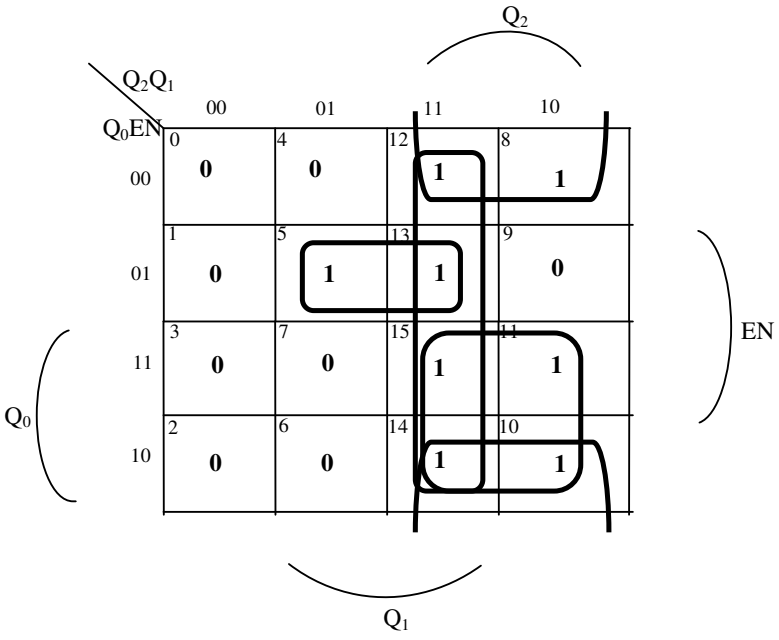


SOLUTION:

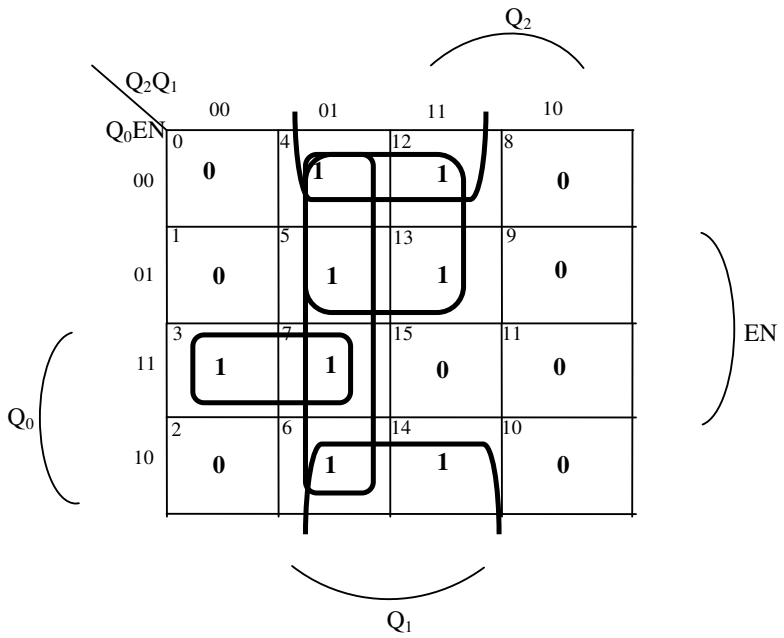


Next state table

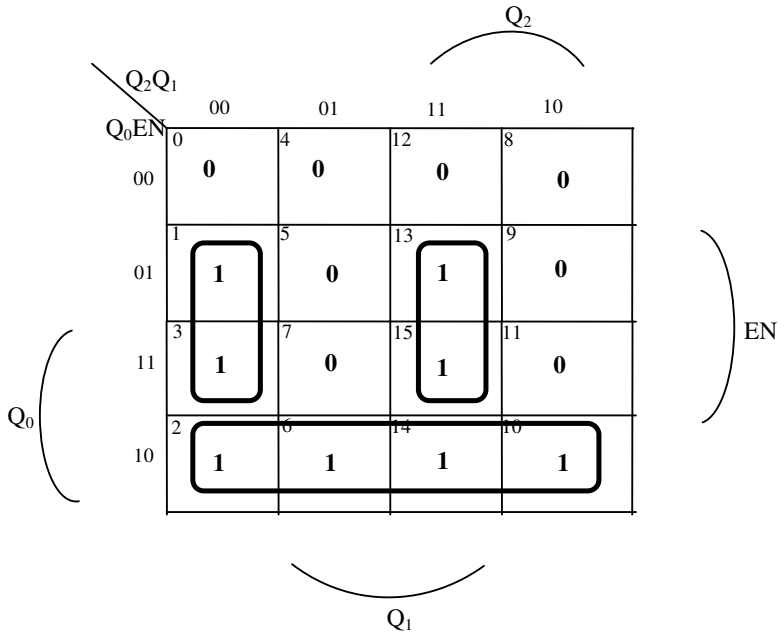
Q2	Q1	Q0	EN	Q2*	Q1*	Q0*
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	1	1
0	1	0	0	0	1	0
0	1	0	1	1	1	0
0	1	1	0	0	1	1
0	1	1	1	0	1	0
1	0	0	0	1	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	1
1	0	1	1	1	0	0
1	1	0	0	1	1	0
1	1	0	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	1	0	1



$$D_2 = Q_2EN' + Q_2Q_0 + Q_1Q_0'EN$$



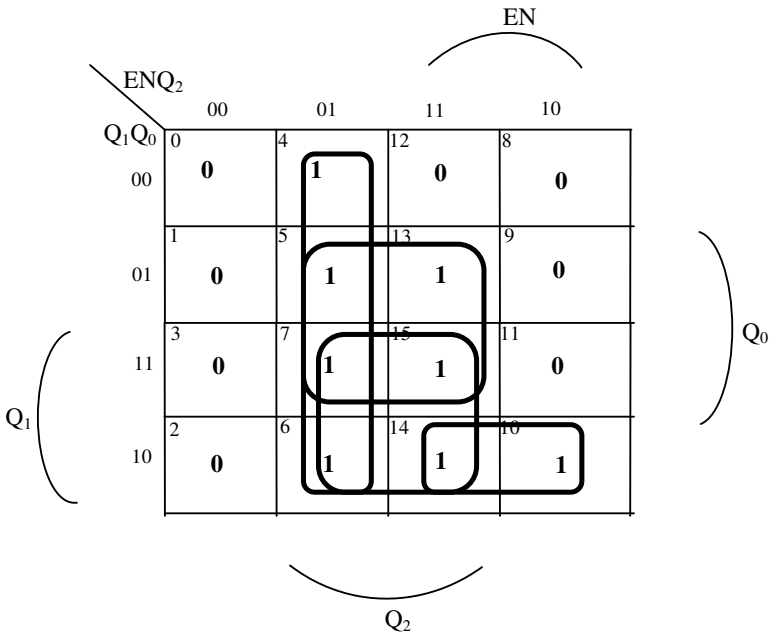
$D_1 = Q_2'Q_0EN + Q_1Q_0' + Q_1EN'$



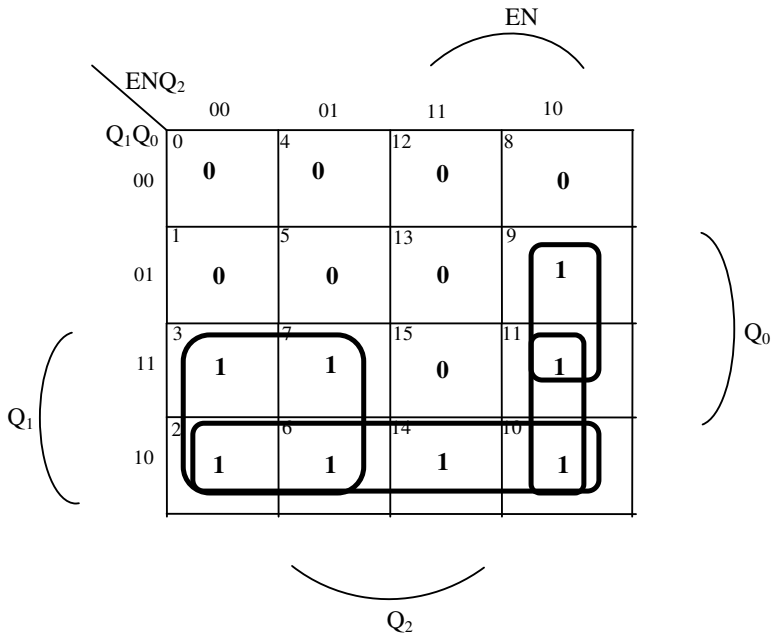
$D_0 = Q_2'Q_1'EN + Q_2Q_1EN + Q_0EN'$

Next state table

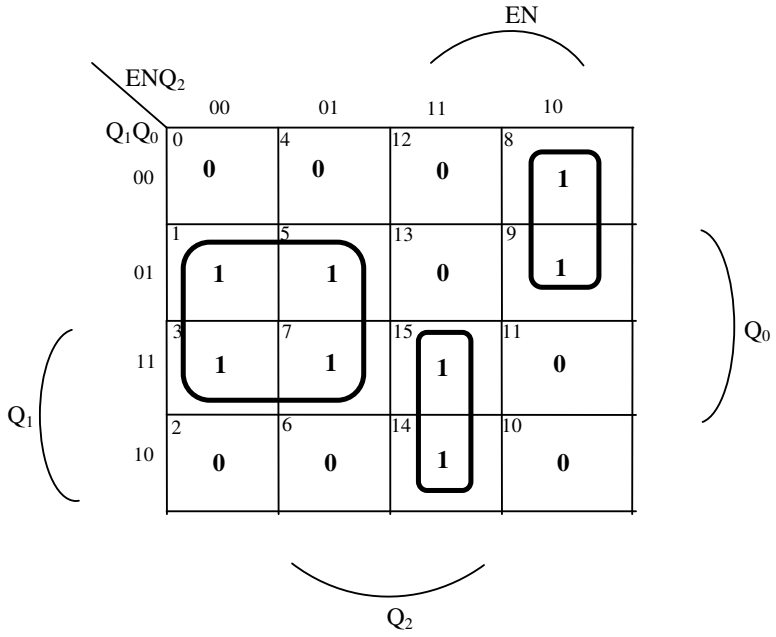
EN	Q2	Q1	Q0	Q2*	Q1*	Q0*
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	1	0	0
0	1	0	1	1	0	1
0	1	1	0	1	1	0
0	1	1	1	1	1	1
1	0	0	0	0	0	1
1	0	0	1	0	1	1
1	0	1	0	1	1	0
1	0	1	1	0	1	0
1	1	0	0	0	0	0
1	1	0	1	1	0	0
1	1	1	0	1	1	1
1	1	1	1	1	0	1



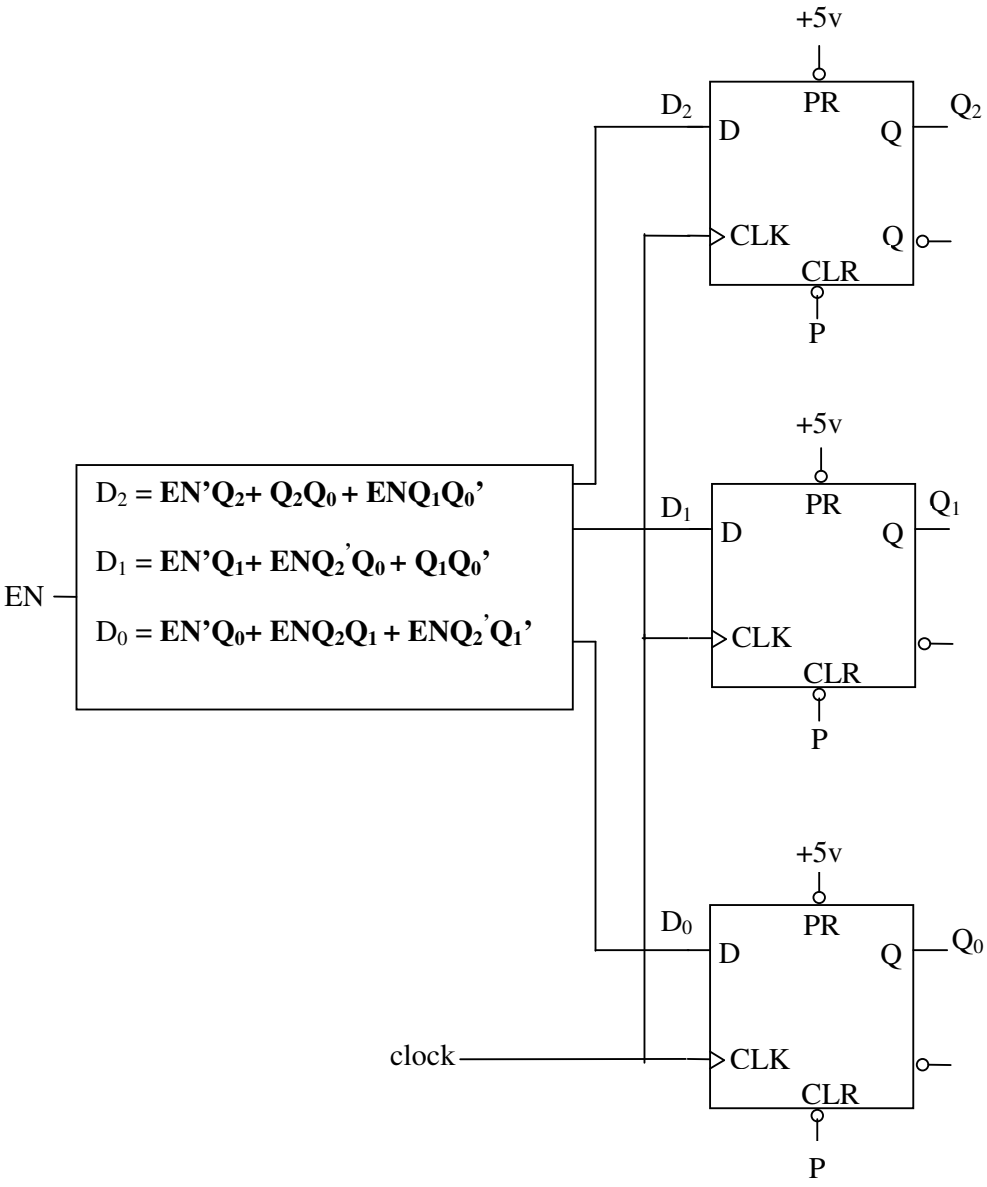
$$D_2 = EN'Q_2 + Q_2Q_0 + ENQ_1Q_0'$$



$D_1 = EN'Q_1 + ENQ_2'Q_0 + Q_1Q_0'$



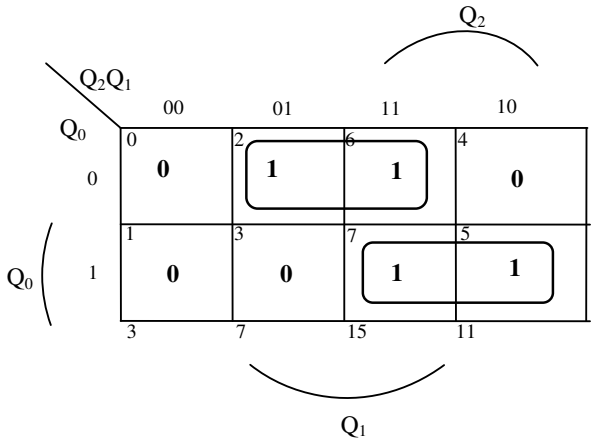
$D_0 = EN'Q_0 + ENQ_2Q_1 + ENQ_2'Q_1'$



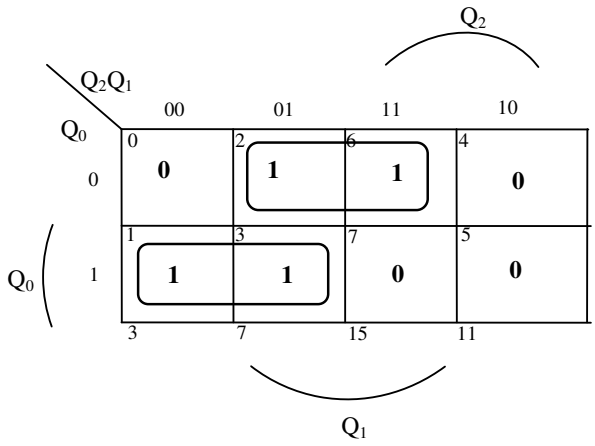
4 points 3 ffs +clock
4 points Initiation using P
12 points Next state logic (4+4+4) (if only for EN=1 then -6 points)

For EN = 1 the next state table is

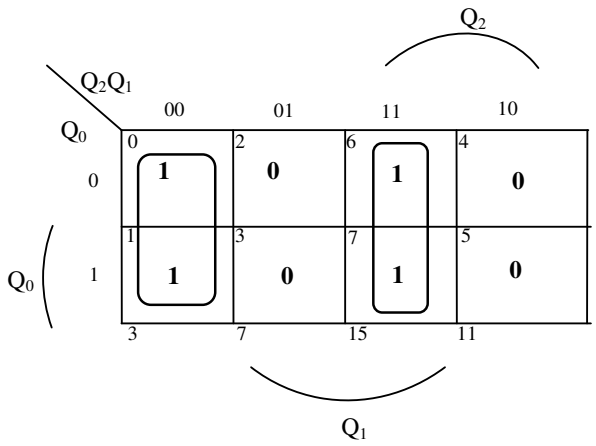
Q2	Q1	Q0	Q2*	Q1*	Q0*
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	1	1	0
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	1	1	1
1	1	1	1	0	1



$D_2 = Q_2Q_0 + Q_1Q_0'$



$D_1 = Q_2'Q_0 + Q_1Q_0'$



$$D_0 = Q_2Q_1 + Q_2'Q_1'$$

Some VHDL Templates:

ENTITY DECLARATION
entity ***entity_name*** **is**
generic (***constant_names : constant type;***
constant_names : constant type;
…
constant_names : constant type**);**
port (***signal_names : mode signal_type;***
signal_names : mode signal_type;
…
signal_names : mode signal_type**);**
end ***entity_name*****;**
ARCHITECTURE DEFINITIONS
architecture ***architecture-name*** **of** ***entity-name*** **is**
type declarations
signal declarations
constant declarations
function definitions
procedure definitions
component declarations
begin
concurrent statement
...
concurrent statement
end ***architecture-name*****;**
COMPONENT DECLARATION
component ***component_name***
port (***signal_names : mode signal type;***
signal_names : mode signal type;
…
signal_names : mode signal type**);**
end component;
COMPONENT INSTANTIATION
label: component_name **port map (*****signal1, signal2, …,signaln*****);**
or,
label: component_name **port map (*****port1 =>signal1, port2 =>signal2, …, portn =>signaln*****);**

DATAFLOW TYPE STATEMENTS:

Simple concurrent assignment statement
signal_name <= expression;
Conditional concurrent assignment statement
signal_name **<=**
expression **when** ***boolean-expression*** **else**
expression **when** ***boolean-expression*** **else**
…
expression **when** ***boolean-expression*** **else**
expression**;**
with-select statement
with ***expression*** **select**
signal_name **<=** ***signal_value*** **when** ***choices*****,**
signal_value **when** ***choices*****,**
…
signal_value **when** ***choices*****;**

Note that conditional concurrent assignment statement and with-select statement cannot be used in a process statement. Instead, in a process, one can use the the sequential conditional assignment statements if and case.

BEVAVIORAL TYPE STATEMENTS:

process statement
process(*signal_name***,** ***signal_name, …, signal_name*)
type_declarations
variable declarations
constant declarations
begin
sequential-statement
…
sequential-statement
end process;
Simple sequential assignment statement
signal_name <= expression;
if statement in its general form
if *boolean_ expression* **then** *sequential_statements*
elsif *boolean_ expression* **then** *sequential_statements*
…
elsif *boolean_ expression* **then** *sequential_statements*
else *sequential_statements*
end if;
Note that you may not use the else and/or the elsif.
case-when statement
case *expression* **is**
when *choices* **=>** *sequential_statements*
…
when *choices* **=>** *sequential_statements*
end case;
loop statement
loop
sequential_statement
…
sequential_statement
end loop;
for-loop statement
for *identifier* **in** *range* **loop**
sequential_statement
…
sequential_statement
end loop;
while statement
while *boolean_expression* **loop**
sequential_statement
…
sequential_statement
end loop;

Note that the if, case, loop, for, and while statements are called sequential statements and they can only be used in a process statement. Also note that each process is one concurrent statement.
Concatenation operator & is used as follows: If A and B are 2 bit numbers then A&B is a four bit number with A being more significant.

STAGES OF FINITE STATE MACHINE DESIGN

1. **Word description**
2. **Sample waveforms. This stage is optional but makes you understand the problem better.**
3. **State definitions and state/output diagram. At this stage the required number of flip flops is also determined.**
4. **State encoding (state assignment)**
5. **Next state table (this table is also called transition table)**
6. **Decide on the type of flip flop**
7. **Excitation table**
8. **Obtain minimized functions for next state logic circuit**
9. **Output table**
10. **Obtain minimized functions for output logic circuit**
11. **Draw the circuit including initialization**