Figure 3-1 Basic ROM Structure

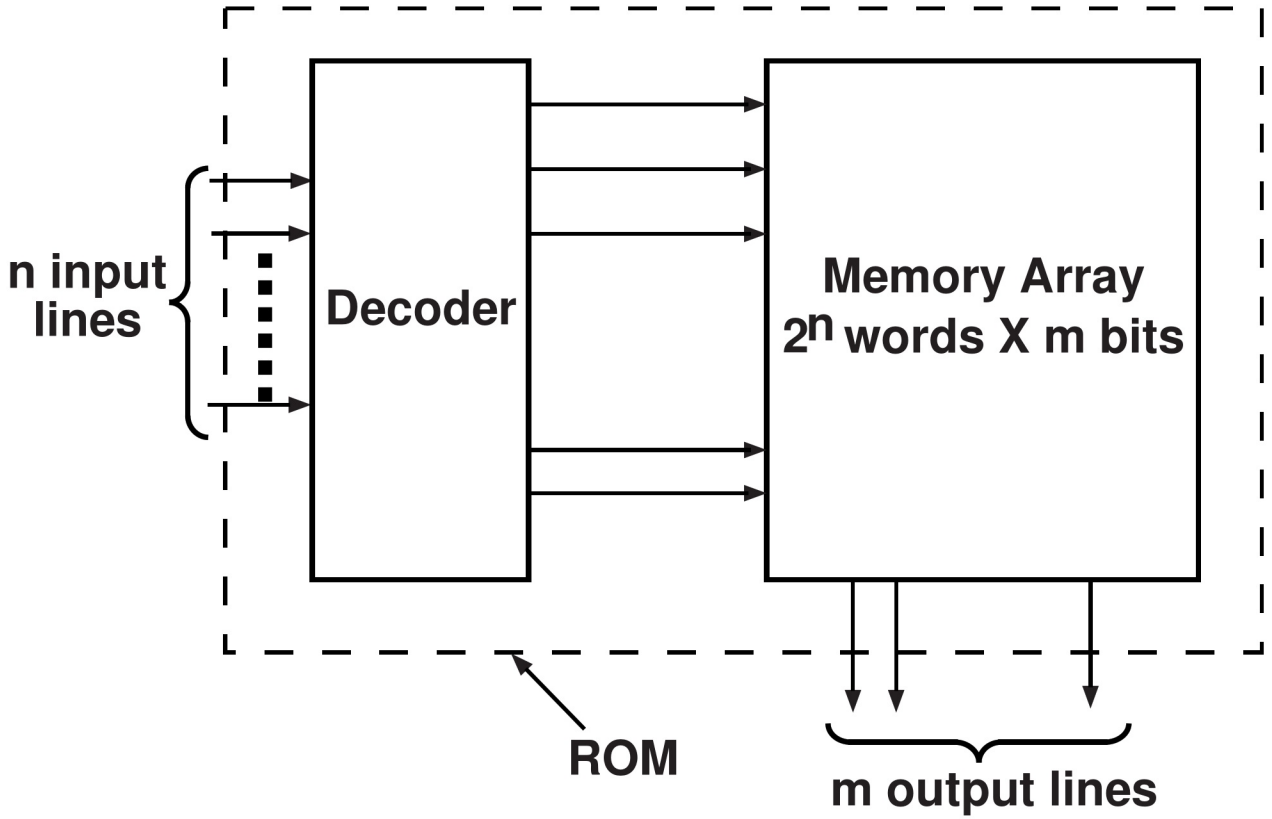


Figure 3-2
Realization of a Mealy Sequential Network with a ROM

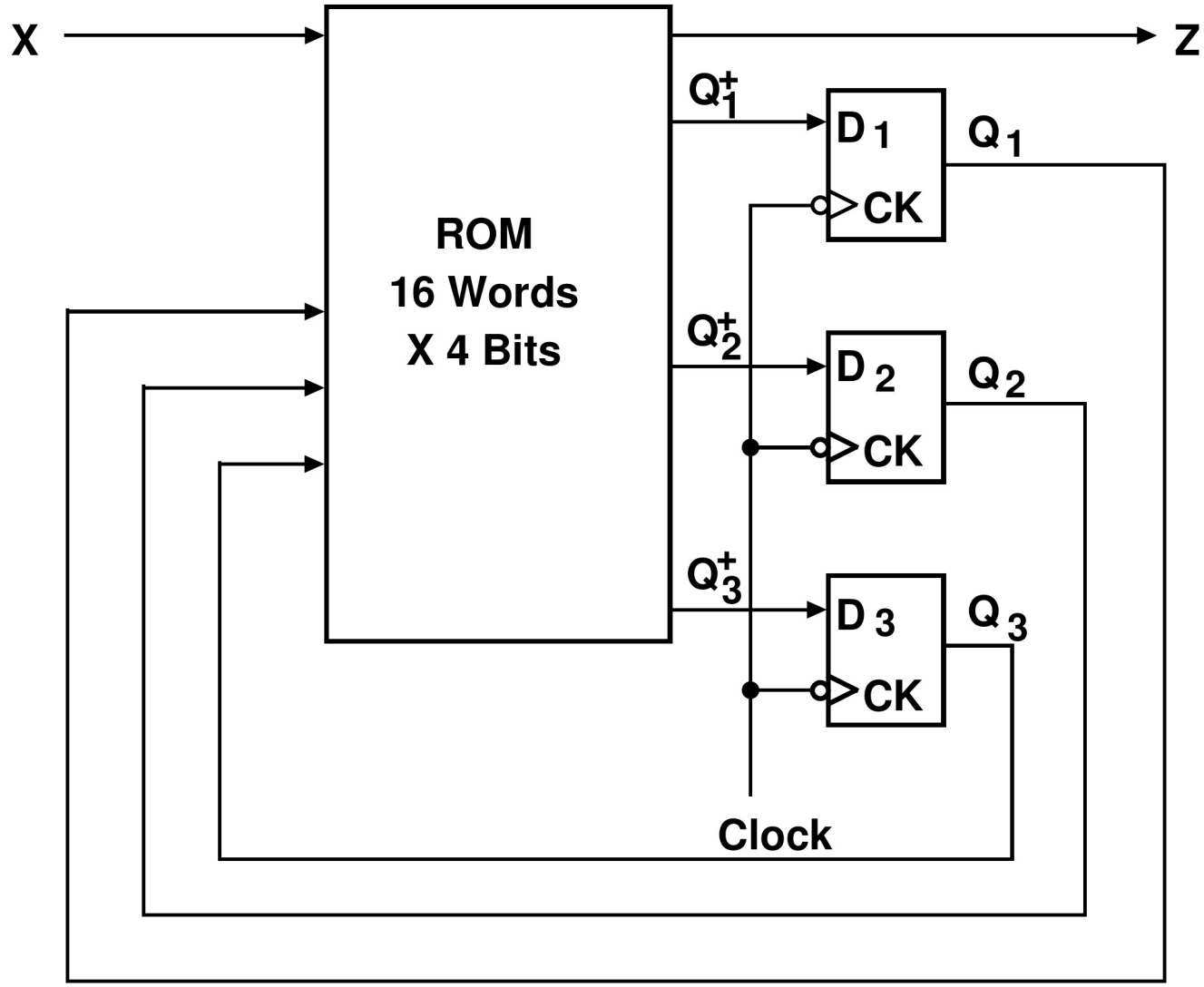


Table 3-1 ROM Truth Table

Q1	Q2	Q3	X	$Q1^+$	$Q2^+$	$Q3^+$	Z
0	0	0	0	1	0	0	1
0	0	0	1	1	0	1	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	1
0	1	0	1	0	0	0	0
0	1	1	0	0	0	0	0
0	1	1	1	0	0	0	1
1	0	0	0	1	1	1	1
1	0	0	1	1	1	0	0
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	1	0	0	0	1	1	1
1	1	0	1	0	1	0	0
1	1	1	0	0	1	1	0
1	1	1	1	0	1	1	1

	$Q1^+$ $Q2^+$ $Q3^+$		**Z**	
Q1 Q2 Q3	**X=0**	**1**	**X=0**	**1**
000	**100**	**101**	**1**	**0**
100	**111**	**110**	**1**	**0**
101	**110**	**110**	**0**	**1**
111	**011**	**011**	**0**	**1**
110	**011**	**010**	**1**	**0**
011	**000**	**000**	**0**	**1**
010	**000**	**xxx**	**1**	**x**
001	**xxx**	**xxx**	**x**	**x**

Figure 3-3 ROM Realization of Figure 1-17

```
library BITLIB;
use BITLIB.bit_pack.all;

entity ROM1_2 is
      port(X,CLK: in bit;
            Z: out bit);
end ROM1_2;

architecture ROM1 of ROM1_2 is
signal Q, Qplus: bit_vector(1 to 3) := "000";
type ROM is array (0 to 15) of bit_vector(3 downto 0);
constant FSM_ROM:  ROM :=
      ("1001","1010","0000","0000","0001","0000","0000","0001",
      "1111","1100","1100","1101","0111","0100","0110","0111");
begin
      process(Q,X)                                    -- determines the next state and output
      variable ROMValue: bit_vector(3 downto 0);
      begin
            ROMValue := FSM_ROM(vec2int(Q & X));      -- read ROM output
            Qplus <= ROMValue(3 downto 1);
            Z <= ROMValue(0);
      end process;

      process(CLK)
      begin
            if CLK='1' then Q <= Qplus; end if;       -- update state register
      end process;
end ROM1;
```

Figure 3-4 Programmable Logic Array Structure

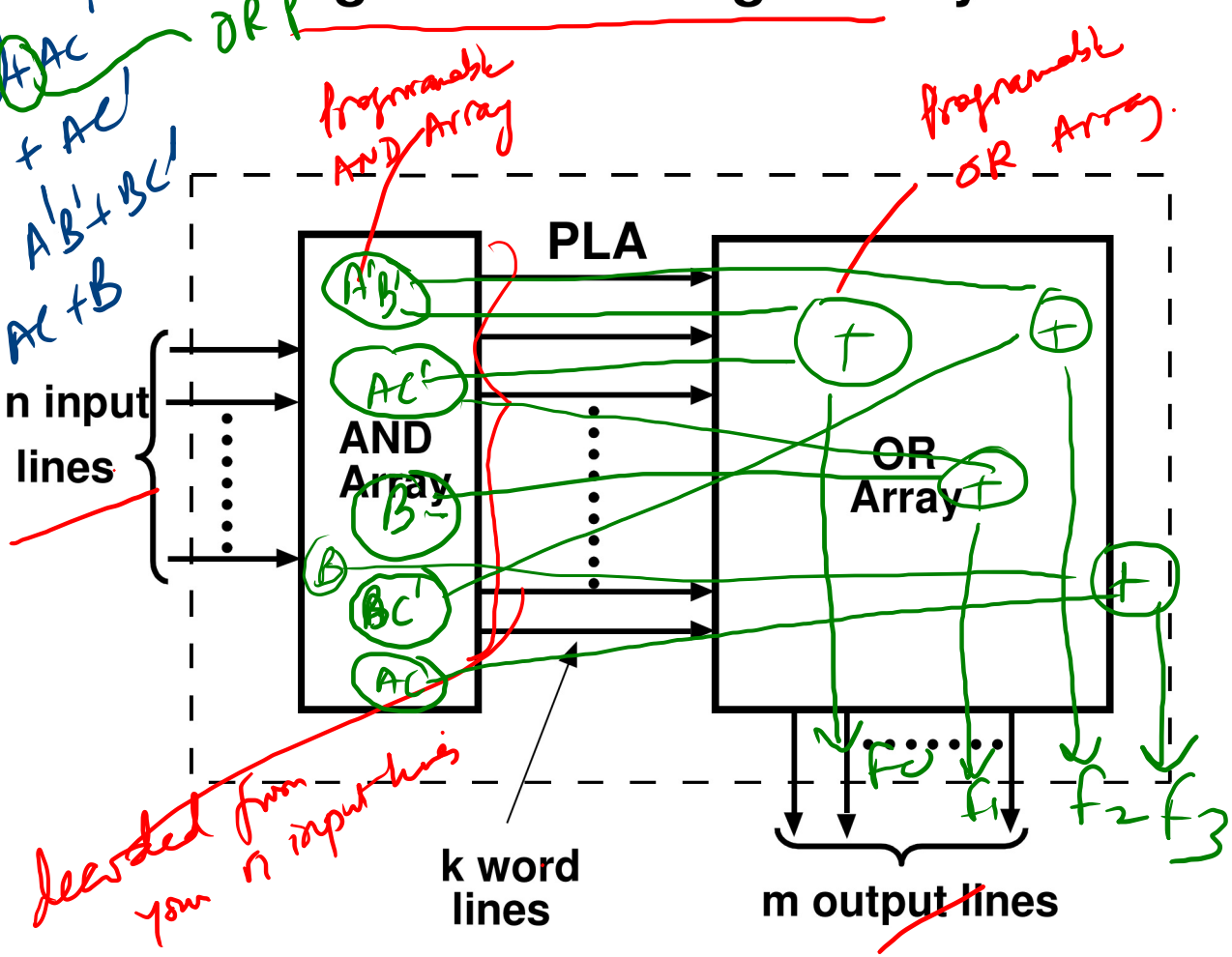


Figure 3-5 PLA with 3 input, 5 Product Terms, and 4 Outputs

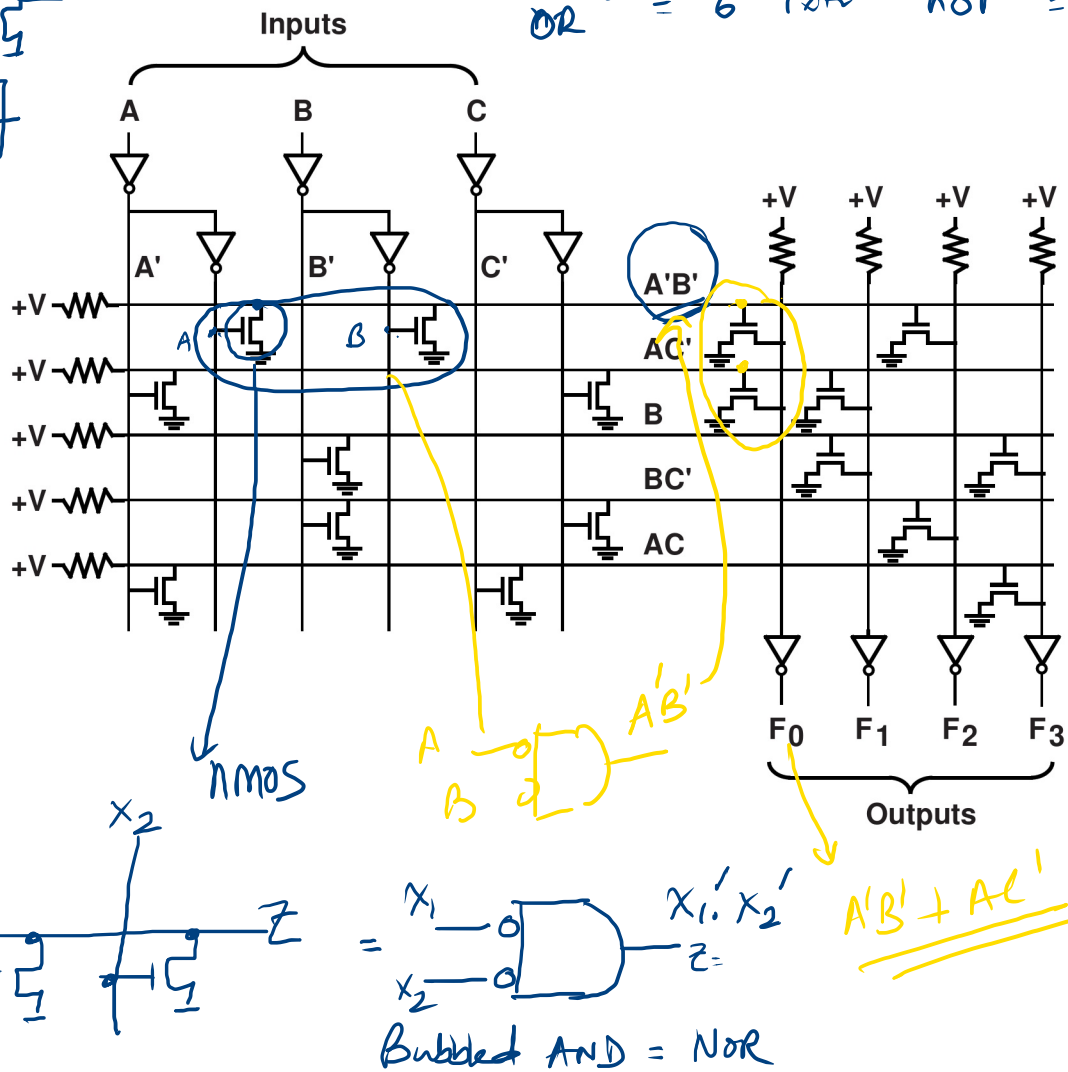


Figure 3-6 nMOS NOR Gate

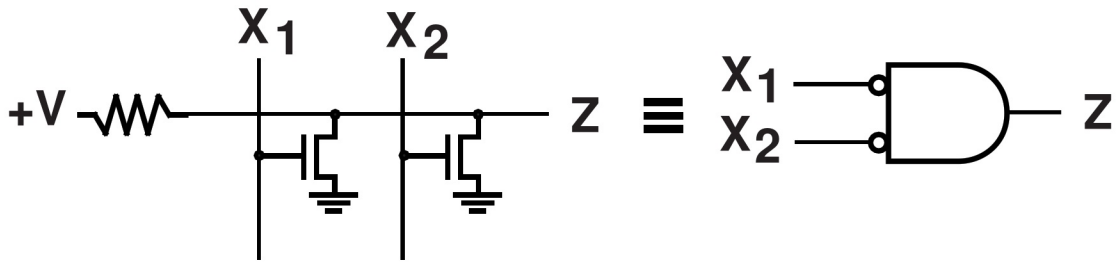


Figure 3-7 Conversion for NOR-NOR to AND-OR

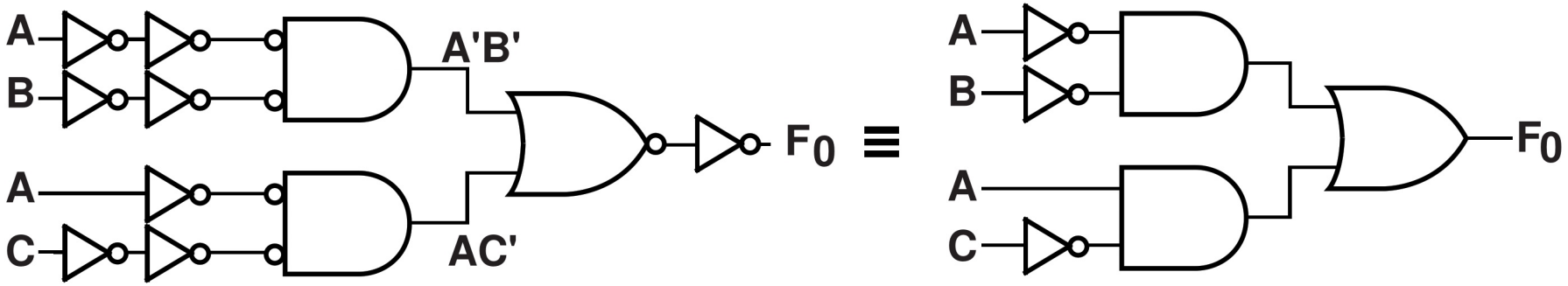


Figure 3-8 AND-OR Array Equivalent to Figure 3-5

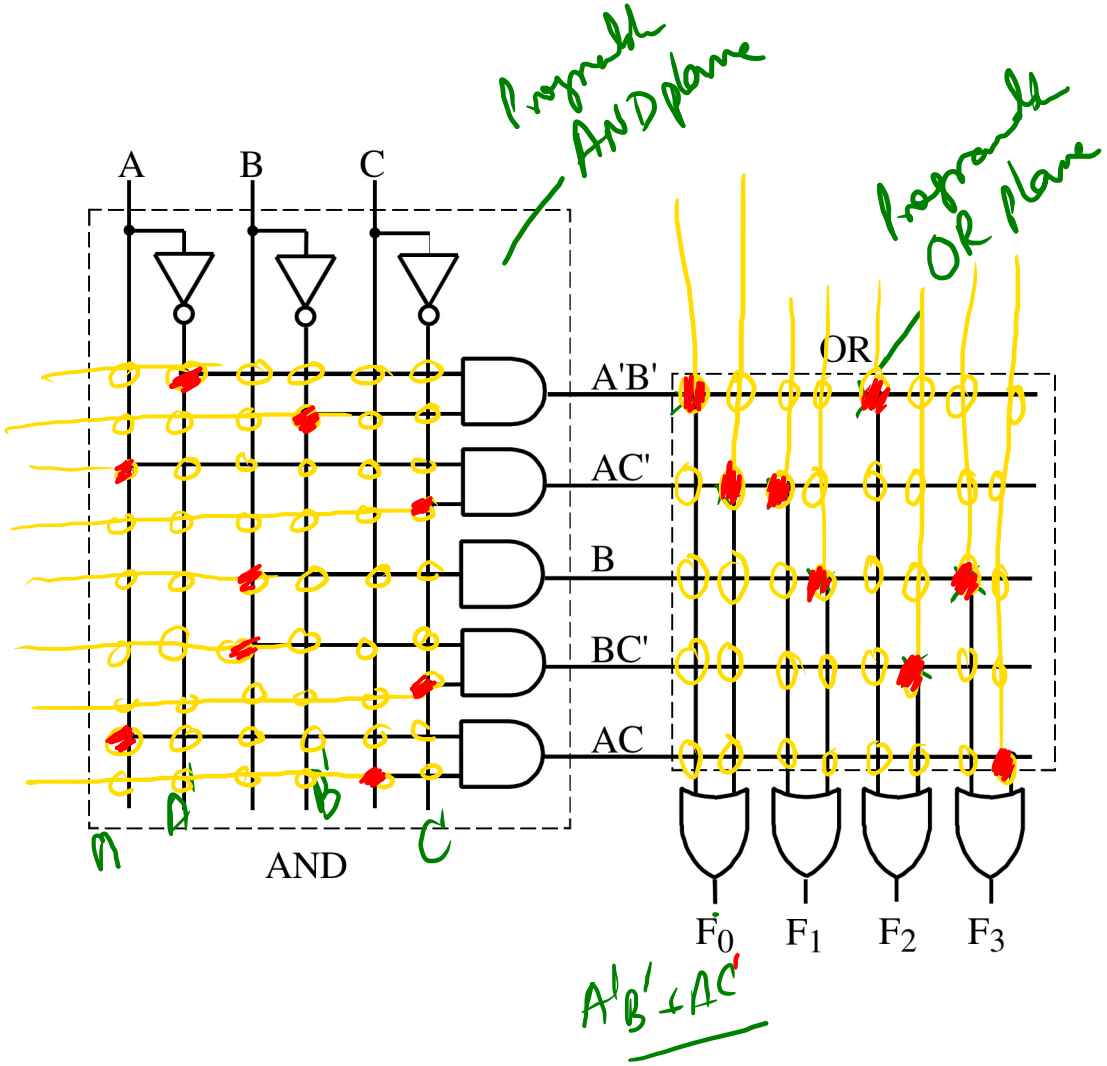
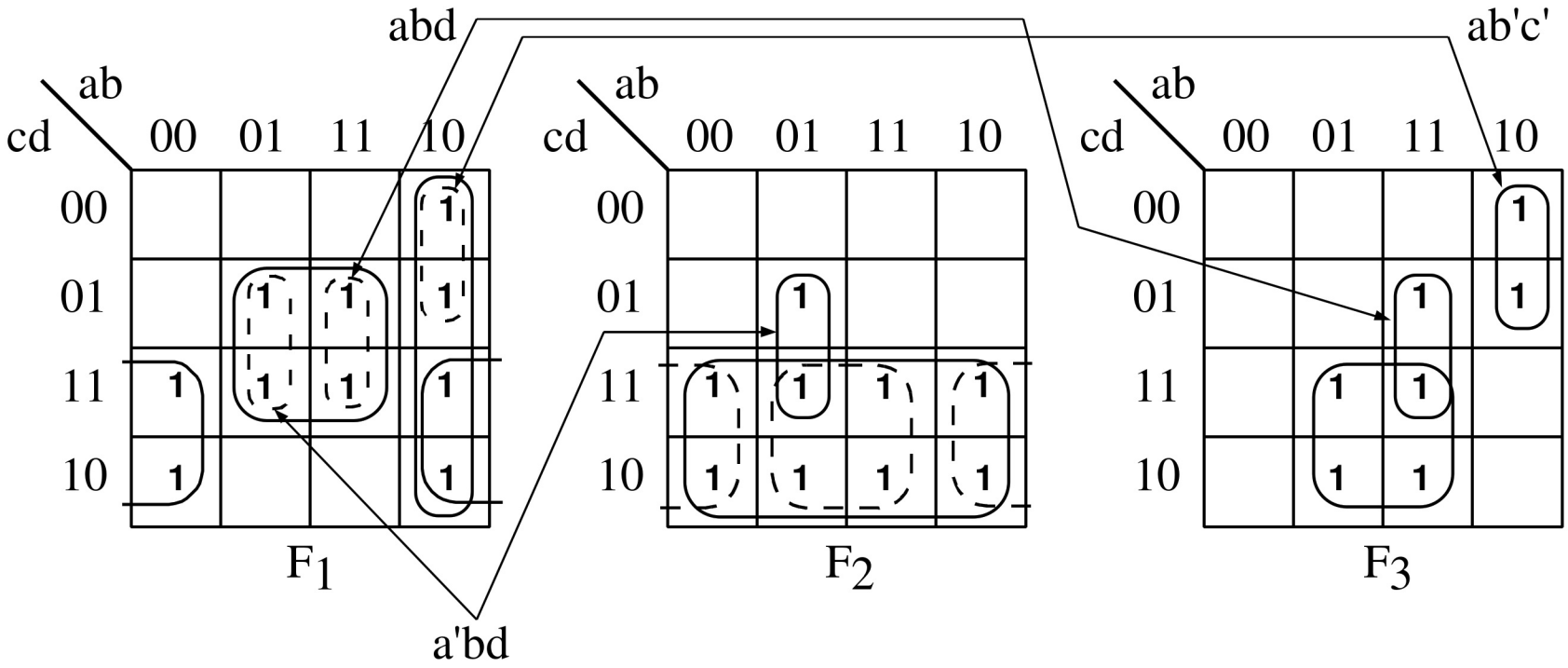


Table 3-2 PLA Table for Figure 3-5

	Inputs			Outputs			
Product	A	B	C	F0	F1	F2	F3
A'B'	0	0	–	1	0	1	0
AC'	1	–	0	1	1	0	0
B	–	1	–	0	1	0	1
BC'	–	1	0	0	0	1	0
AC	1	–	1	0	0	0	1

Figure 3-9 Multiple Output Karnaugh Maps



a	b	c	d	F1	F2	F3
0	1	–	1	1	1	0
1	1	–	1	1	0	1
1	0	0	–	1	0	1
–	0	1	–	1	1	0
–	1	1	–	0	1	1

F1 = a'bd + abd + ab'c' + b'c

F2 = a'bd + b'c + bc

F3 = abd + ab'c' + bc

Table 3-3 Reduced PLA Table

a	b	c	d	F_1	F_2	F_3
0	1	–	1	1	1	0
1	1	–	1	1	0	1
1	0	0	–	1	0	1
–	0	1	–	1	1	0
–	1	1	–	0	1	1

$F_1 = a'bd + abd + ab'c' + b'c$

$F_2 = a'bd + b'c + bc$

$F_3 = abd + ab'c' + bc$

Figure 3-10 PLA Realization of Equations

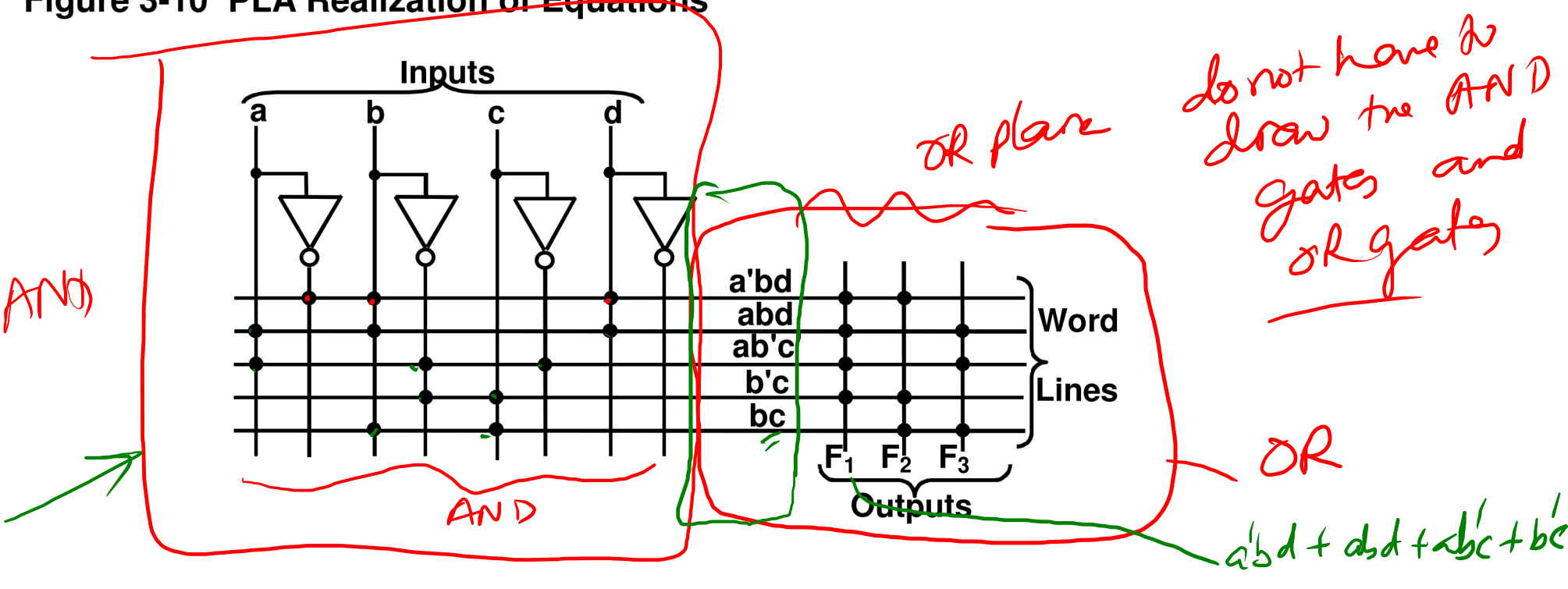
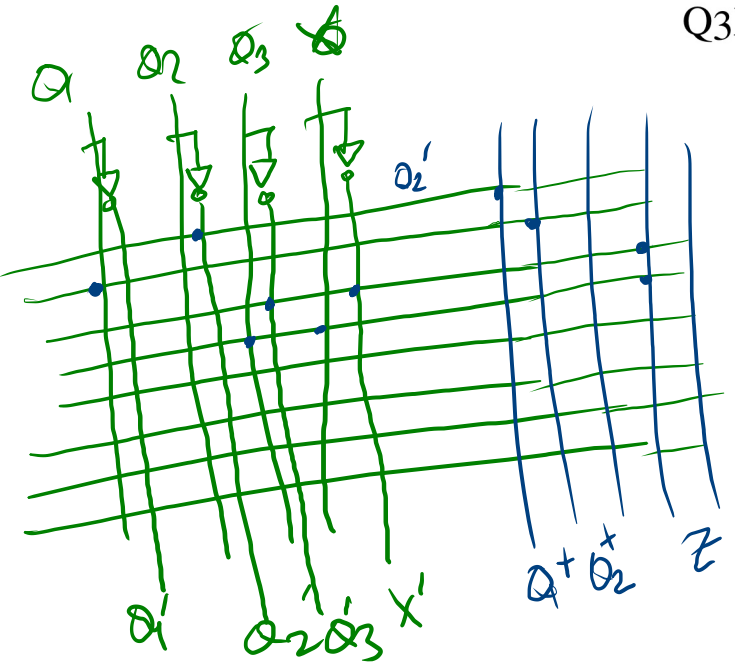


Table 3-4 PLA Table (Based on Figures 1-17 and 1-19)

Product Term	Q1	Q2	Q3	X	$Q1^+$	$Q2^+$	$Q3^+$	Z
Q2'	–	0	–	–	1	0	0	0
Q1	1	–	–	–	0	1	0	0
Q1Q2Q3	1	1	1	–	0	0	1	0
Q1Q3'X'	1	–	0	0	0	0	1	0
Q1'Q2'X	0	0	–	1	0	0	1	0
Q3'X'	–	–	0	0	0	0	0	1
Q3X	–	–	1	1	0	0	0	1



$Q1^+ = Q2'$

$Q2^+ = Q1$

$Q3^+ = Q1Q2\ Q3 + X'Q1Q3' + XQ1'Q2'$

$Z = X'Q3' + XQ3$

Figure 3-11 PLA Realization of Figure 1-17

```
library ieee;
use ieee.std_logic_1164.all;                    -- IEEE standard logic package
library MVLLIB;                                 -- includes PLAmtrx type and
use MVLLIB.mvl_pack.all;                        -- PLAout function

entity PLA1_2 is
      port(X,CLK: in std_logic;
            Z: out std_logic);
end PLA1_2;

architecture PLA of PLA1_2 is
signal Q, Qplus: std_logic_vector(1 to 3) := "000";
constant FSM_PLA:  PLAmtrx(0 to 6, 7 downto 0) :=
                  ("X0XX1000","1XXX0100","111X0010","1X000010",
                  "00X10010","XX000001","XX110001");
begin
      process(Q,X)
      variable PLAValue: std_logic_vector(3 downto 0);
      begin
            PLAValue := PLAout(FSM_PLA,Q & X);          -- read PLA output
            Qplus <= PLAValue(3 downto 1);
            Z <= PLAValue(0);
      end process;

      process(CLK)
      begin
            if CLK='1' then Q <= Qplus; end if;         -- update state register
      end process;
end PLA;
```