Figure 3-1 Basic ROM Structure

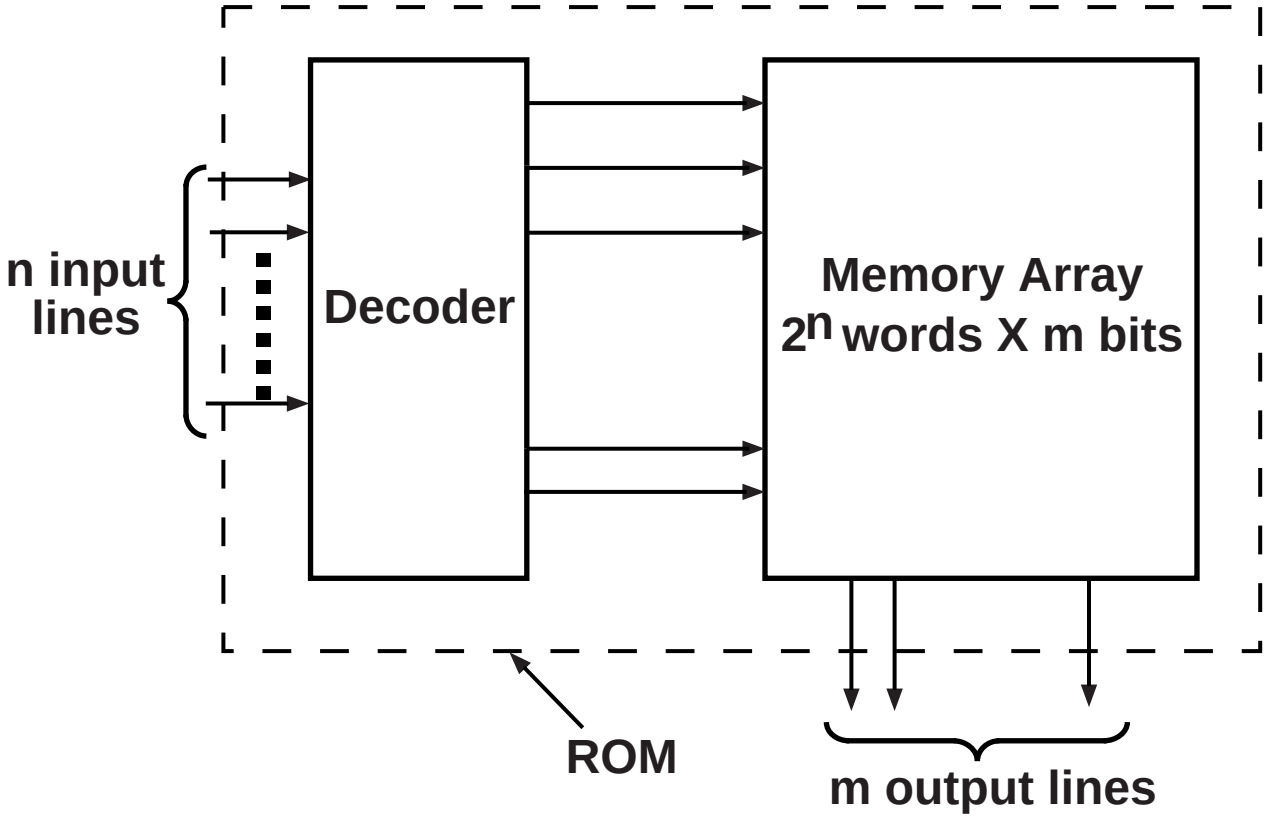


Figure 3-2
Realization of a Mealy Sequential Network with a ROM

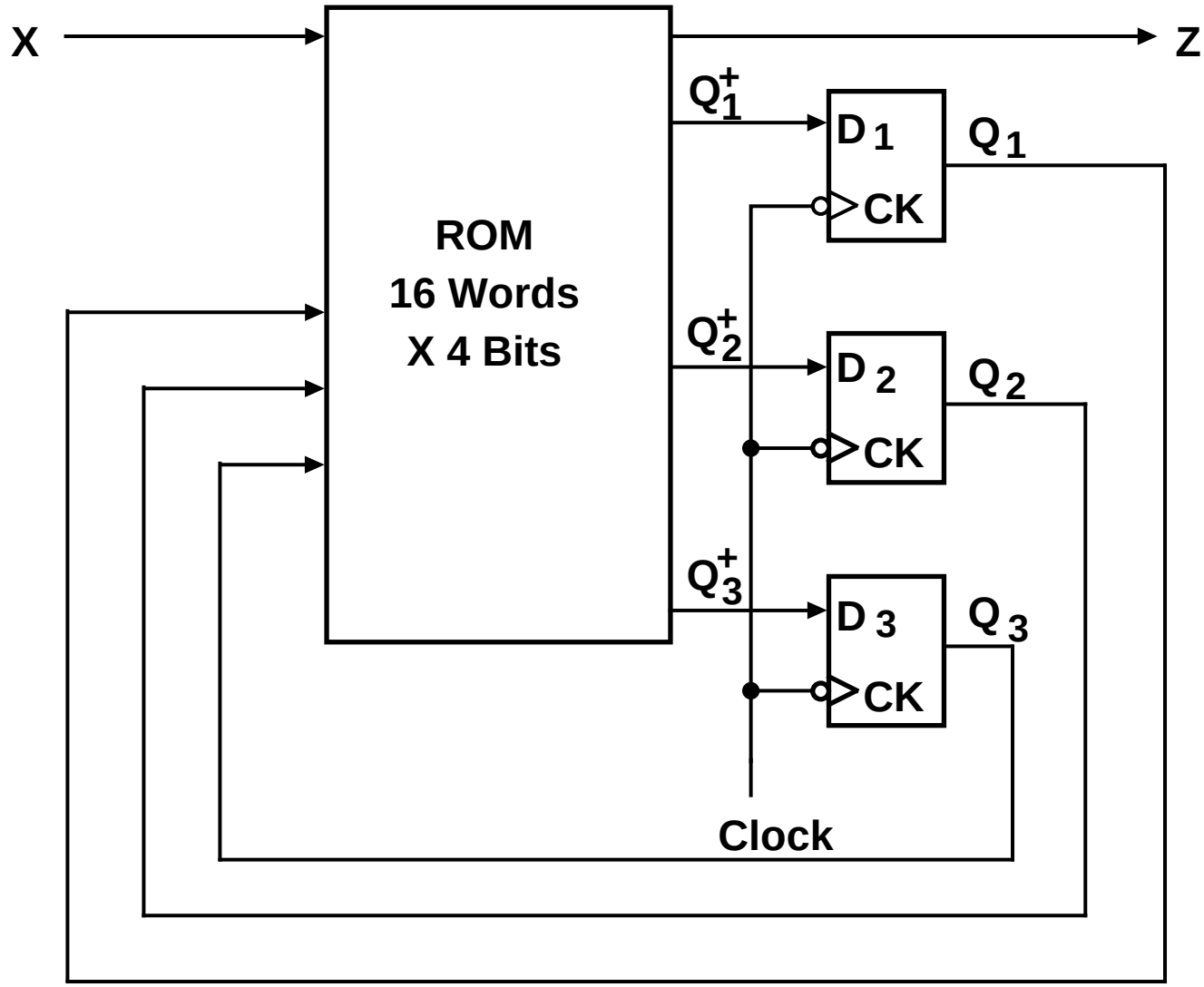


Table 3-1 ROM Truth Table

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

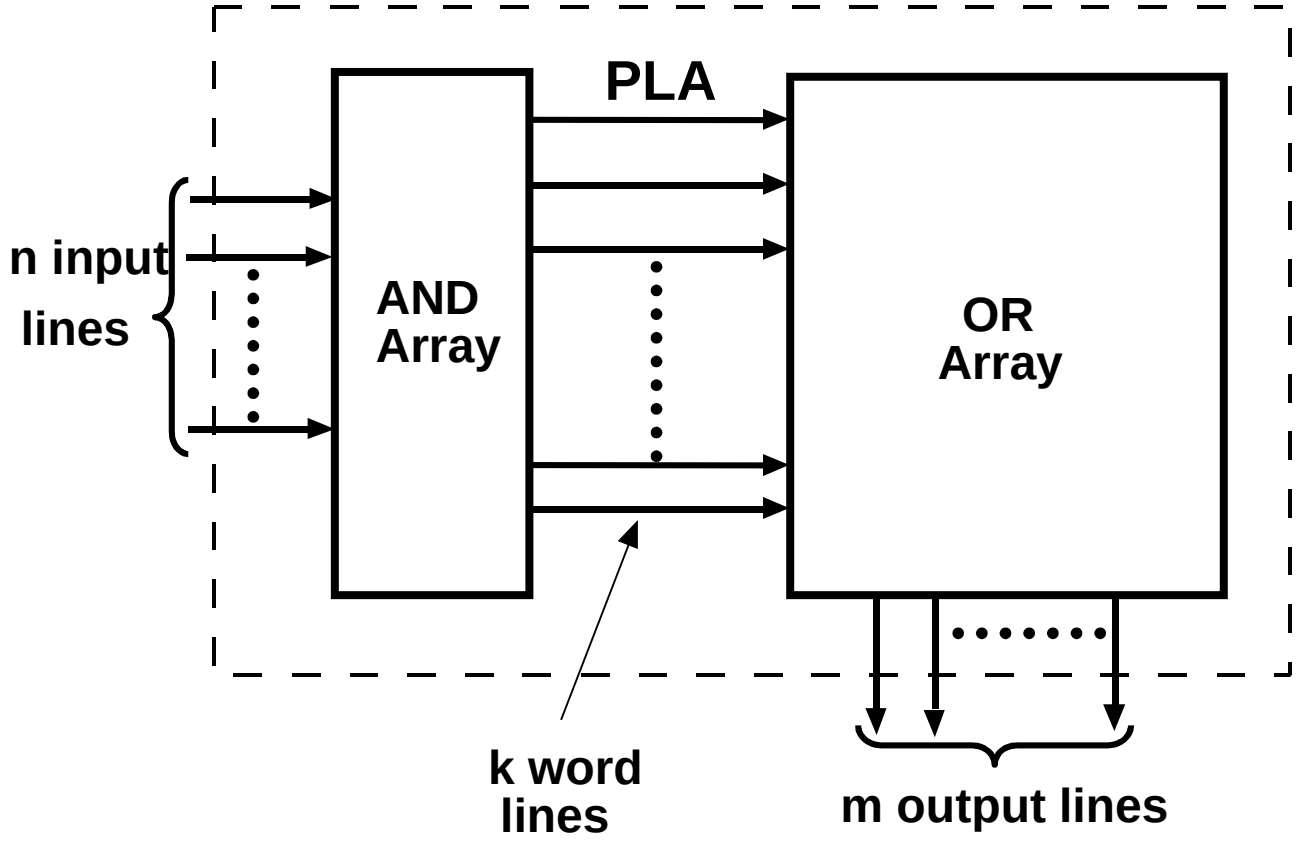


Figure 3-5 PLA with 3 input, 5 Product Terms, and 4 Outputs

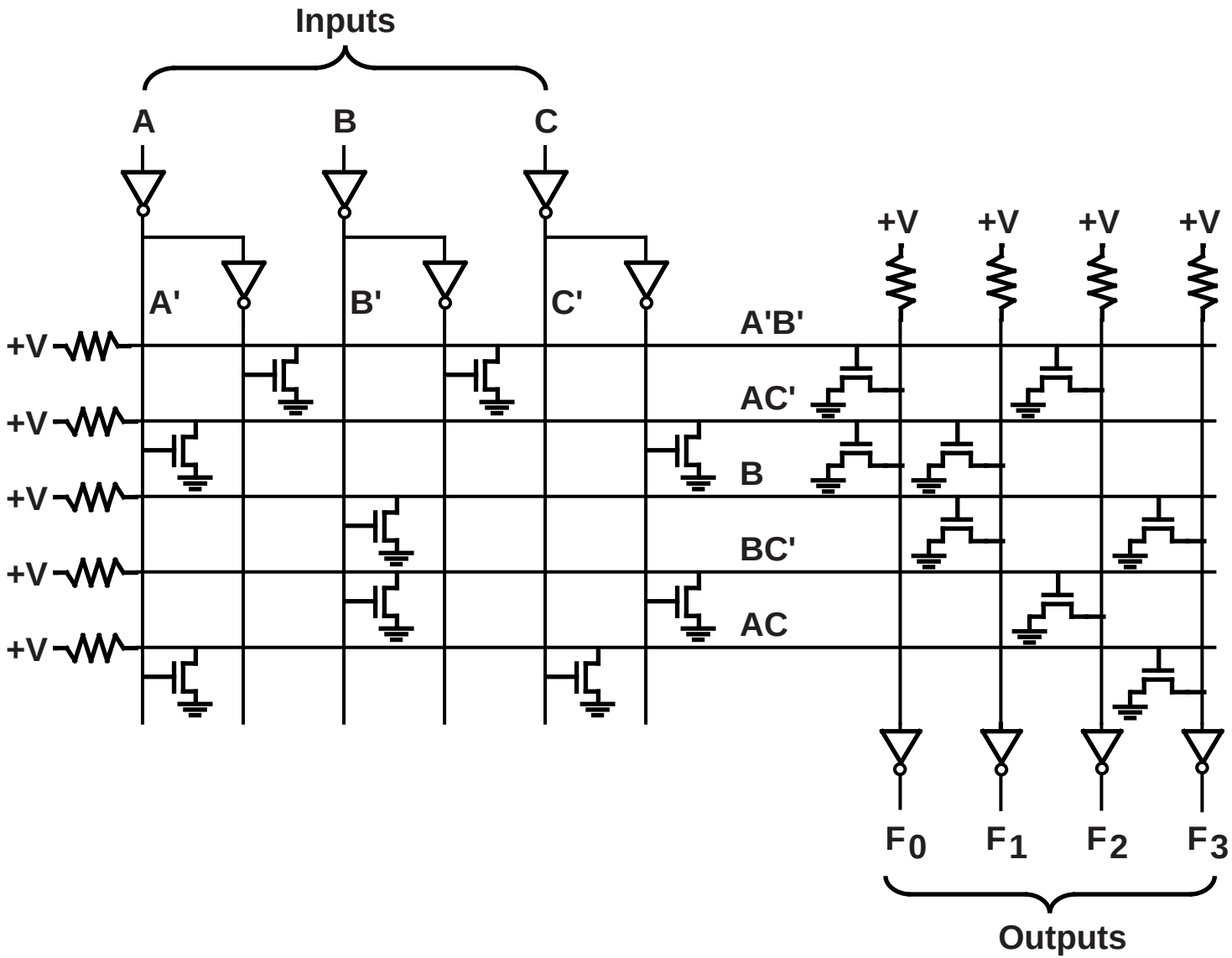


Figure 3-6 nMOS NOR Gate

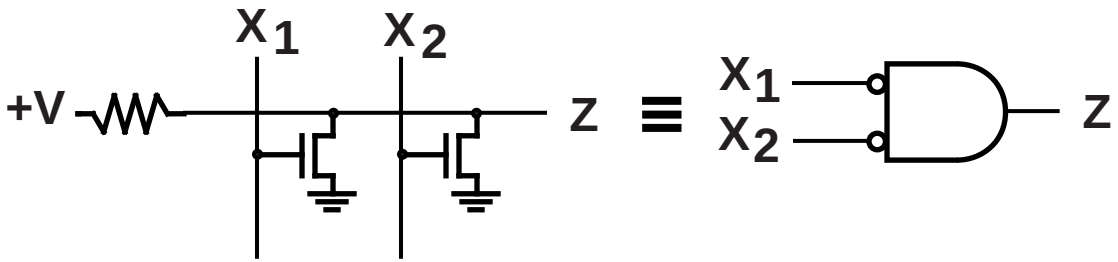


Figure 3-7 Conversion for NOR-NOR to AND-OR

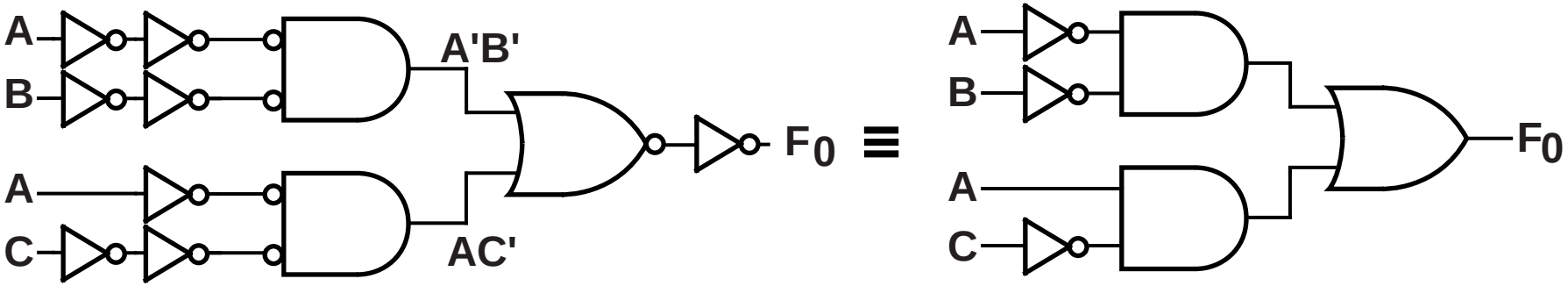


Figure 3-8 AND-OR Array Equivalent to Figure 3-5

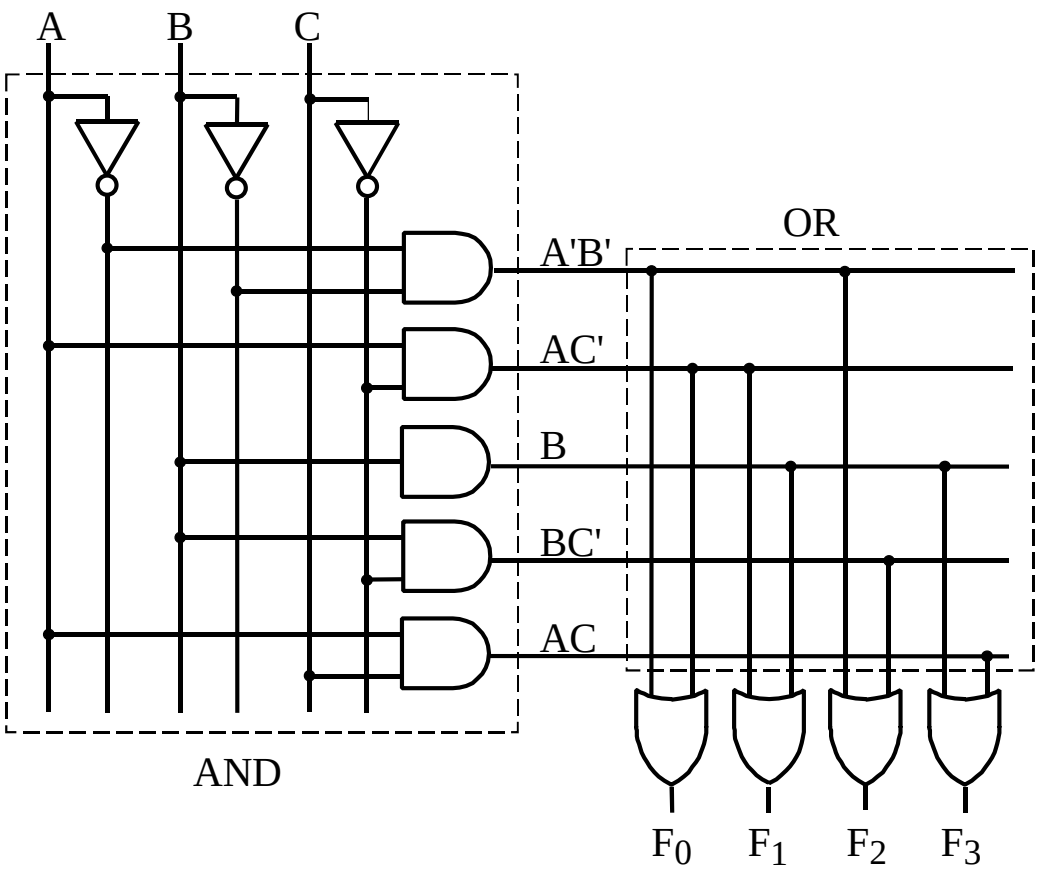
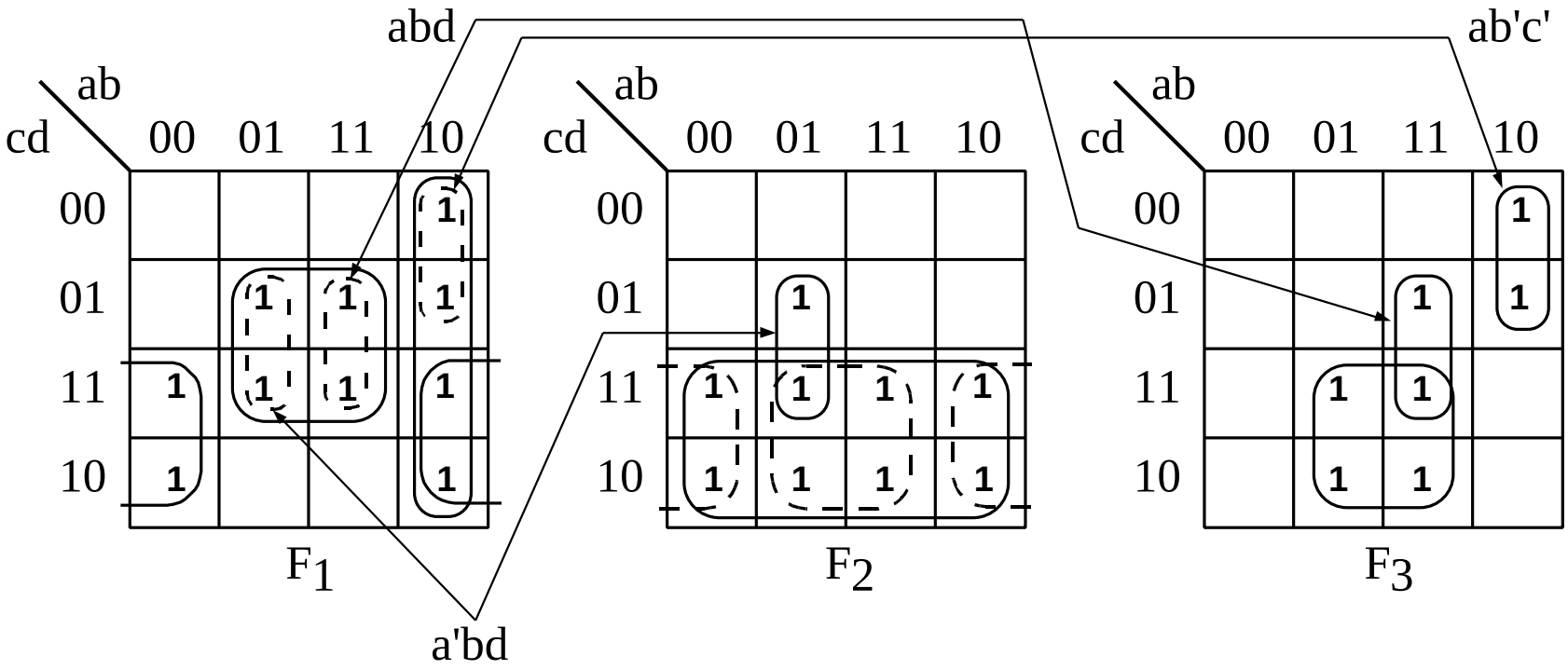


Table 3-2 PLA Table for Figure 3-5

	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

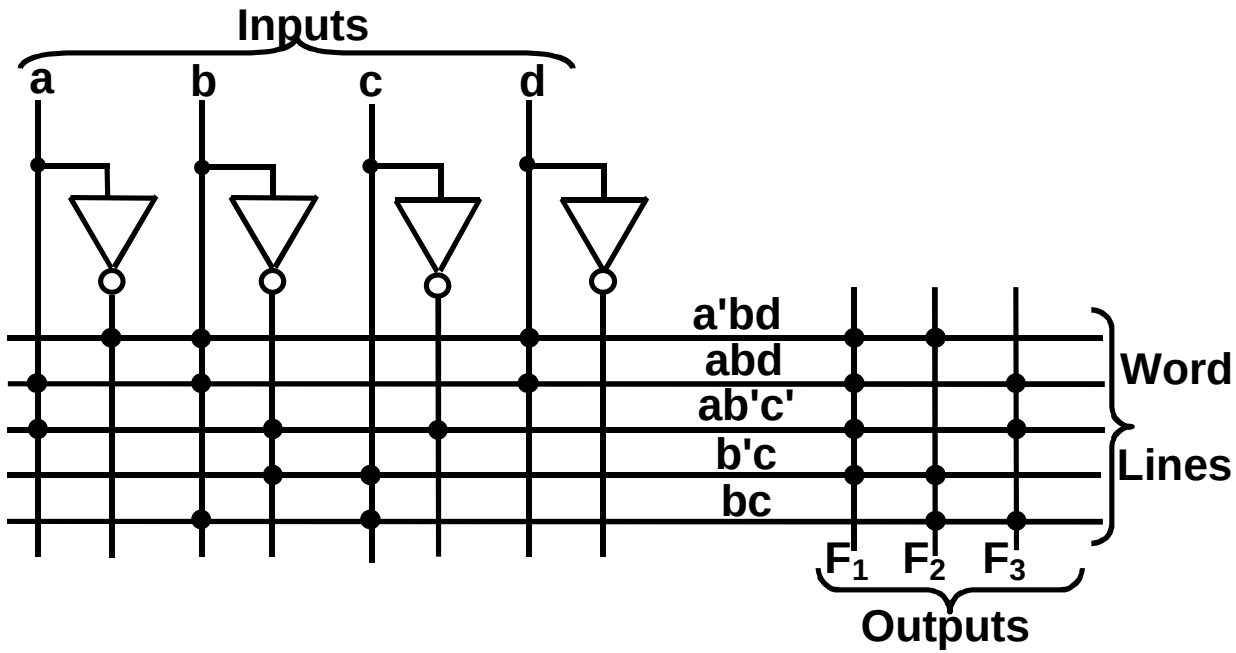


Table 3-4 PLA Table (Based on Figures 1-17 and 1-19)

Product Term	Q_1	Q_2	Q_3	X	Q_1^+	Q_2^+	Q_3^+	Z
Q_2'	–	0	–	–	1	0	0	0
Q_1	1	–	–	–	0	1	0	0
$Q_1Q_2Q_3$	1	1	1	–	0	0	1	0
$Q_1Q_3'X'$	1	–	0	0	0	0	1	0
$Q_1'Q_2'X$	0	0	–	1	0	0	1	0
$Q_3'X'$	–	–	0	0	0	0	0	1
Q_3X	–	–	1	1	0	0	0	1

$$Q1^+ = Q2'$$

$$Q2^+ = Q1$$

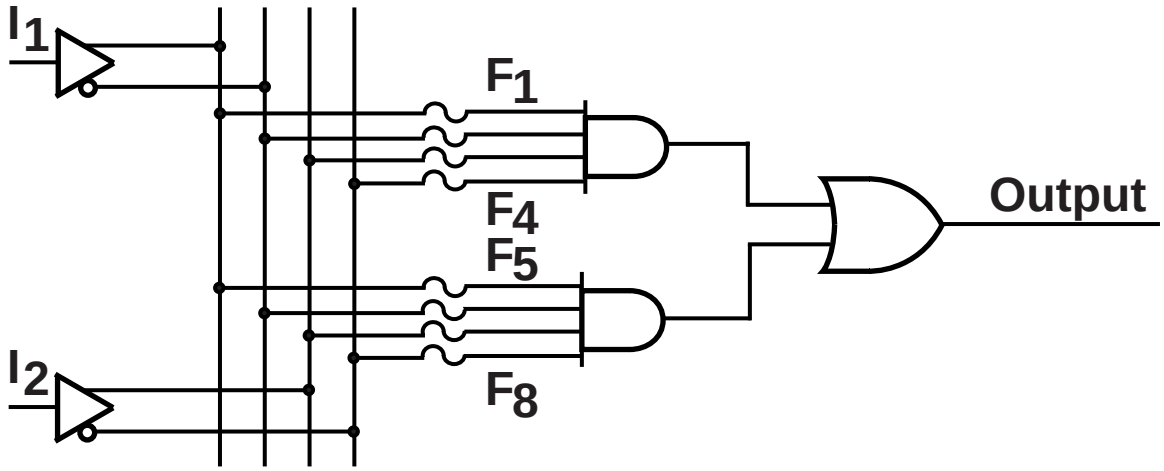
$$Q3^+ = Q1Q2\ Q3 + X'Q1Q3' + XQ1'Q2'$$

$$Z = X'Q3' + XQ3$$

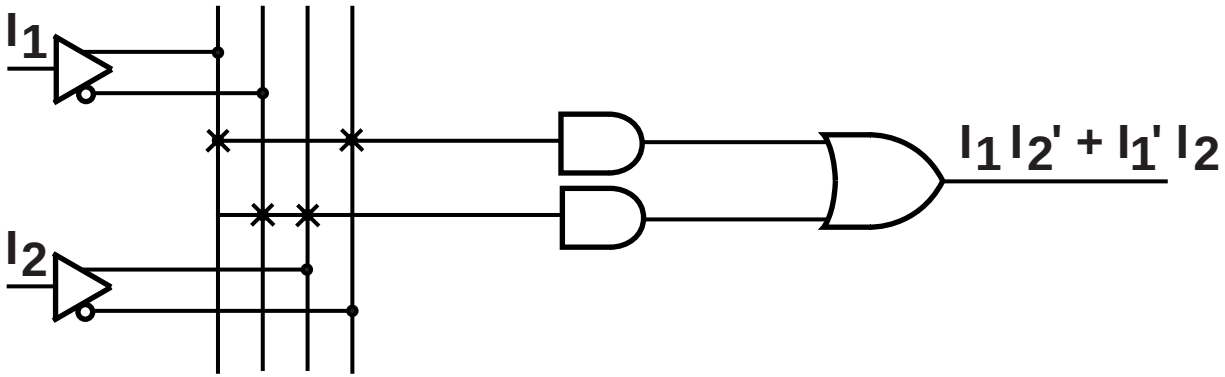
Figure 3-11 PLA Realization of Figure 1-17

```
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;
```

Figure 3-12 Combinational PAL Segment



(a) Unprogrammed



(b) Programmed

Figure 3-13 Segment of a Sequential PAL

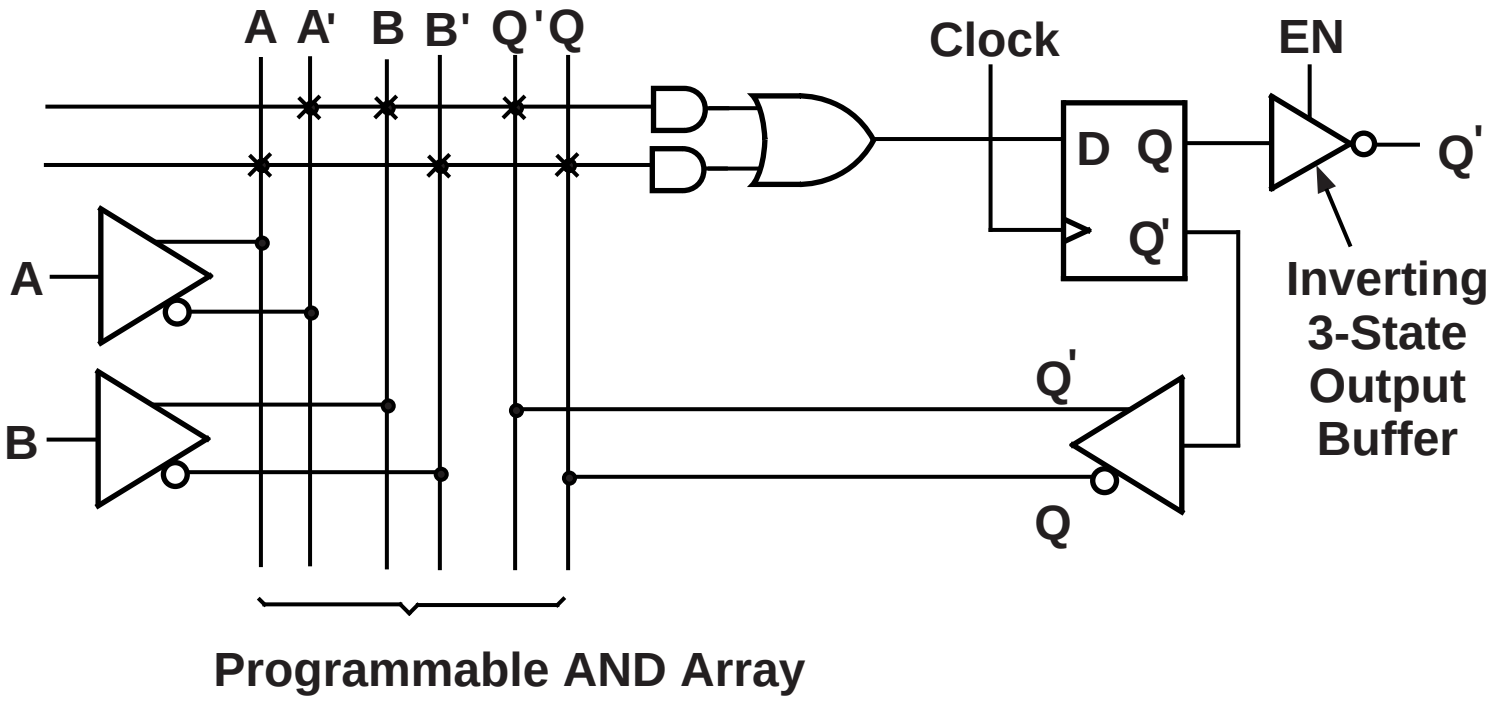


Figure 3-14a Logic Diagram for 16R4 PAL

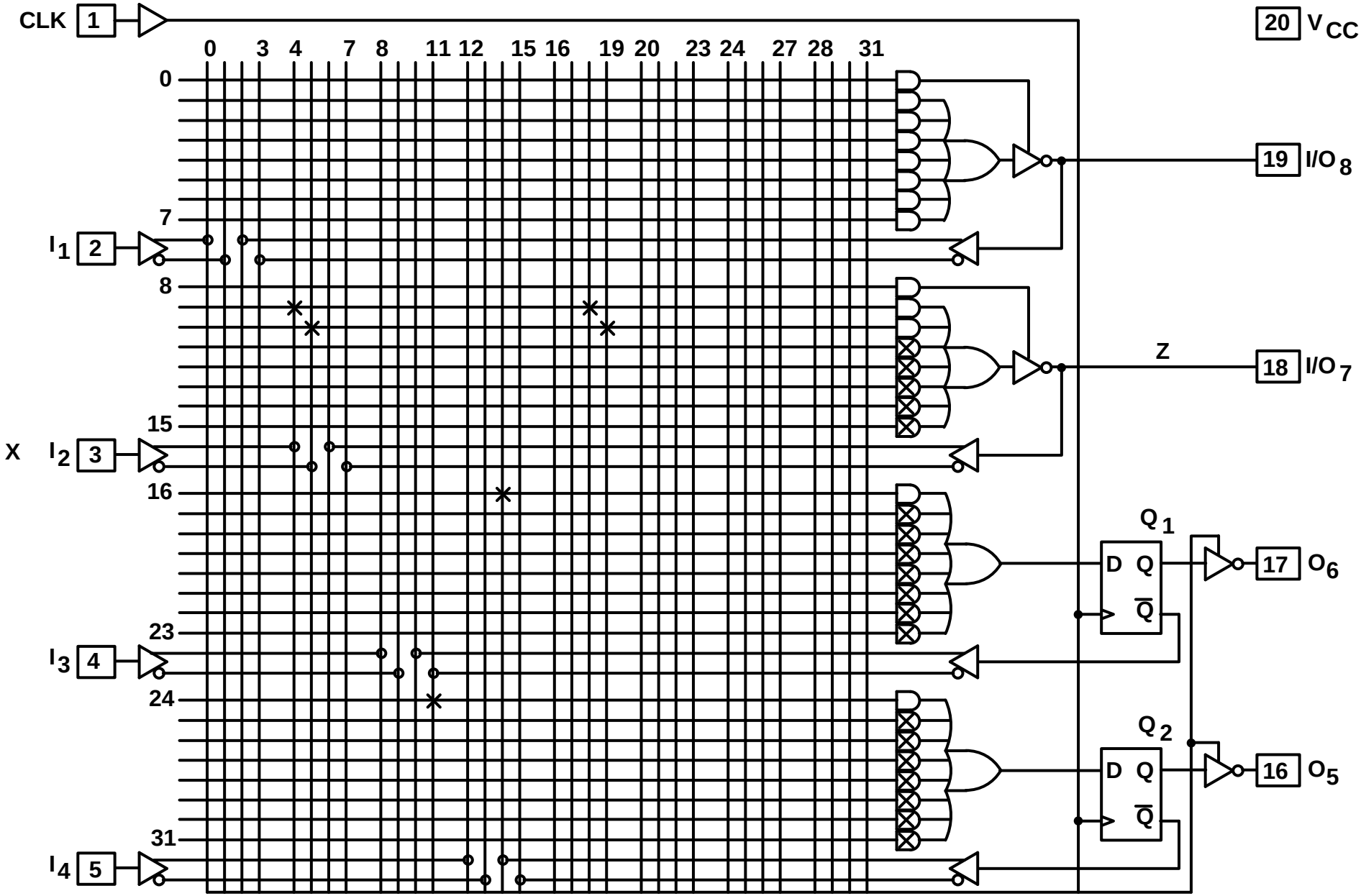


Figure 3-14b Logic Diagram for 16R4 PAL

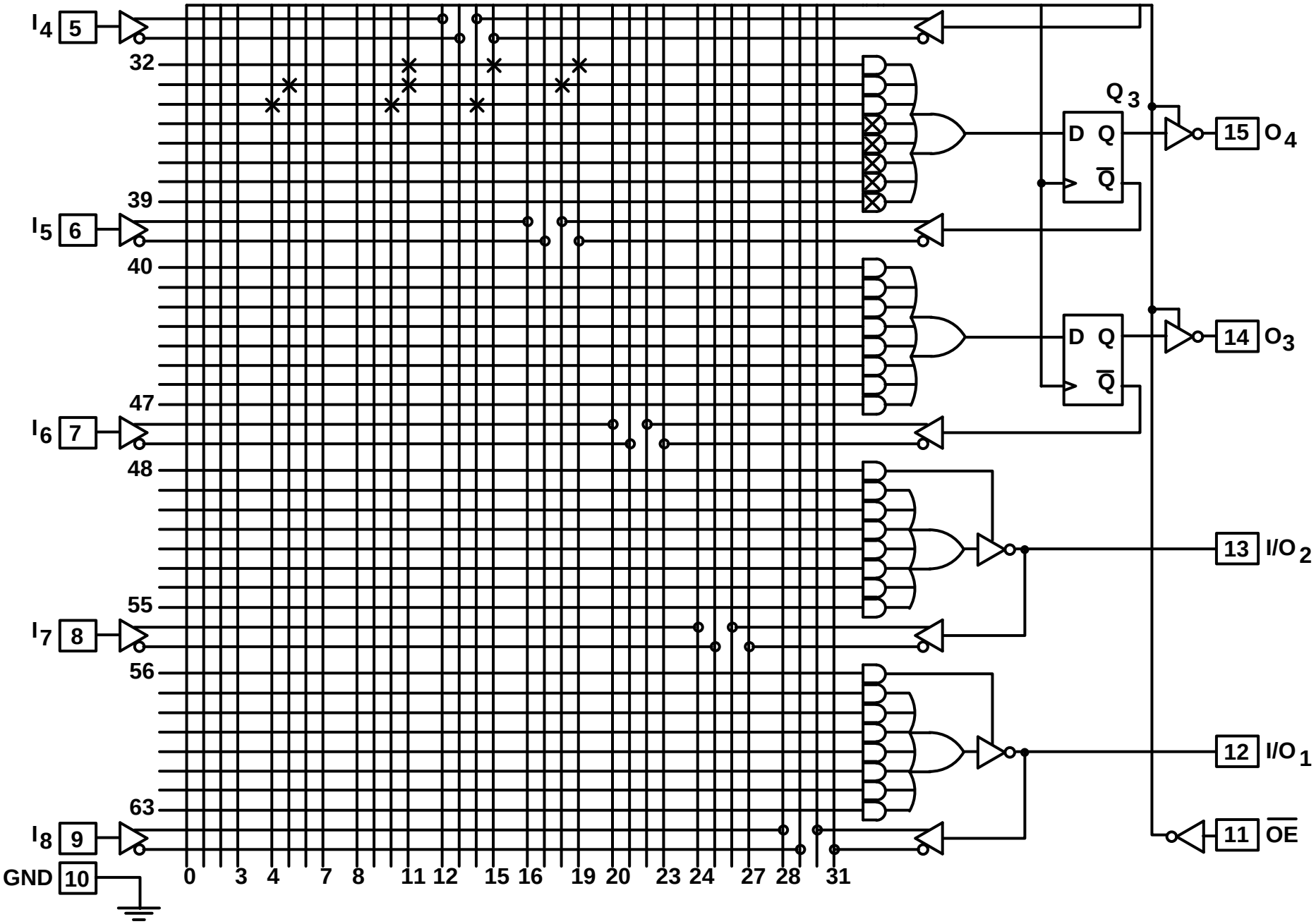


Figure 3-15 Block Diagram for 22V10

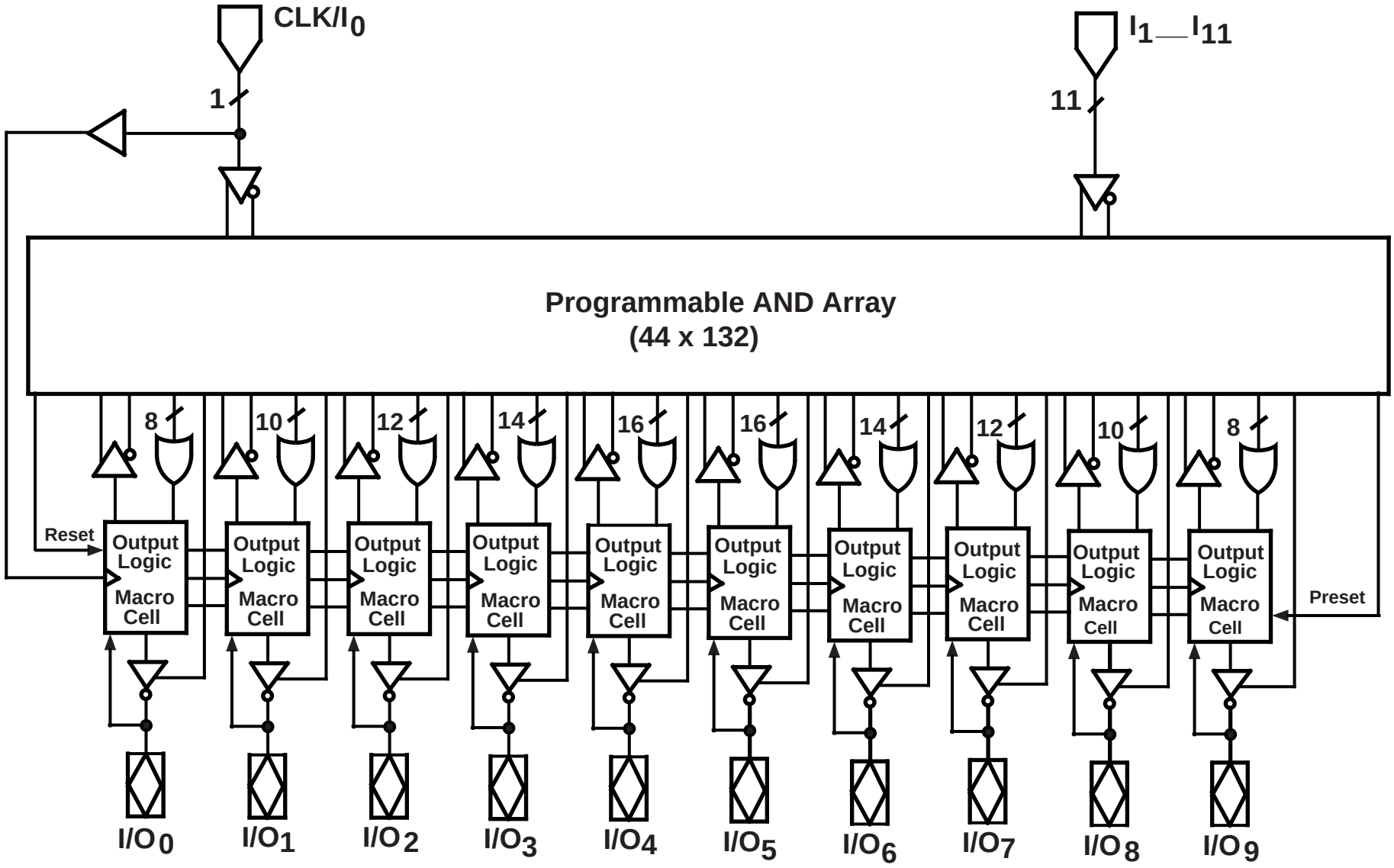
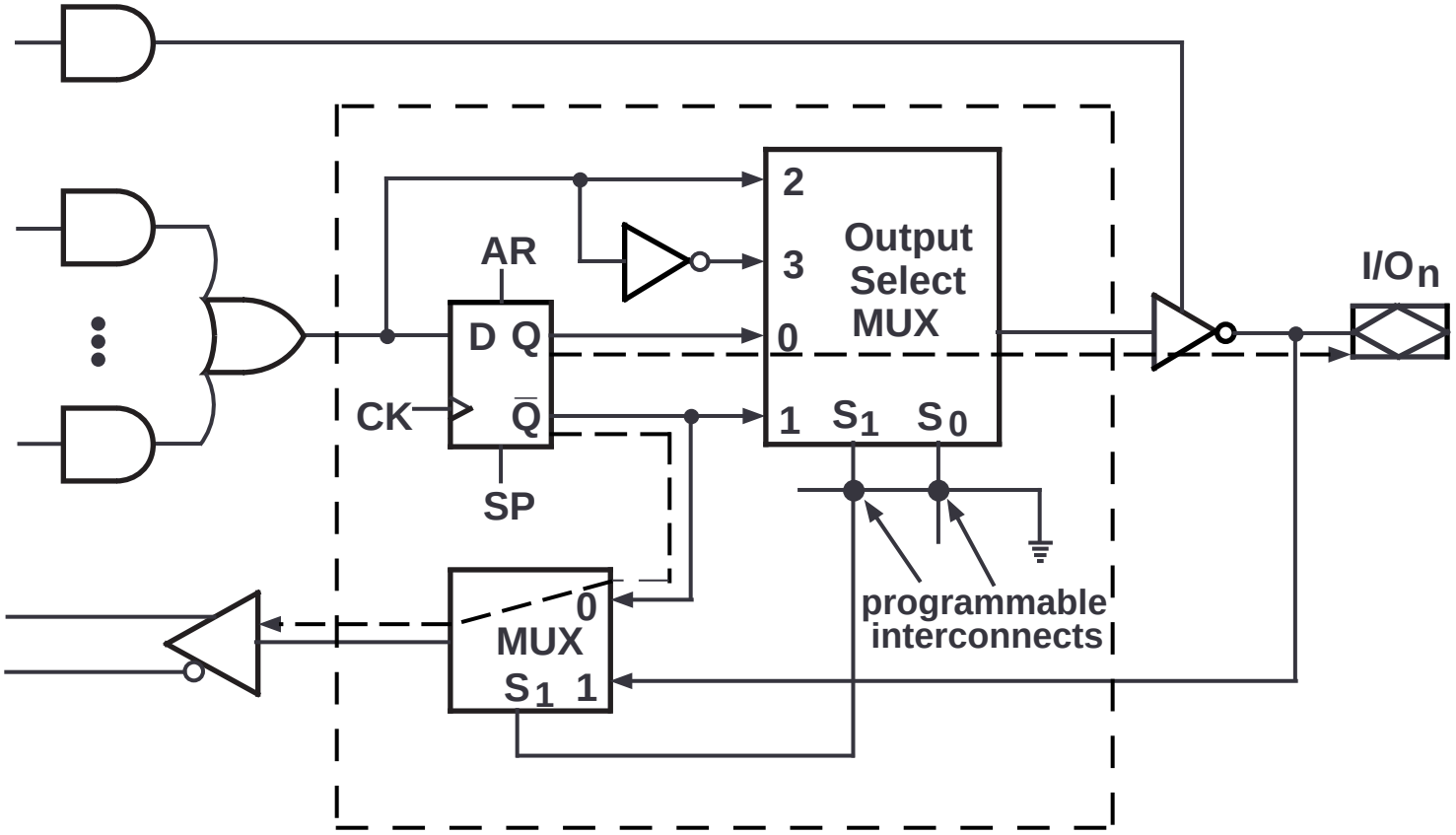


Table 3-5 Characteristics of Simple CMOS PLD's

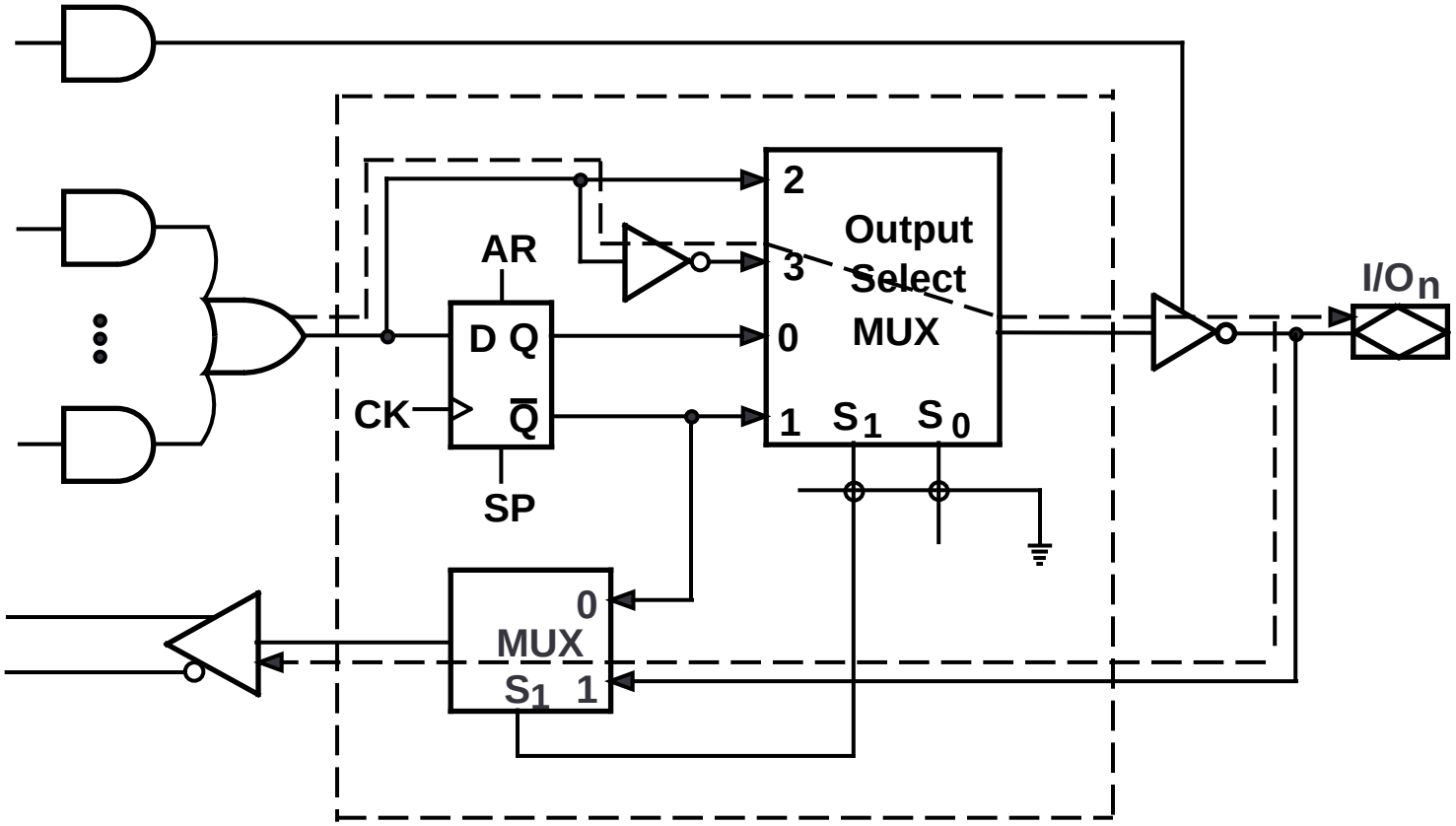
Type No.	No. of inputs	I/O	Macrocells = FFs	AND gates per OR gate
PALCE16V10	8 + $\overline{\text{OE}}$ + Clk	8	8	8
PALCE20V8	14	8	8	8
PALCE22V10	12	10	10	8–16
PALCE24V10	14	10	10	8
PALCE29MA16	5 + Clk	16	16	4–12
CY7C335	12+ $\overline{\text{OE}}$ + Clk	12	12 in/12 out	9–19

Figure 3-16a Output Macrocell



(a) paths with $S_1 = S_0 = 0$

Figure 3-16b Output Macrocell



(b) paths with $S_1 = S_0 = 1$

Figure 3-17 Block Diagram of Traffic-Light Controller

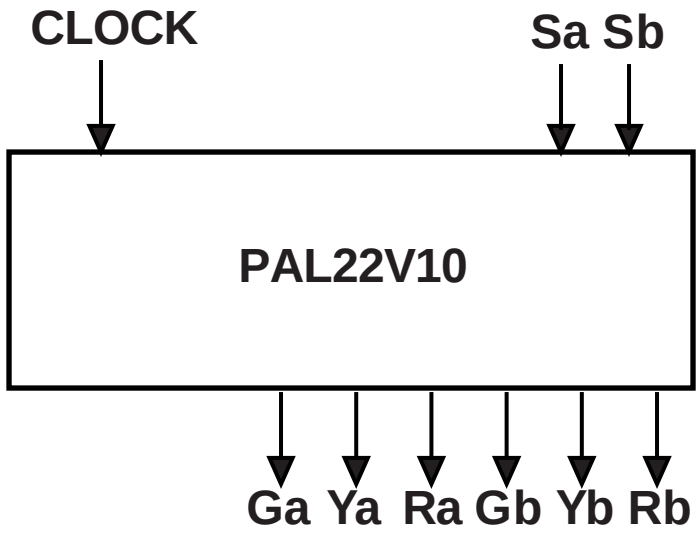


Figure 3-18 State Graph for Traffic-Light Controller

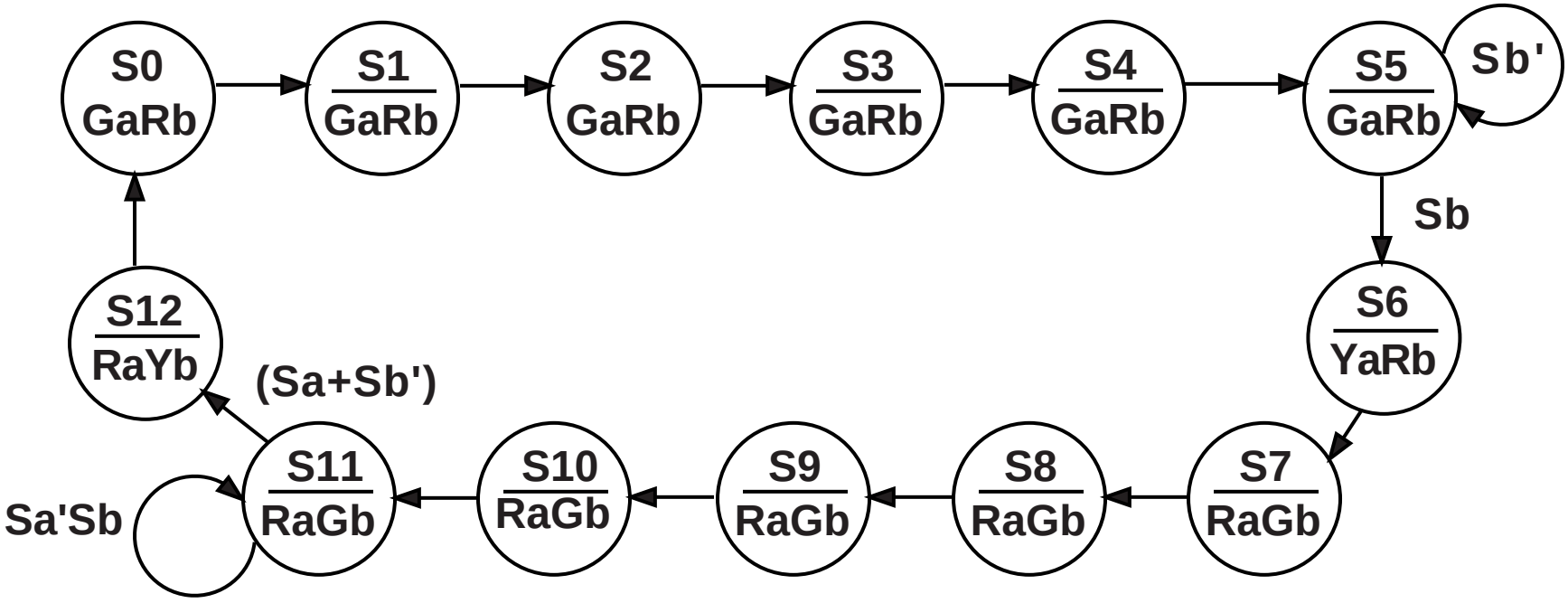


Figure 3-19 VHDL Code for Traffic-Light Controller

```
entity traffic_light is
      port (clk, Sa, Sb: in bit;
                  Ra, Rb, Ga, Gb, Ya, Yb: inout bit);
end traffic_light;

architecture behave of traffic_light is
      signal state, nextstate: integer range 0 to 12;  type light is (R, Y, G);
      signal lightA, lightB: light;                          -- define signals for waveform output
begin
      process(state, Sa, Sb)
      begin
            Ra <= '0'; Rb <= '0'; Ga <= '0'; Gb <= '0'; Ya <= '0'; Yb <= '0';
            case state is
                  when 0 to 4 => Ga <= '1'; Rb <= '1'; nextstate <= state+1;
                  when 5 => Ga <= '1'; Rb <= '1';  if Sb = '1' then nextstate <= 6; end if;
                  when 6 => Ya <= '1'; Rb <= '1';  nextstate <= 7;
                  when 7 to 10 => Ra <= '1'; Gb <= '1'; nextstate <= state+1;
                  when 11 => Ra <= '1'; Gb <= '1';
                                    if (Sa='1' or Sb='0') then nextstate <= 12; end if;
                  when 12 => Ra <= '1'; Yb <= '1'; nextstate <= 0;
            end case;
      end process;
      process(clk)
      begin
            if clk = '1' then state <= nextstate; end if;
      end process;
      lightA <= R when Ra='1' else Y when Ya='1' else G when Ga='1';
      lightB <= R when Rb='1' else Y when Yb='1' else G when Gb='1';
end behave;
```

Figure 3-20 Test Results for Traffic-Light Controller

wave clk SA SB state lightA lightB
force clk 0 0,1 5 sec -repeat 10 sec
force SA 1 0, 0 40, 1 170, 0 230, 1 250 sec
force SB 0 0, 1 70, 0, 100, 1 120, 0 150, 1 210, 0 250, 1 270 sec

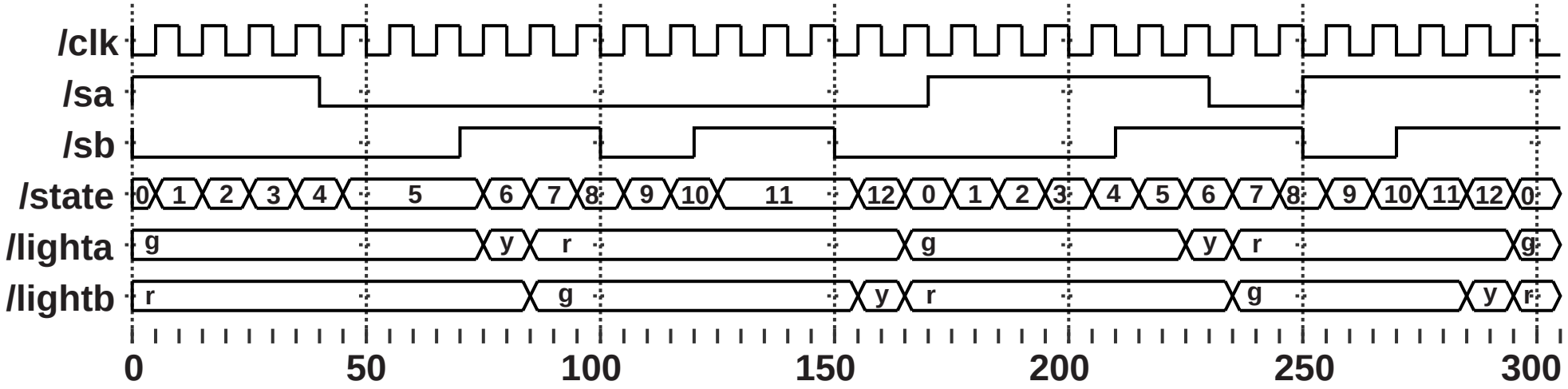
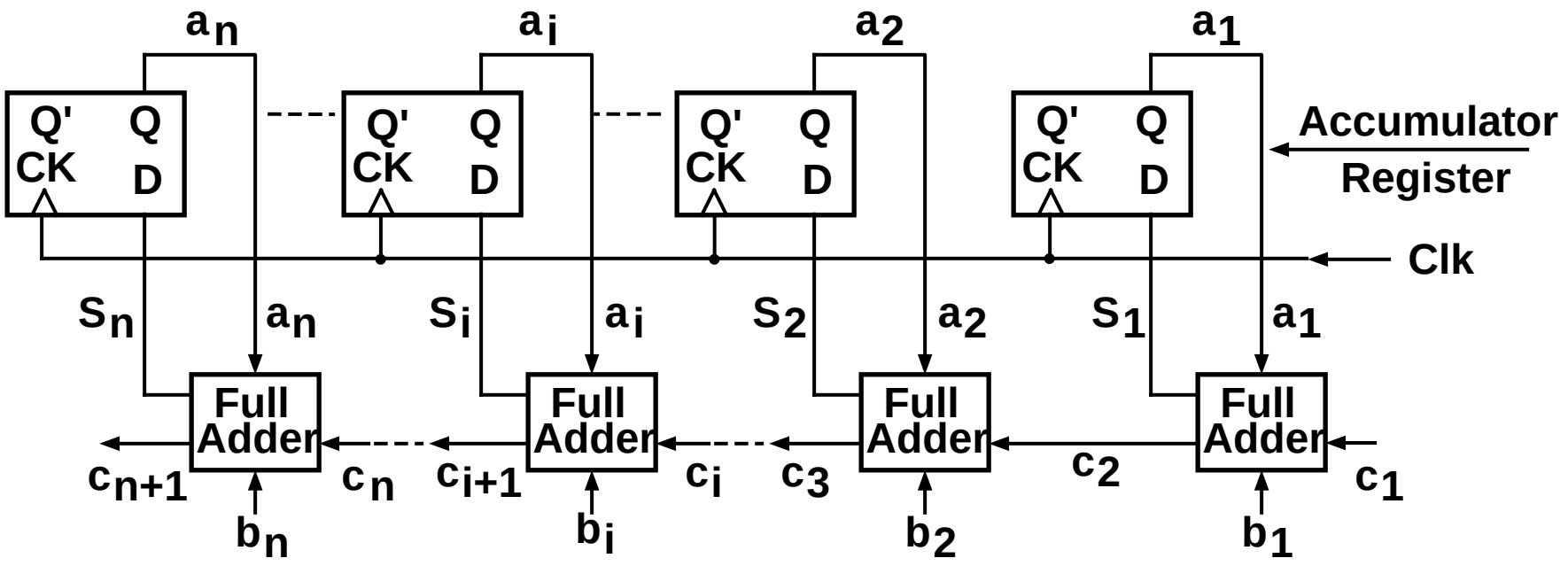


Table 3-6 State Table for Traffic-Light Controller

	SaSb										
	00	01	10	11	*Ga*	*Ya*	*Ra*	*Gb*	*Yb*	*Rb*	
S0	S1	S1	S1	S1	1	0	0	0	0	1	{Green A, Red B}
S1	S2	S2	S2	S2	1	0	0	0	0	1	
S2	S3	S3	S3	S3	1	0	0	0	0	1	
S3	S4	S4	S4	S4	1	0	0	0	0	1	
S4	S5	S5	S5	S5	1	0	0	0	0	1	
S5	S5	S6	S5	S6	1	0	0	0	0	1	
S6	S7	S7	S7	S7	0	1	0	0	0	1	{Ya,Rb}
S7	S8	S8	S8	S8	0	0	1	1	0	0	{Ra,Gb}
S8	S9	S9	S9	S9	0	0	1	1	0	0	
S9	S10	S10	S10	S10	0	0	1	1	0	0	
S10	S11	S11	S11	S11	0	0	1	1	0	0	
S11	S12	S11	S12	S12	0	0	1	1	0	0	
S12	S0	S0	S0	S0	0	0	1	0	1	0	{Ra,Yb}

D1 = Q1 Q2' + Q2 Q3 Q4
D2 = Q1'Q2'Q3 Q4 + Sa Q1 Q3 Q4 + Sb'Q1 Q3 Q4 + Q1'Q2 Q4' + Q1'Q2 Q3'
D3 = Q3 Q4' + Sb Q3'Q4 + Q2'Q3'Q4 + Sa'Sb Q1 Q4
D4 = Sa'Sb Q1 Q3 + Q2'Q4' + Q1'Q4' + Sa Sb'Q2 Q3'Q4
Ga = Q1' Q3' + Q1'Q2' Ya = Q2Q3Q4' Ra = Q1 + Q2Q3Q4
Gb = Q1 Q2' + Q2Q3Q4 Yb = Q1 Q2 Rb = Q1'Q2' + Q1'Q4'+ Q1'Q3'

Figure 3-21 Parallel Adder with Accumulator



From Page 109

A_2	A_1	B_2	B_1	C_1	C_3	S_2	S_1
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	1
0	0	0	1	0	0	0	1
0	0	0	1	1	0	1	0
0	0	1	0	0	0	1	0
–	–	–	–	–	–	–	–
1	1	0	1	0	1	0	0
1	1	0	1	1	1	0	1
1	1	1	0	0	1	0	1
1	1	1	0	1	1	1	0
1	1	1	1	0	1	1	0
1	1	1	1	1	1	1	1

$$C_3 = B_1\,B_2\,C_1 + A_1\,B_2\,C_1 + A_1\,B_2\,B_1 + A_2\,B_1\,C_1 + A_2\,B_2 + A_1\,A_2\,C_1 + A_1\,A_2\,B_1 + A_1\,A_2\,C_1 + A_1\,A_2\,B_1$$

$$S_2 = (A_2'B_1\,B_2'C_1 + A_1'A_2'B_1'B_2 + A_1'A_2'B_2\,C_1' + A_1\,A_2'B_2'C_1 + A_1\,A_2'B_1\,B_2' + A_2'B_1'B_2\,C_1' + A_1'A_2\,B_1'B_2' + A_1'A_2\,B_2'C_1' + A_2\,B_1\,B_2\,C_1 + A_2\,B_1'B_2'C_1' + A_1\,A_2\,B_2\,C_1 + A_1\,A_2\,B_1\,B_2)Ad + Ad'A_2$$

$$S_1 = (A_1'B_1'C_1 + A_1'B_1\,C_1' + A_1\,B_1'C_1' + A_1\,B_1\,C_1)Ad + Ad'A_1$$

Figure 3-22 Block Diagram for Keypad Scanner

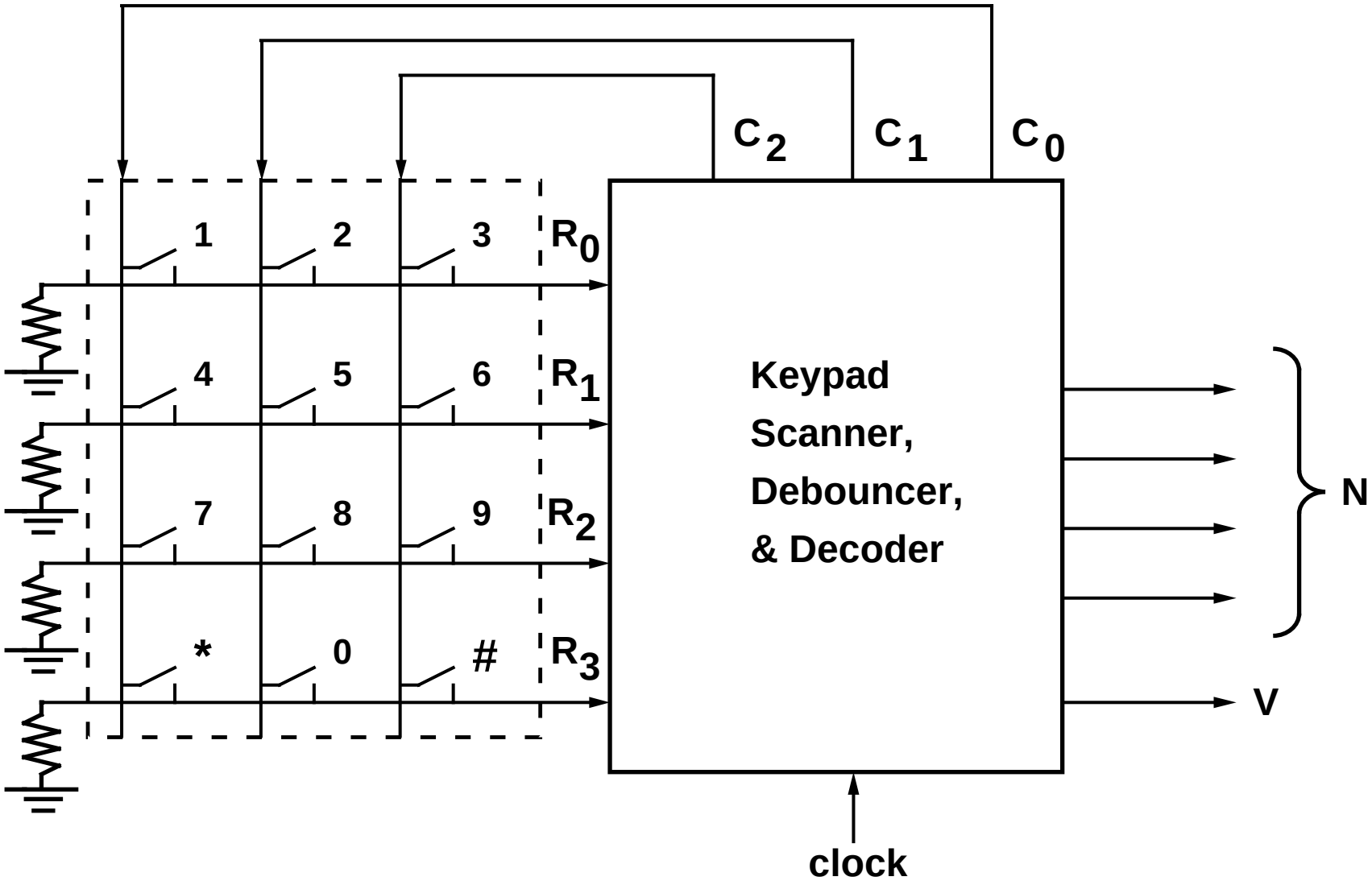


Figure 3-23 Debouncing and Synchronizing Circuit

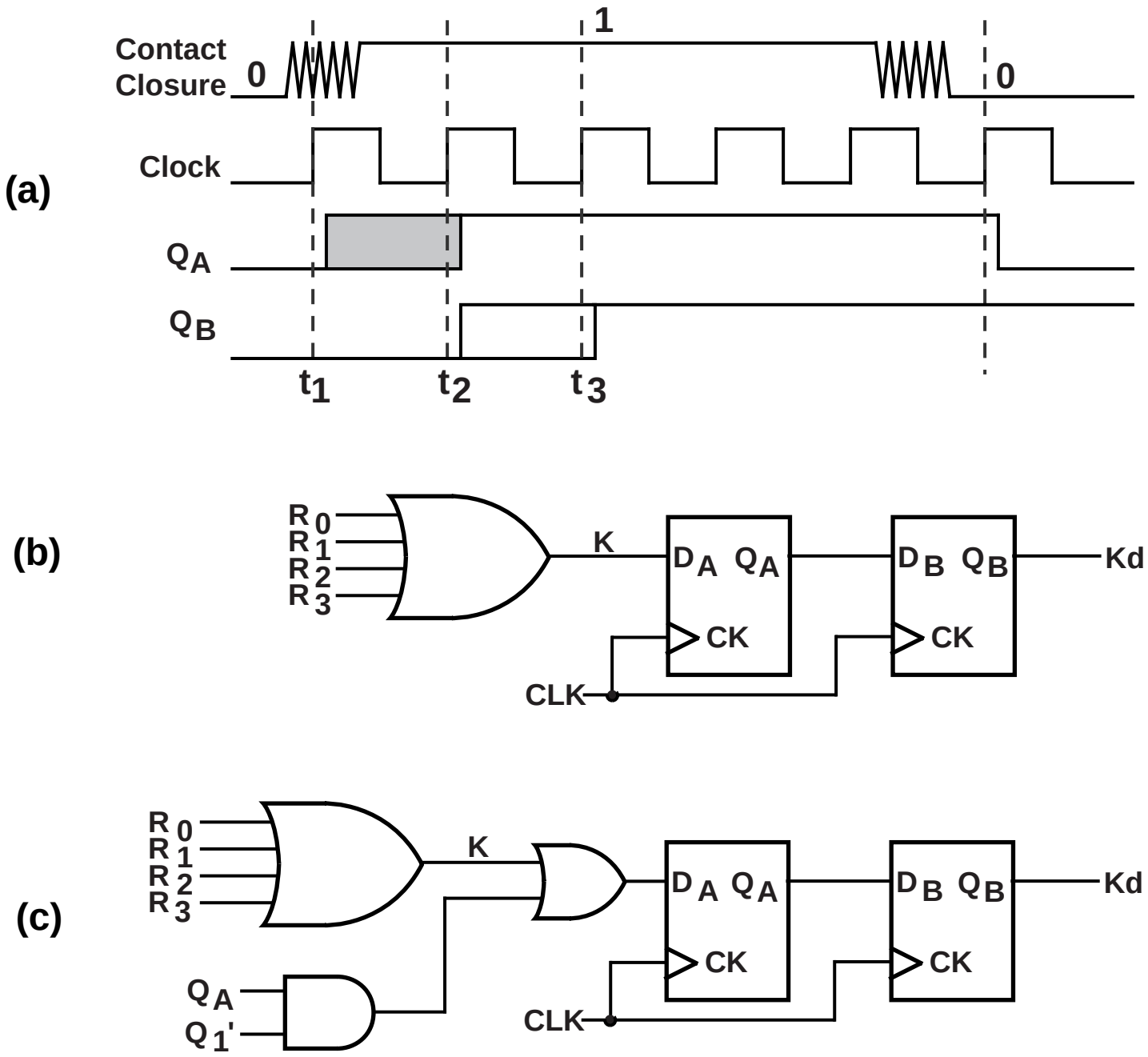


Figure 3-24 Scanner Modules

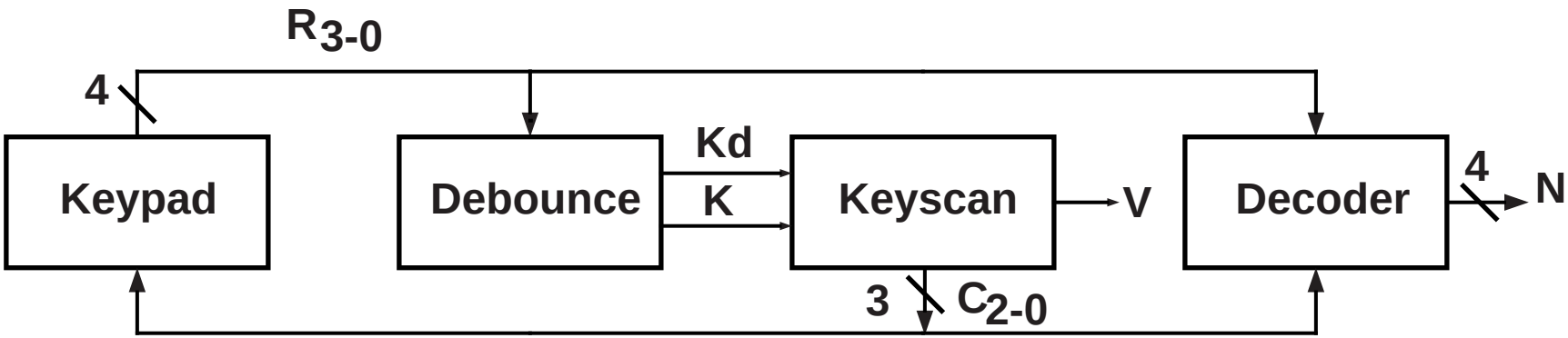
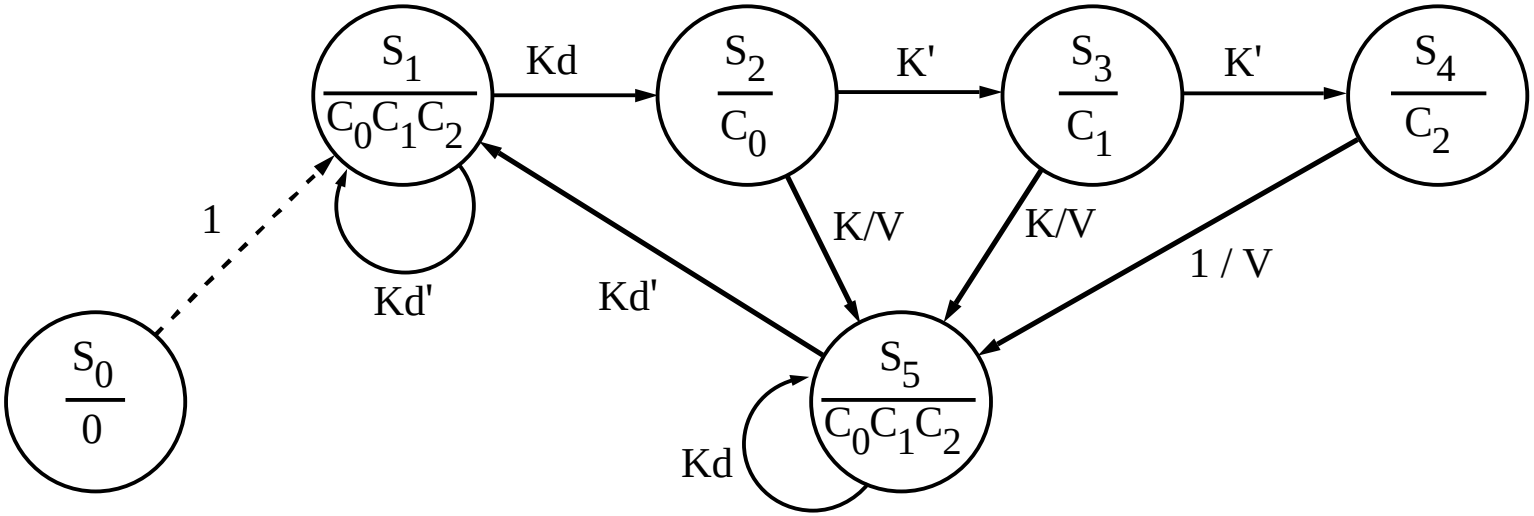


Figure 3-25 State Graph for Scanner

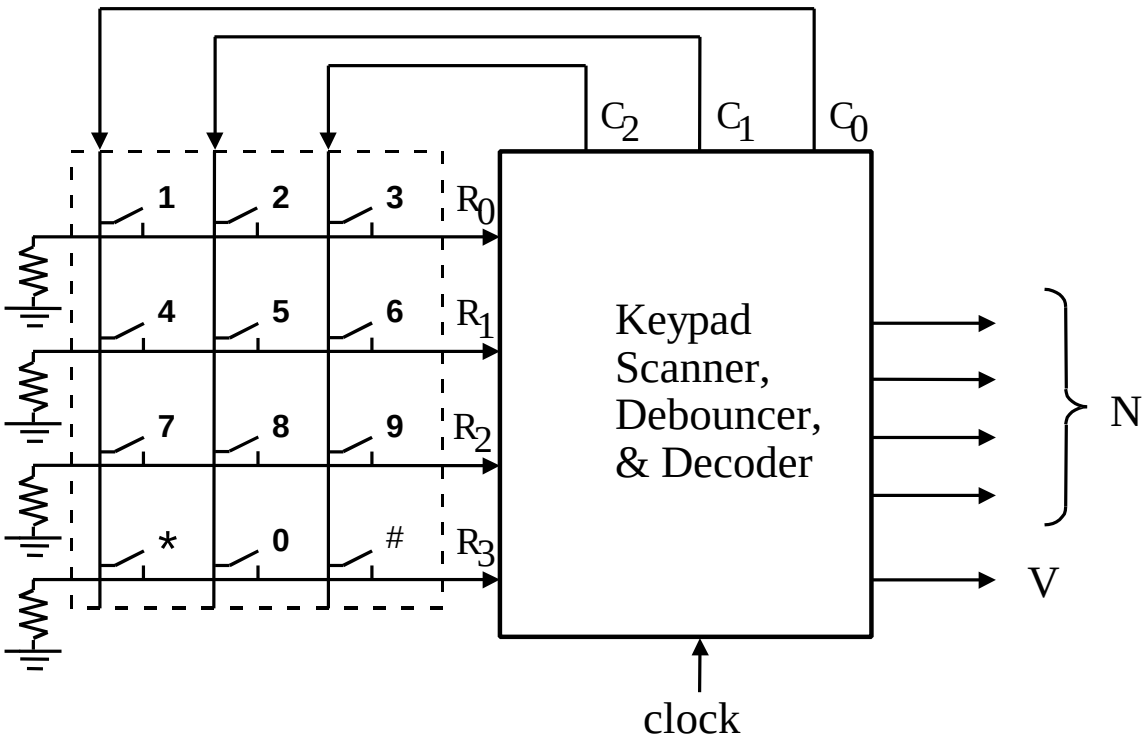


State Assignment for Q1 Q2 Q3 Q4:

$S_1 \leftarrow 0111$, $S_2 \leftarrow 0100$, $S_3 \leftarrow 0010$, $S_4 \leftarrow 0001$, $S_5 \leftarrow 1111$

$Q_1^+ = Q_1\,Kd + Q_2\,Q_3'K + Q_2'Q_3\,K + Q_2'Q_4$
$Q_2^+ = Q_2'Q_3' + K + Q_4$
$Q_3^+ = Q_3' + Q_1 + Q_4\,Kd' + Q_2'K$
$Q_4^+ = Q_2' + Q_1 + Q_3\,Kd' + Q_3'K$
$V = KQ_2Q_3' + KQ_2'Q_3 + Q_2'Q_4$

Table 3-7 Truth Table for Decoder



R_3	R_2	R_1	R_0	C_0	C_1	C_2	N_3	N_2	N_1	N_0	
0	0	0	1	1	0	0	0	0	0	1	
0	0	0	1	0	1	0	0	0	1	0	
0	0	0	1	0	0	1	0	0	1	1	
0	0	1	0	1	0	0	0	1	0	0	
0	0	1	0	0	1	0	0	1	0	1	
0	0	1	0	0	0	1	0	1	1	0	
0	1	0	0	1	0	0	0	1	1	1	
0	1	0	0	0	1	0	1	0	0	0	
0	1	0	0	0	0	1	1	0	0	1	
1	0	0	0	1	0	0	1	0	1	0	(*)
1	0	0	0	0	1	0	0	0	0	0	
1	0	0	0	0	0	1	1	0	1	1	(#)

Logic equations for decoder:

N3 = R2 C0' + R3 C1'
N2 = R1 + R2 C0
N1 = R0 C0' + R2'C2 + R1'R0'C0
N0 = R1 C1 + R1'C2 + R3'R1'C1'

Figure 3-26 VHDL Code for Scanner

```
entity scanner is
      port (R0,R1,R2,R3,CLK: in bit;
            C0,C1,C2: inout bit;
            N0,N1,N2,N3,V: out bit);
end scanner;
architecture scan1 of scanner is
signal Q1,QA, K, Kd: bit;
alias Q2: bit is C0;                                  -- column outputs will be the same
alias Q3: bit is C1;                                  -- as the state variables because
alias Q4: bit is C2;                                  -- of state assignment
begin
      K <= R0 or R1 or R2 or R3;                                  -- this is the decoder section
      N3 <= (R2 and not C0) or (R3 and not C1);  N2 <= R1 or (R2 and C0);
      N1 <= (R0 and not C0) or (not R2 and C2) or (not R1 and not R0 and C0);
      N0 <= (R1 and C1) or (not R1 and C2) or (not R3 and not R1 and not C1);
      V <= (Q2 and not Q3 and K) or (not Q2 and Q3 and K) or (not Q2 and Q4);
      process(CLK)                                            -- process to update flip-flops
      begin
            if CLK = '1' then
                  Q1 <= (Q1 and Kd) or (Q2 and not Q3 and K) or (not Q2 and Q3 and K)
                        or (not Q2 and Q4);
                  Q2 <= (not Q2 and not Q3) or K or Q4;
                  Q3 <= not Q3 or Q1 or (Q4 and not Kd) or (not Q2 and K);
                  Q4 <= not Q2 or Q1 or (Q3 and not Kd) or (not Q3 and K);
                  QA <= K or (QA and not Q1);                 -- first debounce flip-flop
                  Kd <= QA;                                   -- second debounce flip-flop
            end if;
      end process;
end scan1;
```

Figure 3-27 Interface for Scantest

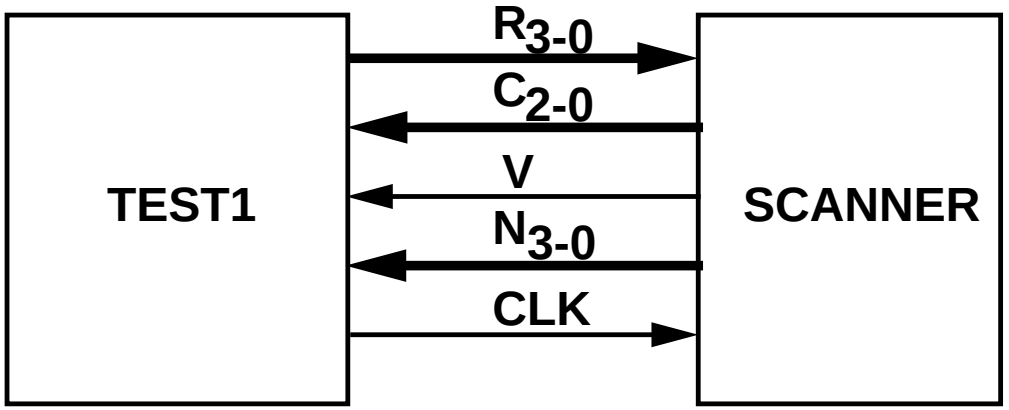


Figure 3-28(a) VHDL for Scantest

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

entity scantest is
end scantest;

architecture test1 of scantest is
component scanner
      port (R0,R1,R2,R3,CLK: in bit;
            C0,C1,C2: inout bit;
            N0,N1,N2,N3,V: out bit);
end component;

type arr is array(0 to 11) of integer;                -- array of keys to test
constant KARRAY:arr := (2,5,8,0,3,6,9,11,1,4,7,10);
signal C0,C1,C2,V,CLK,R0,R1,R2,R3: bit;               -- interface signals
signal N: bit_vector(3 downto 0);
signal KN: integer;                                   -- key number to test
begin
      CLK <= not CLK after 20 ns;                     -- generate clock signal
                                                      -- this section emulates the keypad
      R0 <= '1' when (C0='1' and KN=1) or (C1='1' and KN=2) or (C2='1' and KN=3)
      else '0';
      R1 <= '1' when (C0='1' and KN=4) or (C1='1' and KN=5) or (C2='1' and KN=6)
      else '0';
      R2 <= '1' when (C0='1' and KN=7) or (C1='1' and KN=8) or (C2='1' and KN=9)
      else '0';
      R3 <= '1' when (C0='1' and KN=10) or (C1='1' and KN=0) or (C2='1' and KN=11)
      else '0';
```

Figure 3-28(b) VHDL for Scantest

```
    process                                     -- this section tests scanner
    begin
        for i in 0 to 11 loop                   -- test every number in key array
            KN <= KARRAY(i);                    --  simulates keypress
            wait until (V='1' and rising_edge(CLK));
            assert (vec2int(N) = KN)            --  check if output matches
                report "Numbers don't match"
                severity error;
            KN <= 15;                           -- equivalent to no key pressed
            wait until rising_edge(CLK);        -- wait for scanner to reset
            wait until rising_edge(CLK);
            wait until rising_edge(CLK);
        end loop;
        report "Test complete.";
    end process;
    scanner1: scanner                           -- connect test1 to scanner
        port map(R0,R1,R2,R3,CLK,C0,C1,C2,N(0),N(1),N(2),N(3),V);
end test1;
```