Gregory L. Moss

Lab Solutions Manual

for

Lab Manual: A Design Approach

to accompany

DIGITAL SYSTEMS:

PRINCIPLES AND APPLICATIONS

Eleventh Edition

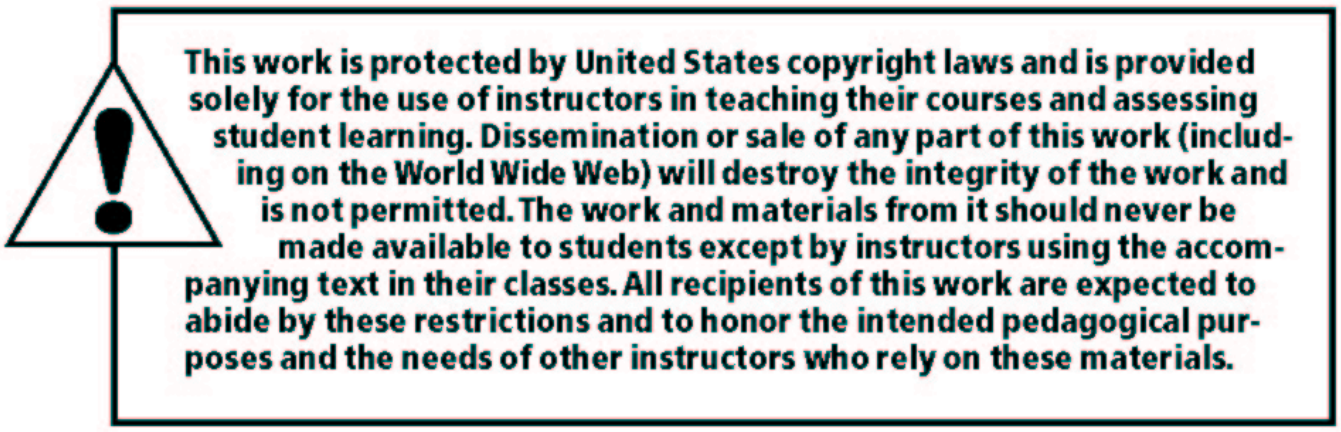
By Ronald J. Tocci, Neal S. Widmer, & Gregory L. Moss

Prentice Hall

Boston Columbus Indianapolis New York San Francisco Upper Saddle River

Amsterdam Cape Town Dubai London Madrid Milan Munich Paris Montreal Toronto

Delhi Mexico City Sao Paulo Sydney Hong Kong Seoul Singapore Taipei Tokyo



10 9 8 7 6 5 4 3 2 1

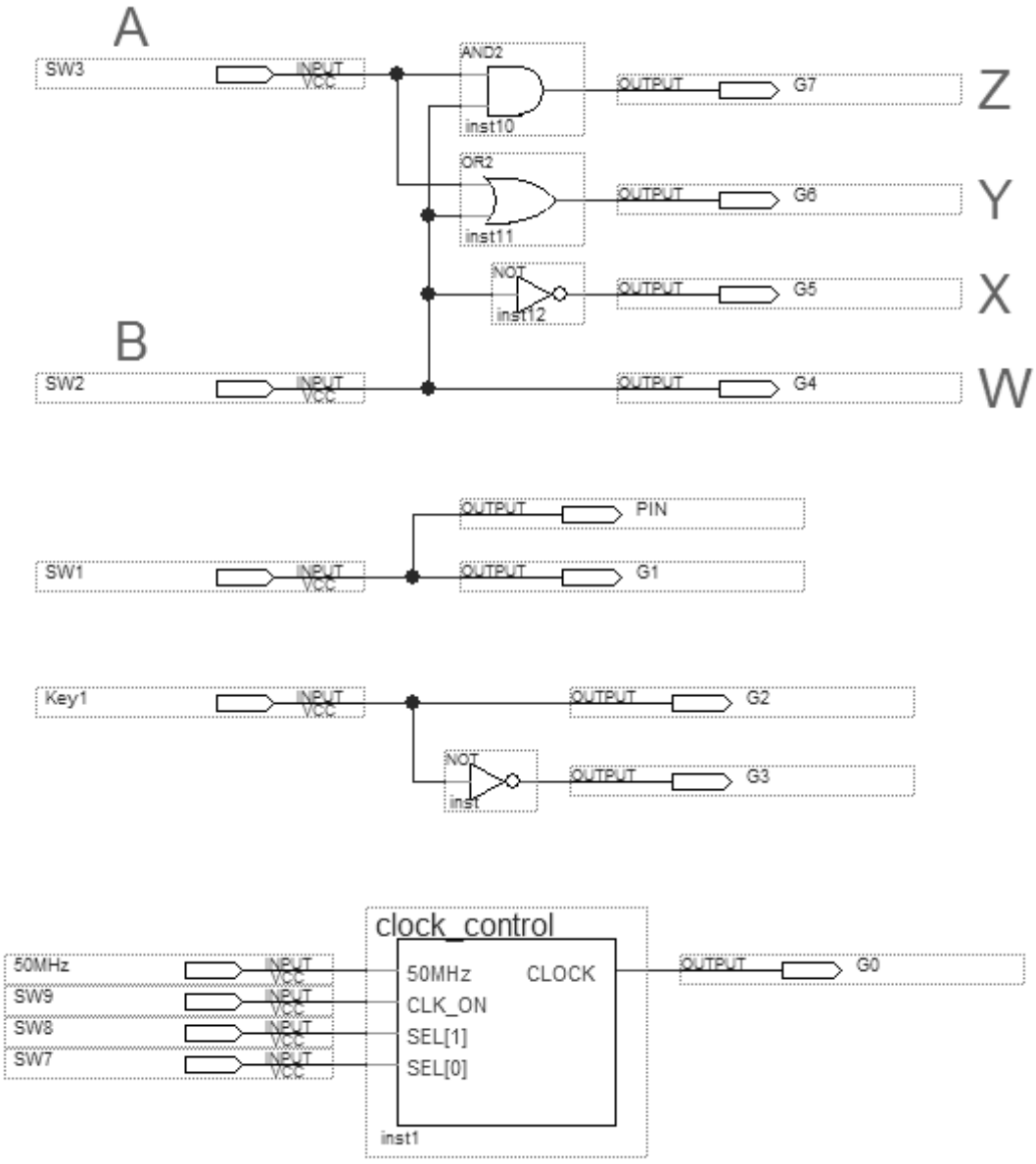
PEARSON

ISBN-13: 978-0-13-512382-9
ISBN-10: 0-13-512382-8

CONTENTS

Unit 1 Introduction to the DE0, DE1, or DE2 Development & Education Board

Project: Intro2DE0, Intro2DE1, or Intro2DE2



1.3 Logic Switches

Board	DE0	DE1	DE2
# switches	10	10	18

Logic Switch SW1	LEDG1 (on/off)	Logic level (high/low)	Voltage at connector pin
Down	Off	Low	~0 V
Up	On	High	~3.3 V

1.4 LEDs

LED label	Color	DE0	DE1	DE2
LEDR	Red	0	10	18
LEDG	Green	10	8	9

1.5 Pushbuttons → Normally High

Board	DE0	DE1	DE2
Pushbutton	3	4	4

Pushbutton #1	LEDG2 (on/off)	LEDG3 (on/off)
Normal	On	Off
Pressed	Off	On

1.6 Clock

CLK_ON (SW9)	1	1	1	1	0
SEL[1..0] (SW8, SW7)	00	01	10	11	XX
$freq_{G0}$	0.5 Hz	5 Hz	25 Hz	50 Hz	0 Hz

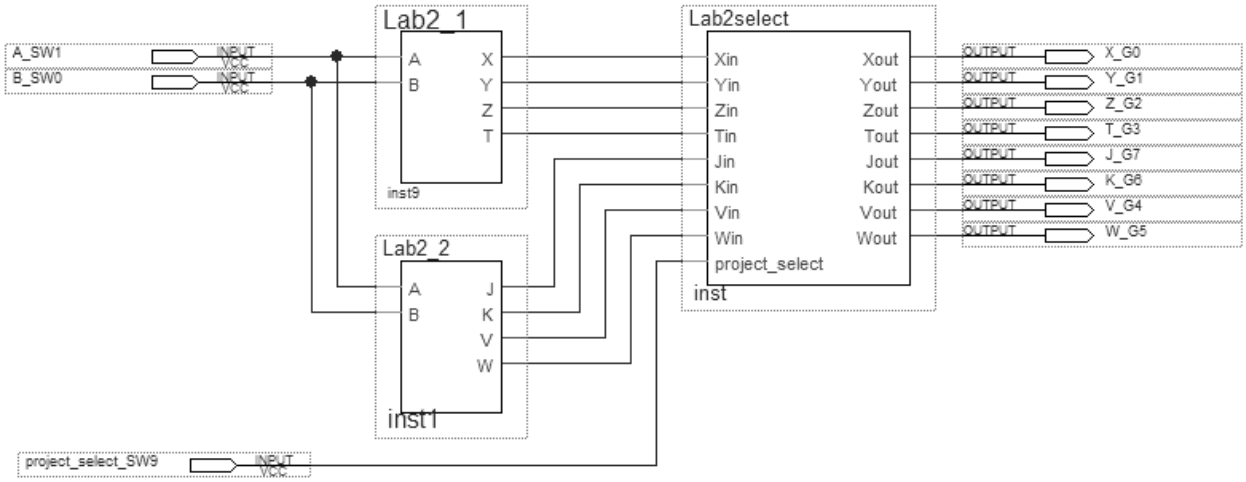
1.7 Simple logic circuits

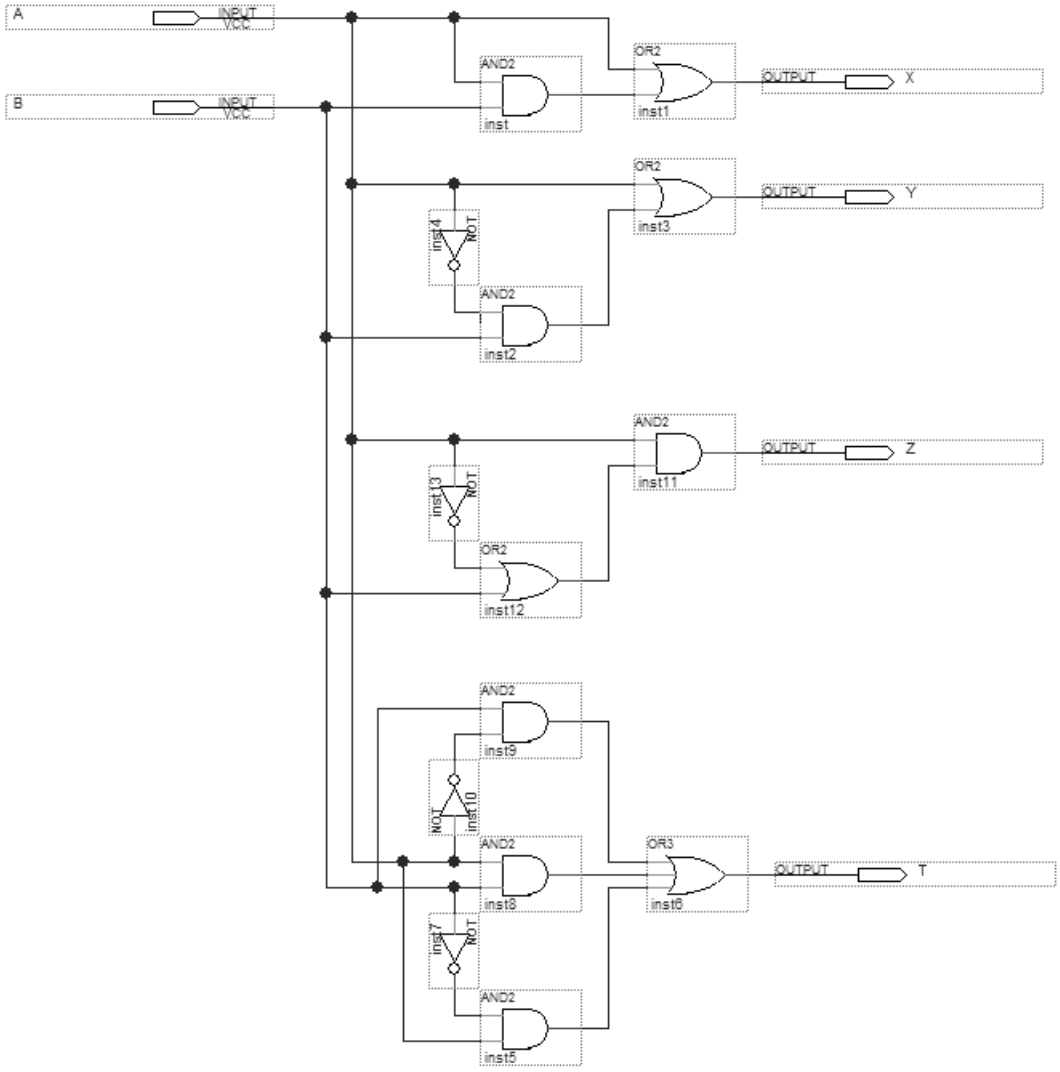
A (SW3)	B (SW2)	W (LEDG4)	X (LEDG5)	Y (LEDG6)	Z (LEDG7)
0	0	0	1	0	0
0	1	1	0	1	0
1	0	0	1	1	0
1	1	1	0	1	1

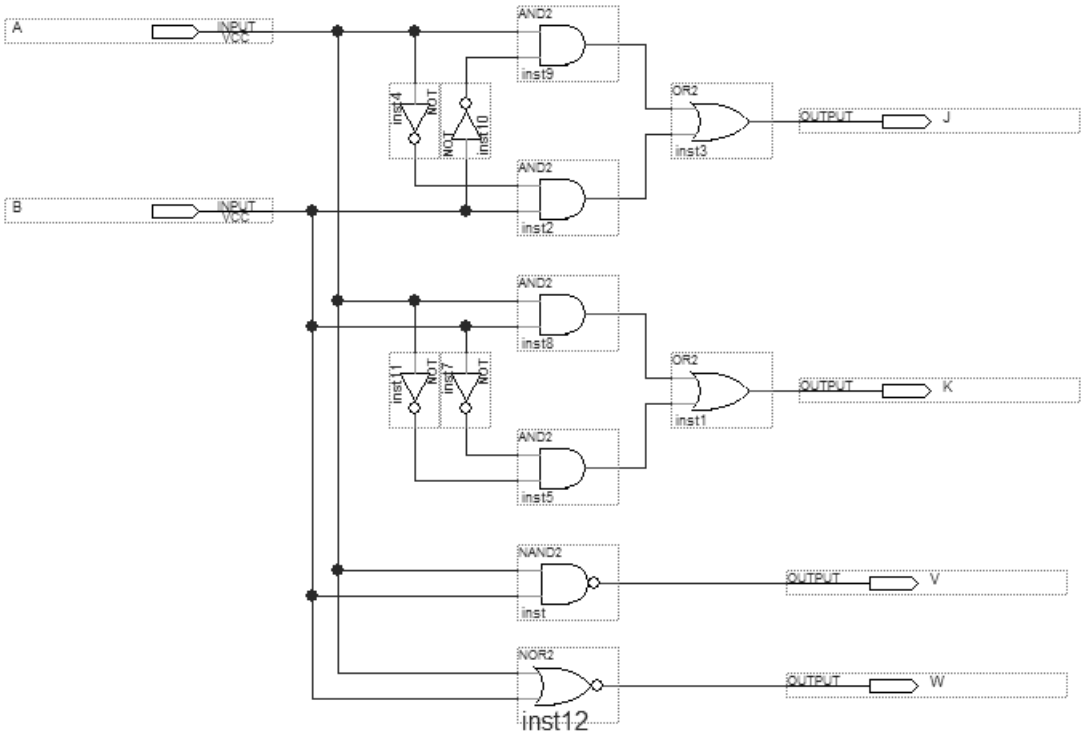
$W = B$ $X = \overline{B}$ $Y = A + B$ $Z = A\ B$

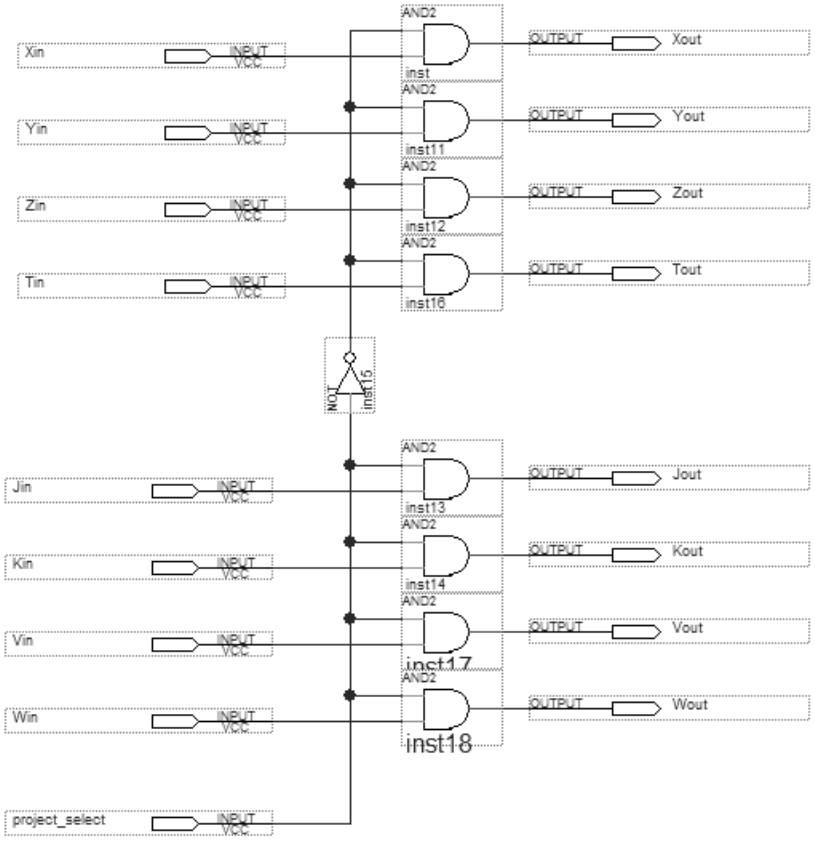
Unit 2 Testing Combinational Logic Circuits Using DE0, DE1, or DE2 Boards

Project: Lab2DE0, Lab2DE1, or Lab2DE2






A
INPUT
VCC
B
INPUT
VCC
AND2
inst9
AND2
inst2
NOT
inst4
NOT
inst10
OR2
inst3
OUTPUT
J
AND2
inst8
AND2
inst5
NOT
inst11
NOT
inst7
OR2
inst1
OUTPUT
K
NAND2
inst
OUTPUT
V
NOR2
inst12
OUTPUT
W


Xin
INPUT
VCC
Yin
INPUT
VCC
Zin
INPUT
VCC
Tin
INPUT
VCC
AND2
inst
AND2
inst11
AND2
inst12
AND2
inst16
OUTPUT
Xout
OUTPUT
Yout
OUTPUT
Zout
OUTPUT
Tout
NOT
inst15
Jin
INPUT
VCC
Kin
INPUT
VCC
Vin
INPUT
VCC
Win
INPUT
VCC
project_select
INPUT
VCC
AND2
inst13
AND2
inst14
AND2
inst17
AND2
inst18
OUTPUT
Jout
OUTPUT
Kout
OUTPUT
Vout
OUTPUT
Wout

2.1 Simple circuits

$X = A + A\,B = A$ $\qquad$ $Y = A + \overline{A}\,B = A + B$

$Z = A\,(\overline{A} + B) = A\,B$ $\qquad$ $T = \overline{A}\,B + A\,B + A\,\overline{B} = A + B$

A	B	T	Z	Y	X	J	K	W	V
0	0	0	0	0	0	0	1	1	1
0	1	1	0	1	0	1	0	0	1
1	0	1	0	1	1	1	0	0	1
1	1	1	1	1	1	0	1	0	0

2.2 More circuit functions

$V = \overline{\overline{A}\,\overline{B}}$ $\qquad$ $W = \overline{\overline{A} + B}$

$J = A\,\overline{B} + \overline{A}\,B = A \oplus B$ $\qquad$ $K = A\,B + \overline{A}\,\overline{B} = \overline{A \oplus B}$

Unit 3 Schematic Capture & Analysis of Combinational Logic Circuits

3.1 Example 3-1 (see Lab Manual Example 3-1 & Quartus Tutorial 1 – Schematic)

3.2 Equivalent circuits

(a) V = W? true

V = A (B + C)

W = A C + A B

(b) X = Y? true

$X = \overline{A}\ \overline{B} + A\ B + A\ C$

$Y = \overline{A}\ \overline{B} + A\ B + \overline{B}\ C$

A	B	C	V	W	X	Y
0	0	0	0	0	1	1
0	0	1	0	0	1	1
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	0	0	0	0
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1

3.3 DeMorgan's theorem

(a) p1 = p2 = p3? true

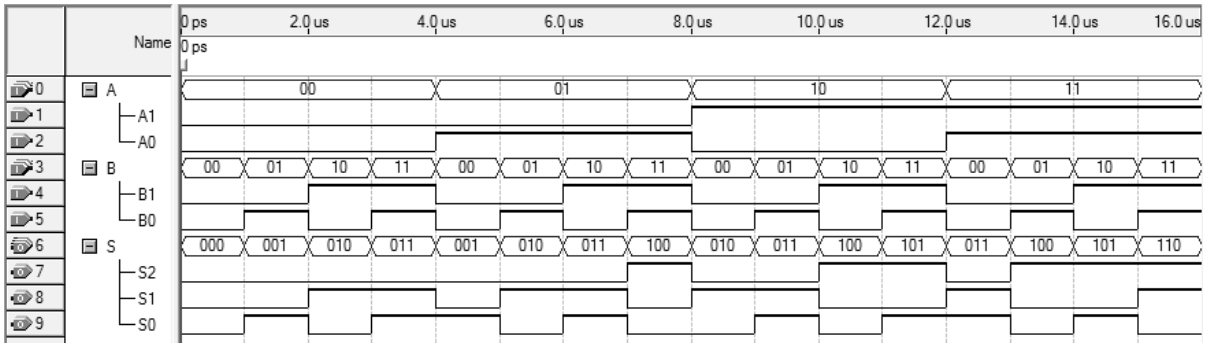
$p1 = \overline{a}\ \overline{b}$ $p2 = \overline{a}\ \overline{b}$ $p3 = \overline{a + b}$

(b) q1 = q2 = q3? true

$q1 = \overline{a} + \overline{b}$ $q2 = \overline{a} + \overline{b}$ $q3 = \overline{a\ b}$

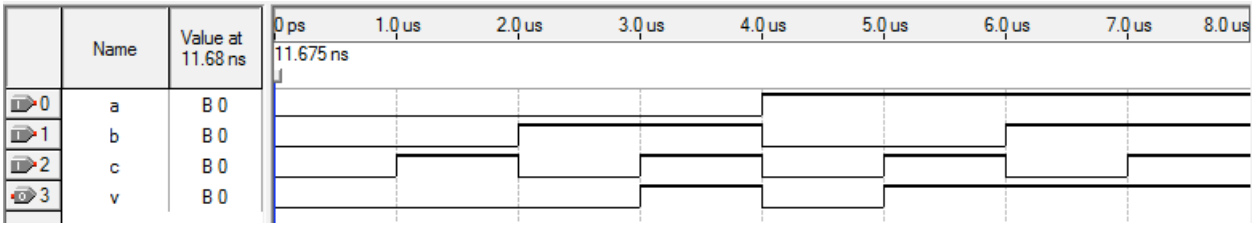
a	b	p	q
0	0	1	1
0	1	0	1
1	0	0	1
1	1	0	0

3.4 2-bit adder



Unit 4 Design and Simulation of Combinational Circuits

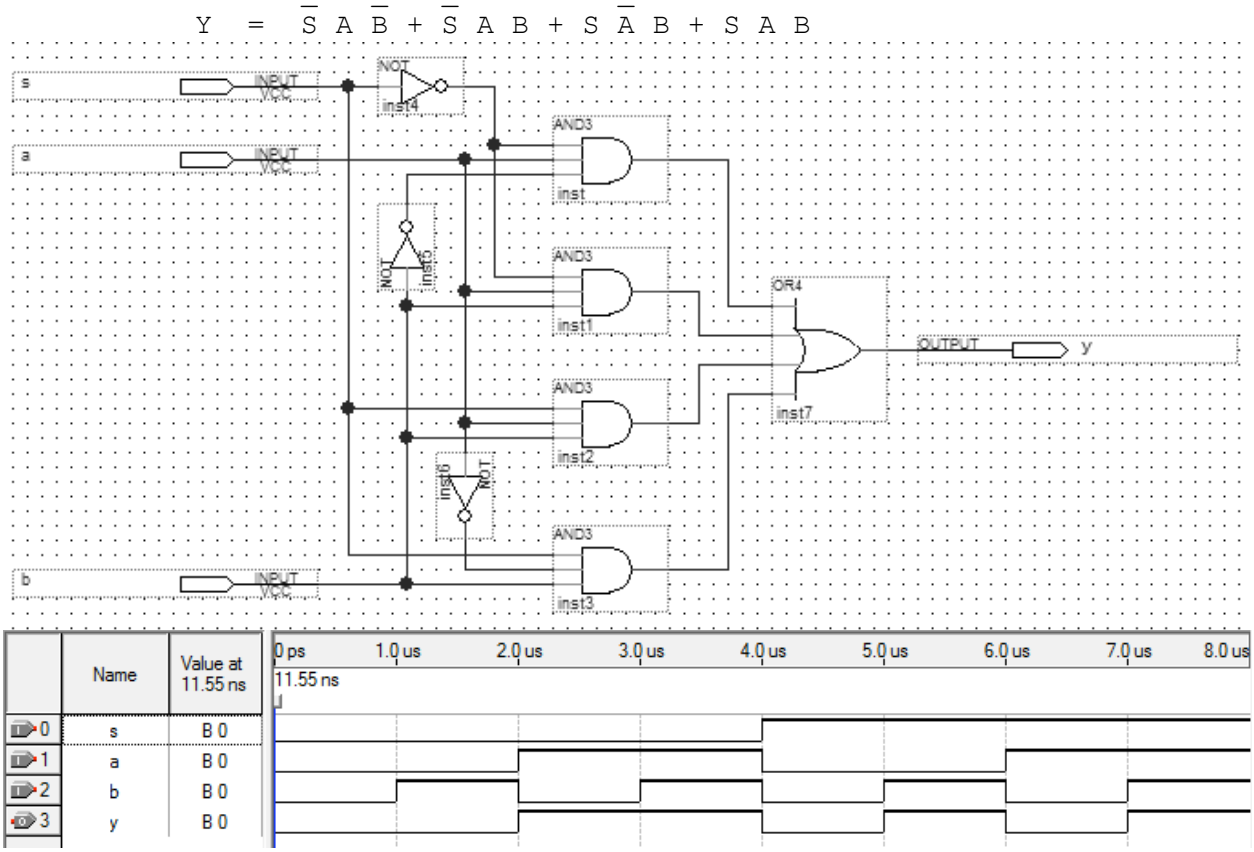
4.1 Majority vote (see Lab Manual Example 4-1 & Quartus Tutorial 2 – Simulation)



4.2 Two-input multiplexer

S	A	B	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$$Y = \overline{S}\,A\,\overline{B} + \overline{S}\,A\,B + S\,\overline{A}\,B + S\,A\,B$$

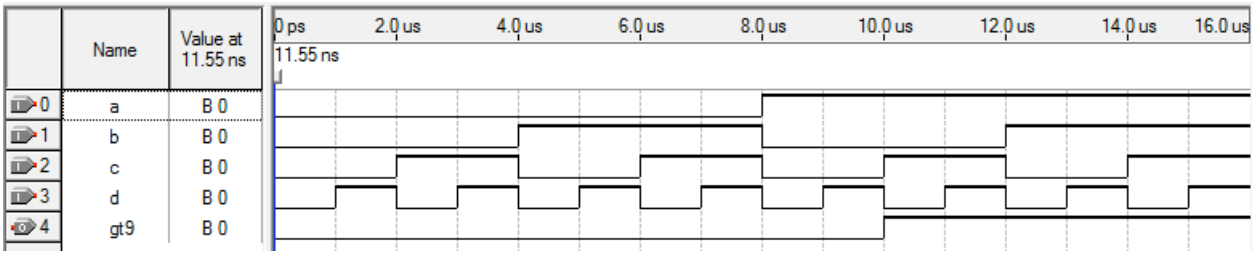


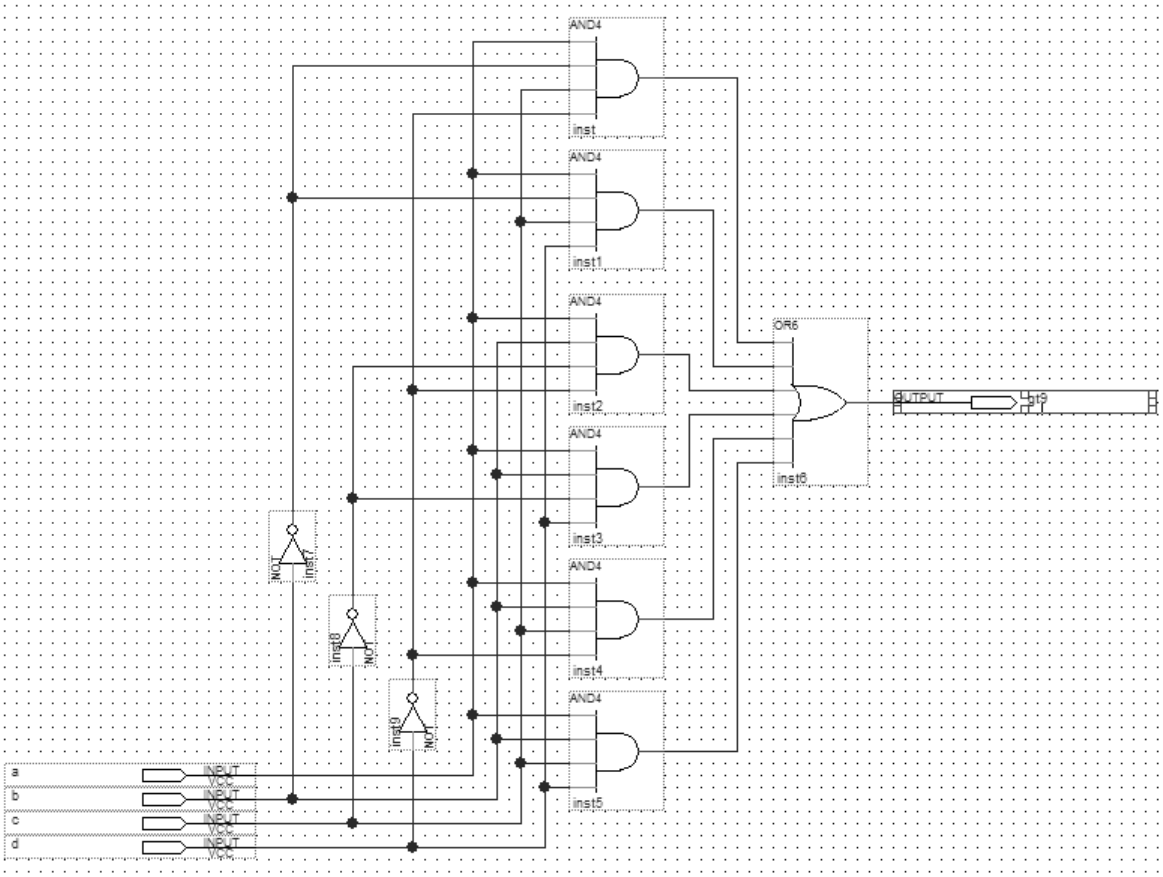
4.3 Non-BCD input

A	B	C	D	GT9
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

$$GT9 = A\,\overline{B}\,C\,\overline{D} + A\,\overline{B}\,C\,D + A\,B\,\overline{C}\,\overline{D} + A\,B\,\overline{C}\,D$$

$$+ A\,B\,C\,\overline{D} + A\,B\,C\,D$$

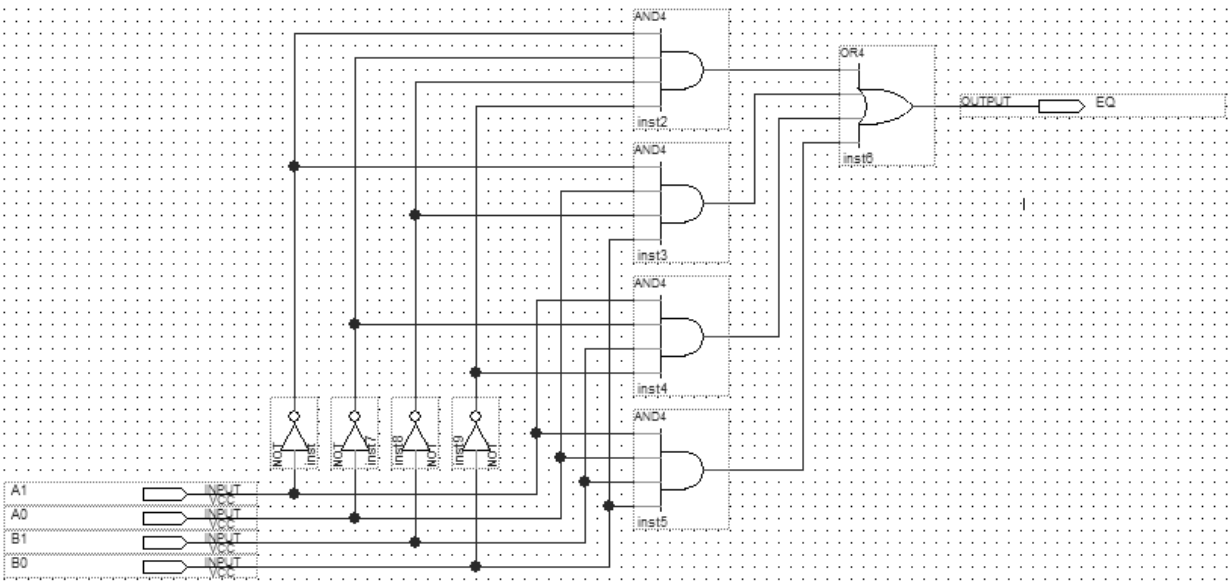




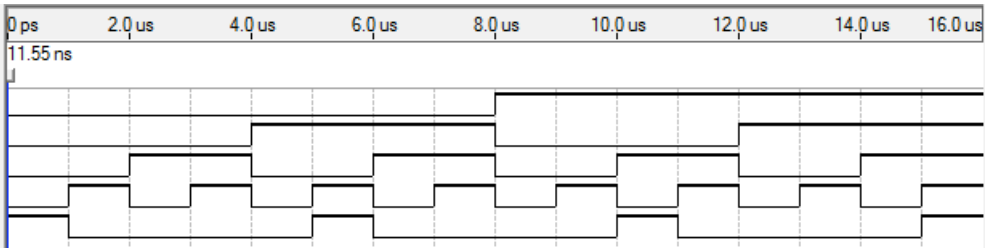
4.4 Two-bit comparator

A1	A0	B1	B0	EQ
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

$$EQ = \overline{A1}\ \overline{A0}\ \overline{B1}\ \overline{B0} + \overline{A1}\ A0\ \overline{B1}\ B0 + A1\ \overline{A0}\ B1\ \overline{B0} + A1\ A0\ B1\ B0$$



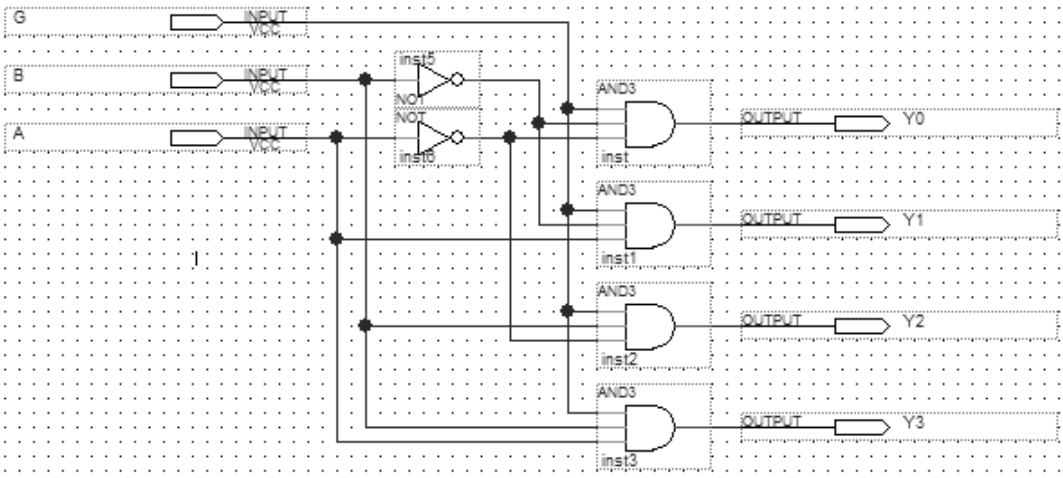
	Name	Value at 11.55 ns
0	A1	B 0
1	A0	B 0
2	B1	B 0
3	B0	B 0
4	EQ	B 1

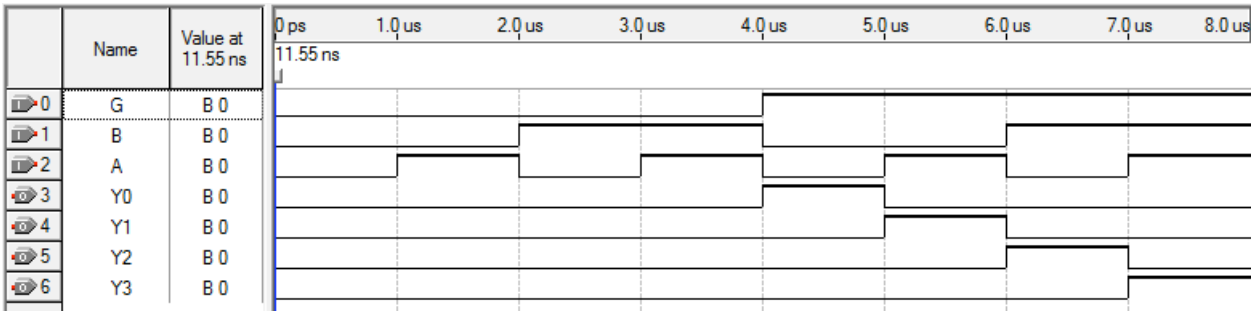


4.5 Binary number detector

$Y0 = G\,\overline{B}\,\overline{A}$ $\quad$ $Y1 = G\,\overline{B}\,A$

$Y2 = G\,B\,\overline{A}$ $\quad$ $Y3 = G\,B\,A$



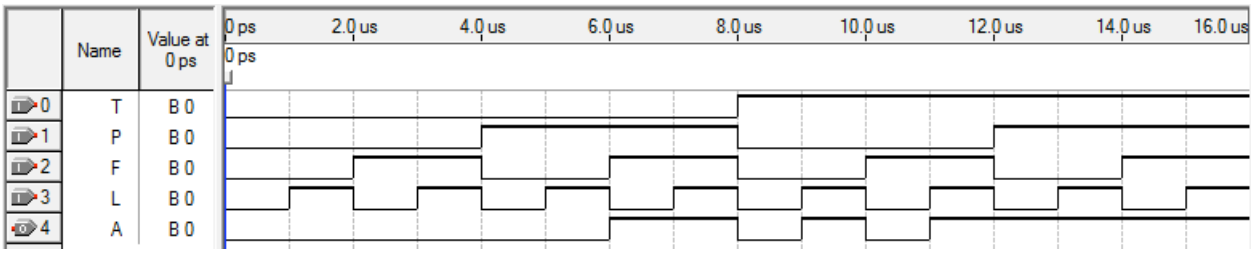


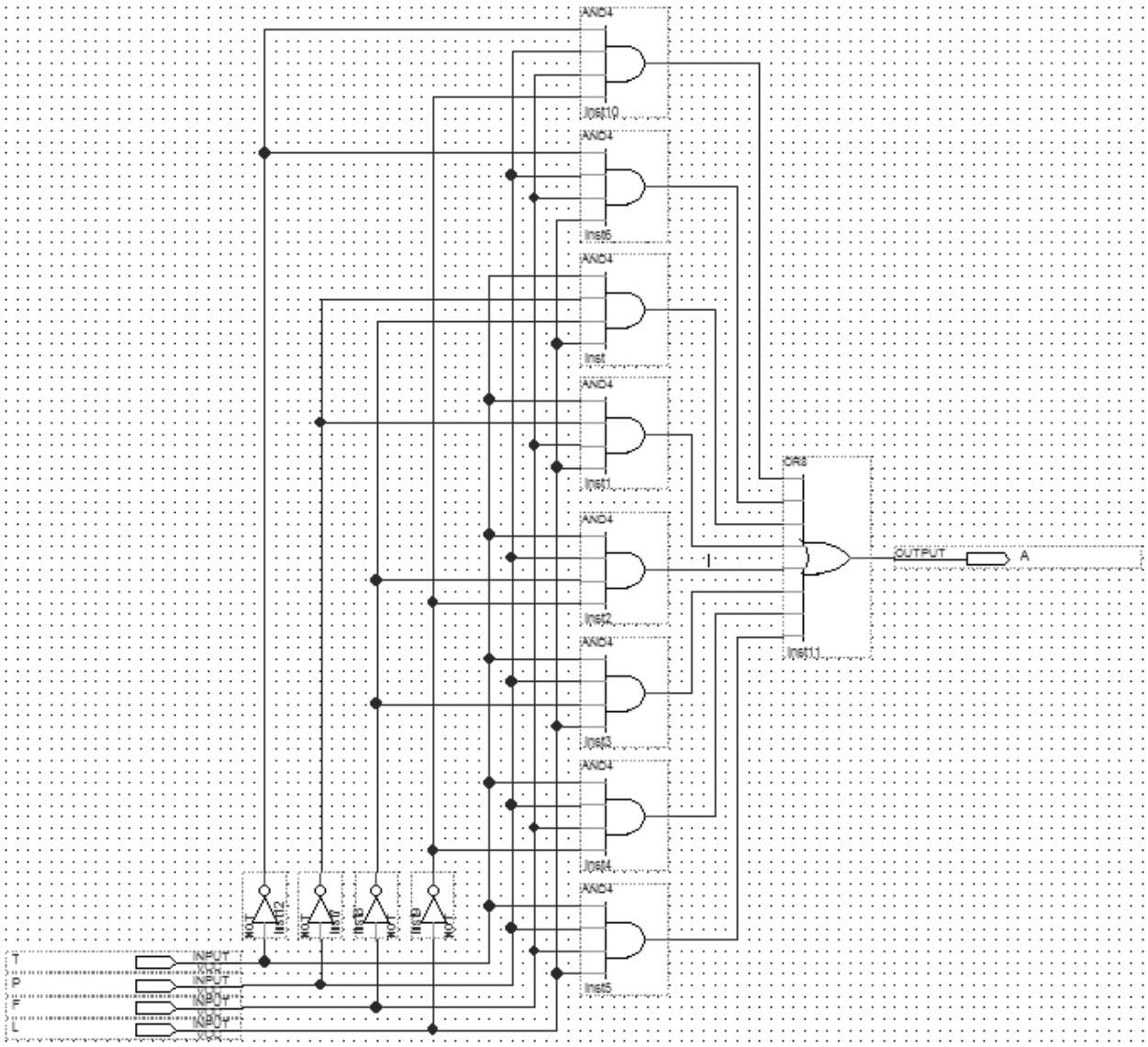
4.6 Alarm circuit

T	P	F	L	A
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

$$A = \overline{T}\,P\,F\,\overline{L} + \overline{T}\,P\,F\,L + T\,\overline{P}\,\overline{F}\,L + T\,\overline{P}\,F\,L$$

$$+ \; T\,P\,\overline{F}\,\overline{L} + T\,P\,\overline{F}\,L + T\,P\,F\,\overline{L} + T\,P\,F\,L$$

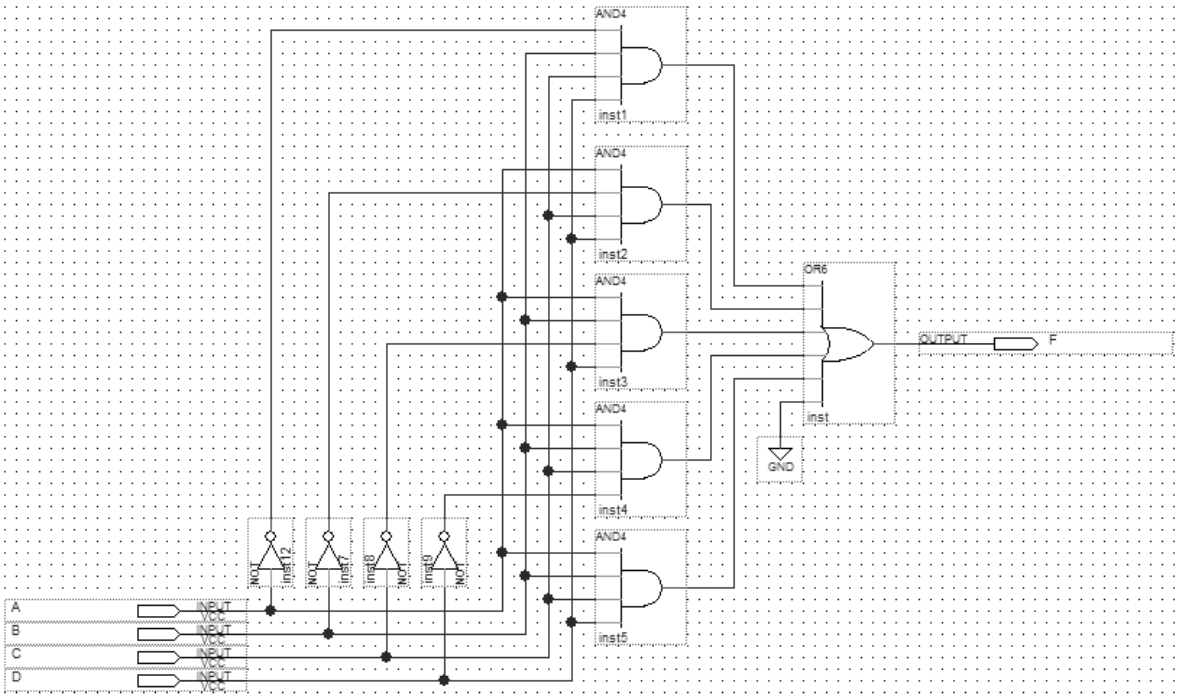


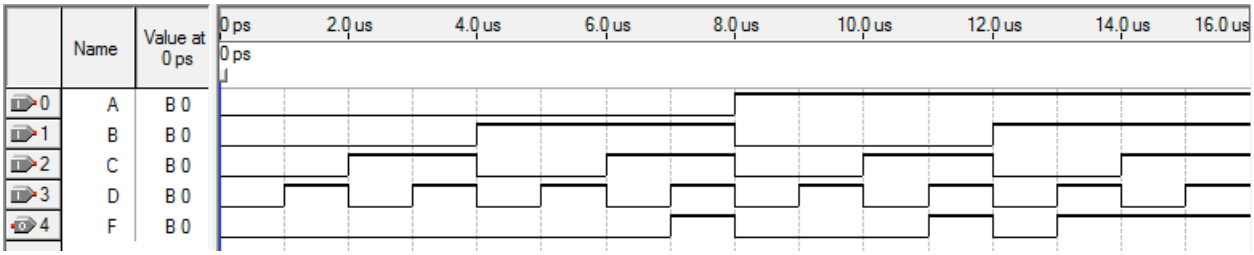

AND4
I06\10
AND4
Inst6
AND4
Inst
AND4
I\61
AND4
Inst2
AND4
I06\3
AND4
I06\4
AND4
I\65
OR6
Inst11
OUTPUT
A
INPUT
INPUT
INPUT
INPUT
T
P
F
L

4.7 Elevator control

A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

$$F = \overline{A}\,B\,C\,D + A\,\overline{B}\,C\,D + A\,B\,\overline{C}\,D + A\,B\,C\,\overline{D} + A\,B\,C\,D$$

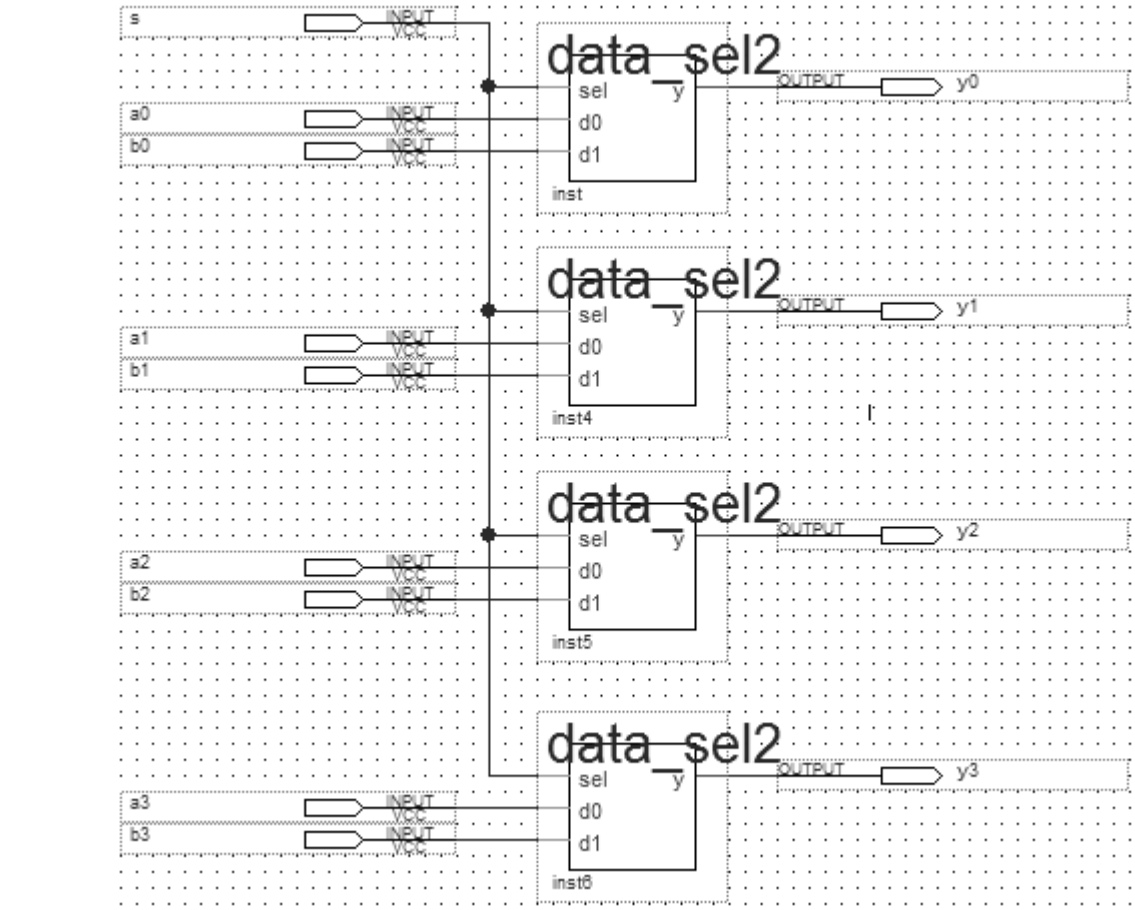


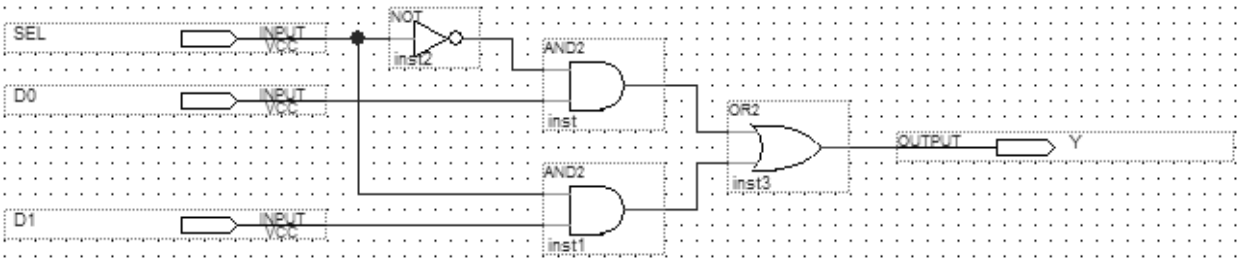


Unit 5 Creating Hierarchical Logic Circuits

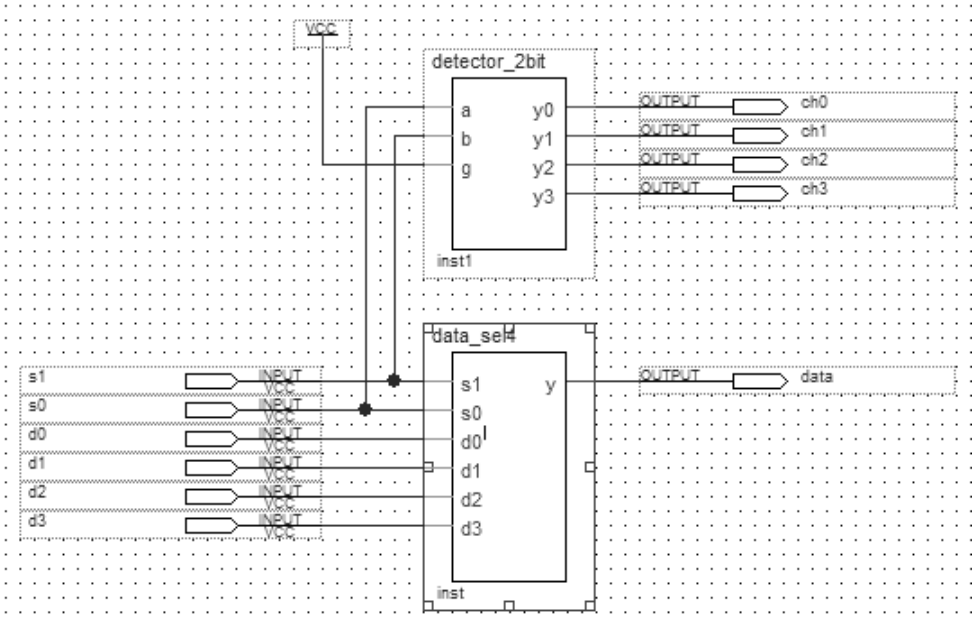
5.1 1-out-of-4 data selector (see Lab Manual Example 5-1 & Quartus Tutorial 3 – Hierarchical)

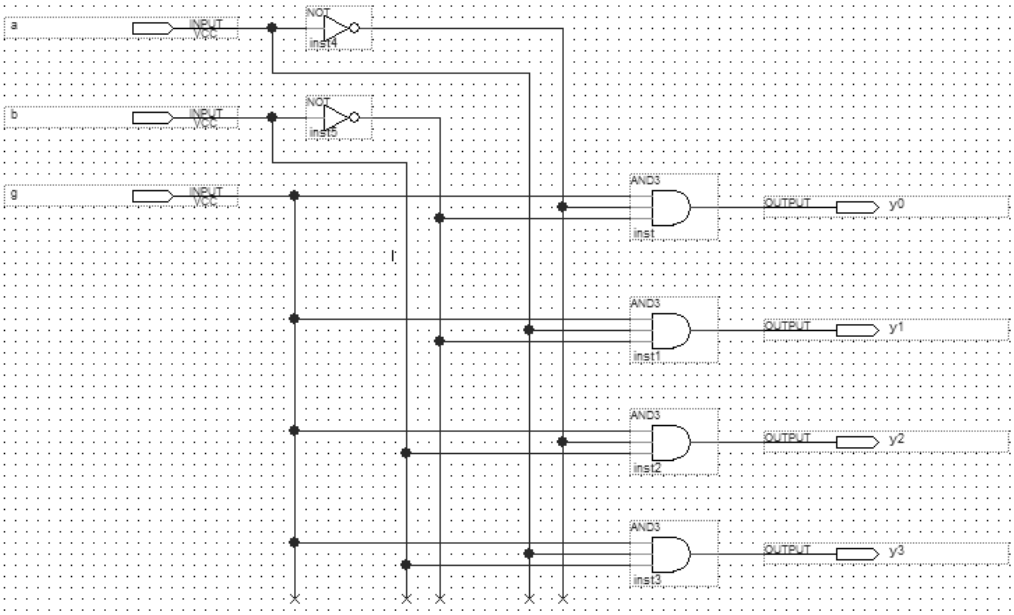
5.2 2-channel, 4-bit multiplexer

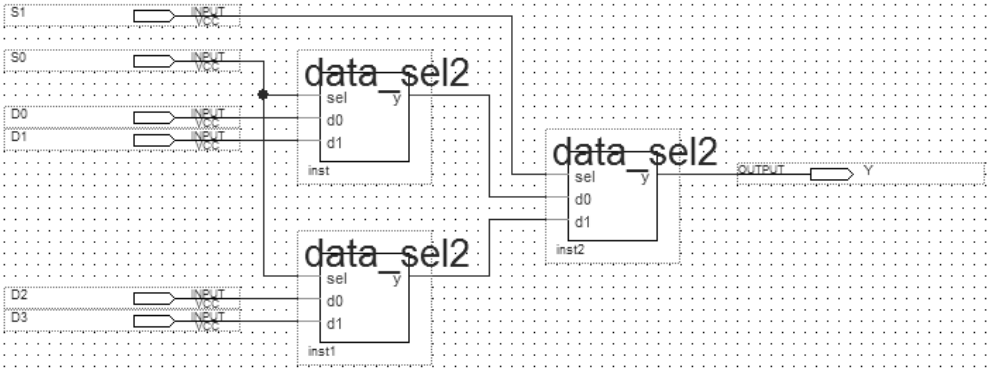




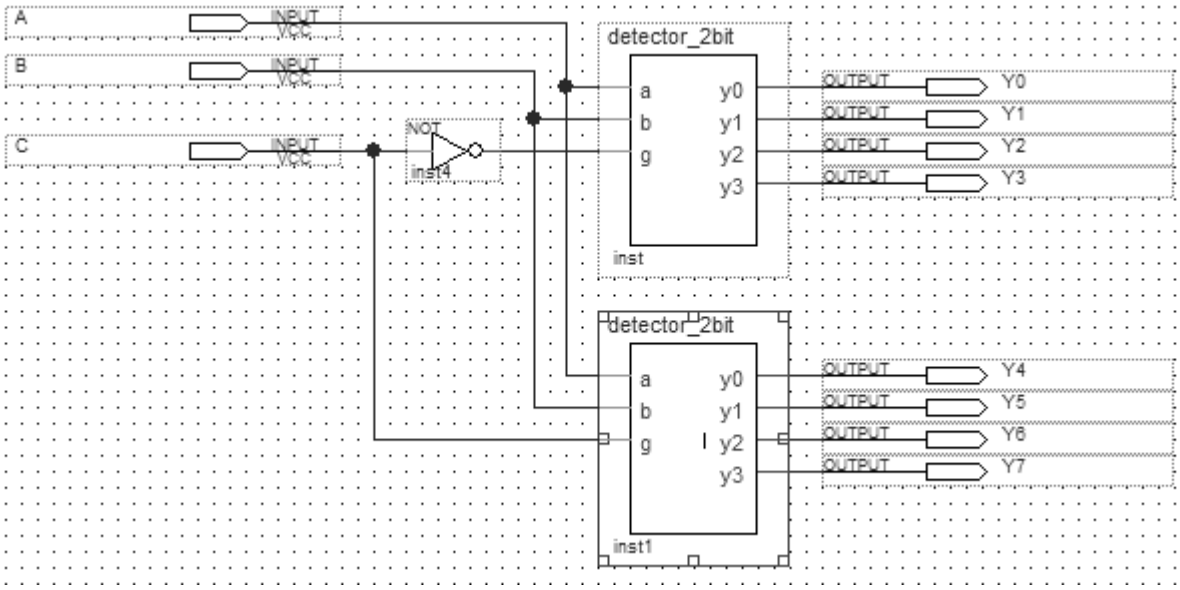
5.3 Multiplexer & decoder

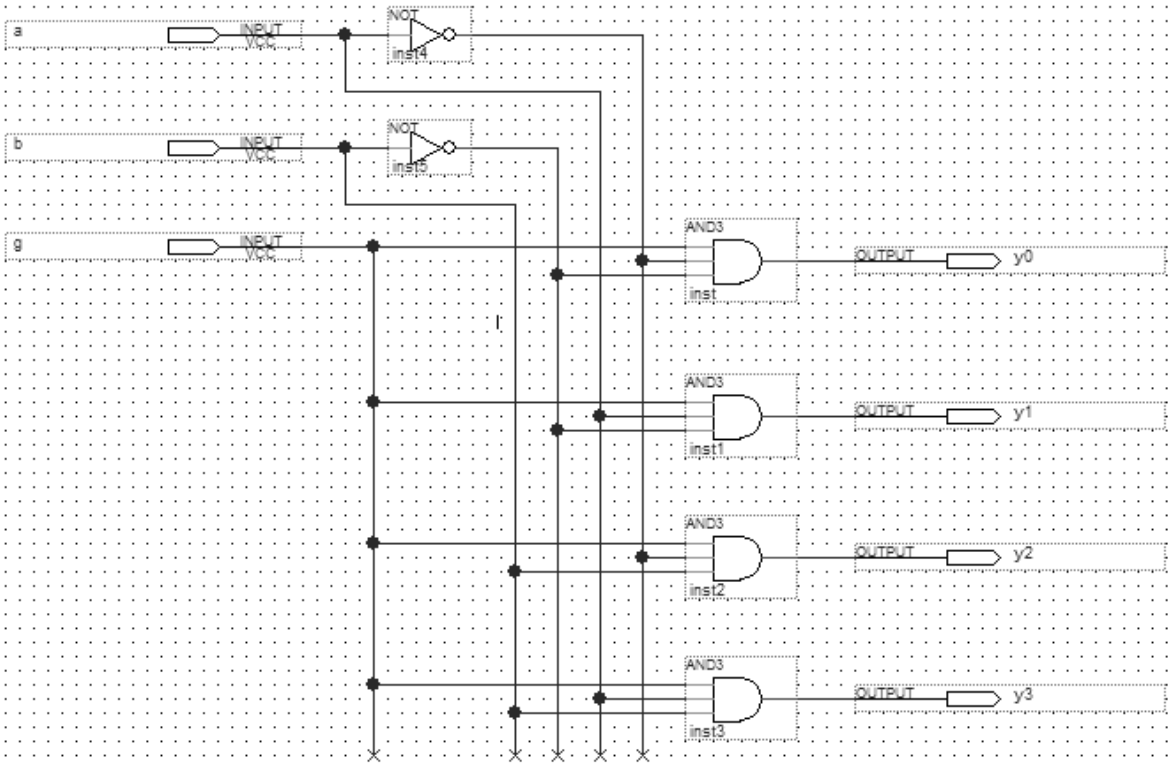




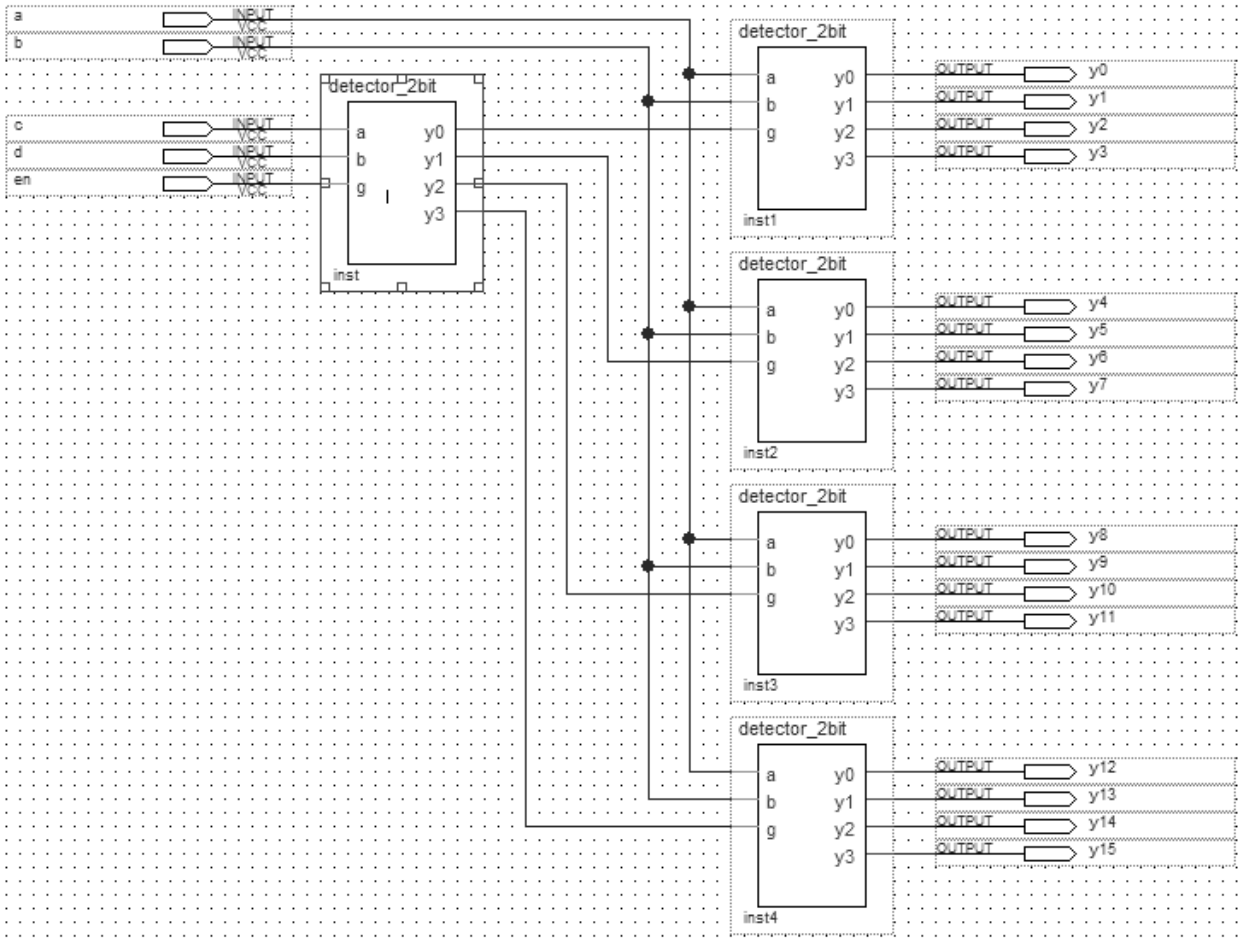


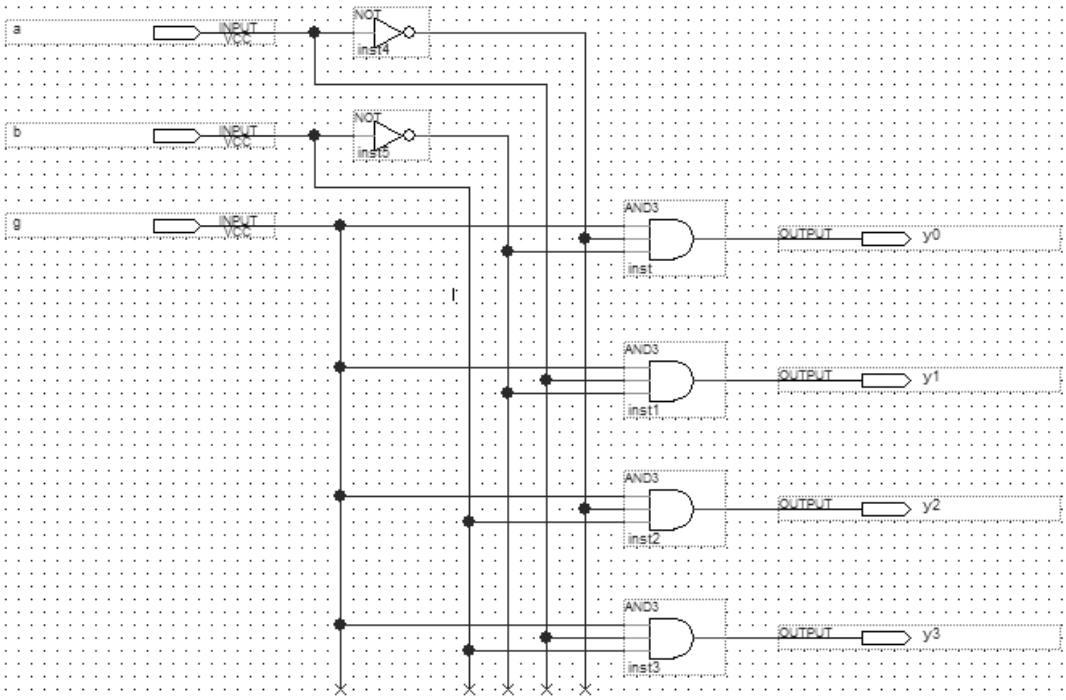
5.4 3-bit decoder



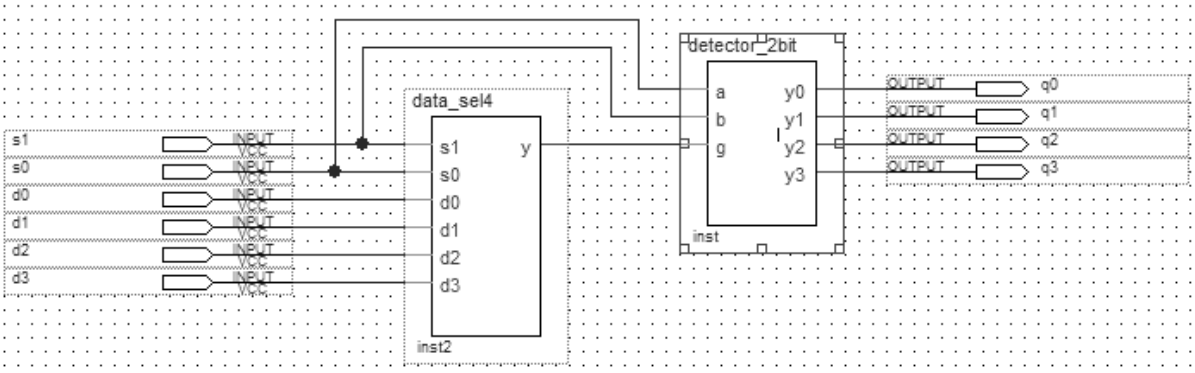


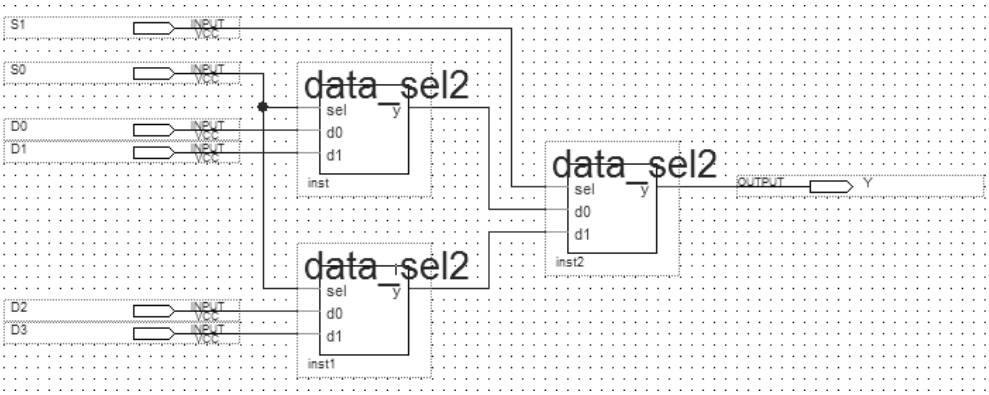
5.5 4-bit decoder

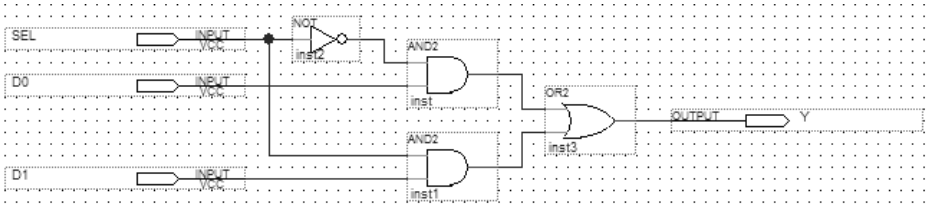


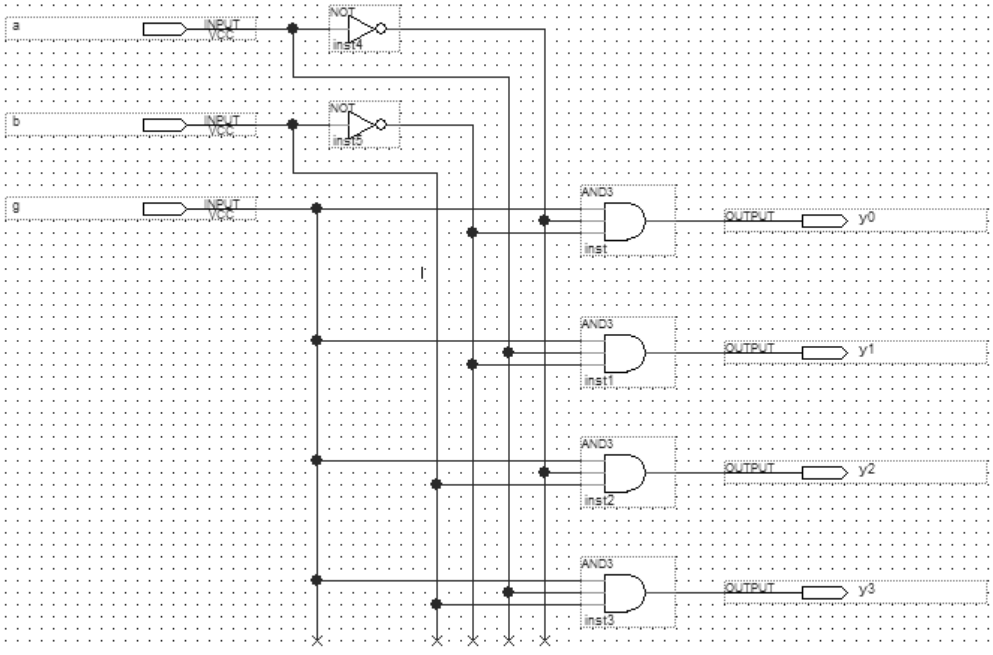


5.6 Data transmission

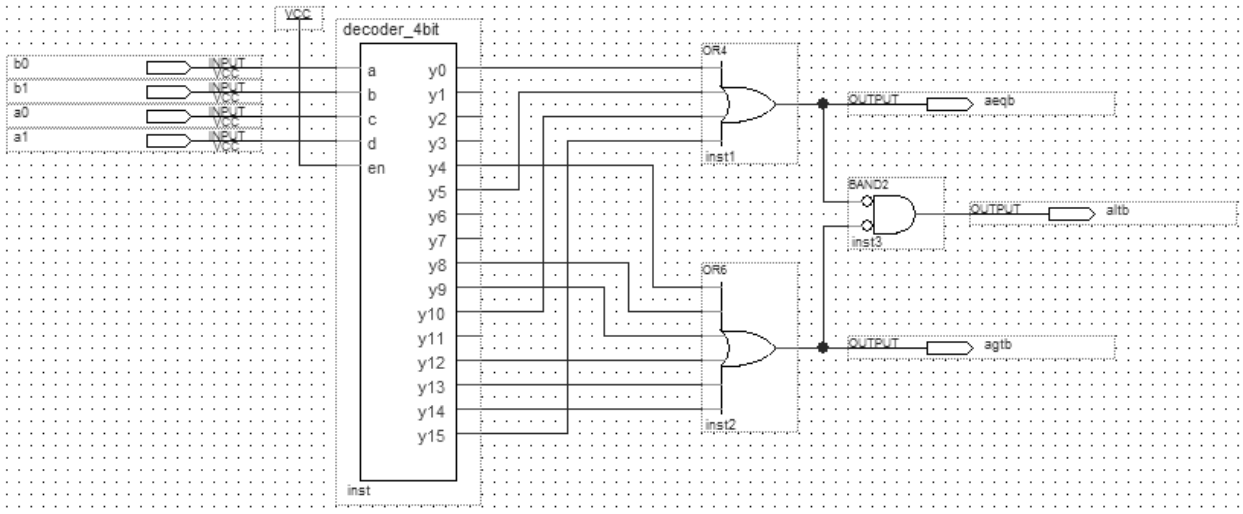


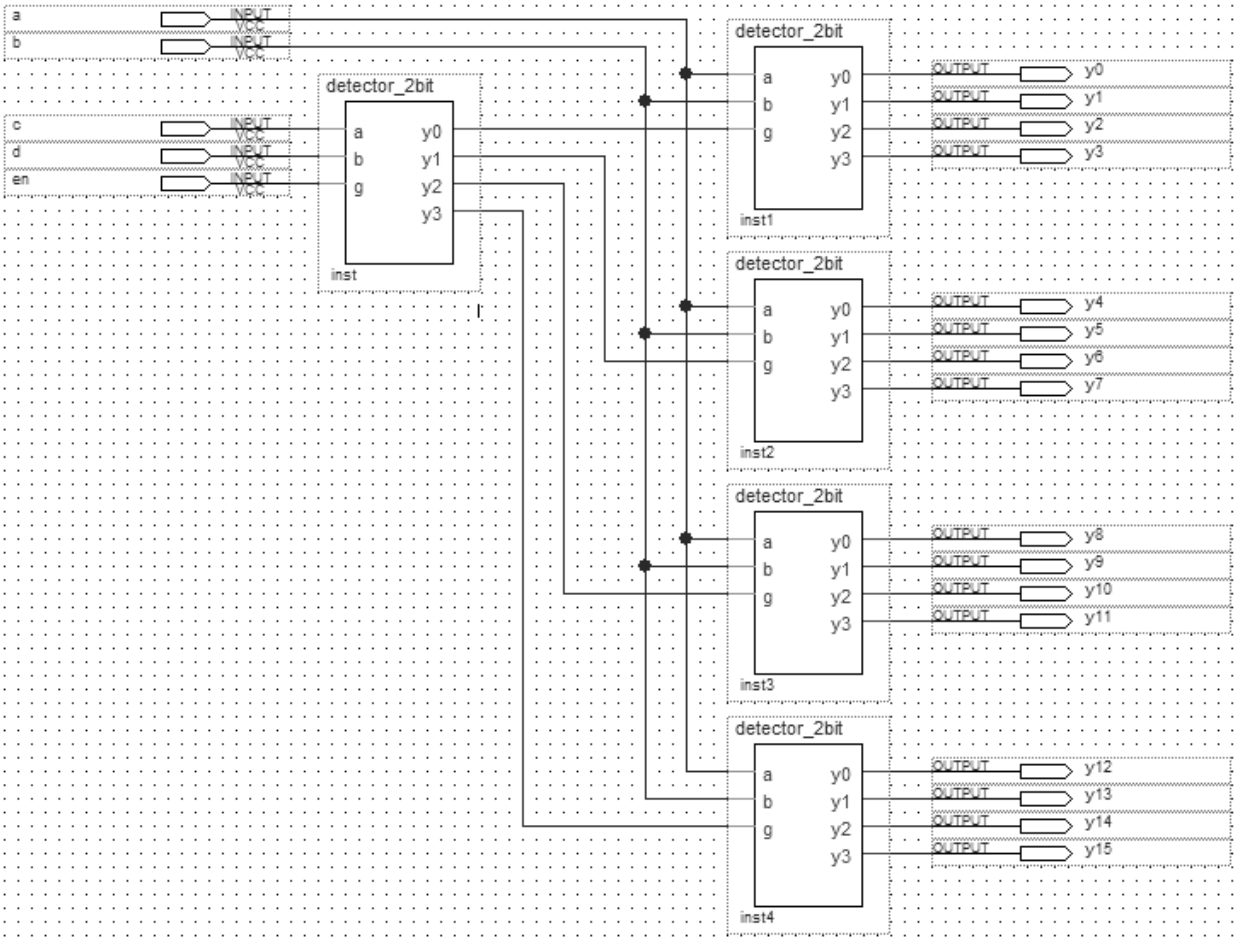






5.7 2-bit magnitude comparator





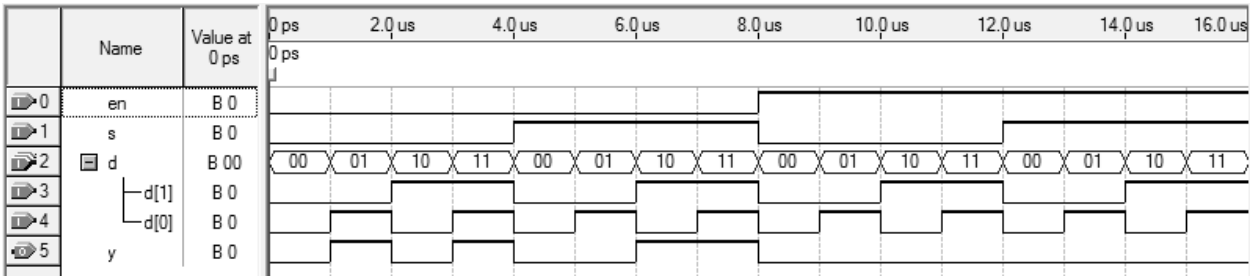
Unit 6A Combinational Circuit Design with AHDL

6A.1 2-bit comparator (see Lab Manual Example 6-1 & Quartus Tutorial 4 – HDL)

6A.2 Code converter (see Lab Manual Example 6-2 & Quartus Tutorial 4 – HDL)

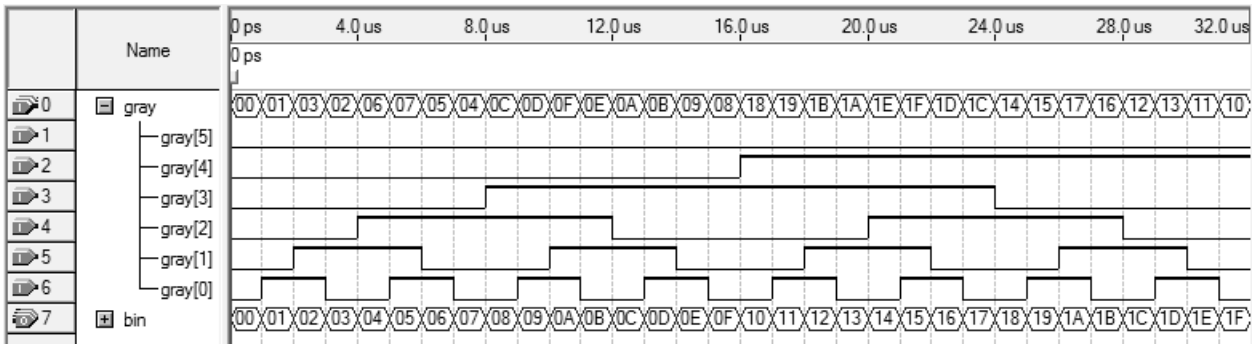
6A.3 Modified data selector

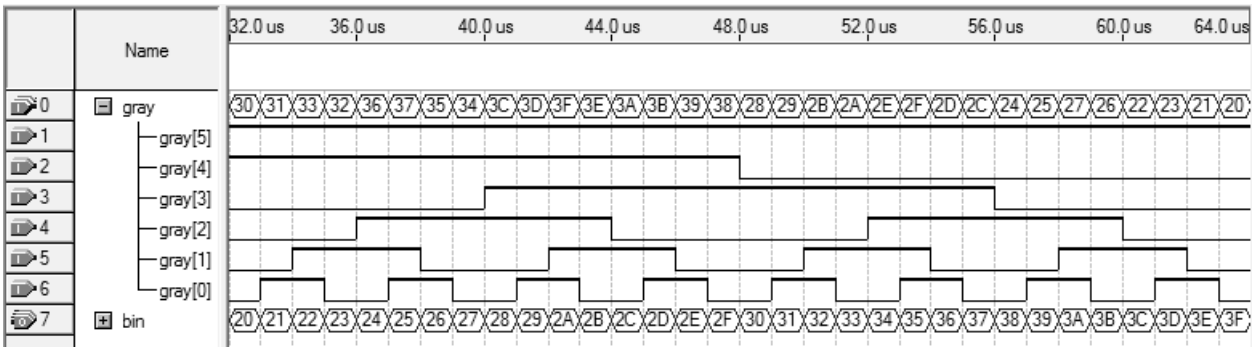
```
SUBDESIGN multiplexer       -- project 6A.3
(
      en, d[1..0], s              :INPUT;
      y                           :OUTPUT;
)
BEGIN
                        -- CASE nested inside IF statement
      IF !en THEN       -- evaluate CASE when en is low
            CASE s IS
                  WHEN 0      =>    y = d0;
                  WHEN 1      =>    y = d1;
            END CASE;
      ELSE                        y = GND;   -- disabled
      END IF;
END;
```



6A.4 Gray-code-to-binary conversion

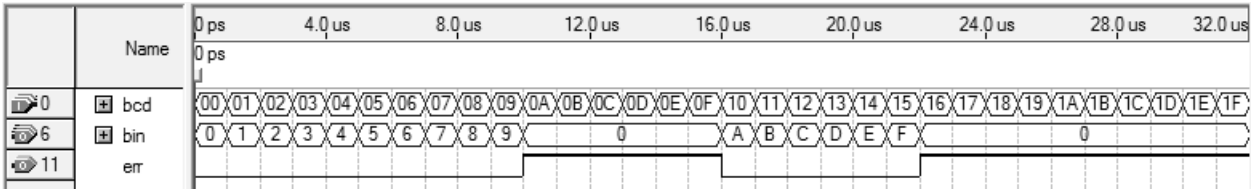
```
SUBDESIGN gray
(
      gray[5..0]       :INPUT;
      bin[5..0]        :OUTPUT;
)
BEGIN
      bin5 = gray5;                -- MSBs are the same
      bin4 = bin5 $ gray4;         -- XOR prev out $ next in
      bin3 = bin4 $ gray3;
      bin2 = bin3 $ gray2;
      bin1 = bin2 $ gray1;
      bin0 = bin1 $ gray0;
END;
```





6A.5 BCD-to-binary converter

```
SUBDESIGN BCD2bin
(
      bcd[4..0]               :INPUT;
      bin[3..0], err          :OUTPUT;
)
BEGIN
      CASE bcd[] IS
            WHEN H"00"        =>    bin[] = H"0"; err = GND;
            WHEN H"01"        =>    bin[] = H"1"; err = GND;
            WHEN H"02"        =>    bin[] = H"2"; err = GND;
            WHEN H"03"        =>    bin[] = H"3"; err = GND;
            WHEN H"04"        =>    bin[] = H"4"; err = GND;
            WHEN H"05"        =>    bin[] = H"5"; err = GND;
            WHEN H"06"        =>    bin[] = H"6"; err = GND;
            WHEN H"07"        =>    bin[] = H"7"; err = GND;
            WHEN H"08"        =>    bin[] = H"8"; err = GND;
            WHEN H"09"        =>    bin[] = H"9"; err = GND;
            WHEN H"10"        =>    bin[] = H"A"; err = GND;
            WHEN H"11"        =>    bin[] = H"B"; err = GND;
            WHEN H"12"        =>    bin[] = H"C"; err = GND;
            WHEN H"13"        =>    bin[] = H"D"; err = GND;
            WHEN H"14"        =>    bin[] = H"E"; err = GND;
            WHEN H"15"        =>    bin[] = H"F"; err = GND;
            WHEN OTHERS       =>    bin[] = H"0"; err = VCC;
      END CASE;
END;
```



6A.6 Binary-to-BCD converter

```
SUBDESIGN bin2BCD
(
      bin[4..0]        :INPUT;
      bcd[5..0]        :OUTPUT;
)
BEGIN
      TABLE            -- use hex to define each BCD digit
            bin[]      =>   bcd[];
            B"00000"   =>   H"00";
            B"00001"   =>   H"01";
            B"00010"   =>   H"02";
            B"00011"   =>   H"03";
            B"00100"   =>   H"04";
            B"00101"   =>   H"05";
            B"00110"   =>   H"06";
            B"00111"   =>   H"07";
            B"01000"   =>   H"08";
            B"01001"   =>   H"09";
            B"01010"   =>   H"10";
            B"01011"   =>   H"11";
            B"01100"   =>   H"12";
            B"01101"   =>   H"13";
            B"01110"   =>   H"14";
            B"01111"   =>   H"15";
            B"10000"   =>   H"16";
            B"10001"   =>   H"17";
            B"10010"   =>   H"18";
            B"10011"   =>   H"19";
            B"10100"   =>   H"20";
            B"10101"   =>   H"21";
            B"10110"   =>   H"22";
            B"10111"   =>   H"23";
            B"11000"   =>   H"24";
            B"11001"   =>   H"25";
            B"11010"   =>   H"26";
            B"11011"   =>   H"27";
            B"11100"   =>   H"28";
            B"11101"   =>   H"29";
            B"11110"   =>   H"30";
            B"11111"   =>   H"31";
      END TABLE;
END;
```

6A.7 Tens digit detector

```
SUBDESIGN decade
(
      num[5..0]         :INPUT;
      tens[6..0]        :OUTPUT;
)
BEGIN
            -- 1st true condition determines output
      IF     num[] <=  9   THEN   tens[] = B"0000001";
      ELSIF  num[] <= 19   THEN   tens[] = B"0000010";
      ELSIF  num[] <= 29   THEN   tens[] = B"0000100";
      ELSIF  num[] <= 39   THEN   tens[] = B"0001000";
      ELSIF  num[] <= 49   THEN   tens[] = B"0010000";
      ELSIF  num[] <= 59   THEN   tens[] = B"0100000";
      ELSE                        tens[] = B"1000000";
      END IF;
END;
```

6A.8 Lamp display

```
SUBDESIGN lamps
(
      in[2..0]          :INPUT;
      out[7..1]         :OUTPUT;
)
BEGIN
            -- test every in[] condition
      CASE in[]  IS
            WHEN 0      =>    out[] = B"0000000";
            WHEN 1      =>    out[] = B"0000001";
            WHEN 2      =>    out[] = B"0000011";
            WHEN 3      =>    out[] = B"0000111";
            WHEN 4      =>    out[] = B"0001111";
            WHEN 5      =>    out[] = B"0011111";
            WHEN 6      =>    out[] = B"0111111";
            WHEN 7      =>    out[] = B"1111111";
      END CASE;
END;
```

6A.9 Programmable logic unit

```
SUBDESIGN logic_unit
(     a[3..0], b[3..0], s[1..0], en          :INPUT;
      f[3..0]                                :OUTPUT;   )
BEGIN
            -- control input selects function
      IF     s[] == 0 & en  THEN   f[] = a[] # b[];
      ELSIF  s[] == 1 & en  THEN   f[] = a[] & b[];
      ELSIF  s[] == 2 & en  THEN   f[] = a[] $ b[];
      ELSIF  s[] == 3 & en  THEN   f[] = !a[];
      ELSE  f[] = B"0000";       -- output disabled
      END IF;
END;
```

6A.10 Number range detector

```
SUBDESIGN range
(     num[4..0], en    :INPUT;
      range[4..1]      :OUTPUT;   )
BEGIN
            -- determine if num[] in ranges when enabled
            -- active-hi outs
      range1 =   num[] >=  4 & num[] <= 12 & en;
      range2 =   num[] >= 15 & num[] <= 20 & en;
      range3 =   num[] >= 18 & num[] <= 24 & en;
            -- active-low out
      range4 = !(num[] >= 26 & num[] <= 30 & en);
END;
```

6A.11 Data switcher

```
SUBDESIGN switcher
(     en, in[1..0], s[1..0]       :INPUT;
      out[1..0]                   :OUTPUT;   )
BEGIN
      IF !en     THEN       -- detect active-low enable
            CASE s[]  IS     -- determine control input
                  WHEN  0  =>  out1 = in1; out0 = in0;
                  WHEN  1  =>  out1 = in0; out0 = in0;
                  WHEN  2  =>  out1 = in1; out0 = in1;
                  WHEN  3  =>  out1 = in0; out0 = in1;
            END CASE;
      ELSE  out[] = B"00";  -- disabled
      END IF;
END;
```

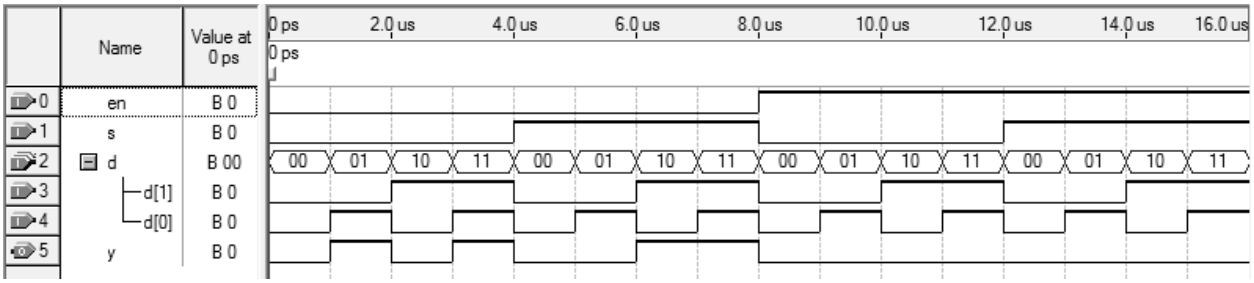
Unit 6V Combinational Circuit Design with VHDL

6V.1 2-bit comparator (see Lab Manual Example 6-1 & Quartus Tutorial 4 – HDL)

6V.2 Code converter (see Lab Manual Example 6-2 & Quartus Tutorial 4 – HDL)

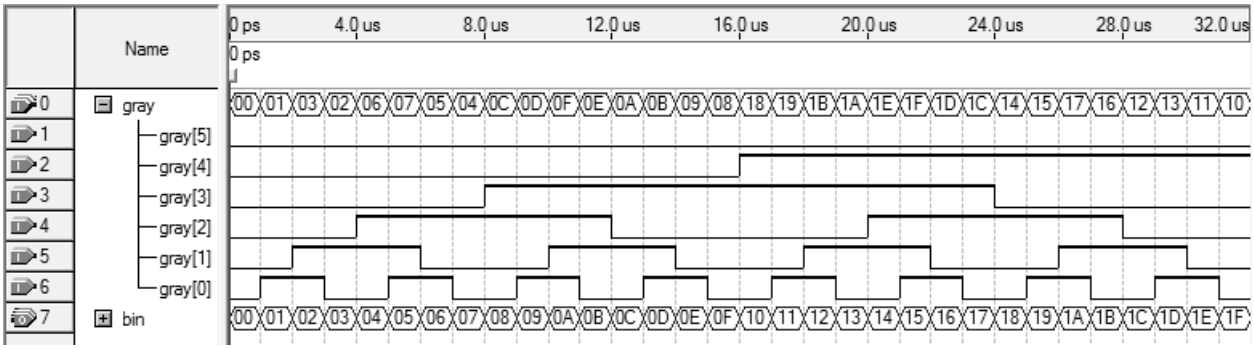
6V.3 Modified data selector

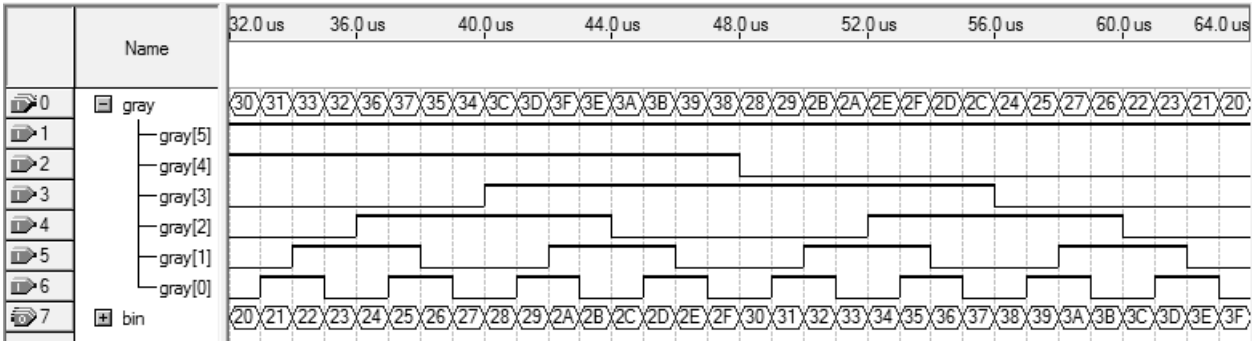
```
ENTITY multiplexer IS
PORT ( en        :IN BIT;
       d         :IN BIT_VECTOR (1 DOWNTO 0);
       s         :IN BIT;
       y         :OUT BIT );
END multiplexer;

ARCHITECTURE modify OF multiplexer IS
BEGIN
     PROCESS (en, d, s)    -- invoke on changes
     BEGIN
          IF (en = '0') THEN    -- en low?
               CASE s IS        -- nested CASE
                    WHEN '0'   =>    y <= d(0);
                    WHEN '1'   =>    y <= d(1);
               END CASE;
          ELSE  -- otherwise disabled output
                                   y <= '0';
          END IF;
     END PROCESS;
END modify;
```



6V.4 Gray-code-to-binary conversion

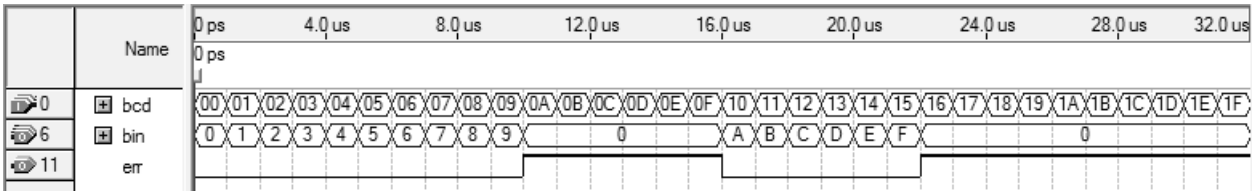
```
ENTITY gray IS
PORT (     gray       :IN BIT_VECTOR (5 DOWNTO 0);
           bin        :OUT BIT_VECTOR (5 DOWNTO 0)     );
END gray;
ARCHITECTURE converter OF gray IS
SIGNAL temp           :BIT_VECTOR (5 DOWNTO 0);
BEGIN
      temp(5) <= gray(5);         -- MSBs are the same
      temp(4) <= temp(5) XOR gray(4);
      temp(3) <= temp(4) XOR gray(3);
      temp(2) <= temp(3) XOR gray(2);
      temp(1) <= temp(2) XOR gray(1);
      temp(0) <= temp(1) XOR gray(0);
      bin <= temp;                -- assign temp to output
END converter;
```





6V.5 BCD-to-binary converter

```
ENTITY bcd2bin IS
PORT (     bcd        :IN BIT_VECTOR (4 DOWNTO 0);
           bin        :OUT BIT_VECTOR (3 DOWNTO 0);
           err        :OUT BIT        );
END bcd2bin;
ARCHITECTURE a OF bcd2bin IS
BEGIN
   PROCESS (bcd)            -- bcd change invokes process
   BEGIN
        CASE bcd IS         -- test all bcd conditions
           WHEN "00000" =>    bin <= "0000";    err <= '0';
           WHEN "00001" =>    bin <= "0001";    err <= '0';
           WHEN "00010" =>    bin <= "0010";    err <= '0';
           WHEN "00011" =>    bin <= "0011";    err <= '0';
           WHEN "00100" =>    bin <= "0100";    err <= '0';
           WHEN "00101" =>    bin <= "0101";    err <= '0';
           WHEN "00110" =>    bin <= "0110";    err <= '0';
           WHEN "00111" =>    bin <= "0111";    err <= '0';
           WHEN "01000" =>    bin <= "1000";    err <= '0';
           WHEN "01001" =>    bin <= "1001";    err <= '0';
           WHEN "10000" =>    bin <= "1010";    err <= '0';
           WHEN "10001" =>    bin <= "1011";    err <= '0';
           WHEN "10010" =>    bin <= "1100";    err <= '0';
           WHEN "10011" =>    bin <= "1101";    err <= '0';
           WHEN "10100" =>    bin <= "1110";    err <= '0';
           WHEN "10101" =>    bin <= "1111";    err <= '0';
           WHEN OTHERS  =>    bin <= "0000";    err <= '1';
        END CASE;
   END PROCESS;
END a;
```



6V.6 Binary-to-BCD converter

```
ENTITY bin2bcd IS
PORT (     bin        :IN BIT_VECTOR (4 DOWNTO 0);
           bcd        :OUT BIT_VECTOR (5 DOWNTO 0)     );
END bin2bcd;
ARCHITECTURE a OF bin2bcd IS
BEGIN
      PROCESS (bin)         -- bin change invokes process
      BEGIN
            CASE bin IS     -- check all input values
                  WHEN "00000" => bcd <= "000000";
                  WHEN "00001" => bcd <= "000001";
                  WHEN "00010" => bcd <= "000010";
                  WHEN "00011" => bcd <= "000011";
                  WHEN "00100" => bcd <= "000100";
                  WHEN "00101" => bcd <= "000101";
                  WHEN "00110" => bcd <= "000110";
                  WHEN "00111" => bcd <= "000111";
                  WHEN "01000" => bcd <= "001000";
                  WHEN "01001" => bcd <= "001001";
                  WHEN "01010" => bcd <= "010000";
                  WHEN "01011" => bcd <= "010001";
                  WHEN "01100" => bcd <= "010010";
                  WHEN "01101" => bcd <= "010011";
                  WHEN "01110" => bcd <= "010100";
                  WHEN "01111" => bcd <= "010101";
                  WHEN "10000" => bcd <= "010110";
                  WHEN "10001" => bcd <= "010111";
                  WHEN "10010" => bcd <= "011000";
                  WHEN "10011" => bcd <= "011001";
                  WHEN "10100" => bcd <= "100000";
                  WHEN "10101" => bcd <= "100001";
                  WHEN "10110" => bcd <= "100010";
                  WHEN "10111" => bcd <= "100011";
                  WHEN "11000" => bcd <= "100100";
                  WHEN "11001" => bcd <= "100101";
                  WHEN "11010" => bcd <= "100110";
                  WHEN "11011" => bcd <= "100111";
                  WHEN "11100" => bcd <= "101000";
                  WHEN "11101" => bcd <= "101001";
                  WHEN "11110" => bcd <= "110000";
                  WHEN "11111" => bcd <= "110001";
            END CASE;
      END PROCESS;
END a;
```

6V.7 Tens digit detector

```
ENTITY decade IS
PORT (     num        :IN INTEGER RANGE 0 TO 63;
           tens       :OUT BIT_VECTOR (6 DOWNTO 0)     );
END decade;

ARCHITECTURE b OF decade IS
BEGIN
      PROCESS (num)          -- num invokes process
      BEGIN -- 1st true condition determines output
            IF     num <=  9  THEN   tens <= B"0000001";
            ELSIF  num <= 19  THEN   tens <= B"0000010";
            ELSIF  num <= 29  THEN   tens <= B"0000100";
            ELSIF  num <= 39  THEN   tens <= B"0001000";
            ELSIF  num <= 49  THEN   tens <= B"0010000";
            ELSIF  num <= 59  THEN   tens <= B"0100000";
            ELSE                     tens <= B"1000000";
            END IF;
      END PROCESS;
END b;
```

6V.8 Lamp display

```
ENTITY lamps IS
PORT (     num        :IN INTEGER RANGE 0 TO 7;
           lites      :OUT BIT_VECTOR (7 DOWNTO 1)     );
END lamps;

ARCHITECTURE cases OF lamps IS
BEGIN
      PROCESS (num)          -- num invokes process
      BEGIN
            CASE num IS      -- test every num condition
                  WHEN 0     =>    lites <= "0000000";
                  WHEN 1     =>    lites <= "0000001";
                  WHEN 2     =>    lites <= "0000011";
                  WHEN 3     =>    lites <= "0000111";
                  WHEN 4     =>    lites <= "0001111";
                  WHEN 5     =>    lites <= "0011111";
                  WHEN 6     =>    lites <= "0111111";
                  WHEN 7     =>    lites <= "1111111";
            END CASE;
      END PROCESS;
END cases;
```

6V.9 Programmable logic unit

```
ENTITY logic_unit IS
PORT (     a, b       :IN BIT_VECTOR (3 DOWNTO 0);
           s          :IN BIT_VECTOR (1 DOWNTO 0);
           en         :IN BIT;
           f          :OUT BIT_VECTOR (3 DOWNTO 0)     );
END logic_unit;

ARCHITECTURE program OF logic_unit IS
BEGIN
      PROCESS (a, b, s, en)       -- any input change
      BEGIN            -- control input selects function
           IF          s = "00" AND en = '1'
                            THEN       f <= a OR b;
           ELSIF       s = "01" AND en = '1'
                            THEN       f <= a AND b;
           ELSIF       s = "10" AND en = '1'
                            THEN       f <= a XOR b;
           ELSIF       s = "11" AND en = '1'
                            THEN       f <= NOT a;
           ELSE        f <= "0000";     -- disabled
           END IF;
      END PROCESS;
END program;
```

6V.10 Number range detector

```
ENTITY val_range IS
PORT (     en          :IN BIT;
           num         :IN INTEGER RANGE 0 TO 31;
           values      :OUT BIT_VECTOR (4 DOWNTO 1)     );
END val_range;

ARCHITECTURE proj OF val_range IS
BEGIN
      PROCESS (en, num)
      BEGIN
            IF en = '1' THEN                 -- enabled
                  IF    (num >=  4 AND num <= 12)
                              THEN values <= "1001";
                  ELSIF (num >= 15 AND num <= 20)
                              THEN values <= "1010";
                  ELSIF (num >= 21 AND num <= 24)
                              THEN values <= "1100";
                  ELSIF (num >= 26 AND num <= 30)
                              THEN values <= "0000";
                  ELSE             values <= "1000";
                  END IF;    -- determine num
            ELSE       values <= "1000";     -- disabled
            END IF;
      END PROCESS;
END proj;
```

6V.11 Data switcher

```
ENTITY switcher IS
PORT (  en            :IN BIT;
        input, sel    :IN BIT_VECTOR (1 DOWNTO 0);
        output        :OUT BIT_VECTOR (1 DOWNTO 0)     );
END switcher;

ARCHITECTURE switch OF switcher IS
BEGIN
     PROCESS (en, input, sel)         -- sensitivity list
     BEGIN
          IF en = '0'     THEN        -- active-low enable
               CASE sel IS            -- control input
                    WHEN  "00"  =>
                          output(1) <= input(1);
                          output(0) <= input(0);
                    WHEN  "01"  =>
                          output(1) <= input(0);
                          output(0) <= input(0);
                    WHEN  "10"  =>
                          output(1) <= input(1);
                          output(0) <= input(1);
                    WHEN  "11"  =>
                          output(1) <= input(0);
                          output(0) <= input(1);
               END CASE;
          ELSE             output <= "00";      -- disabled
          END IF;
     END PROCESS;
END switch;
```

Unit 7 Testing Flip-Flops and Sequential Circuit Applications

7.1 NAND SR latch

Set	Reset	Q	Qbar	
0	0	1	1	invalid
0	1	1	0	set
1	0	0	1	reset
1	1	Q	Qbar	hold

active-low control inputs

7.2 D latch

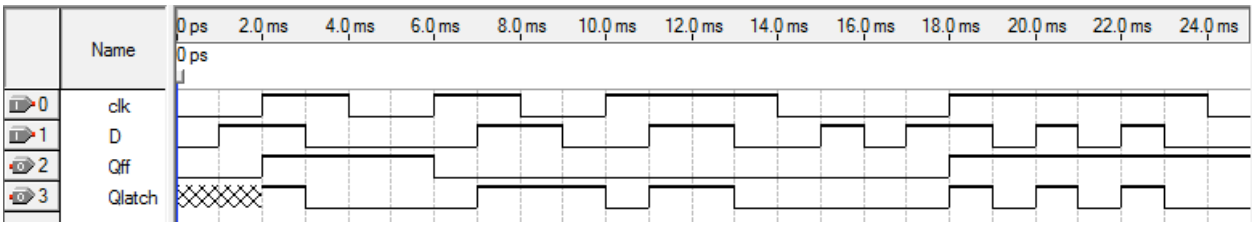
D latch is transparent when EN = 1, stores data when EN = 0

7.3 JK flip-flop

J & K inputs are synchronous, active-low PRE & CLR are asynchronous, synchronous controls require clock for triggering, asynchronous controls have priority, PGT clock is triggering control

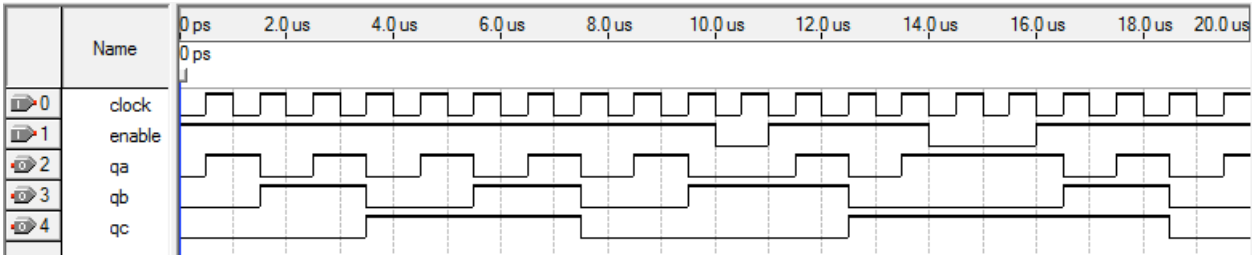
J	K	CLOCK	PRESET	CLEAR	Qn+1	Function
0	0	↑	1	1	Qn	Hold
0	1	↑	1	1	0	Reset
1	0	↑	1	1	1	Set
1	1	↑	1	1	!Qn	Toggle
X	X	X	0	1	1	Asynch preset
X	X	X	1	0	0	Asynch clear

7.4 D latch vs. D flip-flop



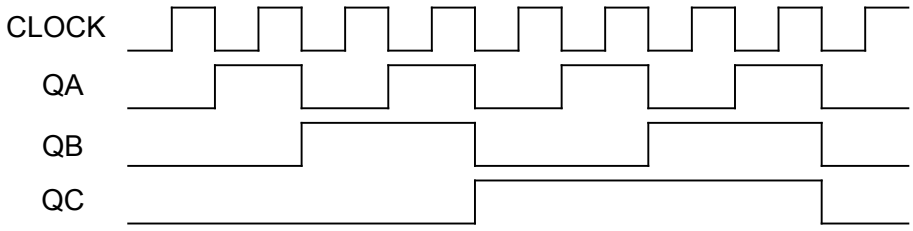
D latch is level-enabled (active-high) & D flip-flop is edge-triggered (positive going).

7.5 Binary counters



count sequence: 000 thru 111 & recycle → mod-8 up counter
synchronous counter → FFs are clocked simultaneously active-high enable

7.6 Counter timing diagram

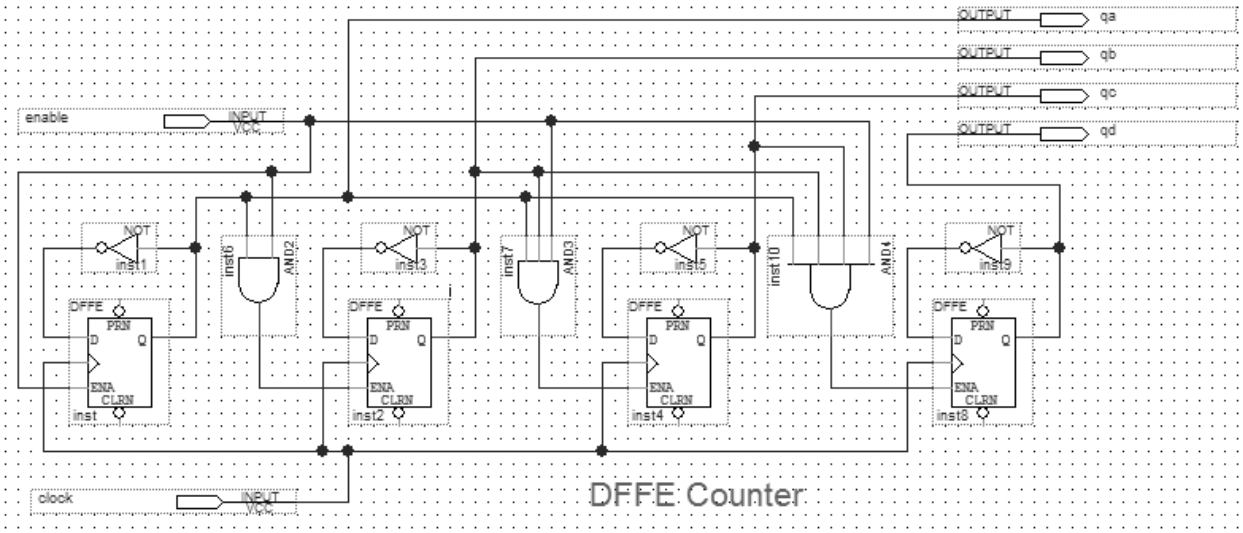


7.7 Frequency division

(assuming 10 kHz clock input frequency)

Signal	Frequency	Frequency Relationship
Clock	10 kHz	Input (assumed frequency)
QA	5 kHz	$Freq_{QA} = Freq_{Clock} \div 2$
QB	2.5 kHz	$Freq_{QB} = Freq_{Clock} \div 4$
QC	1.25 kHz	$Freq_{QC} = Freq_{Clock} \div 8$

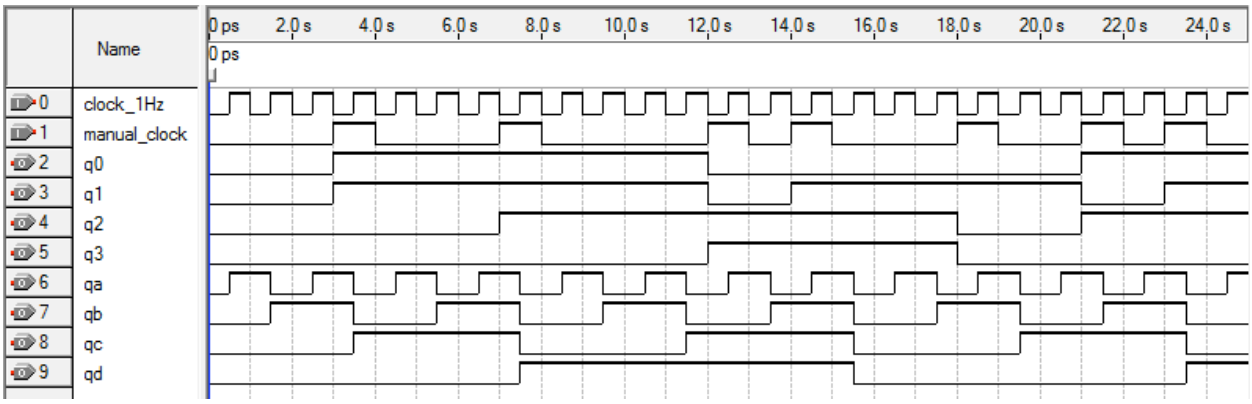
7.8 4-bit binary counter



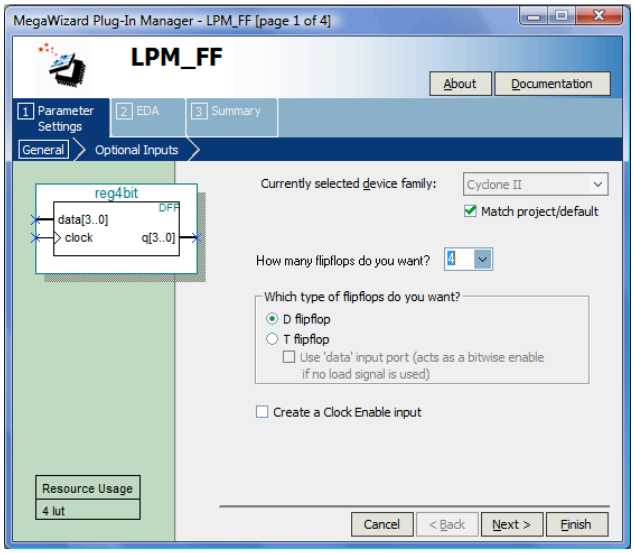
count sequence: 0000 thru 1111 & recycle → mod-16 up counter

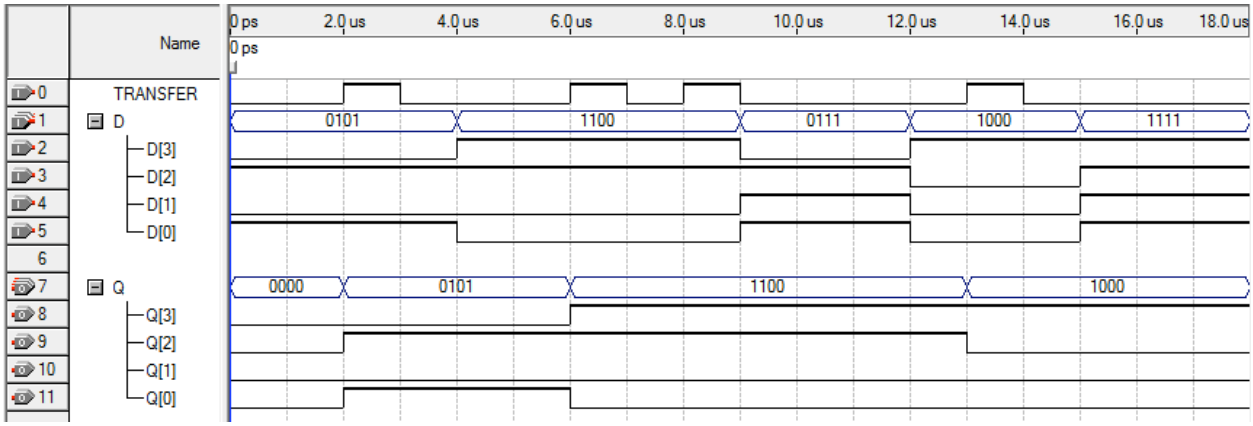
7.9 4-bit register

q0 is the LSB because the D input for that FF is connected to qa, which is the LSB for the counter. q3 is the MSB because D input is connected to qd, the counter MSB.
Register is ↑-edge triggered.
Register holds data between clocks.
If level-enabled, register would track the data input for the entire time that it is enabled.



7.10 LPM_FF megafunction





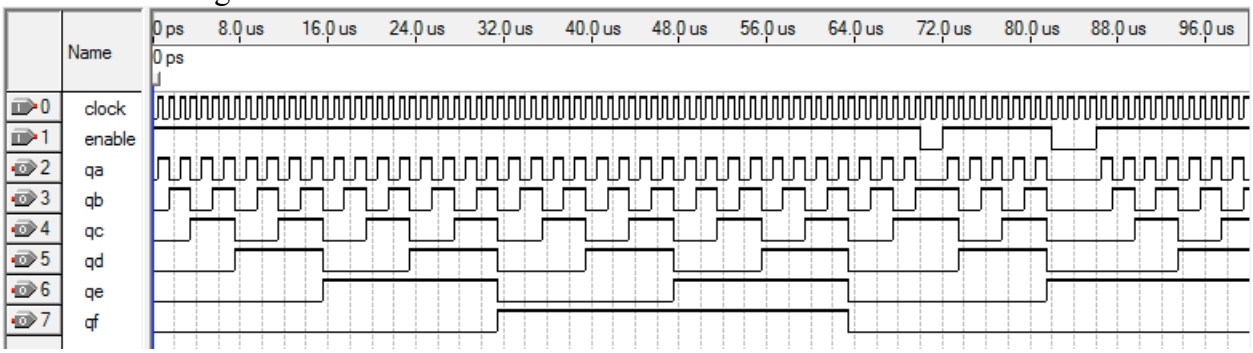
7.11 Binary counter

```
SUBDESIGN look_up_table
( q[3..0]      :INPUT;
  d[3..0]      :OUTPUT;)
BEGIN
CASE q[] IS
     WHEN H"0" =>   d[] = H"1";
     WHEN H"1" =>   d[] = H"2";
     WHEN H"2" =>   d[] = H"3";
     WHEN H"3" =>   d[] = H"4";
     WHEN H"4" =>   d[] = H"5";
     WHEN H"5" =>   d[] = H"6";
     WHEN H"6" =>   d[] = H"7";
     WHEN H"7" =>   d[] = H"8";
     WHEN H"8" =>   d[] = H"9";
     WHEN H"9" =>   d[] = H"A";
     WHEN H"A" =>   d[] = H"B";
     WHEN H"B" =>   d[] = H"C";
     WHEN H"C" =>   d[] = H"D";
     WHEN H"D" =>   d[] = H"E";
     WHEN H"E" =>   d[] = H"F";
     WHEN H"F" =>   d[] = H"0";
END CASE;
END;
```

```
ENTITY look_up_table IS
PORT ( q       :IN BIT_VECTOR (3 DOWNTO 0);
       d       :OUT BIT_VECTOR (3 DOWNTO 0));
END look_up_table;
ARCHITECTURE vhdl OF look_up_table IS
BEGIN
PROCESS (q)
BEGIN
CASE q IS
     WHEN X"0" =>   d <= X"1";
     WHEN X"1" =>   d <= X"2";
     WHEN X"2" =>   d <= X"3";
     WHEN X"3" =>   d <= X"4";
     WHEN X"4" =>   d <= X"5";
     WHEN X"5" =>   d <= X"6";
     WHEN X"6" =>   d <= X"7";
     WHEN X"7" =>   d <= X"8";
     WHEN X"8" =>   d <= X"9";
     WHEN X"9" =>   d <= X"A";
     WHEN X"A" =>   d <= X"B";
     WHEN X"B" =>   d <= X"C";
```

```
      WHEN X"C" =>    d <= X"D";
      WHEN X"D" =>    d <= X"E";
      WHEN X"E" =>    d <= X"F";
      WHEN X"F" =>    d <= X"0";
END CASE;
END PROCESS;
END vhdl;
```

7.12 Cascading counters



MSB: QF
AND gate detects 111 (terminal counter state) for QC QB QA
Enable controls QF QE QD counter through AND gate

7.13 Debouncing a logic switch

(a) counter appears to count erratically due to receiving multiple pulses on clock input from bouncing switch

(b) counter should count correctly (0000 through 1111) with latch
latch see a “hold” command when switch bounces open
pull-up resistors ensure that open latch inputs are held high

Unit 8 Timing and Waveshaping Circuits

8.1 Schmitt trigger waveshaper

Rounded output waveform for 7404
Square output waveform for 74LS14; triggers at approximately 1.6 V and 0.8 V on input waveform.

8.2 Pulse stretcher

$t_p = 0.693\ RC$ example solution:
$t_p = 0.47$ s. using C = 10 μf & R = 68 kΩ

8.3 Variable frequency clock

duty cycle = $(R_A + R_B)/(R_A + 2R_B) \approx 50\%$ if $R_A << R_B$ with $R_A \geq 1$ kΩ
frequency = $1.44/(R_A + 2R_B)C$
example solution:
duty cycle = 50.4% with R_A = 1 kΩ & R_B = 68 kΩ

capacitor	frequency
10 μf	1.05 Hz
0.01 μf	1.05 kHz
0.001 μf	10.53 kHz

8.4 Waveform generator

SIGNAL1: T = 1 ms. & duty cycle = 80% → freq. = 1 kHz

on 555: $t_L = 200\ \mu s = 0.693\ R_B\ C = 0.693\ R_B\ (0.01\ \mu f)$ → $R_B = 28.8$ kΩ
$t_H = 800\ \mu s = 0.693\ (R_A + R_B)\ (0.01\ \mu f)$ → $R_A = 86.6$ kΩ
using R_A = 82 kΩ, R_B = 27 kΩ, C = 0.01 μf:
T = 0.94 ms & duty cycle = 80.1%

on 74221: 1B = 2A = SIGNAL1
1A = 0 & 2B = 1 & 1CLR = 2CLR = 1
$t_p = 0.693\ RC$

SIGNAL2: $t_p = 300\ \mu s$ → use 1Q output
using C = 0.01 μf → R = 43.3 kΩ → try 47 kΩ

SIGNAL3: $t_p = 100\ \mu s$ → use $\overline{2Q}$ output
using C = 0.0047 μf → R = 30.7 kΩ → try 33 kΩ

8.5 Variable duty-cycle square wave generator

Assuming $V_F = 0.7v$:
$T = t_L + t_H = 1.00(R_A+R_B)C$ $\therefore f = 1/[(R_A+R_B)C]$
for f = 10 kHz, try 0.0047 μf $\rightarrow (R_A+R_B) = 21.3\ k\Omega$
for D.C. = 15% = $R_A/(R_A+R_B) = 0.15$ $\rightarrow R_A = 0.15\ (21.3\ k\Omega) = 3.2\ k\Omega$
for D.C. = 85% = $R_A/(R_A+R_B) = 0.85$ $\rightarrow R_A = 0.85\ (21.3\ k\Omega) = 18.1\ k\Omega$
$\therefore$ use 3.3 kΩ + 10 kΩ pot + 8.2 kΩ resistors = 21.5 kΩ = R_A+R_B

8.6 Delayed pulse

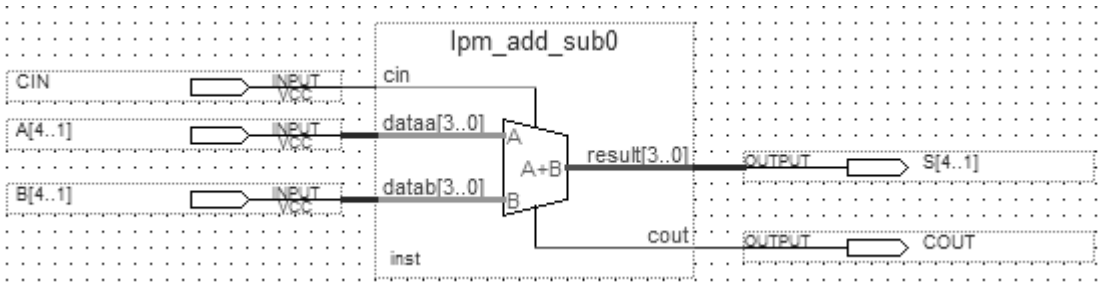
use both one-shots in 74221: 1A = trigger, 2A = 1Q
$t_w = 0.22\ ms = 0.693R_2C_2$ $\rightarrow$ if $C_2 = 0.0047\ \mu f$, $R_2 = 68\ k\Omega$
$t_d = 0.33\ ms = 0.693R_1C_1$ $\rightarrow$ if $C_2 = 0.01\ \mu f$, $R_1 = 47\ k\Omega$

8.7 Clock generator using one-shots

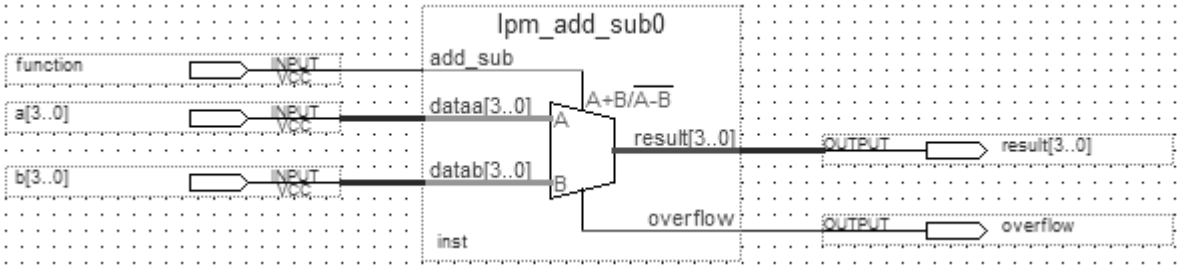
on 74221: 1A = 2Q, 2A = 1Q
$t_1 = 0.693R_1C = 0.693R_1(0.1\ \mu f)$ $\rightarrow$ for 5 ms, $R_1 = 72\ k\Omega$
try $R_1 = 72\ k\Omega$ $\rightarrow t_1 = 5.68$ ms
$t_2 = 0.693R_2(0.1\ \mu f)$ $\rightarrow$ for 10 ms-5.68 ms = 4.32 ms
$R_2 = 62\ k\Omega$

Unit 9 Arithmetic Circuit Applications

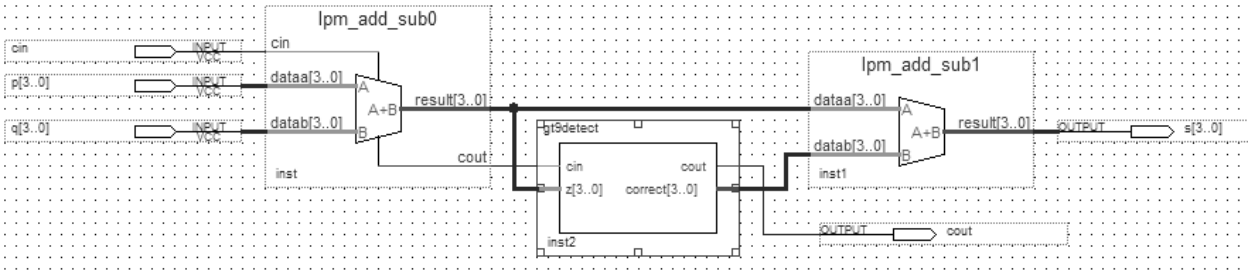
9.1 Four-bit adder



9.2 Four-bit binary adder/subtractor



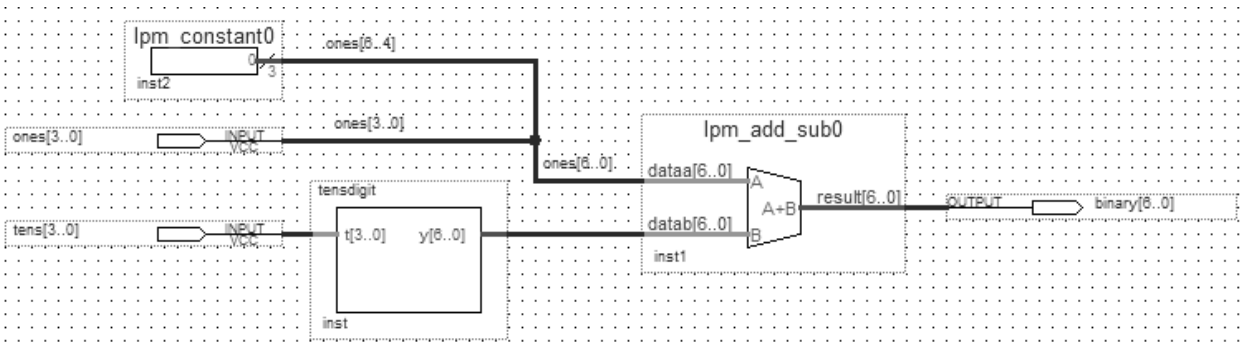
9.3 BCD adder (one digit)




```
SUBDESIGN  gt9detect
(
      cin, z[3..0]              :INPUT;
      cout, correct[3..0]       :OUTPUT;
)
BEGIN
      IF cin == VCC # z[] > 9        THEN
                cout = VCC;
                correct[] = 6;
      ELSE      cout = GND;
                correct[] = 0;
      END IF;
END;
```

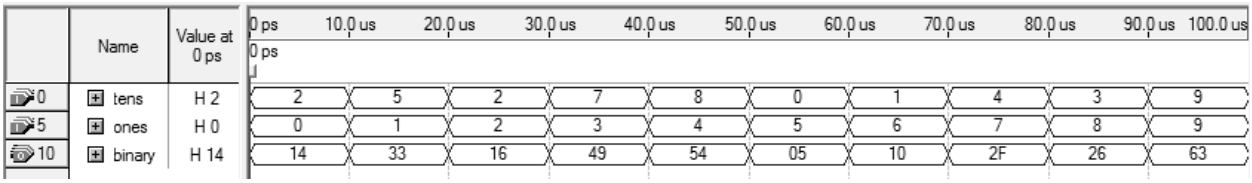
```
ENTITY gt9detect IS
PORT (
      cin         :IN BIT;
      z           :IN INTEGER RANGE 0 TO 15;
      cout        :OUT BIT;
      correct     :OUT INTEGER RANGE 0 TO 15
);
END gt9detect;
ARCHITECTURE vhdl OF gt9detect IS
BEGIN
    PROCESS (c4, z)
    BEGIN              -- detect if binary sum >9
        IF c4 = '1' OR z > 9  THEN
                cout <= '1';
                correct <= 6;
        ELSE
                cout <= '0';
                correct <= 0;
        END IF;
    END PROCESS;
END vhdl;
```

9.4 BCD to binary converter

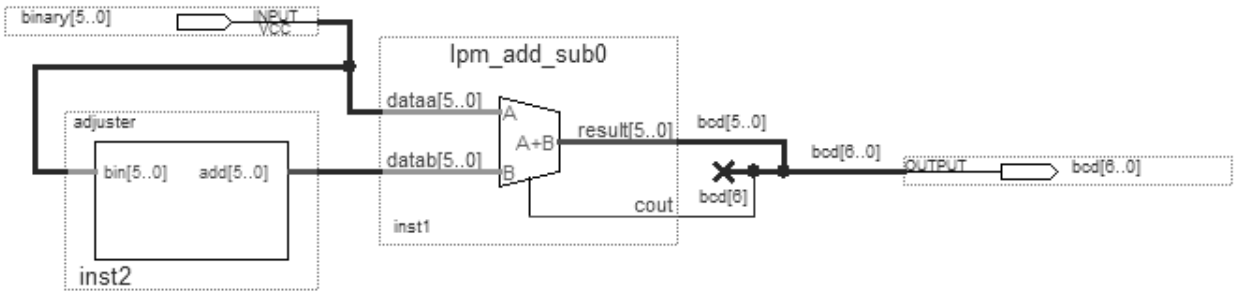



```
SUBDESIGN  tensdigit
(     t[3..0]          :INPUT;
      y[6..0]          :OUTPUT;    )
BEGIN
CASE  t[]  IS
      WHEN 0      =>    y[] = 0;
      WHEN 1      =>    y[] = 10;
      WHEN 2      =>    y[] = 20;
      WHEN 3      =>    y[] = 30;
      WHEN 4      =>    y[] = 40;
      WHEN 5      =>    y[] = 50;
      WHEN 6      =>    y[] = 60;
      WHEN 7      =>    y[] = 70;
      WHEN 8      =>    y[] = 80;
      WHEN 9      =>    y[] = 90;
      WHEN OTHERS      =>    y[] = B"1111111";
END CASE;
END;
```

```
ENTITY tensdigit IS
PORT ( t   :IN INTEGER RANGE 0 TO 15;
       y   :OUT INTEGER RANGE 0 TO 90 );
END tensdigit;
ARCHITECTURE vhdl OF tensdigit IS
BEGIN
      y <= t * 10;
END;
```



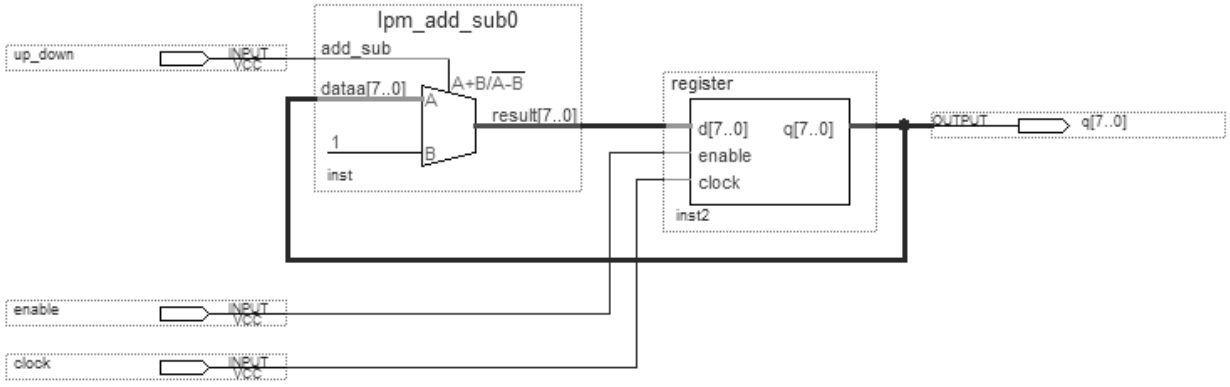
9.5 Binary to BCD converter

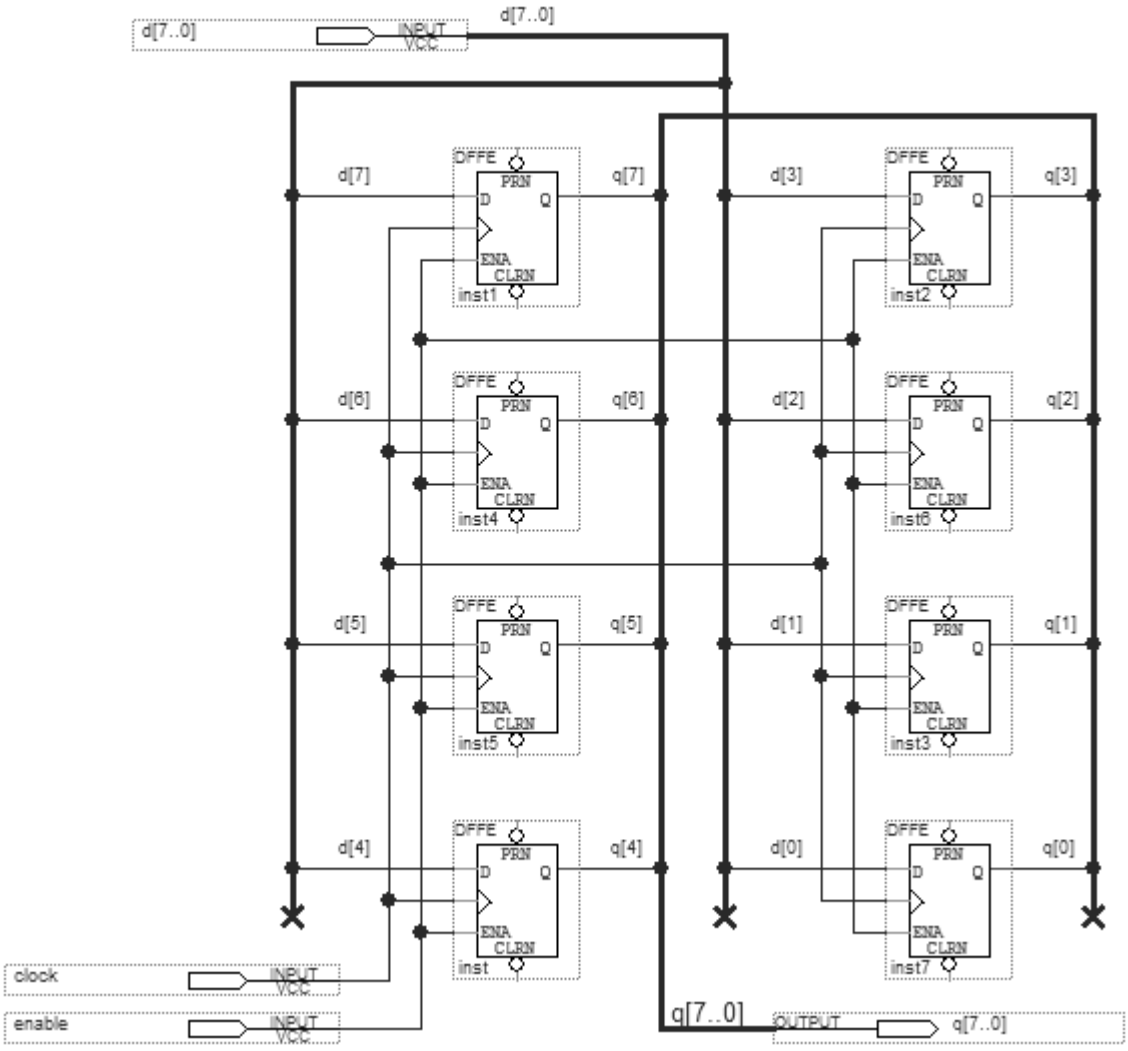



```
SUBDESIGN  adjuster
(
      bin[5..0]        :INPUT;
      add[5..0]        :OUTPUT;
)
BEGIN
      IF    bin[] < 10      THEN  add[] = 0;
      ELSIF bin[] < 20      THEN  add[] = 6;
      ELSIF bin[] < 30      THEN  add[] = 12;
      ELSIF bin[] < 40      THEN  add[] = 18;
      ELSIF bin[] < 50      THEN  add[] = 24;
      ELSIF bin[] < 60      THEN  add[] = 30;
      ELSE                        add[] = 36;
      END IF;
END;
```

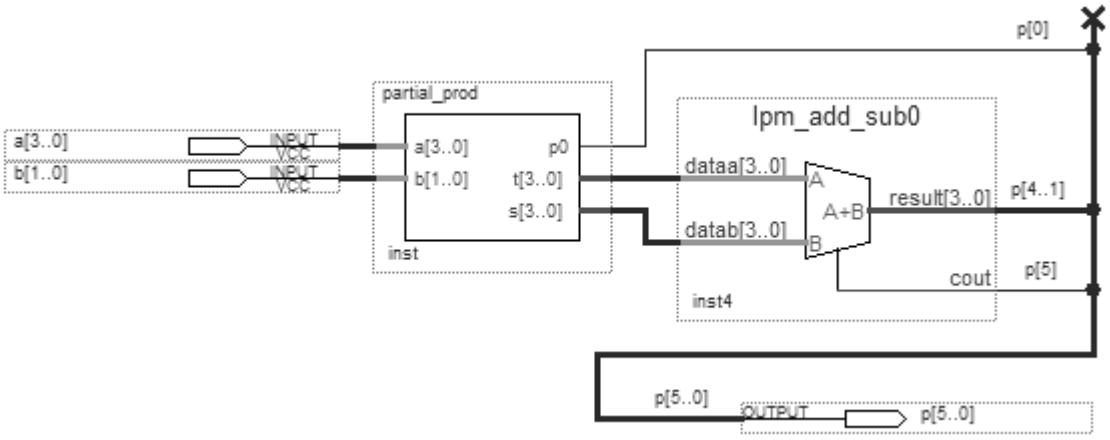
```
ENTITY adjuster IS
PORT (
      bin         :IN INTEGER RANGE 0 TO 63;
      add         :OUT INTEGER RANGE 0 TO 63
);
END adjuster;

ARCHITECTURE vhdl OF adjuster IS
BEGIN
      PROCESS (bin)
      BEGIN
      IF    bin < 10        THEN       add <= 0;
      ELSIF bin < 20        THEN       add <= 6;
      ELSIF bin < 30        THEN       add <= 12;
      ELSIF bin < 40        THEN       add <= 18;
      ELSIF bin < 50        THEN       add <= 24;
      ELSIF bin < 60        THEN       add <= 30;
      ELSE                             add <= 36;
      END IF;
      END PROCESS;
END vhdl;
```

9.6 Up/down counter



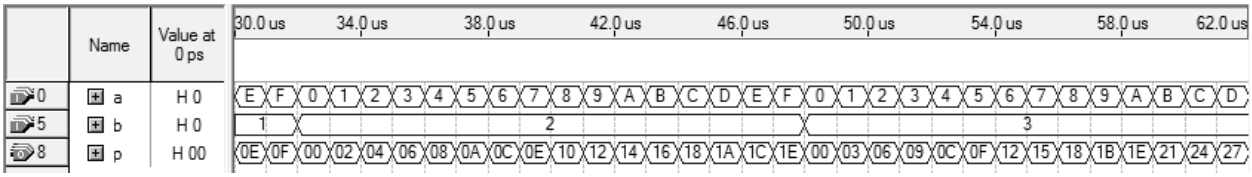


9.7 Binary multiplier

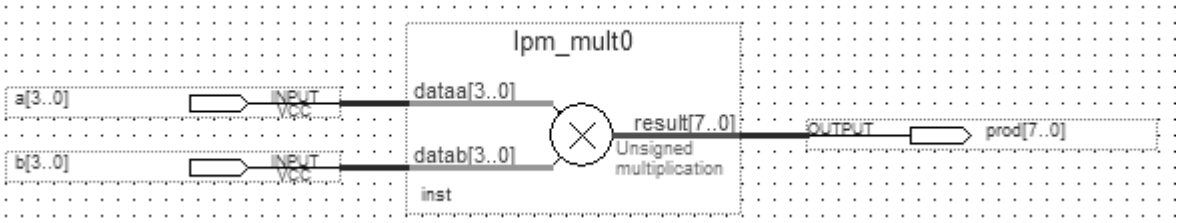


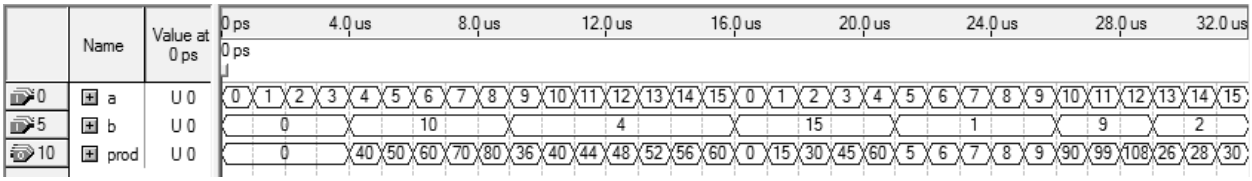

```
SUBDESIGN  partial_prod
(
      a[3..0], b[1..0]      :INPUT;
      p0, t[3..0], s[3..0]  :OUTPUT;
)
VARIABLE
      r[3..0]               :NODE;
BEGIN
      r[3..0] = a[3..0] & b0;
      s[3..0] = a[3..0] & b1;
      t[3..0] = (GND, r[3..1]);
      p0 = r0;
END;
```

```
ENTITY partial_prod IS
PORT (
            a           :IN BIT_VECTOR (3 DOWNTO 0);
            b           :IN BIT_VECTOR (1 DOWNTO 0);
            p0          :OUT BIT;
            s, t        :OUT BIT_VECTOR (3 DOWNTO 0)
);
END partial_prod;

ARCHITECTURE vhdl OF partial_prod IS
SIGNAL      r           :BIT_VECTOR (3 DOWNTO 0);
BEGIN
      r(3 DOWNTO 0) <= a(3 DOWNTO 0) AND (b(0)&b(0)&b(0)&b(0));
      s(3 DOWNTO 0) <= a(3 DOWNTO 0) AND (b(1)&b(1)&b(1)&b(1));
      t(3 DOWNTO 0) <= '0' & r(3 DOWNTO 1);
      p0 <= r(0);
END vhdl;
```



9.8 Multiplier megafunction

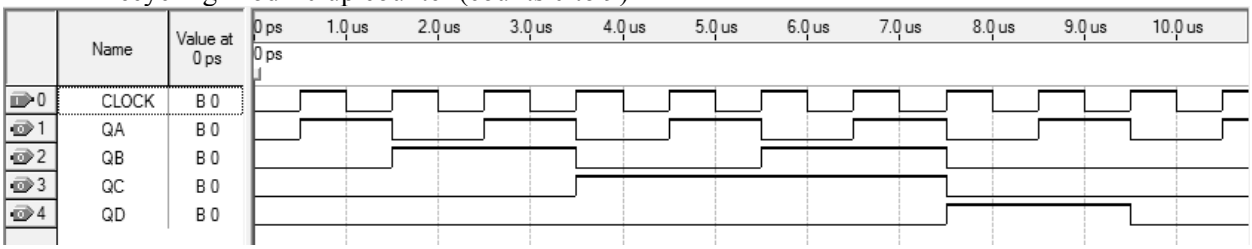




Unit 10 Analyzing and Testing Synchronous Counters

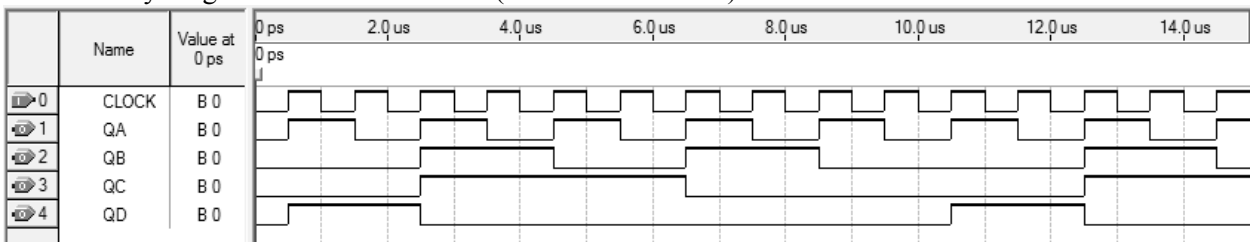
10.1 Counter 1 (a & b)

Recycling Mod-10 up counter (counts 0 to 9)



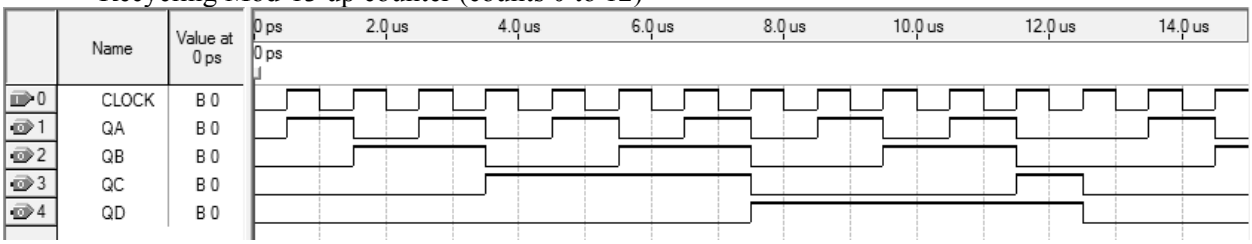
10.2 Counter 2 (a & b)

Recycling Mod-10 down counter (counts 9 down to 0)



10.3 Counter 3 (a & b)

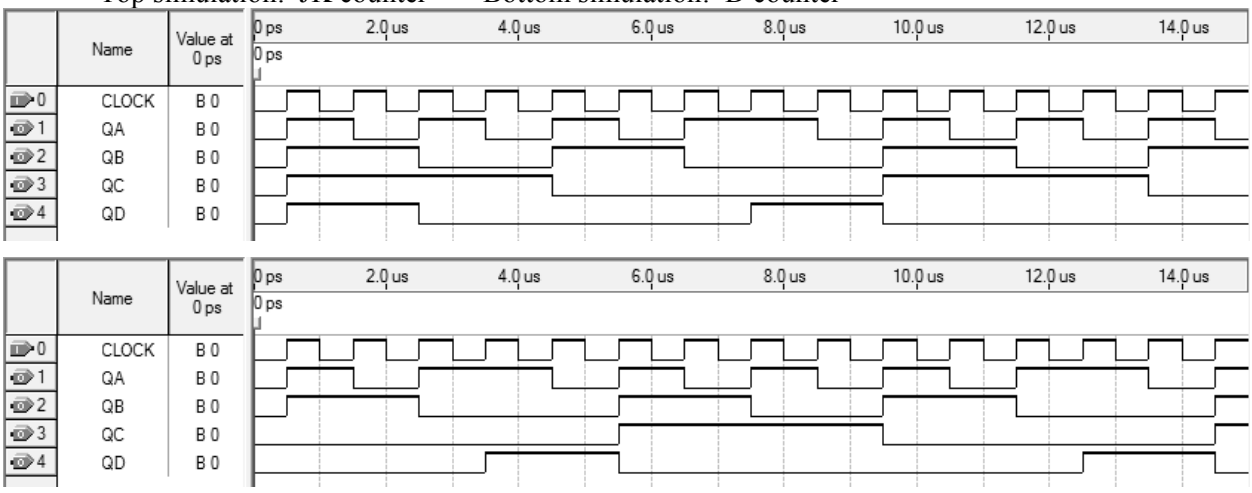
Recycling Mod-13 up counter (counts 0 to 12)



10.4 Counter 4 (a & b)

Recycling Mod-9 down counter (counts 9 to 1 after start-up sequence)

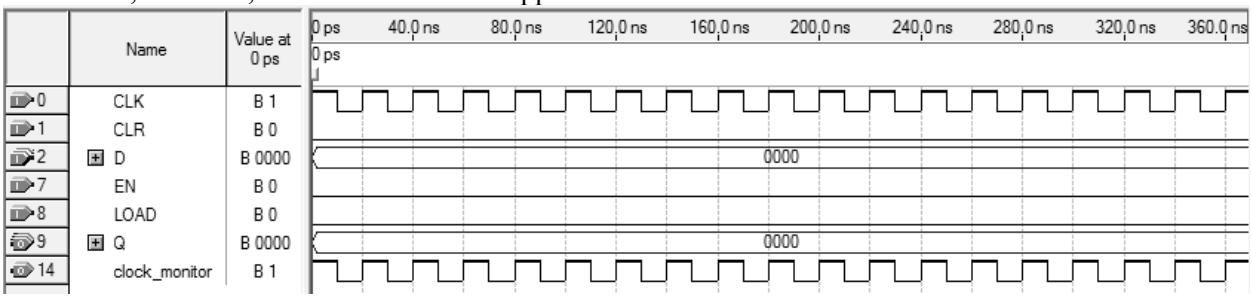
Top simulation: JK counter Bottom simulation: D counter



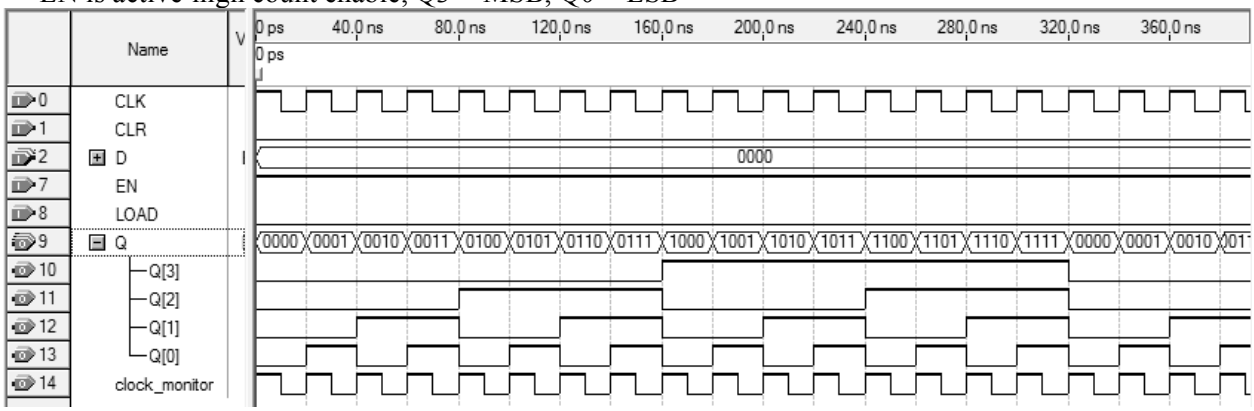
Unit 11 Creating and Testing Counter Functions

11.1 Mod-16 binary counter

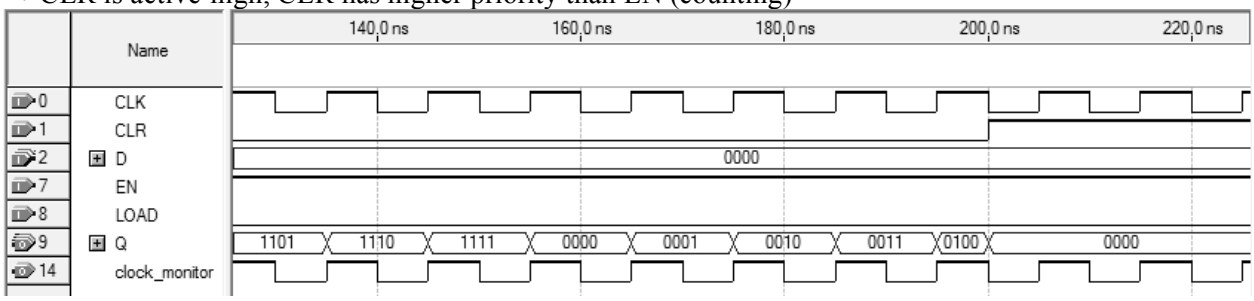
LOAD = 0, CLR = 0, EN = 0 → counter stopped



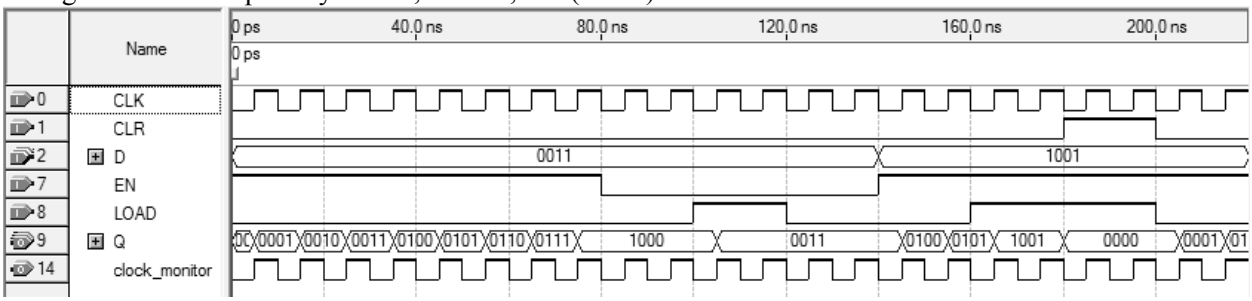
LOAD = 0, CLR = 0, EN = 1
→ counter counts up 0000 to 1111 & recycles, clocked on positive-edge
→ EN is active-high count enable, Q3 = MSB, Q0 = LSB



LOAD = 0, CLR = 1, EN = 1
→ counter clears immediately when CLR goes high, CLR is asynchronous & not dependent upon EN
→ CLR is active-high, CLR has higher priority than EN (counting)



LOAD = 1, CLR = 0, EN = 0
→ counter loads D[3..0] when LOAD goes high, LOAD is synchronous & overrides counting
→ highest to lowest priority: CLR, LOAD, EN (count)

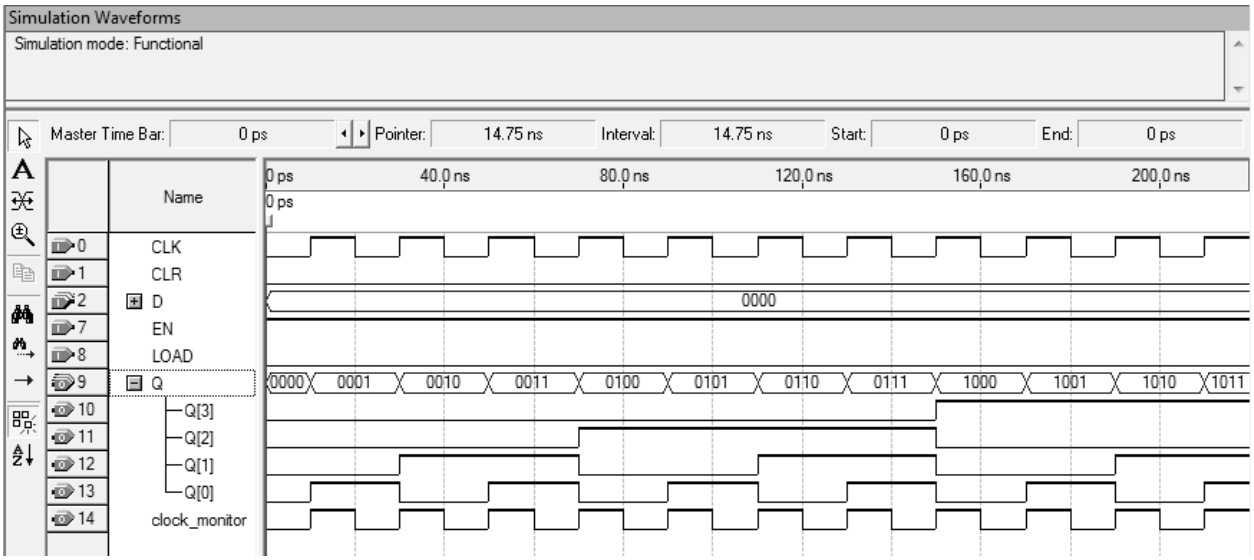


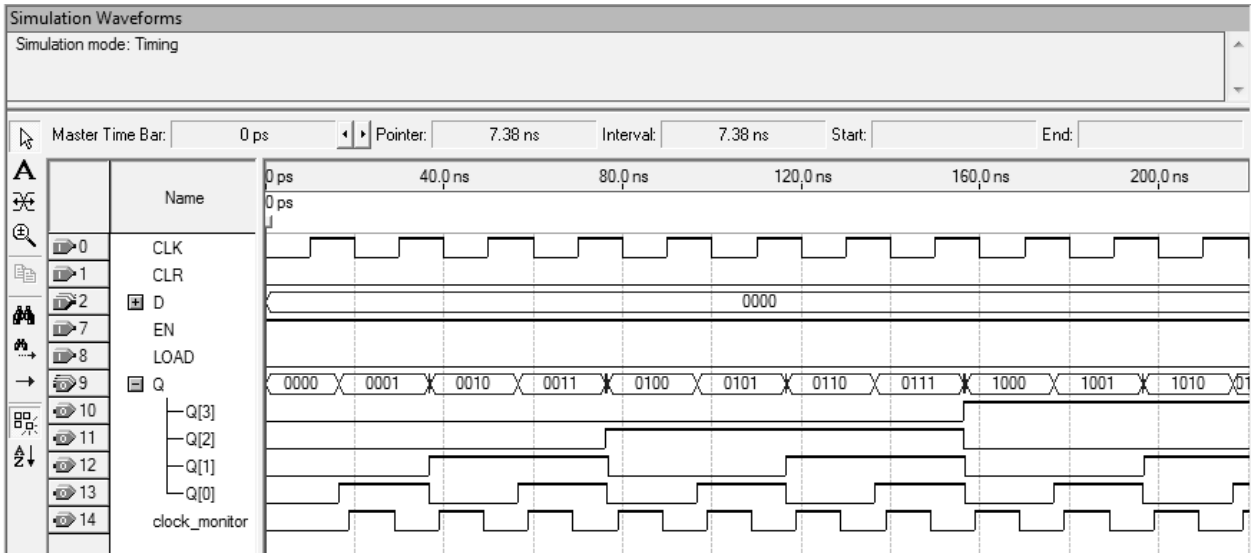
11.2 Binary counter signal frequencies

signal	Clock	Q0	Q1	Q2	Q3
frequency	16kHz	8kHz	4kHz	2kHz	1kHz

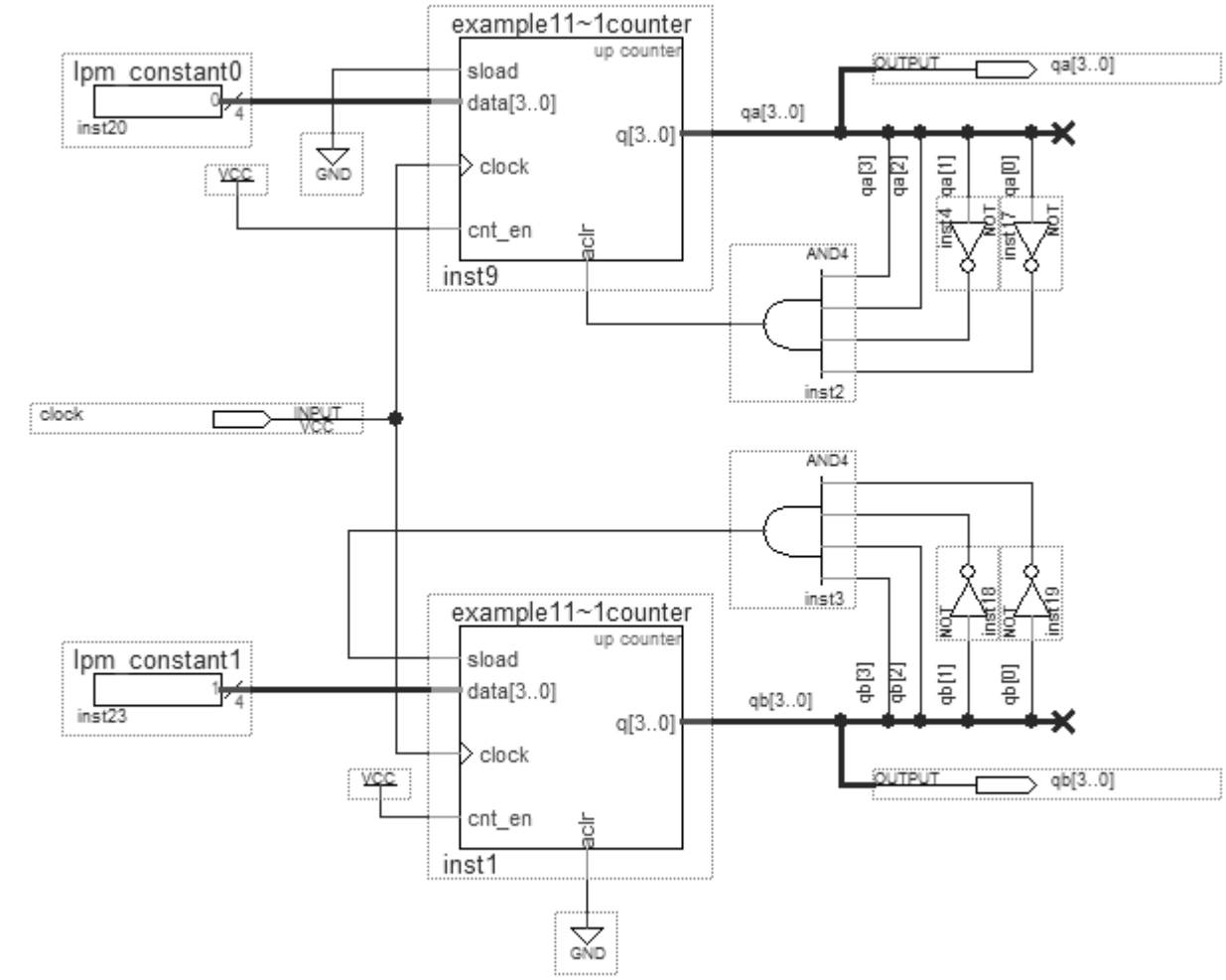
$freq_{Q3} = ½\ freq_{Q2} = ½\ freq_{Q1} = ½\ freq_{Q0} = ½\ freq_{Clock}$.

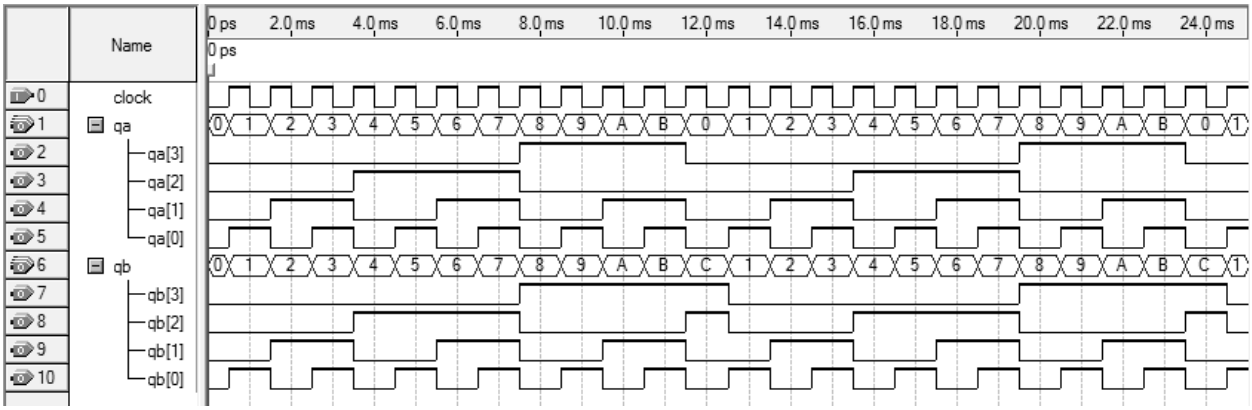
11.3 Functional vs. timing simulation





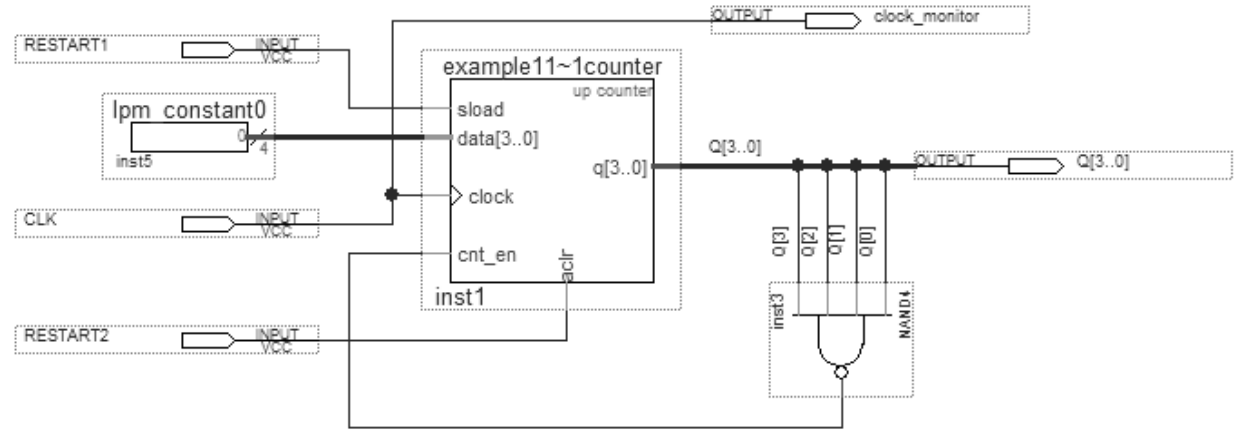
11.4 Mod-12 counters

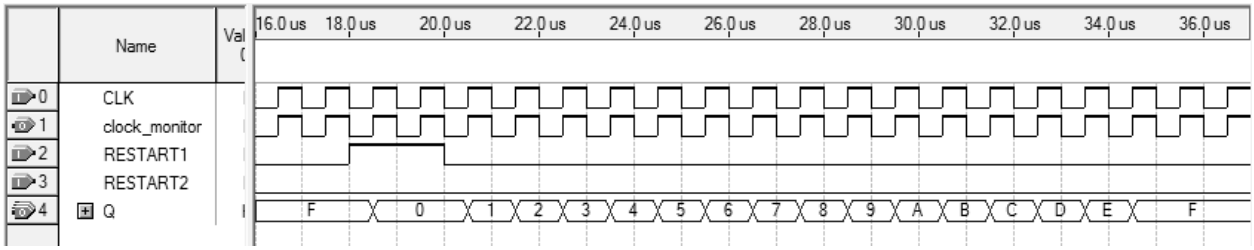




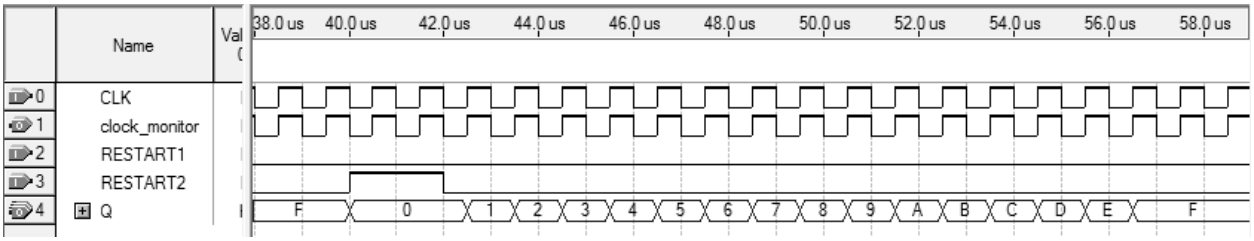
Decode 1100 (12_{10}) on each counter. Data input for counter B is 0001.
Counter A counts sequence 0000 to 1011 due to asynchronous CLR.
Counter B counts sequence 0001 to 1100 due to synchronous LOAD.
To produce same count sequence for counter B, need to decode 1011 (11_{10}) & LOAD 0000.

11.5 Self-stopping counter



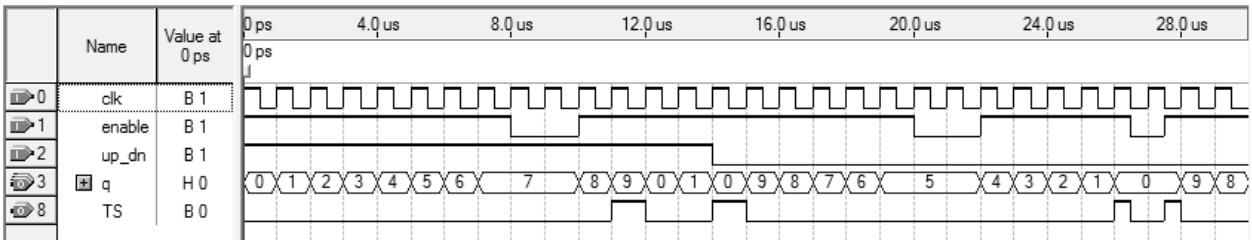


Restart1 uses synchronous LOAD.



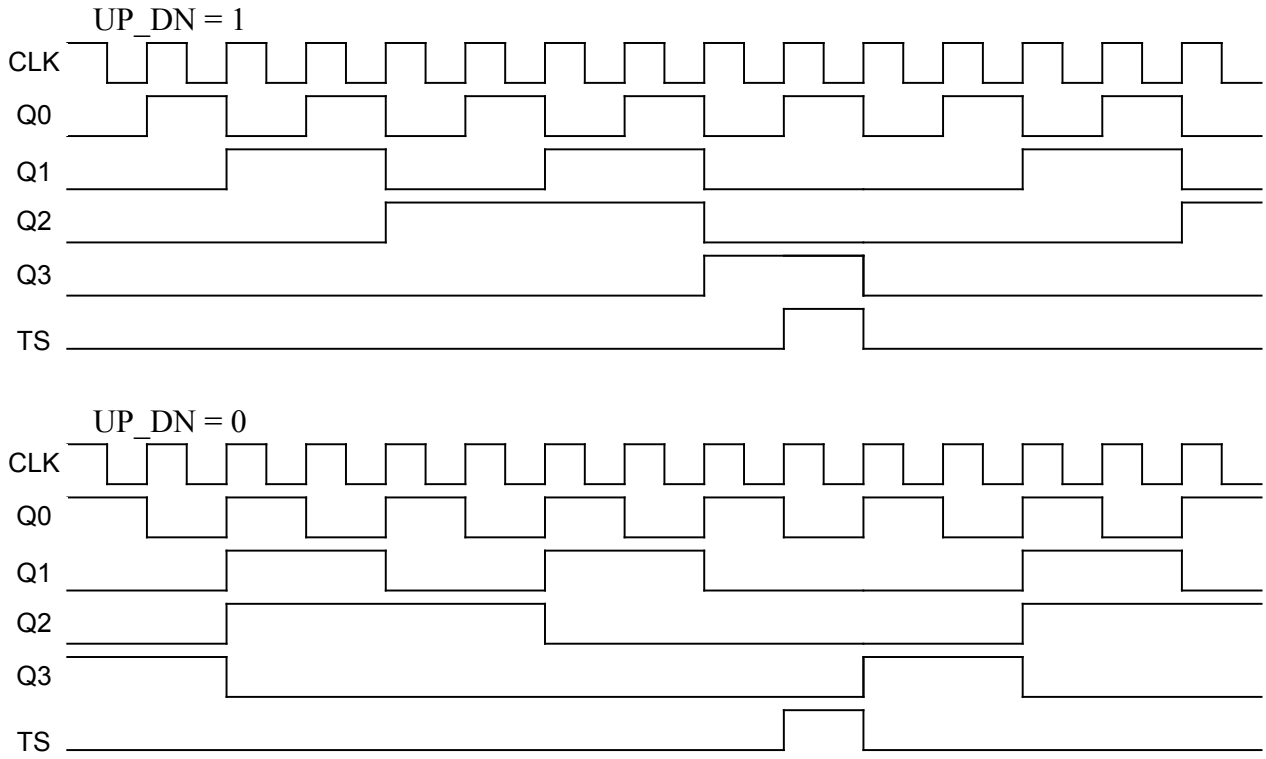
Restart2 uses asynchronous CLR.

11.6 Mod-10 up/down counter

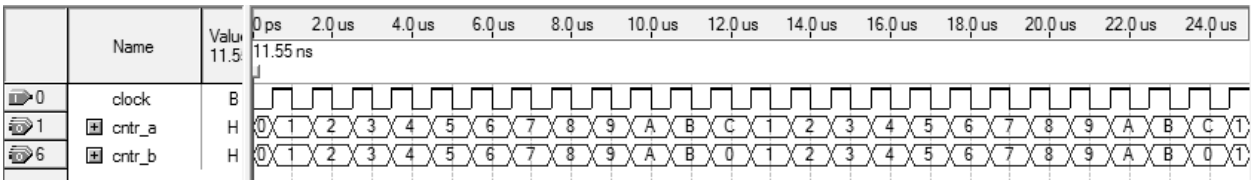


Active-high count enable. UP_DN = 0 counts down, UP_DN = 1 counts up. TS = 1 when counter reaches terminal state (9 when counting up & 0 when counting down). ENABLE also controls TS output.

11.7 Mod-10 timing diagram



11.8 Comparing counters



Cntr_a: count sequence is 0001 to 1100 (mod-12), terminal state is 1100 (12_{10})
Cntr_b: count sequence is 0000 to 1011 (mod-12), terminal state is 1011 (11_{10})
Same modulus for each counter, no transient states present
Cout detects the terminal state for cntr_a
Sset is a synchronous set control that will load the specified value (1_{10}) when enabled

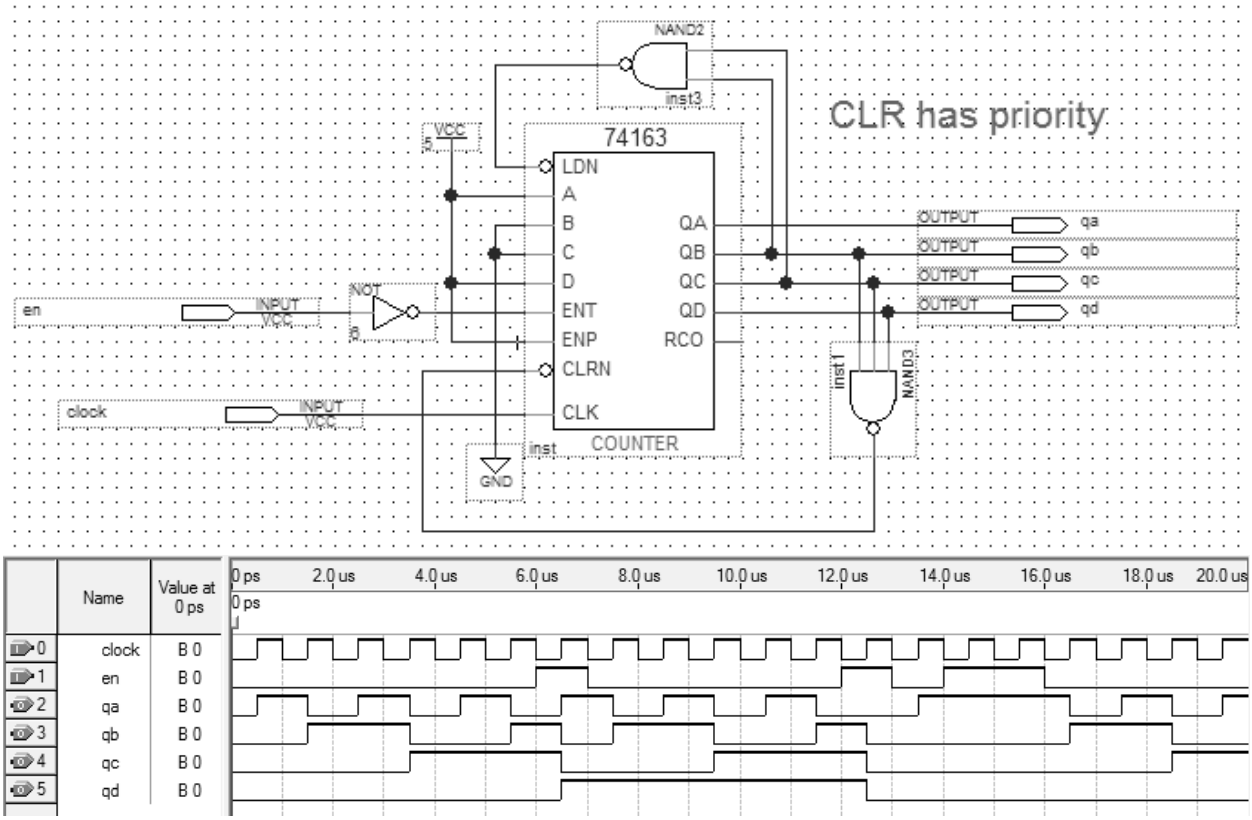
11.9 Counter waveforms

Each output has the same frequency (1kHz).
Both counters are mod-10 counters.
Duty cycles for each output is different due to signal used.
Output signals also have different phases (i.e., pulse occurs at different times).

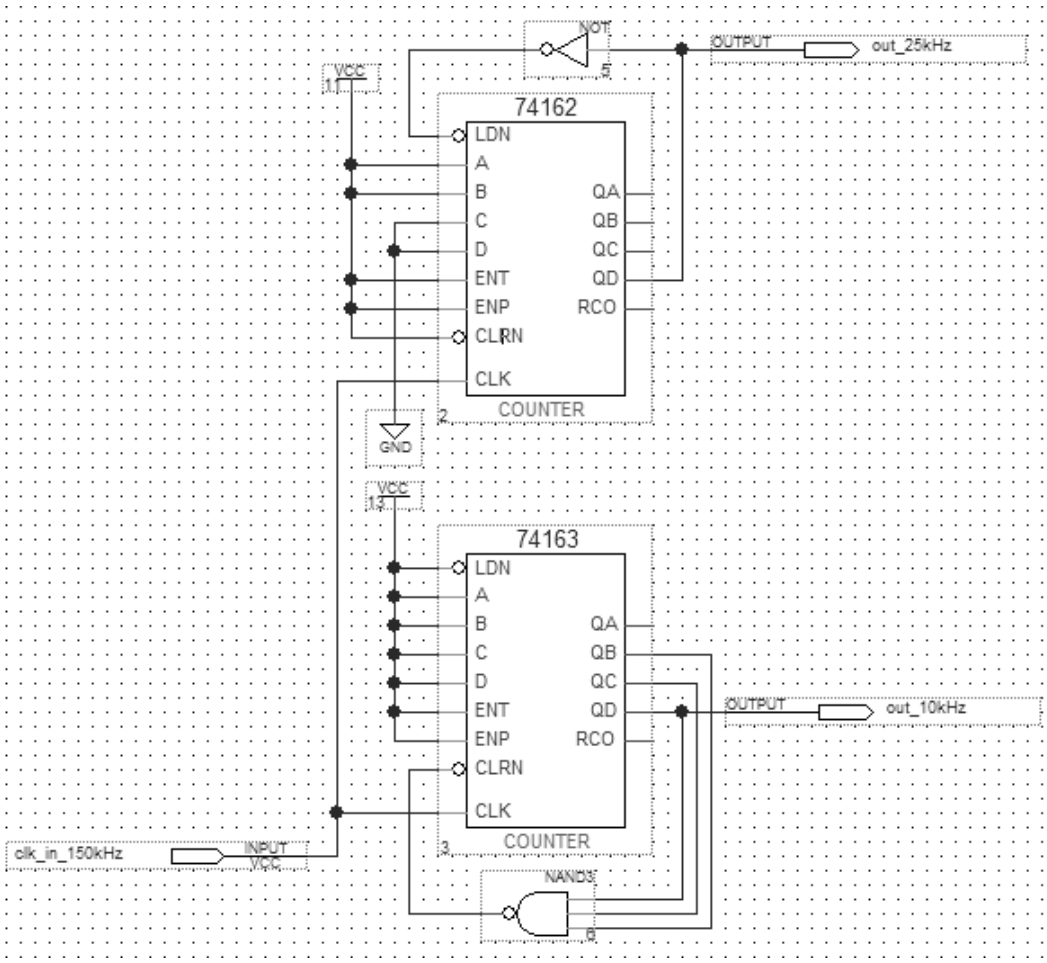
Signal	F1	F2	F3	F4	F5
Duty Cycle	10%	40%	20%	90%	50%

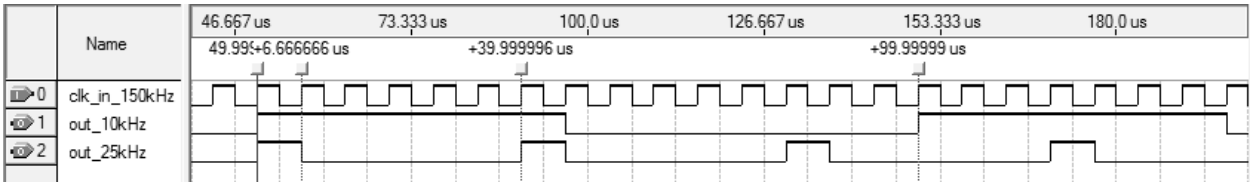
Unit 12 Applications Using Maxplus2 Counters

12.1 Mod-13 count sequence

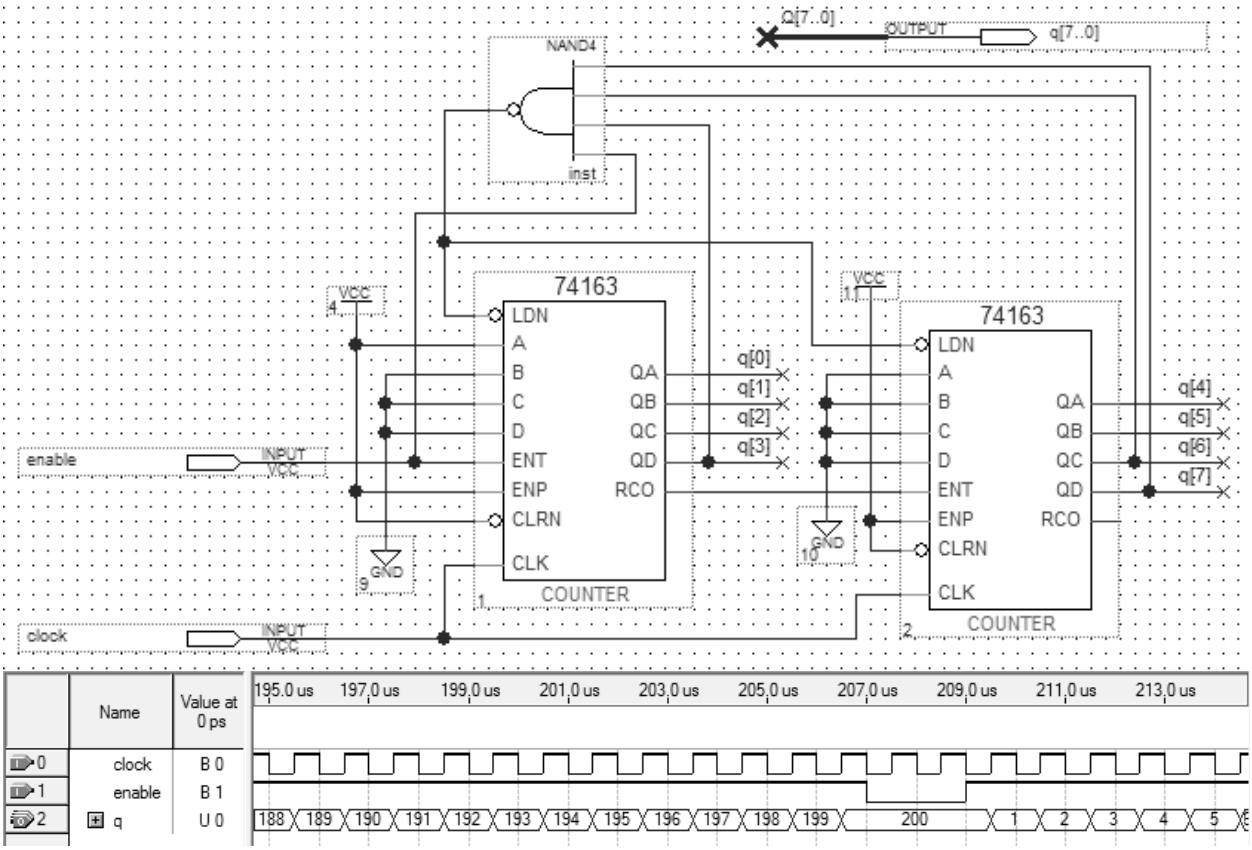


12.2 Frequency divider circuit

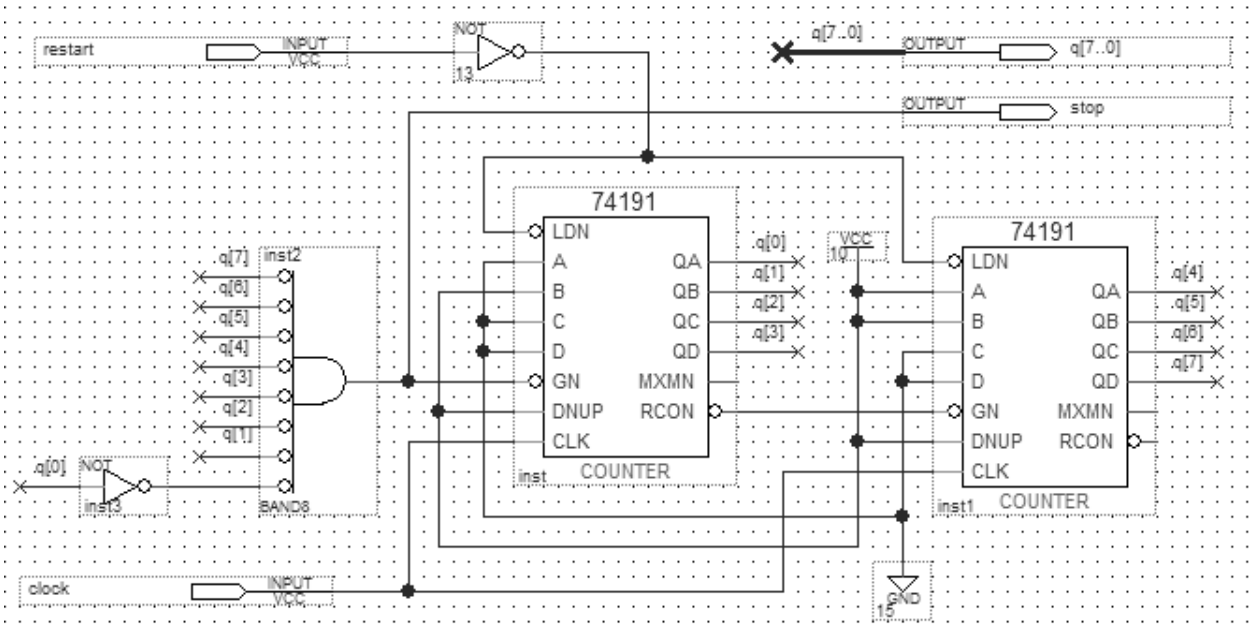


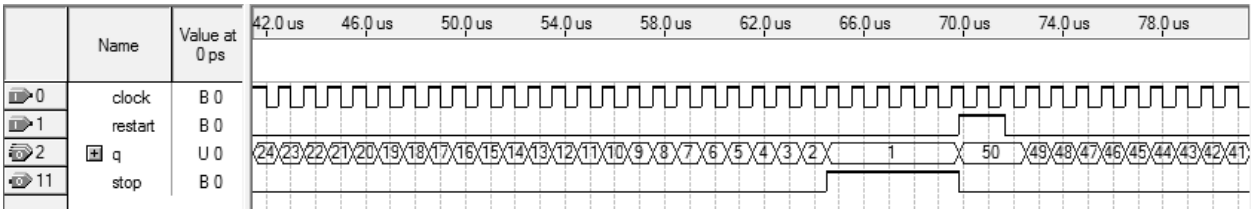


12.3 Mod-200 binary counter

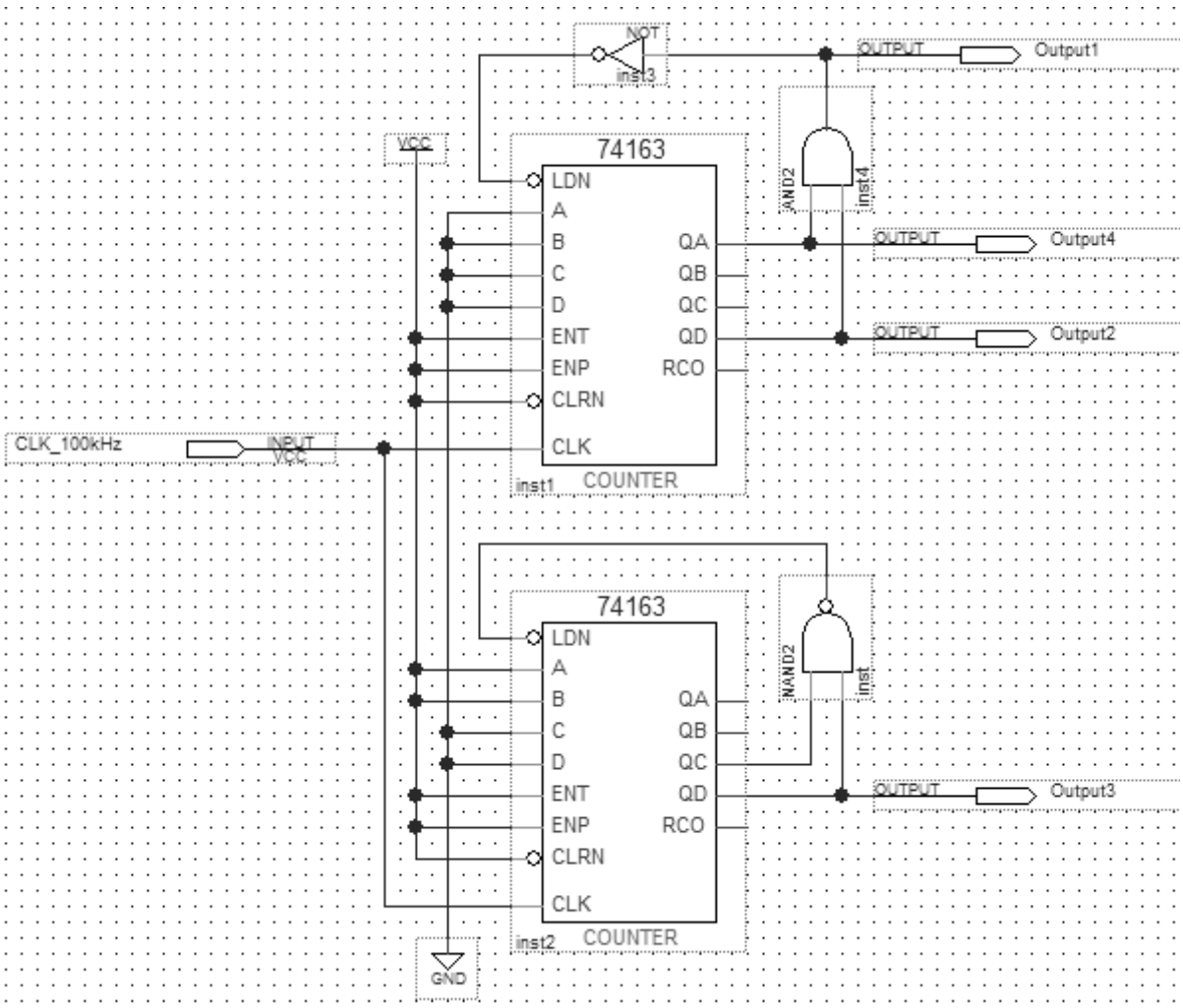


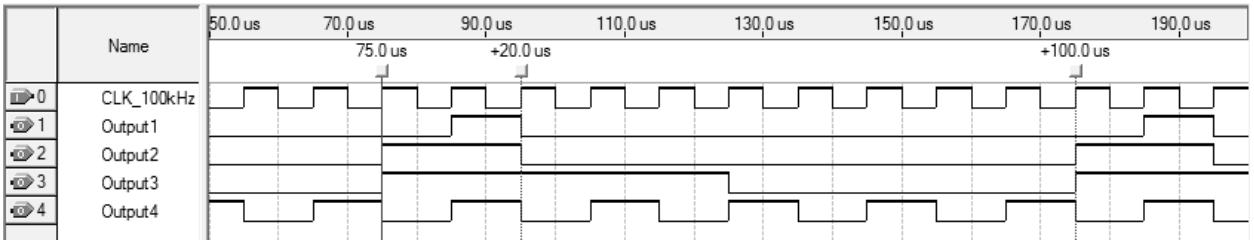
12.4 Mod-50, self-stopping, down-counter



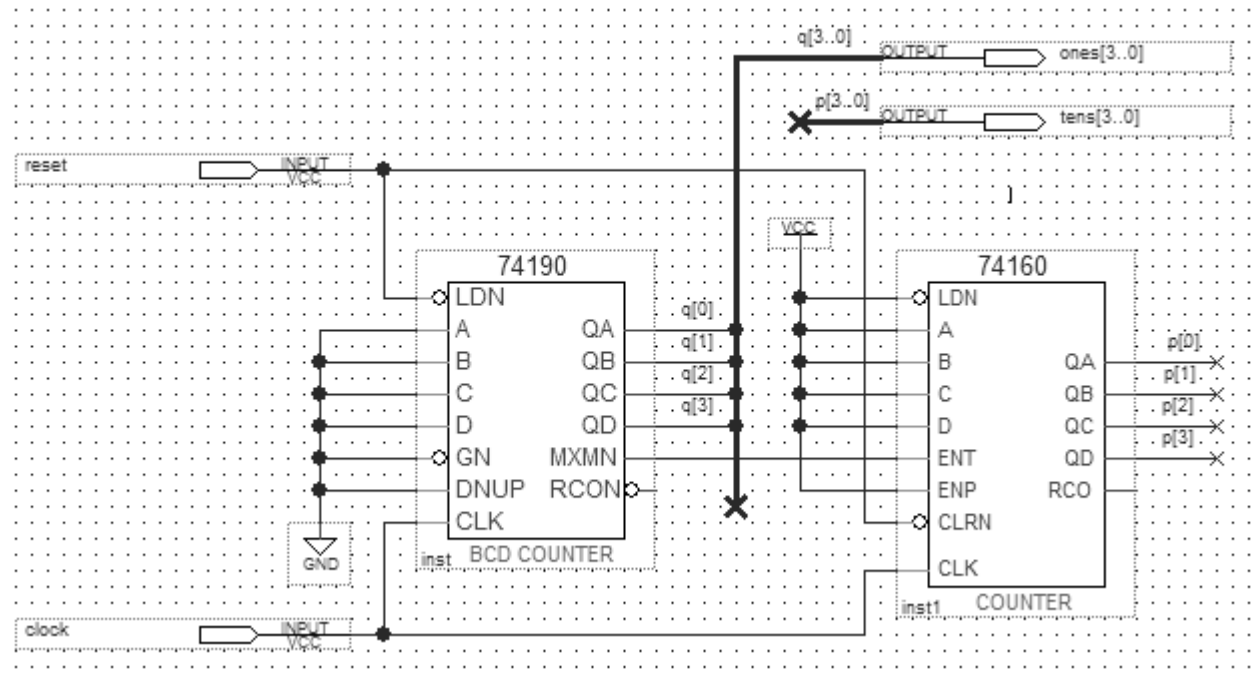


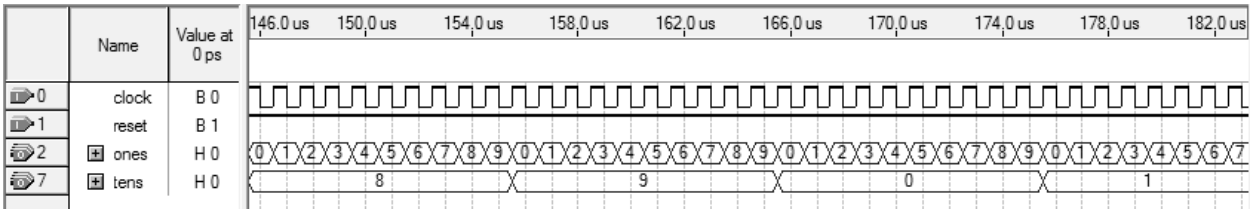
12.5 Waveform generator circuits

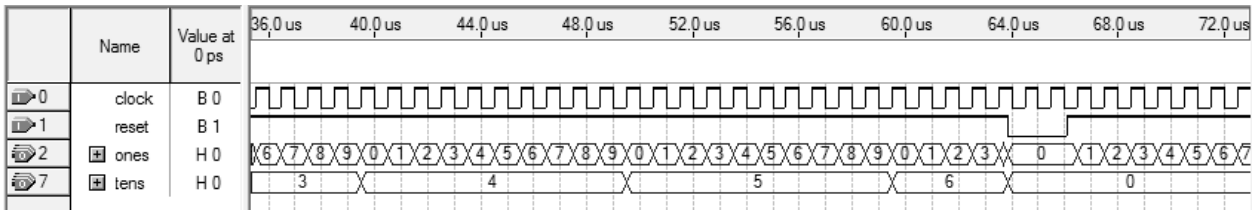




12.6 Mod-100 BCD counter



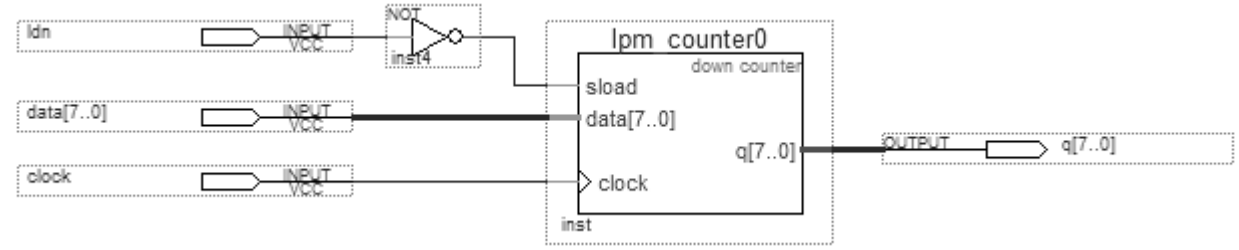


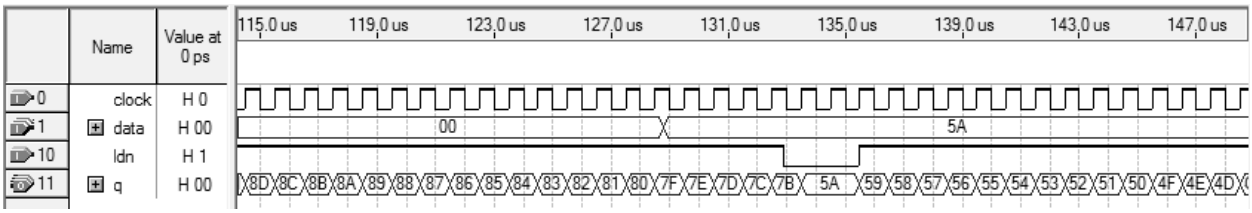


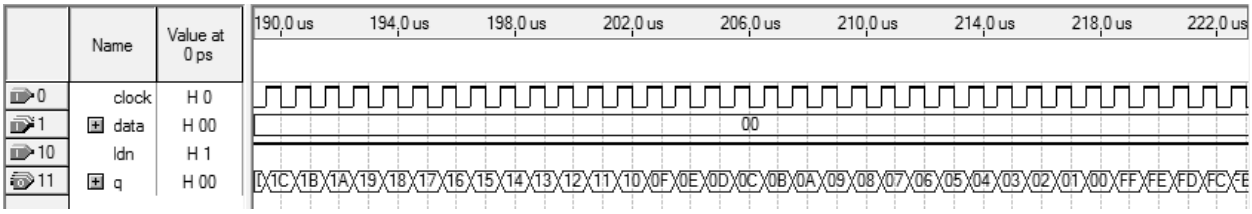
74162 has a synchronous clear, but design requires an asynchronous “reset”

Unit 13 Applications using LPM_Counter Megafunctions

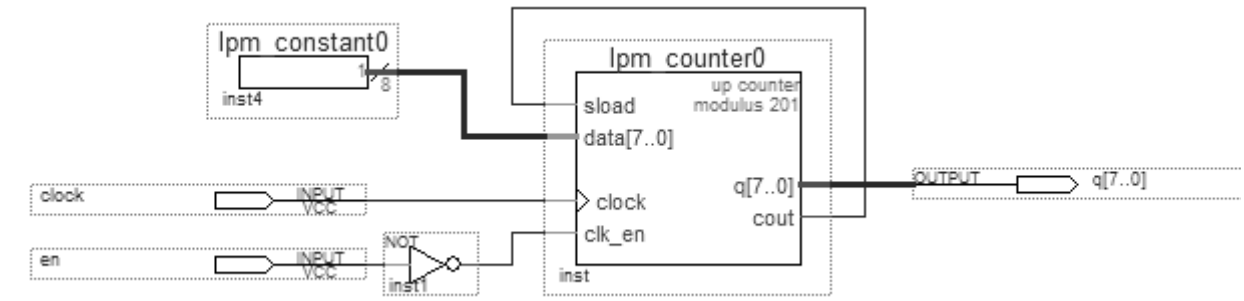
13.1 Mod-256 binary down-counter

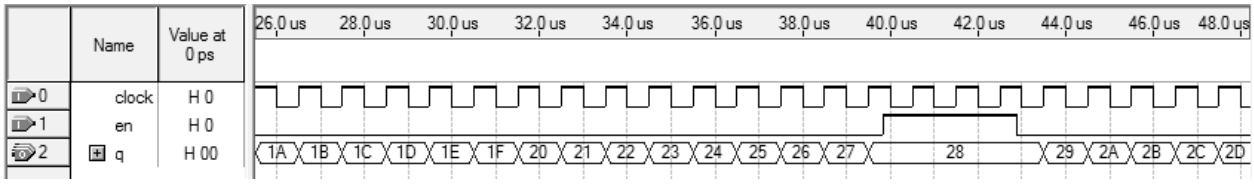


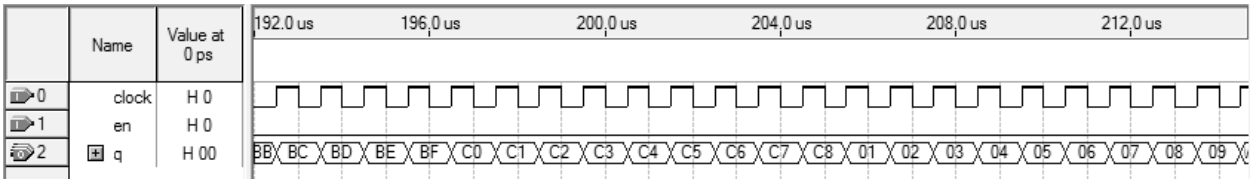




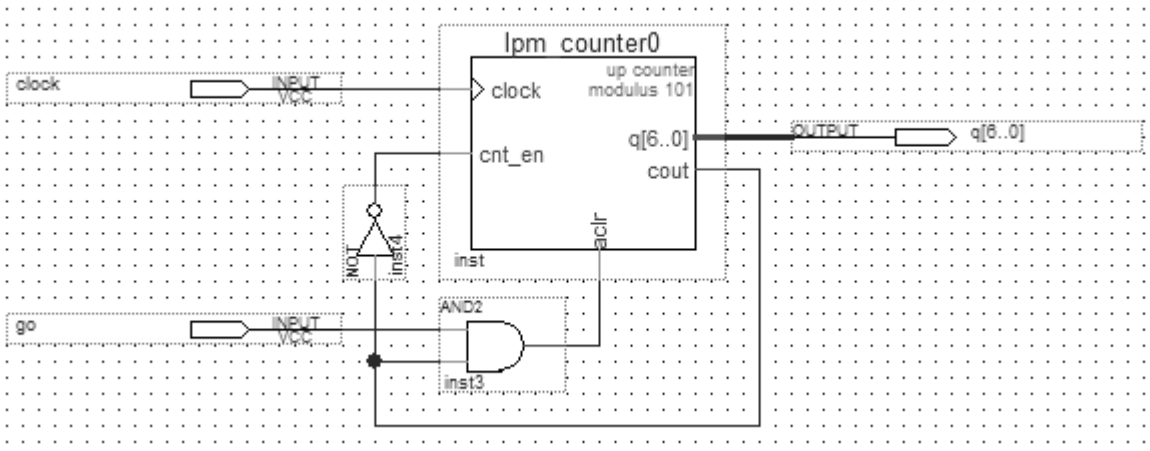
13.2 Mod-200 binary counter

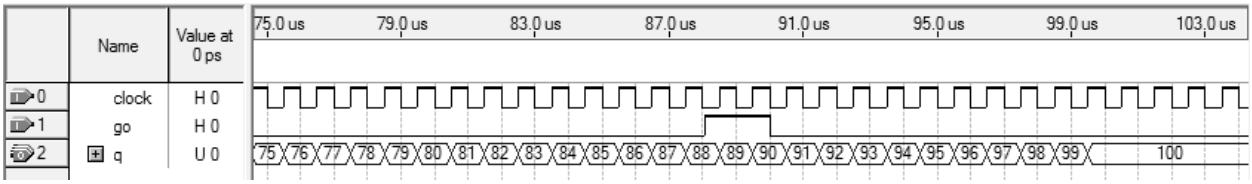


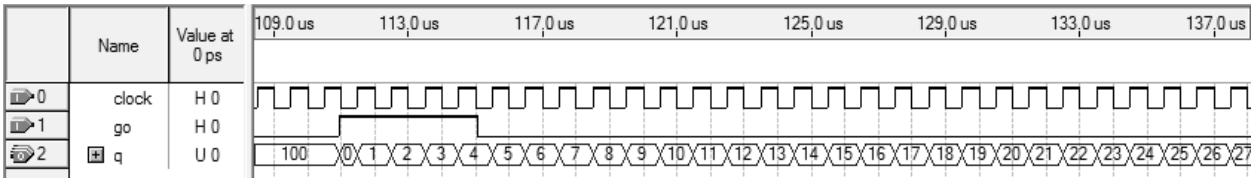




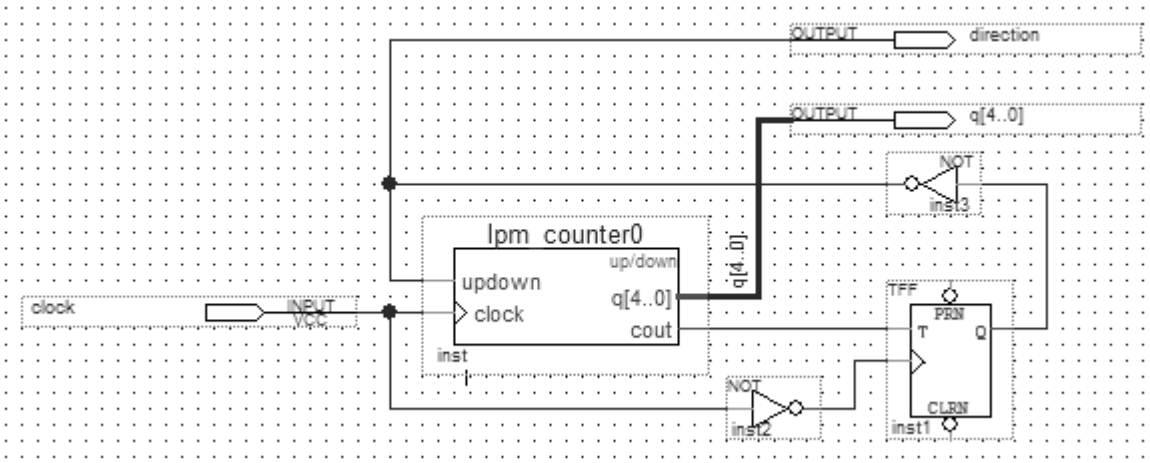
13.3 Self-stopping binary counter

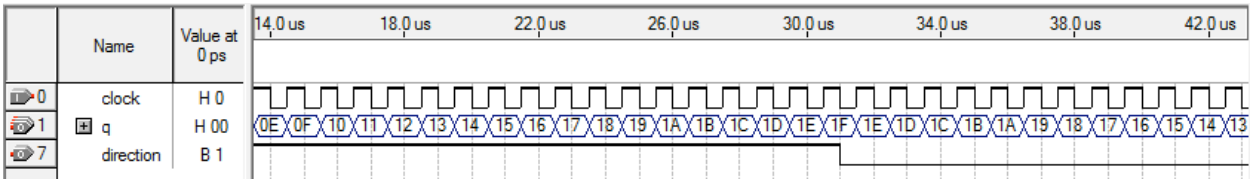


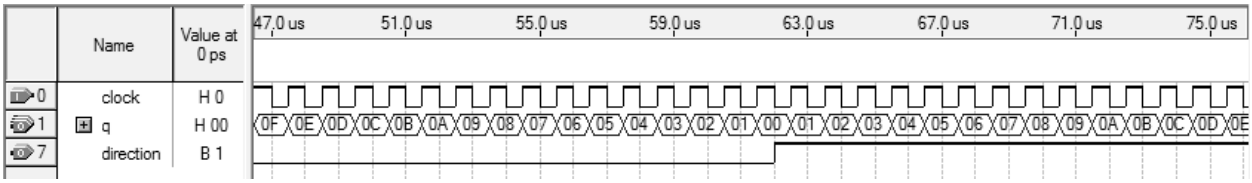




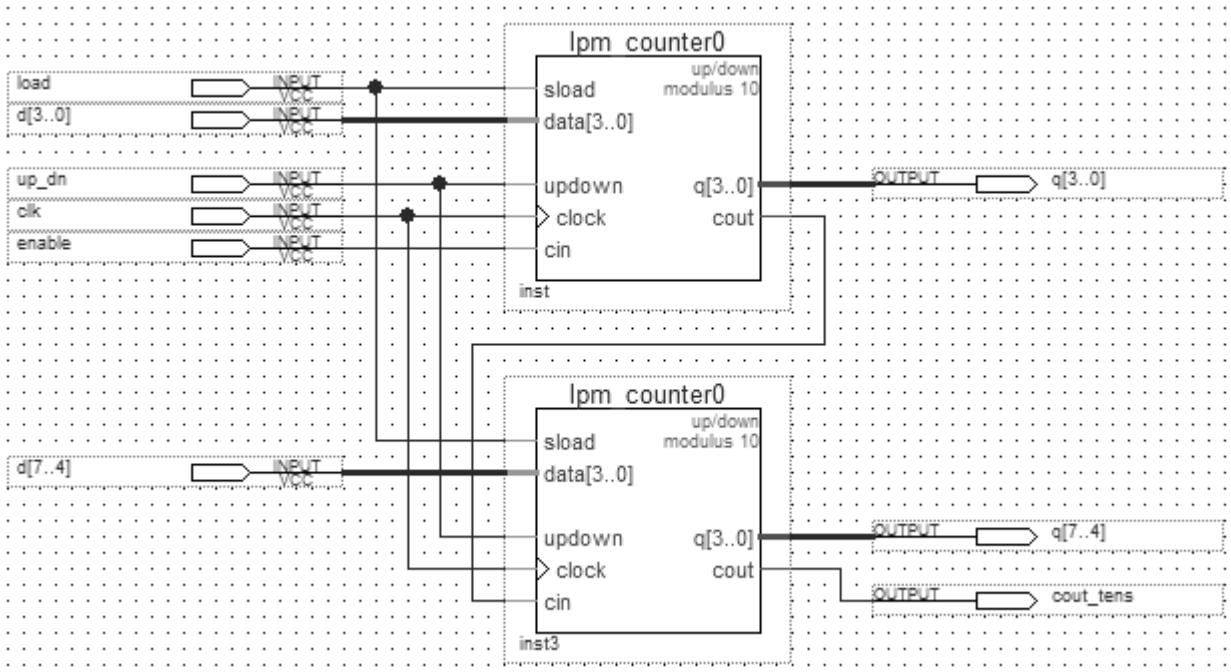
13.4 Automatic up/down counter

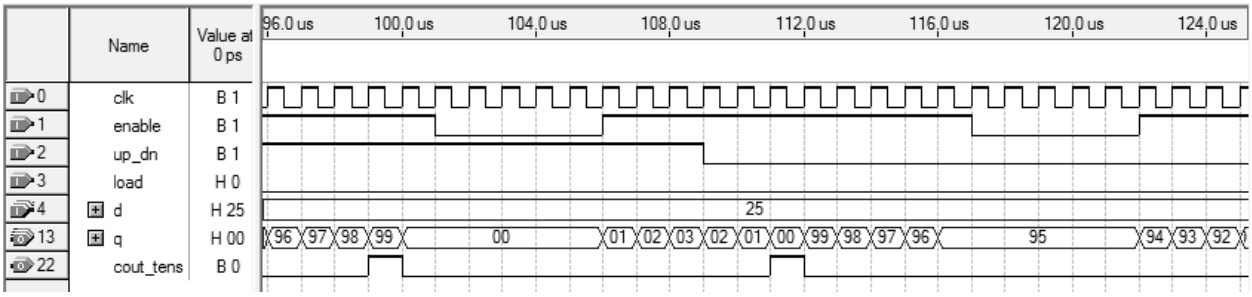


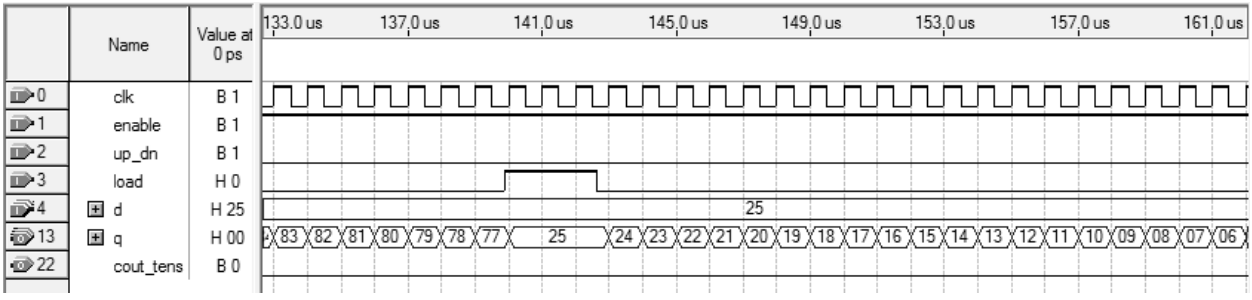




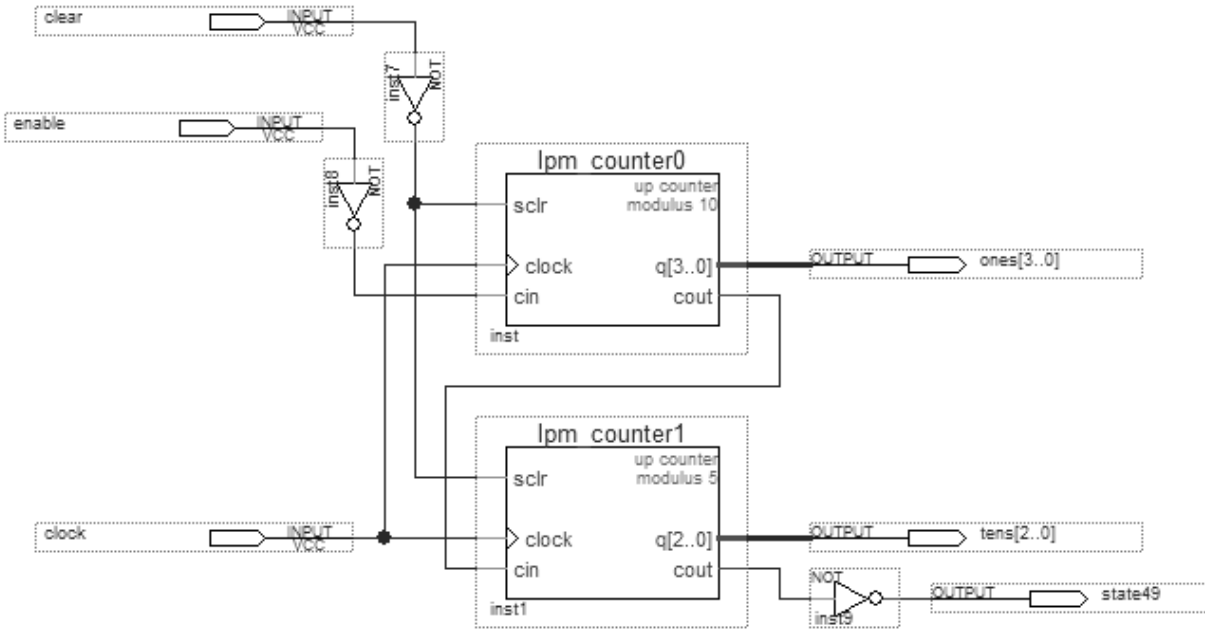
13.5 Mod-100, up/down, BCD counter

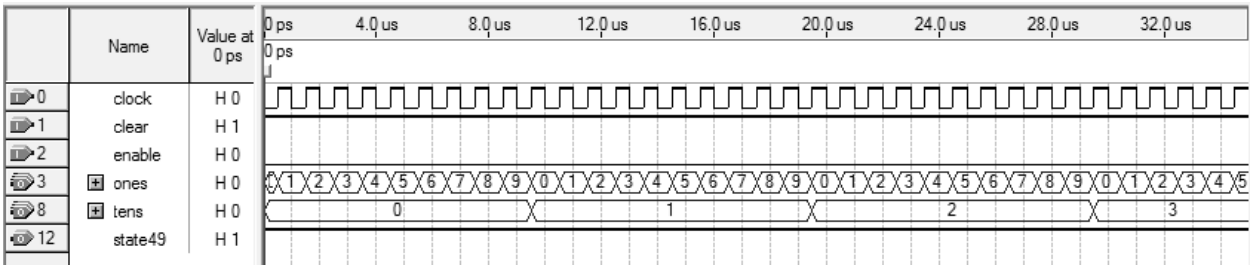


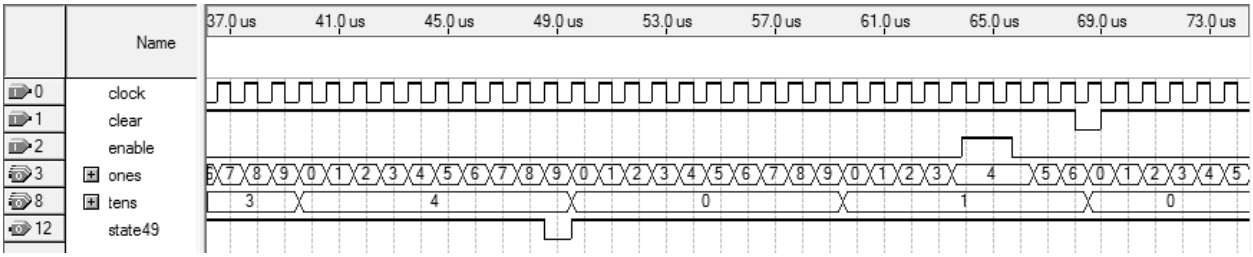




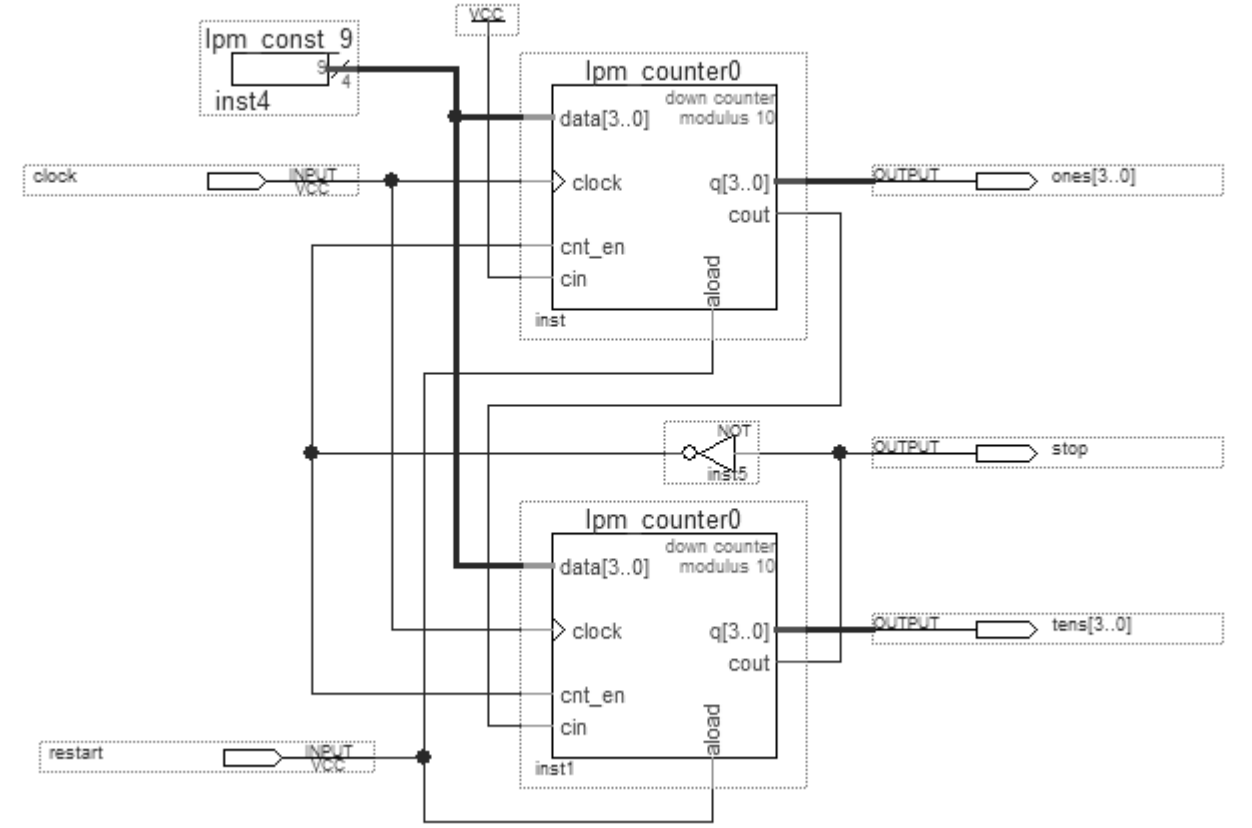
13.6 Mod-50 BCD counter

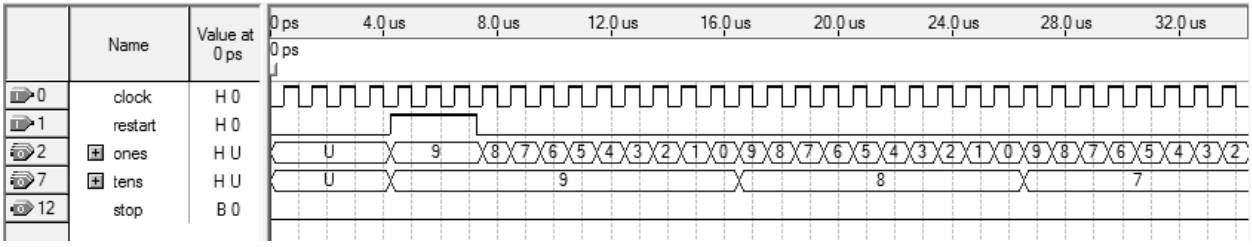


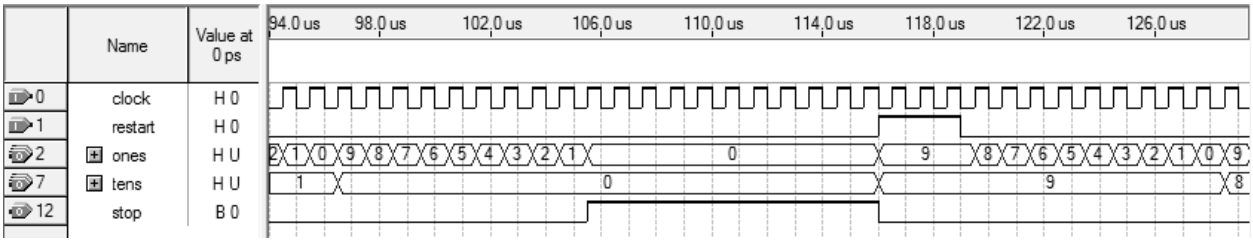




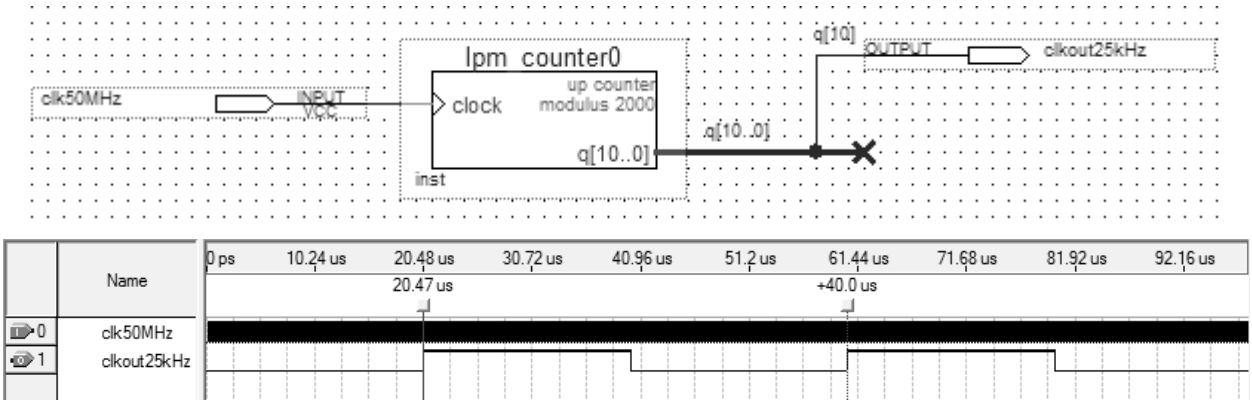
13.7 Mod-100, self-stopping, BCD down-counter



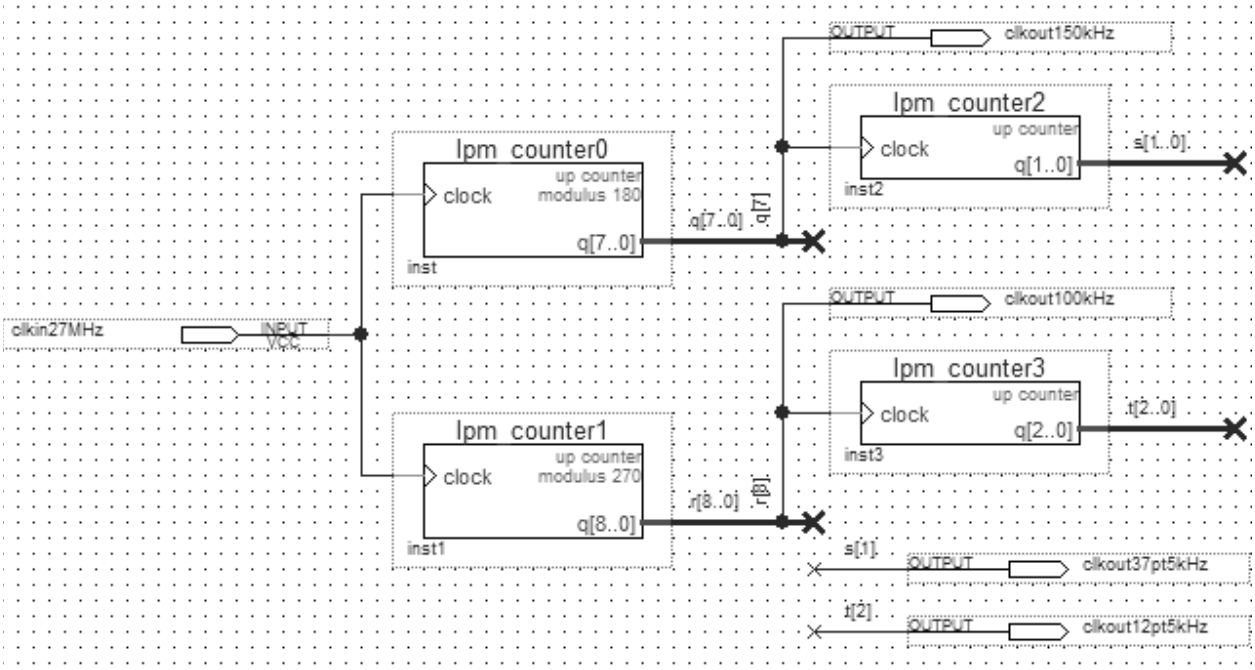


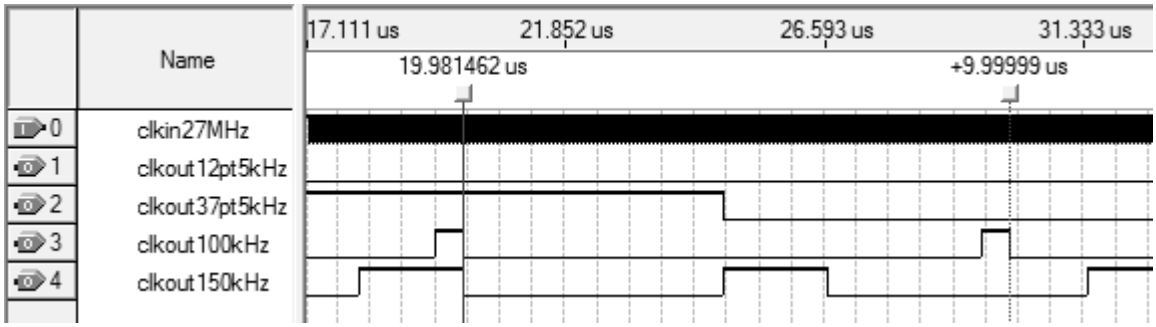


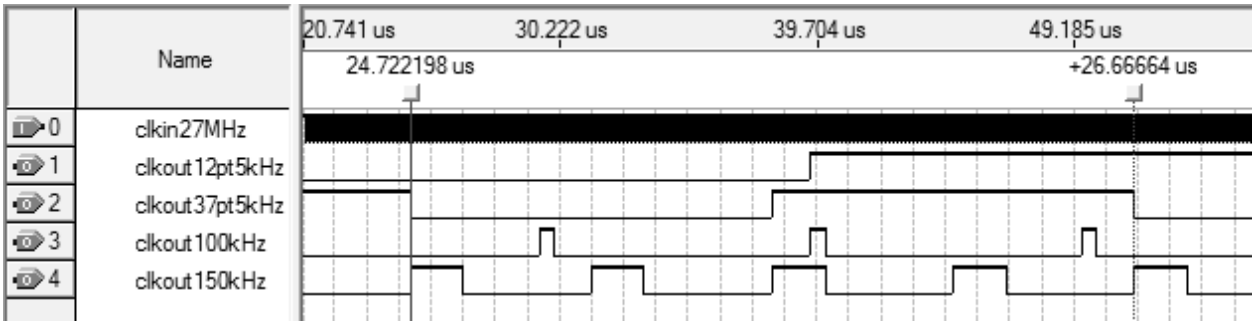
13.8 Clock frequency divider

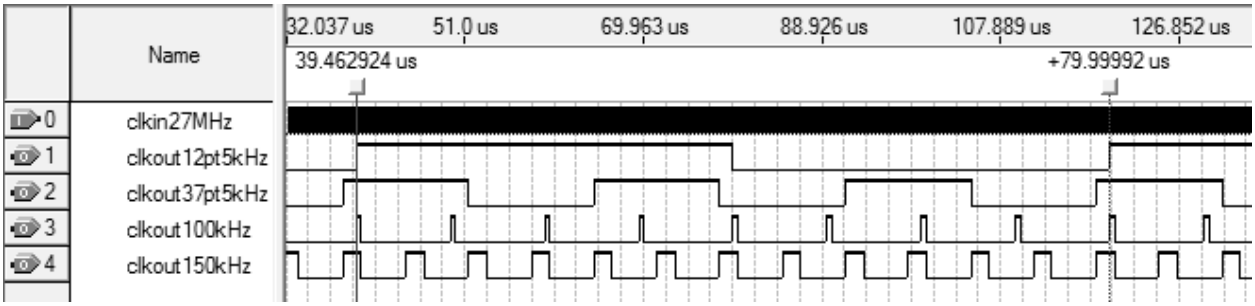


13.9 Frequency divider

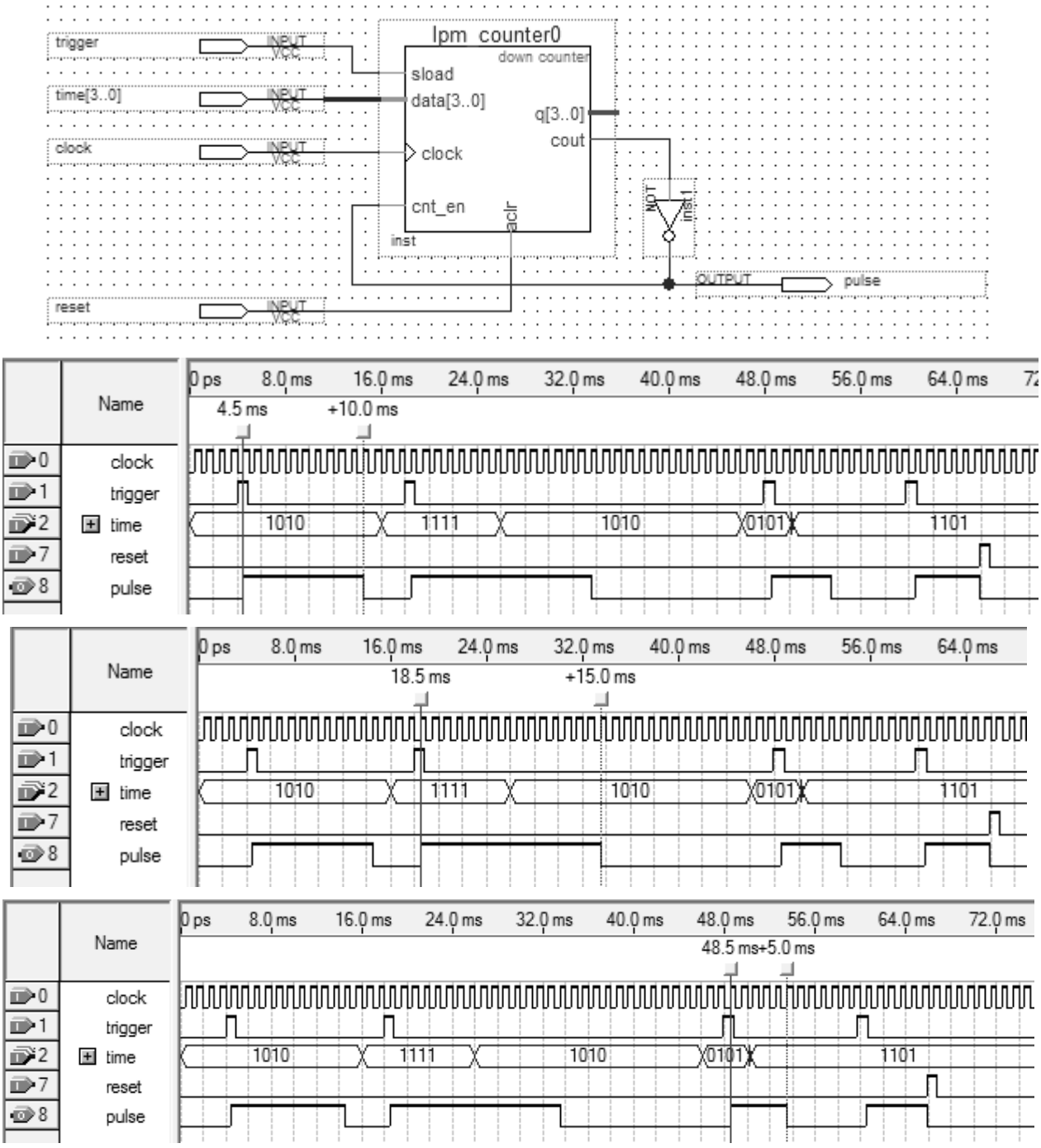



Name
17.111 us
21.852 us
26.593 us
31.333 us
19.981462 us
+9.99999 us
0
clkin27MHz
1
clkout12pt5kHz
2
clkout37pt5kHz
3
clkout100kHz
4
clkout150kHz


Name
20.741 us
30.222 us
39.704 us
49.185 us
24.722198 us
+26.66664 us
0
clkin27MHz
1
clkout12pt5kHz
2
clkout37pt5kHz
3
clkout100kHz
4
clkout150kHz


Name
32.037 us
51.0 us
69.963 us
88.926 us
107.889 us
126.852 us
39.462924 us
+79.99992 us
0
clkin27MHz
1
clkout12pt5kHz
2
clkout37pt5kHz
3
clkout100kHz
4
clkout150kHz

13.10 One-shot counter



Using a 1-kHz clock, this circuit will produce output pulses from 1 ms to 15 ms wide.

Unit 14A Counter Designs using AHDL

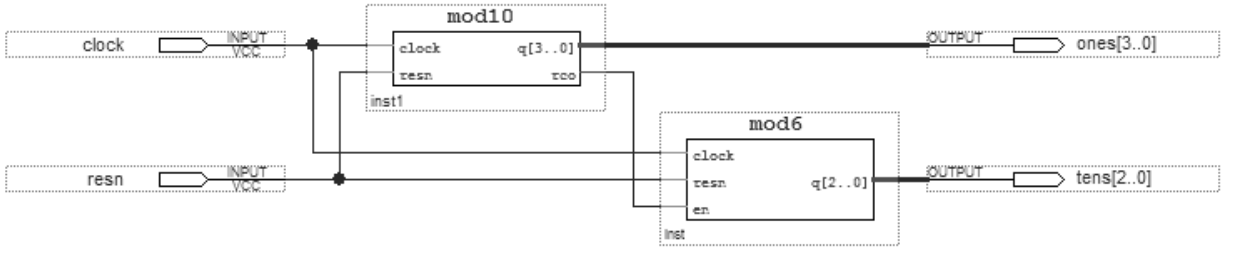
14A.1 Binary down counter

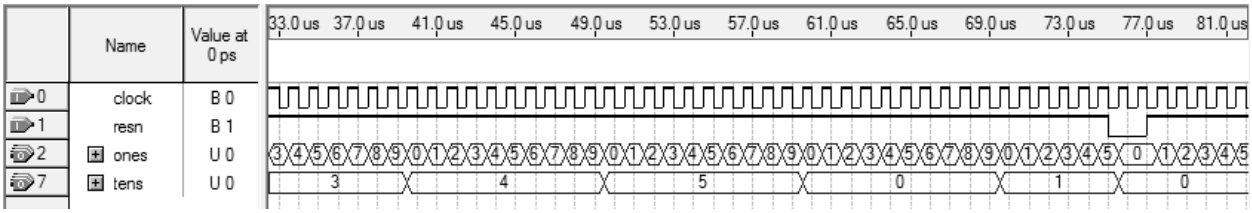
```
SUBDESIGN mod16dwn
(     clock, ld, en, data[3..0]        :INPUT;
      q[3..0], last                    :OUTPUT;   )
VARIABLE
      q[3..0]                          :DFF;
BEGIN
      q[].clk = clock;            -- connect clock

      IF ld == VCC     THEN       -- load is highest priority
            q[].d = data[];       -- synchronous load
      ELSIF en == GND  THEN       -- active-low count enable
            q[].d = q[].q - 1;    -- count down
      ELSE
            q[].d = q[].q;        -- hold count
      END IF;

      IF q[].q == 0    THEN       -- detect terminal state
            last = VCC;           -- active-high output
      ELSE
            last = GND;           -- if not the last state
      END IF;
END;
```

14A.2 Up/down BCD counter

```
SUBDESIGN  updn_mod10
(
      clock, c[1..0]              :INPUT;
      count[3..0], carryn         :OUTPUT;
)
VARIABLE
      count[3..0]                 :DFF;      -- counter FFs
BEGIN
      DEFAULTS
           carryn = VCC;          -- inactive output
      END DEFAULTS;

      count[].clk  = clock;       -- clock connection

      CASE  c[]  IS               -- determine control
         WHEN  H"0"  =>           -- reset
                       count[].d = 0;
         WHEN  H"1"  =>           -- count down
            IF     count[] == 0  THEN   -- at terminal state?
                       count[].d = 9;  carryn = GND;
            ELSE   count[].d = count[].q - 1;      -- decr.
            END IF;
         WHEN  H"2"  =>           -- count up
            IF     count[] == 9  THEN   -- at terminal state?
                       count[].d = 0;  carryn = GND;
            ELSE   count[].d = count[].q + 1;      -- incr.
            END IF;
         WHEN  H"3"  =>           -- hold count
                       count[].d = count[].q;
      END CASE;
END;
```

14A.3 Mod-60 BCD counter





```
SUBDESIGN  mod10
(
      clock, resn           :INPUT;
      q[3..0], rco          :OUTPUT;
)
VARIABLE
      q[3..0]               :DFF;      -- 4 FFs
BEGIN
      DEFAULTS
            rco = GND;            -- inactive output
      END DEFAULTS;

      q[].clk  = clock;

      IF  !resn  THEN           -- active-low reset
            q[].d = 0;
      ELSIF  q[].q == 9  THEN   -- at terminal state?
            q[].d = B"0000";    -- recycle
            rco = VCC;          -- assert rco
      ELSE
            q[].d = q[].q + 1;  -- increment count
      END IF;
END;
```

```
SUBDESIGN  mod6
(
      clock, resn, en           :INPUT;
      q[2..0]                   :OUTPUT;
)
VARIABLE
      q[2..0]                   :DFF;      -- 3 FFs
BEGIN
      q[].clk  = clock;

      IF  !resn  THEN           -- active-low reset
            q[].d = 0;
      ELSIF  en  THEN           -- increment tens digit
            IF  q[].q == 5  THEN      -- at terminal state?
                  q[].d = B"000";     -- recycle
            ELSE
                  q[].d = q[].q + 1;  -- increment count
            END IF;
      ELSE
            q[].d = q[].q;            -- hold current state
      END IF;
END;
```

14A.4 Mod-100 binary counter

```
SUBDESIGN  mod100
(
     clock, enable, w      :INPUT;
     q[7..0], pulse        :OUTPUT;
)
VARIABLE
     q[7..0]               :DFF;      -- 8 FFs
BEGIN
     q[].clk = !clock;                -- clock on NGT

     IF  enable  THEN                 -- count enabled?
          IF  q[].q == 99  THEN       -- at terminal state?
               q[].d = 0;             -- recycle
          ELSE
               q[].d = q[].q + 1;     -- increment count
          END IF;
     ELSE
          q[].d = q[].q;             -- hold count
     END IF;

          -- control pulse width of output
     IF    !w & q[].q >= 95  THEN  pulse = VCC;
     ELSIF  w & q[].q >= 90  THEN  pulse = VCC;
     ELSE   pulse = GND;
     END IF;
END;
```

14A.5 Stepper motor sequence controller

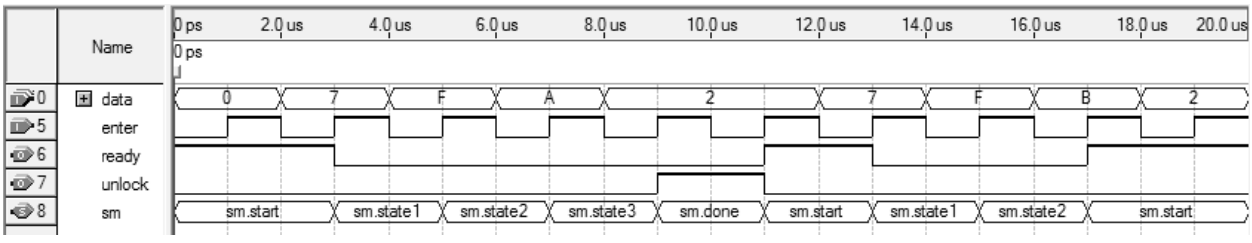
```
SUBDESIGN  'half-step'
(
      step, cw, go          :INPUT;
      q[3..0]               :OUTPUT;
)

VARIABLE
      stepper:  MACHINE OF BITS (q[3..0])   -- state machine
                 WITH STATES (   -- define stepper states
                 initial = B"0000",
                       s1 = B"0101",
                       s2 = B"0001",
                       s3 = B"1001",
                       s4 = B"1000",
                       s5 = B"1010",
                       s6 = B"0010",
                       s7 = B"0110",
                       s8 = B"0100");
BEGIN

      stepper.clk = step;         -- clock port
      stepper.ena = go;           -- active-high enable port

      TABLE            -- present state/next state table
            stepper,         cw         =>         stepper;
            initial,         X          =>         s1;
            s1,              1          =>         s2;
            s1,              0          =>         s8;
            s2,              1          =>         s3;
            s2,              0          =>         s1;
            s3,              1          =>         s4;
            s3,              0          =>         s2;
            s4,              1          =>         s5;
            s4,              0          =>         s3;
            s5,              1          =>         s6;
            s5,              0          =>         s4;
            s6,              1          =>         s7;
            s6,              0          =>         s5;
            s7,              1          =>         s8;
            s7,              0          =>         s6;
            s8,              1          =>         s1;
            s8,              0          =>         s7;
      END TABLE;
END;
```

14A.6 Variable frequency divider

```
SUBDESIGN  divider
(
    freq_in, m[1..0]                              :INPUT;
    freq_out                                    :OUTPUT;
)
VARIABLE
    count[4..0]                                 :DFF;
BEGIN
    count[].clk = freq_in;

    CASE m[] IS
        WHEN 0 =>
            freq_out = count[2];
            IF count[] == 4  THEN     -- terminal state?
                count[].d = 0;        -- recycle
            ELSE
                count[].d = count[].q + 1;  -- count up
            END IF;
        WHEN 1 =>
            freq_out = count[3];
            IF count[] == 9  THEN     -- terminal state?
                count[].d = 0;        -- recycle
            ELSE
                count[].d = count[].q + 1;  -- count up
            END IF;
        WHEN 2 =>
            freq_out = count[4];
            IF count[] == 19  THEN    -- terminal state?
                count[].d = 0;        -- recycle
            ELSE
                count[].d = count[].q + 1;  -- count up
            END IF;
        WHEN 3 =>
            freq_out = count[4];
            IF count[] == 24  THEN    -- terminal state?
                count[].d = 0;        -- recycle
            ELSE
                count[].d = count[].q + 1;  -- count up
            END IF;
    END CASE;
END;
```

14A.7 Digital lock

```
CONSTANT  comb1 = H"7";
CONSTANT  comb2 = H"F";
CONSTANT  comb3 = H"A";
CONSTANT  comb4 = H"2";
          -- define desired lock combination
SUBDESIGN  lock
(enter                :INPUT;
 data[3..0]           :INPUT;
 unlock, ready        :OUTPUT;)
VARIABLE
      sm              :MACHINE
            WITH STATES (start, state1, state2, state3, done);
BEGIN
      DEFAULTS
            unlock = GND;
            ready = GND;
      END DEFAULTS;
      sm.clk = enter;
      CASE  sm  IS
            WHEN start =>          -- beginning state
                  ready = VCC;     -- ready to enter code
                  IF  data[] == comb1  THEN  sm = state1;
                  ELSE  sm = start;
                  END IF;
            WHEN state1 =>         -- 1st number ok
                  IF  data[] == comb2  THEN  sm = state2;
                  ELSE  sm = start;     -- wrong number
                  END IF;
            WHEN state2 =>         -- 2nd number ok
                  IF  data[] == comb3  THEN  sm = state3;
                  ELSE  sm = start;
                  END IF;
            WHEN state3 =>         -- 3rd number ok
                  IF  data[] == comb4  THEN  sm = done;
                  ELSE  sm = start;
                  END IF;
            WHEN done=>      -- 4th number ok
                  sm = start;      -- relock when press enter
                  unlock = VCC;    -- unlock
      END CASE;
END;
```



14A.8 Programmable frequency divider

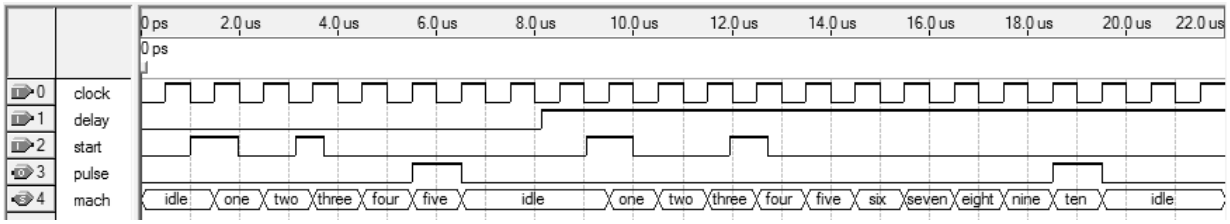
```
SUBDESIGN  divide_by
(
      freq_in, b[7..0]      :INPUT;
      freq_out              :OUTPUT;
)

VARIABLE
      divide_by[7..0]       :DFF;      -- 8 FFs

BEGIN
      divide_by[].clk  = freq_in;

      IF  divide_by[] <= 1  THEN       -- detect last state
            divide_by[].d = b[];       -- reload number
            freq_out = VCC;            -- make pulse
      ELSE  divide_by[].d = divide_by[].q - 1;    -- decr.
      END IF;
END;
```

14A.9 State machine

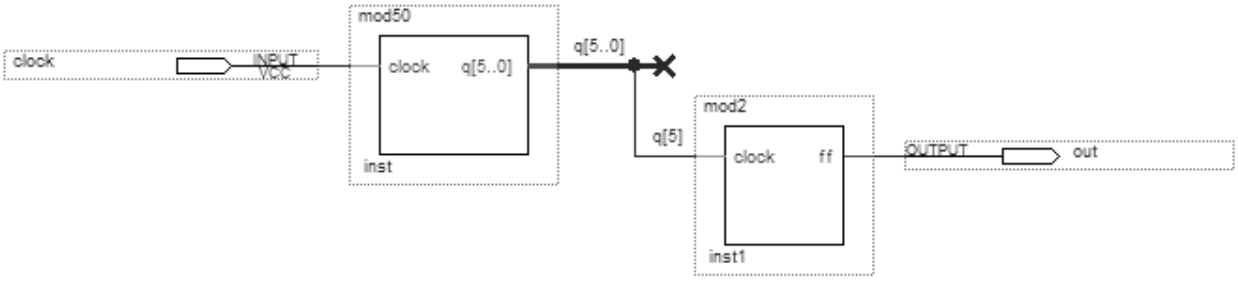
```
SUBDESIGN  state
(
      clock, r, s, t        :INPUT;
      q[2..0]               :OUTPUT;
)

VARIABLE
      sm    :MACHINE OF BITS (q[2..0])
                  WITH STATES   (s0 = B"000",
                                 s1 = B"001",
                                 s2 = B"010",
                                 s3 = B"011",
                                 s4 = B"100",
                                 s5 = B"101",
                                 s6 = B"110",
                                 s7 = B"111");

BEGIN
      sm.clk = clock;

      TABLE      -- present state/next state table
            sm, r, s, t           =>         sm;
            s0, X, X, X           =>         s1;
            s1, X, X, X           =>         s7;
            s2, X, X, X           =>         s1;
            s3, 1, X, X           =>         s3;
            s3, 0, 1, X           =>         s4;
            s3, 0, 0, X           =>         s7;
            s5, X, X, X           =>         s1;
            s4, X, X, X           =>         s3;
            s6, X, X, X           =>         s4;
            s7, X, X, 0           =>         s1;
            s7, X, X, 1           =>         s6;
      END TABLE;

END;
```

14A.10 Variable pulse delay

```
SUBDESIGN  varypulsedelay
(
      clock           :INPUT;
      start           :INPUT;
      delay           :INPUT;
      pulse           :OUTPUT;
)
VARIABLE
      mach       :MACHINE         -- "buried" machine
                 WITH STATES (
                      idle, one, two, three, four, five,
                      six, seven, eight, nine, ten);
BEGIN
      DEFAULTS
      pulse = GND;
      END DEFAULTS;
      mach.clk = clock;

      CASE  mach  IS
            WHEN idle =>     -- mach waits here
                 IF start THEN   mach = one;
                 ELSE            mach = idle;
                 END IF;
            WHEN one =>          mach = two;
            WHEN two =>          mach = three;
            WHEN three =>        mach = four;
            WHEN four =>         mach = five;
            WHEN five =>
                 IF delay == GND  THEN
                                  mach = idle;
                                  pulse = VCC;
                 ELSE             mach = six;
                 END IF;
            WHEN six =>          mach = seven;
            WHEN seven =>        mach = eight;
            WHEN eight =>        mach = nine;
            WHEN nine =>         mach = ten;
            WHEN ten =>          mach = idle;
                 IF delay == VCC  THEN
                                  pulse = VCC;
                 END IF;
      END CASE;
END;
```



14A.11 Frequency divider




```
SUBDESIGN  mod50
(
      clock             :INPUT;
      q[5..0]           :OUTPUT;
)
VARIABLE
      q[5..0]           :DFF;
BEGIN
      q[].clk = clock;

      IF q[] == 49     THEN
            q[].d = 0;
      ELSE
            q[].d = q[].q + 1;
      END IF;
END;
```

```
SUBDESIGN  mod2
(
      clock             :INPUT;
      ff                :OUTPUT;
)
VARIABLE
      ff                :DFF;
BEGIN
      ff.clk = clock;

      ff.d = !ff.q;
END;
```

14A.12 Clock frequency divider

```
SUBDESIGN  clock_divider
(
      clock_50MHz                         :INPUT;
      out_25kHz, out_10kHz                :OUTPUT;
)
VARIABLE
      q_25kHz[10..0], q_10kHz[12..0]   :DFF;
BEGIN
      q_25kHz[].clk = clock_50MHz;
      q_10kHz[].clk = clock_50MHz;

      out_25kHz = q_25kHz[10].q;  -- only need MSB
      out_10kHz = q_10kHz[12].q;  -- from 2 counters

      IF    q_25kHz[].q == 1999   THEN  -- mod 2000
            q_25kHz[].d = 0;
      ELSE
            q_25kHz[].d = q_25kHz[].q + 1;
      END IF;

      IF    q_10kHz[].q == 4999 THEN    -- mod 5000
            q_10kHz[].d = 0;
      ELSE
            q_10kHz[].d = q_10kHz[].q + 1;
      END IF;

END;
```

14A.13 Gray code counter

```
SUBDESIGN  gray
(
      clock, dir            :INPUT;
      q[3..0], index        :OUTPUT;
)

VARIABLE
      gray:  MACHINE OF BITS (q[3..0])      -- state machine
            WITH STATES (         -- gray code bit patterns
                  s0  = B"0000",
                  s1  = B"0001",
                  s2  = B"0011",
                  s3  = B"0010",
                  s4  = B"0110",
                  s5  = B"0111",
                  s6  = B"0101",
                  s7  = B"0100",
                  s8  = B"1100",
```

```
                s9  = B"1101",
                s10 = B"1111",
                s11 = B"1110",
                s12 = B"1010",
                s13 = B"1011",
                s14 = B"1001",
                s15 = B"1000");

BEGIN
     gray.clk = clock;          -- state machine clock

     TABLE           -- present state/next state table
          gray,      dir        =>         gray,      index;
          s0,        0          =>         s1,        0;
          s0,        1          =>         s15,       0;
          s1,        0          =>         s2,        1;
          s1,        1          =>         s0,        1;
          s2,        0          =>         s3,        1;
          s2,        1          =>         s1,        1;
          s3,        0          =>         s4,        1;
          s3,        1          =>         s2,        1;
          s4,        0          =>         s5,        1;
          s4,        1          =>         s3,        1;
          s5,        0          =>         s6,        1;
          s5,        1          =>         s4,        1;
          s6,        0          =>         s7,        1;
          s6,        1          =>         s5,        1;
          s7,        0          =>         s8,        1;
          s7,        1          =>         s6,        1;
          s8,        0          =>         s9,        1;
          s8,        1          =>         s7,        1;
          s9,        0          =>         s10,       1;
          s9,        1          =>         s8,        1;
          s10,       0          =>         s11,       1;
          s10,       1          =>         s9,        1;
          s11,       0          =>         s12,       1;
          s11,       1          =>         s10,       1;
          s12,       0          =>         s13,       1;
          s12,       1          =>         s11,       1;
          s13,       0          =>         s14,       1;
          s13,       1          =>         s12,       1;
          s14,       0          =>         s15,       1;
          s14,       1          =>         s13,       1;
          s15,       0          =>         s0,        1;
          s15,       1          =>         s14,       1;
     END TABLE;
END;
```

Unit 14V Counter Designs using VHDL

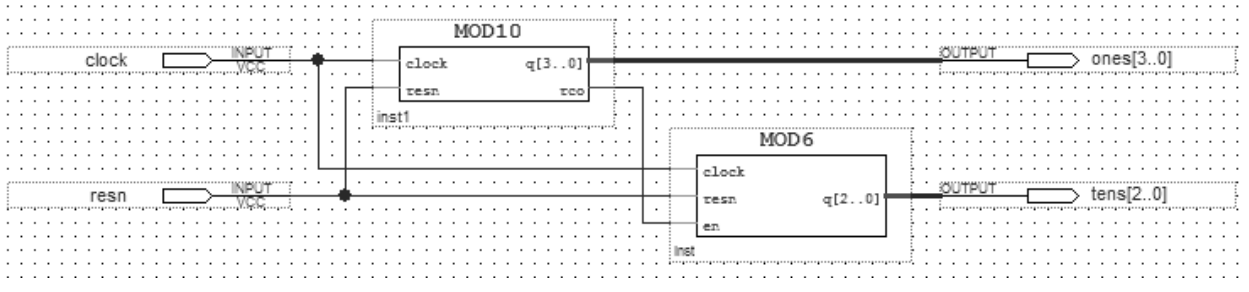
14V.1 Binary down counter

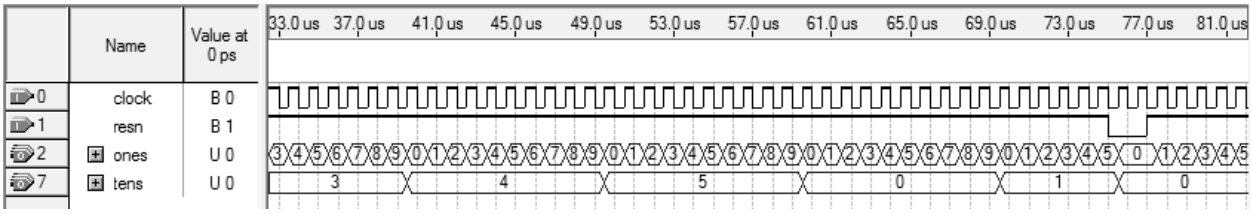
```
ENTITY mod16dwn IS
PORT( clock, load, en       :IN BIT;
           data             :IN INTEGER RANGE 0 TO 15;
           q                :OUT INTEGER RANGE 0 TO 15;
           last             :OUT BIT   );
END mod16dwn;

ARCHITECTURE down_count    OF mod16dwn IS
BEGIN
PROCESS (clock)                         -- wait for clock
VARIABLE   counter         :INTEGER RANGE 0 TO 15;
BEGIN
     IF (clock'EVENT AND clock = '1') THEN
           IF load = '1'    THEN        -- load is highest priority
                counter := data;       -- synchronous load
           ELSIF en = '0'   THEN        -- active-low count enable
                counter := counter - 1;    -- count down
           END IF;                      -- hold is implied
     END IF;
     q <= counter;                      -- connect to output port
     IF  counter = 0  THEN              -- detect terminal state
           last <= '1';                 -- active-high
     ELSE
           last <= '0';
     END IF;
END PROCESS;
END;
```

14V.2 Up/down BCD counter

```
ENTITY updn_mod10 IS
PORT (
   clock            :IN BIT;
   c                :IN BIT_VECTOR (1 DOWNTO 0);
   count            :OUT INTEGER RANGE 0 TO 15;
   carryn           :OUT BIT
);
END updn_mod10;

ARCHITECTURE vhdl OF updn_mod10 IS
BEGIN
PROCESS (clock, c)       -- clock or control invokes process
VARIABLE bcd       :INTEGER RANGE 0 TO 9;  -- counter variable
BEGIN
   IF (clock'EVENT AND clock = '1') THEN       -- on PGT
         CASE  c  IS      -- determine control input
               WHEN  "00"  =>        -- reset
                                          bcd := 0;
               WHEN  "01"  =>        -- count down
                     IF (bcd = 0) THEN     bcd := 9;
                     ELSE                  bcd := bcd - 1;
                     END IF;
               WHEN  "10"  =>        -- count up
                     IF (bcd = 9) THEN     bcd := 0;
                     ELSE                  bcd := bcd + 1;
                     END IF;
               WHEN  "11"  =>        -- hold count
                                          bcd := bcd;
         END CASE;
   END IF;
   count <= bcd;                           -- output counter

         -- assert active-low ripple carry output
   IF    (c = "01" AND bcd = 0)     THEN       carryn <= '0';
   ELSIF (c = "10" AND bcd = 9)     THEN       carryn <= '0';
   ELSE                                        carryn <= '1';
   END IF;
END PROCESS;
END vhdl;
```

14V.3 Mod-60 BCD counter






```
ENTITY mod10 IS
PORT (
      clock, resn           :IN BIT;
      q                     :OUT INTEGER RANGE 0 TO 9;
      rco                   :OUT BIT   );
END mod10;

ARCHITECTURE vhdl OF mod10 IS
BEGIN
      PROCESS (clock)
      VARIABLE count        :INTEGER RANGE 0 TO 9;
      BEGIN

            IF (clock'EVENT AND clock = '1')       THEN
                  IF (resn = '0' OR count = 9)     THEN
                              count := 0;      -- reset/recycle
                  ELSE        count := count + 1;   -- increment
                  END IF;
            END IF;

            IF (count = 9)   THEN  rco <= '1';      -- RCO
            ELSE rco <= '0';
            END IF;

      q <= count;                -- output count

      END PROCESS;
END vhdl;
```

```
ENTITY mod6 IS
PORT (
      clock, resn, en       :IN BIT;
      q                     :OUT INTEGER RANGE 0 TO 5  );
END mod6;

ARCHITECTURE vhdl OF mod6 IS
BEGIN
      PROCESS (clock)
      VARIABLE count        :INTEGER RANGE 0 TO 5;      -- mod-6
      BEGIN

            IF (clock'EVENT AND clock = '1') THEN
                  IF (resn = '0')  THEN  count := 0;      -- reset
                  ELSIF (en = '1') THEN             -- enabled
                        IF (count = 5)   THEN
                              count := 0;           -- recyle
                        ELSE
                              count := count + 1;   -- increment
                        END IF;
                  END IF;
            END IF;

      q <= count;           -- output count

      END PROCESS;
END vhdl;
```

14V.4 Mod-100 binary counter

```
ENTITY mod100 IS
PORT (
     clock, enable, w      :IN BIT;
     q                     :OUT INTEGER RANGE 0 TO 99;
     pulse                 :OUT BIT);
END mod100;

ARCHITECTURE vhdl OF mod100 IS
BEGIN
     PROCESS (clock)
     VARIABLE count        :INTEGER RANGE 0 TO 99;
     BEGIN

          IF (clock'EVENT AND clock = '0') THEN       -- on NGT
               IF (enable = '1')     THEN       -- enabled?
                    IF (count = 99)  THEN
                         count := 0;           -- recycle
                    ELSE
                         count := count + 1;   -- increment
                    END IF;
               END IF;
          END IF;

     q <= count;                        -- output count

          -- control pulse width of output
     IF (w = '0' AND count >= 95)     THEN  pulse <= '1';
     ELSIF (w = '1' AND count >= 90)  THEN  pulse <= '1';
     ELSE  pulse <= '0';
     END IF;

     END PROCESS;
END vhdl;
```

14V.5 Stepper motor sequence controller

```
ENTITY half_step IS
PORT (
     step, cw, go          :IN BIT;
     q                     :OUT BIT_VECTOR (3 DOWNTO 0));
END half_step;

ARCHITECTURE vhdl OF half_step IS
BEGIN
PROCESS (step)
VARIABLE stepper           :BIT_VECTOR (3 DOWNTO 0);
BEGIN

     IF (step'EVENT AND step = '1')   THEN
         IF (go = '1' AND cw = '0')   THEN
             -- enabled for CCW steps
             CASE stepper IS    -- define CCW sequence
                  WHEN "0101"    =>    stepper := "0100";
                  WHEN "0100"    =>    stepper := "0110";
                  WHEN "0110"    =>    stepper := "0010";
                  WHEN "0010"    =>    stepper := "1010";
                  WHEN "1010"    =>    stepper := "1000";
                  WHEN "1000"    =>    stepper := "1001";
                  WHEN "1001"    =>    stepper := "0001";
                  WHEN "0001"    =>    stepper := "0101";
                  WHEN OTHERS    =>    stepper := "0101";
             END CASE;

         ELSIF (go = '1' AND cw = '1')     THEN
             -- enabled for CW steps
             CASE stepper IS    -- define CW sequence
                  WHEN "0101"    =>    stepper := "0001";
                  WHEN "0001"    =>    stepper := "1001";
                  WHEN "1001"    =>    stepper := "1000";
                  WHEN "1000"    =>    stepper := "1010";
                  WHEN "1010"    =>    stepper := "0010";
                  WHEN "0010"    =>    stepper := "0110";
                  WHEN "0110"    =>    stepper := "0100";
                  WHEN "0100"    =>    stepper := "0101";
                  WHEN OTHERS    =>    stepper := "0101";
             END CASE;
         END IF;
     END IF;

     q <= stepper;         -- output count

END PROCESS;
END vhdl;
```

14V.6 Variable frequency divider

```
ENTITY divider IS
PORT (
     freq_in               :IN BIT;
     m                     :IN BIT_VECTOR (1 DOWNTO 0);
     freq_out              :OUT BIT);
END divider;

ARCHITECTURE vhdl OF divider IS
BEGIN
     PROCESS (freq_in)
     VARIABLE count        :INTEGER RANGE 0 TO 24;
                     -- buried counter
     BEGIN

     IF (freq_in'EVENT AND freq_in = '1')  THEN
          -- determine if at selected terminal state
          IF    (m = "00" AND count = 4)   THEN
                          count := 0;
                          freq_out <= '1';
          ELSIF (m = "01" AND count = 9)   THEN
                          count := 0;
                          freq_out <= '1';
          ELSIF (m = "10" AND count = 19)  THEN
                          count := 0;
                          freq_out <= '1';
          ELSIF (m = "11" AND count = 24)  THEN
                          count := 0;
                          freq_out <= '1';
          ELSE       count := count + 1;   -- incr.
                          freq_out <= '0';
          END IF;
     END IF;

     END PROCESS;
END vhdl;
```

14V.7 Digital lock

```
ENTITY  lock  IS
PORT (enter           :IN BIT;
      data            :IN INTEGER RANGE 0 TO 15;
      unlock          :OUT BIT;
      ready           :OUT BIT);
END lock;

ARCHITECTURE vhdl OF lock IS
CONSTANT  comb1       :INTEGER RANGE 0 TO 15 := 7;
CONSTANT  comb2       :INTEGER RANGE 0 TO 15 := 15;
CONSTANT  comb3       :INTEGER RANGE 0 TO 15 := 10;
CONSTANT  comb4       :INTEGER RANGE 0 TO 15 := 2;
      -- define desired lock combination
BEGIN
PROCESS (enter)
TYPE  lock_state  IS (start, state1, state2, state3, done);
VARIABLE  sm     : lock_state;
BEGIN
      IF (sm = done)   THEN       unlock <= '1';   -- unlocked
      ELSE                        unlock <= '0';   -- locked
      END IF;
      IF (sm = start)  THEN       ready <= '1';    -- ready
      ELSE                        ready <= '0';
      END IF;
      IF (enter'EVENT AND enter = '1') THEN
            CASE  sm  IS
                  WHEN start =>         -- beginning state
                        IF  (data = comb1)    THEN  sm := state1;
                        ELSE  sm := start;    -- wrong number
                        END IF;
                  WHEN state1 =>        -- 1st number ok
                        IF  (data = comb2)  THEN  sm := state2;
                        ELSE  sm := start;    -- wrong number
                        END IF;
                  WHEN state2 =>        -- 2nd number ok
                        IF  (data = comb3)  THEN  sm := state3;
                        ELSE  sm := start;    -- wrong number
                        END IF;
                  WHEN state3 =>        -- 3rd number ok
                        IF  (data = comb4)  THEN  sm := done;
                        ELSE  sm := start;    -- wrong number
                        END IF;
                  WHEN done=> sm := start;    -- 4th number ok
            END CASE;
      END IF;
END PROCESS;
END vhdl;
```

14V.8 Programmable frequency divider

```
ENTITY  divide_by  IS
PORT (
      freq_in               :IN BIT;
      b                     :IN INTEGER RANGE 0 TO 255;
      freq_out              :OUT BIT
);
END divide_by;

ARCHITECTURE vhdl OF divide_by IS
BEGIN
      PROCESS (freq_in)
      VARIABLE  divider     :INTEGER RANGE 0 TO 255;
                -- variable modulus counter

      BEGIN
            IF (freq_in'EVENT AND freq_in='1')  THEN
                  IF  divider <= 1  THEN           -- reload
                        divider := b;
                  ELSE
                        divider := divider - 1;    -- decr.
                  END IF;
            END IF;

            IF divider <= 1  THEN      -- detect state 1
                        freq_out <= '1';
            ELSE
                        freq_out <= '0';
            END IF;

      END PROCESS;
END vhdl;
```

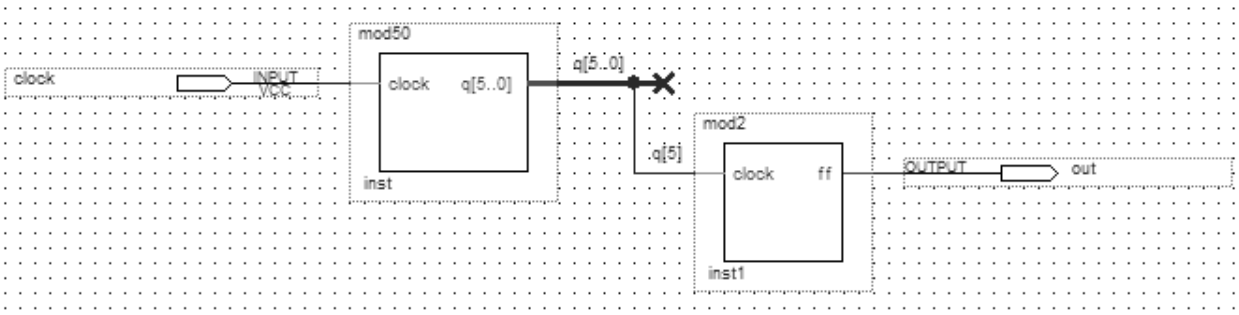
14V.9 State machine

```
ENTITY  state  IS
PORT (
      clock, r, s, t        :IN BIT;
      q                     :OUT BIT_VECTOR (2 DOWNTO 0)
);
END state;

ARCHITECTURE vhdl OF state IS
BEGIN
      PROCESS (clock)
      VARIABLE  sm          :BIT_VECTOR (2 DOWNTO 0);
      BEGIN

            IF (clock'EVENT AND clock = '1')  THEN
                  CASE sm    IS         -- define sequence
                        WHEN "001" =>    sm := "111";
                        WHEN "111" =>    -- conditional
                              IF (t = '1')  THEN
                                          sm := "110";
                              ELSE        sm := "001";
                              END IF;
                        WHEN "110" =>    sm := "100";
                        WHEN "100" =>    sm := "011";
                        WHEN "011" =>    -- conditional
                              IF (r = '0' AND s = '0')  THEN
                                          sm := "111";
                              ELSIF (r = '0' AND s = '1')  THEN
                                          sm := "100";
                              ELSE        sm := "011";
                              END IF;
                        WHEN OTHERS =>   sm := "001";
                              -- self-correcting & starting
                  END CASE;
            END IF;

      q <= sm;              -- output state machine

      END PROCESS;
END vhdl;
```

14V.10 Variable pulse delay

```
ENTITY  varypulsedelay  IS
PORT (    clock     :IN BIT;
          start     :IN BIT;
          delay     :IN BIT;
          pulse     :OUT BIT  );
END varypulsedelay;
ARCHITECTURE project14_10 OF varypulsedelay IS
BEGIN
PROCESS (clock, delay)
TYPE states IS (idle, one, two, three, four, five,
                    six, seven, eight, nine, ten);
VARIABLE mach  : states;
BEGIN
     IF (clock'EVENT AND clock = '1')   THEN
          CASE  mach  IS
               WHEN idle =>   -- mach waits here
                    IF (start = '1')    THEN
                                        mach := one;
                    ELSE                mach := idle;
                    END IF;
               WHEN one   =>       mach := two;
               WHEN two   =>       mach := three;
               WHEN three =>       mach := four;
               WHEN four  =>       mach := five;
               WHEN five  =>
                    IF (delay = '0')    THEN
                                        mach := idle;
                    ELSE           mach := six;
                    END IF;
               WHEN six   =>       mach := seven;
               WHEN seven =>       mach := eight;
               WHEN eight =>       mach := nine;
               WHEN nine  =>       mach := ten;
               WHEN ten   =>       mach := idle;
          END CASE;
     END IF;
     IF (delay = '0' AND mach = five) THEN
               pulse <= '1';
     ELSIF (delay = '1' AND mach = ten) THEN
               pulse <= '1';
     ELSE      pulse <= '0';
     END IF;
END PROCESS;
END project14_10;
```

14V.11 Frequency divider




```
ENTITY  mod50  IS
PORT (clock          :IN BIT;
      q              :OUT INTEGER RANGE 0 TO 49);
END mod50;
ARCHITECTURE project14_11 OF mod50 IS
BEGIN
PROCESS (clock)
VARIABLE count       :INTEGER RANGE 0 TO 49;
BEGIN
     IF (clock'EVENT AND clock = '1')    THEN
          IF (count = 49)      THEN count := 0;
          ELSE                       count := count + 1;
          END IF;
     END IF;
     q <= count;
END PROCESS;
END project14_11;
```

```
ENTITY  mod2  IS
PORT (clock          :IN BIT;
      ff             :OUT BIT);
END mod2;
ARCHITECTURE block2 OF mod2 IS
BEGIN
     PROCESS (clock)
          VARIABLE flipflop   : BIT;
     BEGIN
          IF (clock'EVENT AND clock = '1')    THEN
               flipflop := NOT flipflop;
          END IF;
          ff <= flipflop;
     END PROCESS;
END block2;
```

14V.12 Clock frequency divider

```
ENTITY  clock_divider  IS
PORT(
     clock_50MHz          :IN BIT;
     out_25kHz       :OUT INTEGER RANGE 0 TO 2047;
     out_10kHz       :OUT INTEGER RANGE 0 TO 8191
);
END clock_divider;

ARCHITECTURE vhdl OF clock_divider IS
BEGIN
PROCESS (clock_50MHz)
VARIABLE  mod2000   :INTEGER RANGE 0 TO 2047;
VARIABLE  mod5000   :INTEGER RANGE 0 TO 8191;
BEGIN

IF (clock_50MHz'EVENT AND clock_50MHz = '1') THEN
     IF   mod2000 = 1999 THEN -- mod 2000
          mod2000 := 0;
     ELSE
          mod2000 := mod2000 + 1;
     END IF;

     IF   mod5000 = 4999 THEN -- mod 5000
          mod5000 := 0;
     ELSE
          mod5000 := mod5000 + 1;
     END IF;
END IF;

out_25kHz <= mod2000;    -- use MSB for 25 kHz signal
out_10kHz <= mod5000;    -- use MSB for 10 kHz signal

END PROCESS;
END vhdl;
```

14V.13 Gray code counter

```
ENTITY gray IS
PORT (
      clock, dir            :IN BIT;
      q                     :OUT BIT_VECTOR (3 DOWNTO 0);
      index                 :OUT BIT
);
END gray;

ARCHITECTURE vhdl OF gray IS
BEGIN
      PROCESS (clock)             -- detect clock change
            VARIABLE seq    :BIT_VECTOR (3 DOWNTO 0);
            -- declare bit vector variable for state machine
      BEGIN
            IF (clock'EVENT AND clock = '1')      THEN  -- PGT
                  IF (dir = '0')   THEN             -- count up
                        CASE seq IS     -- determine next state
                              WHEN "0000" =>  seq := "0001";
                              WHEN "0001" =>  seq := "0011";
                              WHEN "0011" =>  seq := "0010";
                              WHEN "0010" =>  seq := "0110";
                              WHEN "0110" =>  seq := "0111";
                              WHEN "0111" =>  seq := "0101";
                              WHEN "0101" =>  seq := "0100";
                              WHEN "0100" =>  seq := "1100";
                              WHEN "1100" =>  seq := "1101";
                              WHEN "1101" =>  seq := "1111";
                              WHEN "1111" =>  seq := "1110";
                              WHEN "1110" =>  seq := "1010";
                              WHEN "1010" =>  seq := "1011";
                              WHEN "1011" =>  seq := "1001";
                              WHEN "1001" =>  seq := "1000";
                              WHEN "1000" =>  seq := "0000";
                        END CASE;
                  ELSE                              -- count down
                        CASE seq IS     -- determine next state
                              WHEN "1000" =>  seq := "1001";
                              WHEN "1001" =>  seq := "1011";
                              WHEN "1011" =>  seq := "1010";
                              WHEN "1010" =>  seq := "1110";
                              WHEN "1110" =>  seq := "1111";
                              WHEN "1111" =>  seq := "1101";
                              WHEN "1101" =>  seq := "1100";
                              WHEN "1100" =>  seq := "0100";
                              WHEN "0100" =>  seq := "0101";
                              WHEN "0101" =>  seq := "0111";
                              WHEN "0111" =>  seq := "0110";
                              WHEN "0110" =>  seq := "0010";
```

```
                              WHEN "0010" =>   seq := "0011";
                              WHEN "0011" =>   seq := "0001";
                              WHEN "0001" =>   seq := "0000";
                              WHEN "0000" =>   seq := "1000";
                         END CASE;
                    END IF;
               END IF;
               q <= seq;        -- output counter variable to port
                    -- produce index signal
               IF (seq = "0000")     THEN       index <= '1';
               ELSE                             index <= '0';
               END IF;
          END PROCESS;
END vhdl;
```

Unit 15 Shift Register Applications

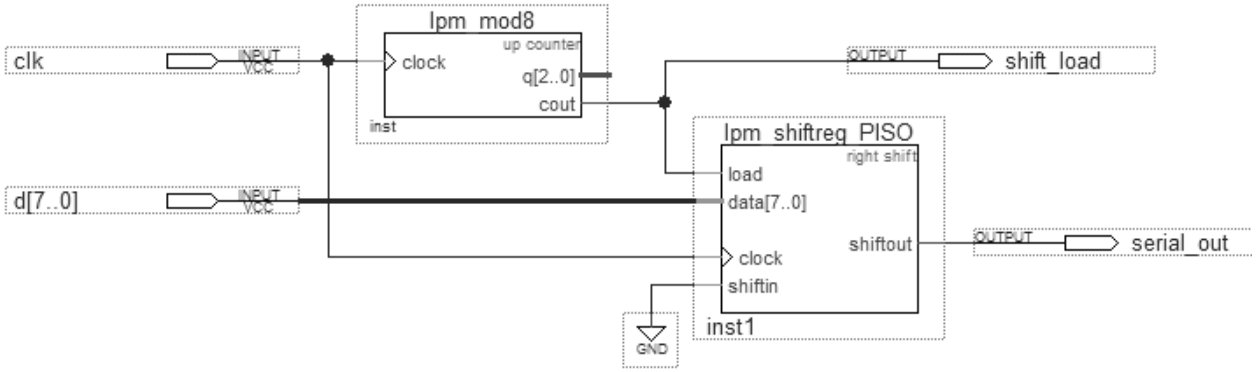
15.1 Waveform pattern generator

(a) HDL solutions

```
%    waveform pattern generator – AHDL solution    %
SUBDESIGN  pattern_gen_a
(
      clk, data[0..7]             :INPUT;
      serial_out, shift/load      :OUTPUT;
)

VARIABLE
      q[0..7]                :DFF;      -- shift register
      control                :MACHINE WITH STATES
                  (sh/ld, s1, s2, s3, s4, s5, s6, s7);
                        -- mod-8 control counter

BEGIN
      q[].clk = clk;
      control.clk = clk;
      serial_out = q7.q;

      TABLE      -- present state/next state table
            control    =>    control;
            sh/ld      =>    s1;
            s1         =>    s2;
            s2         =>    s3;
            s3         =>    s4;
            s4         =>    s5;
            s5         =>    s6;
            s6         =>    s7;
            s7         =>    sh/ld;
      END TABLE;

      IF (control == sh/ld)       THEN
            q[0..7].d = data[0..7];          -- parallel load
            shift/load = VCC;           -- shift/load pulse
      ELSE
            q[0..7].d = (GND, q[0..6].q);    -- shift data
            shift/load = GND;
      END IF;
END;
```

```
--  waveform pattern generator – VHDL solution
ENTITY  pattern_gen_v  IS
PORT (
     clk                         :IN BIT;
     data                        :IN BIT_VECTOR (0 TO 7);
     serial_out, shift_load      :OUT BIT
);
END pattern_gen_v;

ARCHITECTURE vhdl OF pattern_gen_v IS
BEGIN
PROCESS (clk)
     TYPE count_state IS
                (sh_ld, s1, s2, s3, s4, s5, s6, s7);
                     -- enumerated data type
     VARIABLE control            :count_state;
     VARIABLE  q                 :BIT_VECTOR (0 TO 7);
     VARIABLE ser                :BIT;
     BEGIN
     ser := '0';                 -- serial input
     serial_out <= q(7);         -- serial output

                -- define shift/load control signal
     IF (control = sh_ld)  THEN  shift_load <= '1';
     ELSE  shift_load <= '0';
     END IF;

     IF (clk'EVENT AND clk = '1')     THEN
          IF (control = sh_ld)  THEN
               q := data;                  -- parallel load
          ELSE  q := (ser & q(0 TO 6));    -- serial shift
          END IF;
                     -- define control sequence
          CASE  control  IS
               WHEN sh_ld      =>   control := s1;
               WHEN s1         =>   control := s2;
               WHEN s2         =>   control := s3;
               WHEN s3         =>   control := s4;
               WHEN s4         =>   control := s5;
               WHEN s5         =>   control := s6;
               WHEN s6         =>   control := s7;
               WHEN s7         =>   control := sh_ld;
          END CASE;
     END IF;

END PROCESS;
END vhdl;
```

15.1 Waveform pattern generator
(b) LPM schematic solution



15.2 Serial data buffer

```
SUBDESIGN  serial_data_a          -- AHDL solution
(
      clk, ser_data               :INPUT;
      pipo[0..7], new_data        :OUTPUT;
)

VARIABLE
      pipo[0..7]       :DFFE;     -- D FFs with enable
      sipo[0..7]       :DFF;
      control          :MACHINE WITH STATES
                             (s0, s1, s2, s3, s4, s5, s6, s7);

BEGIN
      pipo[].clk = clk;
      pipo[].ena = new_data;      -- enable pipo register
      pipo[].d = sipo[].q;        -- parallel out to parallel in
      sipo[].clk = clk;
      sipo[0..7].d = (ser_data, sipo[0..6].q);    -- ser shift
      control.clk = clk;
      new_data = (control == s7);      -- time to parallel load

      TABLE            -- present state/next state table
            control    =>    control;
            s0         =>    s1;
            s1         =>    s2;
            s2         =>    s3;
            s3         =>    s4;
            s4         =>    s5;
            s5         =>    s6;
            s6         =>    s7;
            s7         =>    s0;
      END TABLE;
END;
```

```
ENTITY  serial_data_v  IS        -- VHDL solution
PORT (
      clk, ser_data         :IN BIT;
      pipo                  :OUT BIT_VECTOR (0 TO 7);
      new_data              :OUT BIT
);
END serial_data_v;

ARCHITECTURE vhdl OF serial_data_v IS
BEGIN
PROCESS (clk)
      TYPE count_state IS (s0, s1, s2, s3, s4, s5, s6, s7);
                              -- enumerated data type
      VARIABLE  control     :count_state;
      VARIABLE  reg         :BIT_VECTOR (0 TO 7);
      VARIABLE  sipo        :BIT_VECTOR (0 TO 7);

      BEGIN
            pipo <= reg;     -- connect buried register to port
                 -- detect data load
            IF  (control = s7)  THEN  new_data <= '1';
            ELSE  new_data <= '0';
            END IF;

            IF  (clk'EVENT AND clk = '1')  THEN
                 sipo := (ser_data & sipo(0 TO 6));     -- shift
                 CASE  control  IS          -- control sequence
                       WHEN s0    =>    control := s1;
                       WHEN s1    =>    control := s2;
                       WHEN s2    =>    control := s3;
                       WHEN s3    =>    control := s4;
                       WHEN s4    =>    control := s5;
                       WHEN s5    =>    control := s6;
                       WHEN s6    =>    control := s7;
                       WHEN s7    =>    control := s0;
                 END CASE;
                 IF  (control = s7)  THEN
                             reg := sipo;     -- parallel load
                 END IF;
            END IF;

      END PROCESS;
END;
```

15.3 Barrel shifter

```
SUBDESIGN  barrel_shift
(        clk, s[2..0], ld, d[0..7]        :INPUT;
         q[0..7]                          :OUTPUT;    )
VARIABLE
         reg[0..7]                        :DFF;    -- register
BEGIN
      reg[].clk = clk;
      IF  ld  THEN  reg[0..7].d = d[0..7];      -- parallel load
      ELSE  -- shift data
            CASE  s[]  IS                       -- rotate s bits
               WHEN  0  =>  reg[0..7].d = reg[0..7].q;
               WHEN  1  =>  reg[0..7].d = (reg[1..7].q, reg[0].q);
               WHEN  2  =>  reg[0..7].d = (reg[2..7].q, reg[0..1].q);
               WHEN  3  =>  reg[0..7].d = (reg[3..7].q, reg[0..2].q);
               WHEN  4  =>  reg[0..7].d = (reg[4..7].q, reg[0..3].q);
               WHEN  5  =>  reg[0..7].d = (reg[5..7].q, reg[0..4].q);
               WHEN  6  =>  reg[0..7].d = (reg[6..7].q, reg[0..5].q);
               WHEN  7  =>  reg[0..7].d = (reg[7].q, reg[0..6].q);
            END CASE;
      END IF;
      q[] = reg[].q;    -- connect register to port
END;
```

```
ENTITY  barrel_shift  IS
PORT (clk, ld           :IN BIT;
      s                 :IN BIT_VECTOR (2 DOWNTO 0);
      d                 :IN BIT_VECTOR (0 TO 7);
      q                 :OUT BIT_VECTOR (0 TO 7));
END barrel_shift;
ARCHITECTURE vhdl OF barrel_shift IS
BEGIN
PROCESS (clk)
     VARIABLE  reg:BIT_VECTOR (0 TO 7);   -- register
     BEGIN
     IF  (clk'EVENT AND clk = '1')  THEN
         IF (ld = '1')  THEN  reg := d;   -- parallel load
         ELSE-- shift data
             CASE  s  IS      -- rotate s bits
                  WHEN  "000"  =>  reg := reg;
                  WHEN  "001"  =>  reg := (reg(1 TO 7) & reg(0));
                  WHEN  "010"  =>  reg := (reg(2 TO 7) & reg(0 TO 1));
                  WHEN  "011"  =>  reg := (reg(3 TO 7) & reg(0 TO 2));
                  WHEN  "100"  =>  reg := (reg(4 TO 7) & reg(0 TO 3));
                  WHEN  "101"  =>  reg := (reg(5 TO 7) & reg(0 TO 4));
                  WHEN  "110"  =>  reg := (reg(6 TO 7) & reg(0 TO 5));
                  WHEN  "111"  =>  reg := (reg(7) & reg(0 TO 6));
             END CASE;
         END IF;
     END IF;
     q <= reg;     -- connect register to port
END PROCESS;
END vhdl;
```

15.4 Mod-10 ring counter

```
SUBDESIGN  ring10          -- AHDL solution
(
      clk, clrn        :INPUT;
      q[0..9]          :OUTPUT;
)
VARIABLE
      q[0..9]          :DFF;      -- 10 bit register
      ser              :NODE;     -- serial input signal

BEGIN
      q[].clk = clk;
      q[].clrn = clrn;           -- asynchronous clear
      IF q[0..8].q == B"111111111"     THEN  ser = GND;
      ELSE                                   ser = VCC;
      END IF;                    -- determine feedback
      q[0..9].d = (ser, q[0..8].q);          -- shift
END;
```

```
ENTITY  ring10  IS          -- VHDL solution
PORT (
      Clk, clrn             :IN BIT;
      q                     :OUT BIT_VECTOR (0 TO 9)
);
END ring10;

ARCHITECTURE vhdl OF ring10 IS
SIGNAL  ser                 :BIT;      -- serial feedback signal
BEGIN
PROCESS (clk, clrn)
      VARIABLE  r           :BIT_VECTOR (0 TO 9);
      BEGIN
            IF r(0 TO 8) = "111111111" THEN  ser <= '0';
            ELSE                             ser <= '1';
            END IF;         -- determine feedback

            IF clrn = '0' THEN
                  r := "0000000000";
            ELSIF (clk'EVENT AND clk = '1')  THEN
                  r := (ser & r(0 TO 8));          -- shift
            END IF;
            q <= r;         -- connect to output port
END PROCESS;
END vhdl;
```

15.5 Mod-10 Johnson counter

```
SUBDESIGN  johnson10a            -- AHDL solution
(
     clock, reset, hold          :INPUT;
     q[0..4], zero               :OUTPUT;
)

VARIABLE
     q[0..4]                     :DFF;      -- 5-bit reg
     ser                         :NODE;     -- feedback

BEGIN
     q[].clk = clock;
     ser = !q4;                  -- feedback function
     zero = !q0 & !q4;           -- decode 00000

     IF    reset  THEN  q[].d = B"00000";        -- reset
     ELSIF  hold  THEN  q[].d = q[].q;           -- hold
     ELSE  q[0..4].d = (ser, q[0..3].q);         -- count
     END IF;
END;
```

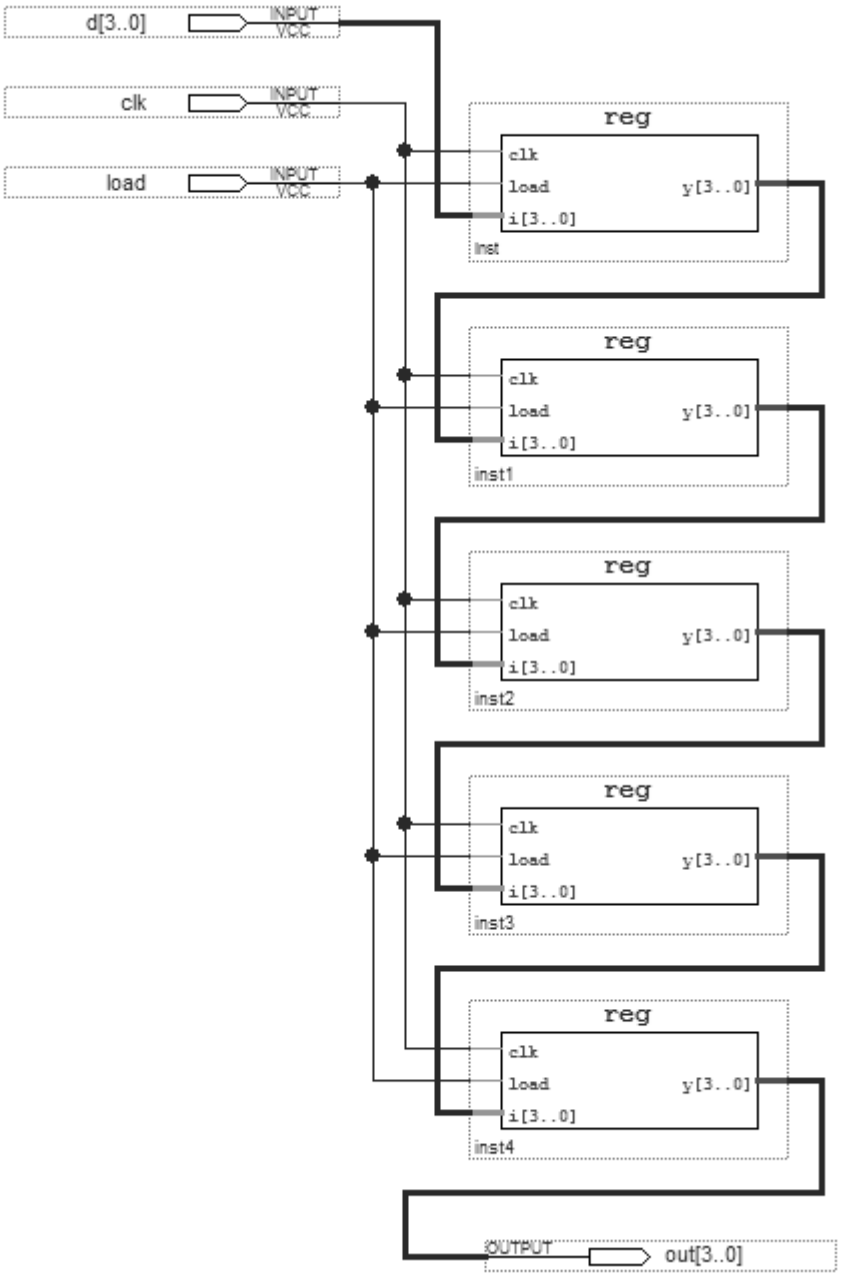
```
ENTITY  johnson10v  IS           -- VHDL solution
PORT (
     clock, reset, hold    :IN BIT;
     q                     :OUT BIT_VECTOR (0 TO 4);
     zero                  :OUT BIT
);
END johnson10v;

ARCHITECTURE vhdl OF johnson10v  IS
SIGNAL  ser                :BIT;      -- feedback
BEGIN
PROCESS (clock)
     VARIABLE  r           :BIT_VECTOR (0 TO 4);
     BEGIN
        ser <= NOT r(4);         -- feedback function

        IF (clock'EVENT AND clock = '1') THEN
           IF (reset = '1') THEN  r := "00000";  -- reset
           ELSIF (hold = '1') THEN  r := r;      -- hold
           ELSE  r := (ser & r(0 TO 3));         -- count
           END IF;
        END IF;

        q <= r;                        -- output register
        zero <= (NOT r(0)) AND (NOT r(4));       -- decode
END PROCESS;
END vhdl;
```

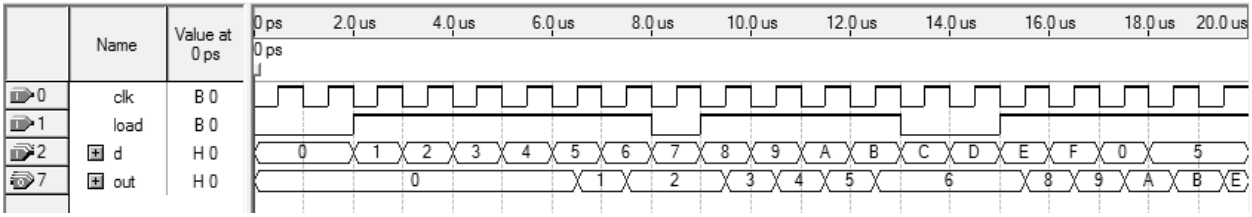
15.6 Mod-31 LFSR counter

```
SUBDESIGN  mod31lfsr              -- AHDL solution
(
      clock, resetn, enable       :INPUT;
      q[0..4]                     :OUTPUT;
)

VARIABLE
      q[0..4]                     :DFF;
      ser                         :NODE;

BEGIN
      q[].clk = clock;
      ser = !(q4 $ q2);                              -- XNOR

      IF  !resetn  THEN  q[].d = B"00000";           -- reset
      ELSIF  !enable  THEN  q[].d = q[].q;           -- disable
      ELSE  q[0..4].d = (ser, q[0..3].q);            -- count
      END IF;
END;
```

```
ENTITY  mod31lfsr_v  IS           -- VHDL solution
PORT (
      clock, resetn, enable       :IN  BIT;
      q                           :OUT BIT_VECTOR (0 TO 4)
);
END mod31lfsr_v;

ARCHITECTURE vhdl OF mod31lfsr_v IS
SIGNAL  ser                       :BIT;
BEGIN
PROCESS (clock)
      VARIABLE  reg               :BIT_VECTOR (0 TO 4);
      BEGIN
      ser <= NOT (reg(4) XOR reg(2));                -- XNOR

      IF (clock'EVENT AND clock = '1')  THEN
            IF  (resetn = '0')  THEN
                  reg := "00000";                    -- reset
            ELSIF  (enable = '0')   THEN
                  reg := reg;                        -- disable
            ELSE  reg := (ser & reg(0 TO 3));        -- count
            END IF;
      END IF;

      q <= reg;        -- output count
END PROCESS;
END vhdl;
```

15.7 Parallel data pipeline





```
SUBDESIGN  reg              -- AHDL solution
(
      clk, load, i[3..0]          :INPUT;
      y[3..0]                     :OUTPUT;
)
VARIABLE
      y[3..0]                     :DFF;      -- 4-bit reg.
BEGIN
      y[].clk = clk;

      IF  load  THEN  y[].d = i[];           -- load
      ELSE  y[3..0].d = y[3..0].q;           -- hold
      END IF;
END;
```

```
ENTITY  reg  IS
PORT (
      clk, load        :IN BIT;
      i                :IN BIT_VECTOR (3 DOWNTO 0);
      y                :OUT BIT_VECTOR (3 DOWNTO 0)
);
END reg;

ARCHITECTURE vhdl OF reg IS
BEGIN
PROCESS (clk)
      VARIABLE  reg1   :BIT_VECTOR (3 DOWNTO 0);
      BEGIN
      IF  (clk'EVENT AND clk = '1')  THEN
            IF  (load = '1')  THEN  reg1 := i;
            END IF;
      END IF;
      y <= reg1;
END PROCESS;
END vhdl;
```

15.8 Special-purpose data register

```
SUBDESIGN  special_data          -- AHDL solution
(
      clk, s[1..0], i[3..0]       :INPUT;
      q[7..0]                     :OUTPUT;
)

VARIABLE
      q[7..0]                     :DFF;      -- 8-bit register

BEGIN
      q[].clk = clk;
```

```
     CASE  s[]  IS                    -- determine operation
          WHEN  0  =>                      -- load low nibble
                q[7..0].d = (q[7..4].q, i[3..0]);
          WHEN  1  =>                      -- load high nibble
                q[7..0].d = (i[3..0], q[3..0].q);
          WHEN  2  =>                      -- swap nibbles
                q[7..0].d = (q[3..0].q, q[7..4].q);
          WHEN  3  =>                      -- rotate right
                q[7..0].d = (q0.q, q[7..1].q);
     END CASE;
END;
```

```
ENTITY  special_data_v  IS
PORT (
     clk             :IN BIT;
     s               :IN INTEGER RANGE 0 TO 3;
     i               :IN BIT_VECTOR (3 DOWNTO 0);
     q               :OUT BIT_VECTOR (7 DOWNTO 0)
);
END special_data_v;

ARCHITECTURE vhdl OF special_data_v IS
BEGIN
PROCESS (clk)
VARIABLE  reg        :BIT_VECTOR (7 DOWNTO 0);
                          -- 8-bit register
BEGIN

    IF  (clk'EVENT AND clk = '1')  THEN
        CASE  s  IS        -- determine operation
            WHEN  0  =>    reg(7 DOWNTO 0) :=
                (reg(7 DOWNTO 4) & i(3 DOWNTO 0));
                          -- load low nibble
            WHEN  1  =>    reg(7 DOWNTO 0) :=
                (i(3 DOWNTO 0) & reg(3 DOWNTO 0));
                          -- load high nibble
            WHEN  2  =>    reg(7 DOWNTO 0) :=
                (reg(3 DOWNTO 0) & reg(7 DOWNTO 4));
                          -- swap nibbles
            WHEN  3  =>    reg(7 DOWNTO 0) :=
                (reg(0) & reg(7 DOWNTO 1));
                          -- rotate right
        END CASE;
    END IF;

        q <= reg;          -- register to output port
END PROCESS;
END vhdl;
```

15.9 Shift/Rotate register

```
SUBDESIGN  shift_rotate          -- AHDL solution

(
     clk, m[2..0]          :INPUT;
     ser, d[7..0]          :INPUT;
     q[7..0]               :OUTPUT;
)

VARIABLE
     q[7..0]               :DFF;      -- 8-bit register

BEGIN
     q[].clk = clk;

     CASE  m[]  IS         -- determine control function
          WHEN  2     =>  q[7..0].d = d[7..0];
                -- load data
          WHEN  3     =>  q[7..0].d = q[7..0].q;
                -- hold data
          WHEN  4     =>  q[7..0].d = (ser, q[7..1].q);
                -- shift right
          WHEN  5     =>  q[7..0].d = (q[6..0].q, ser);
                -- shift left
          WHEN  6     =>  q[7..0].d = (q0.q, q[7..1].q);
                -- rotate right
          WHEN  7     =>  q[7..0].d = (q[6..0].q, q7.q);
                -- rotate left
          WHEN OTHERS =>  q[7..0].d = B"00000000";
                -- clear register with 0 or 1
     END CASE;
END;
```

```
ENTITY  shift_rotate_v  IS       -- VHDL solution
PORT (
     clk, ser         :IN BIT;
     m                :IN INTEGER RANGE 0 TO 7;
     d                :IN BIT_VECTOR (7 DOWNTO 0);
     q                :OUT BIT_VECTOR (7 DOWNTO 0)
);
END shift_rotate_v;

ARCHITECTURE vhdl OF shift_rotate_v IS
BEGIN
PROCESS (clk)
VARIABLE  reg         :BIT_VECTOR (7 DOWNTO 0);
                                -- 8-bit register
BEGIN
IF  (clk'EVENT AND clk = '1')  THEN
   CASE  m  IS        -- determine function
     WHEN  2  =>  reg(7 DOWNTO 0) := d(7 DOWNTO 0);
          -- load data
     WHEN  3  =>  reg(7 DOWNTO 0) := reg(7 DOWNTO 0);
          -- hold data
     WHEN  4  =>  reg(7 DOWNTO 0) := (ser & reg(7 DOWNTO 1));
          -- shift right
     WHEN  5  =>  reg(7 DOWNTO 0) := (reg(6 DOWNTO 0) & ser);
          -- shift left
     WHEN  6  =>  reg(7 DOWNTO 0) := (reg(0) & reg(7 DOWNTO 1));
          -- rotate right
     WHEN  7  =>  reg(7 DOWNTO 0) := (reg(6 DOWNTO 0) & reg(7));
          -- rotate left
     WHEN OTHERS  =>  reg(7 DOWNTO 0) := "00000000";
          -- clear register with 0 or 1
   END CASE;
END IF;
q <= reg;        -- connect register to port
END PROCESS;
END vhdl;
```

Unit 16 Synchronous Counter Design with Flip-Flops

16.1 Mod-6 synchronous counter design

Present States			Next States			Flip-flop Inputs					
QC_n	QB_n	QA_n	QC_{n+1}	QB_{n+1}	QA_{n+1}	JC	KC	JB	KB	JA	KA
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	0	0	0	X	1	0	X	X	1
1	1	0	X	X	X	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X	X	X	X

QB QA

QC	00	01	11	10
0	0	0	1	0
1	X	X	X	X

JC

QB QA

QC	00	01	11	10
0	X	X	X	X
1	0	1	X	X

KC

QB QA

QC	00	01	11	10
0	0	1	X	X
1	0	0	X	X

JB

QB QA

QC	00	01	11	10
0	X	X	1	0
1	X	X	X	X

KB

QB QA

QC	00	01	11	10
0	1	X	X	1
1	1	X	X	X

JA

QB QA

QC	00	01	11	10
0	X	1	1	X
1	X	1	X	X

KA

$JC = QB\ QA$ $\qquad$ $KC = QA$

$JB = \overline{QC}\ QA$ $\qquad$ $KB = QA$

$JA = KA = 1$

QB QA

QC	00	01	11	10
0	0	0	1	0
1	1	0	X	X

$D_{QC} = QC\ \overline{QA} + QB\ QA$

QB QA

QC	00	01	11	10
0	0	1	0	1
1	0	0	X	X

$D_{QB} = QB\ \overline{QA} + \overline{QC}\ \overline{QB}\ QA$

QB QA

QC	00	01	11	10
0	1	0	0	1
1	1	0	X	X

$D_{QA} = \overline{QA}$

16.2 Mod-5 synchronous, down counter design

Present States			Next States			Flip-flop Inputs					
QX_n	QY_n	QZ_n	QX_{n+1}	QY_{n+1}	QZ_{n+1}	JX	KX	JY	KY	JZ	KZ
0	0	0	1	0	0	1	X	0	X	0	X
0	0	1	0	0	0	0	X	0	X	X	1
0	1	0	0	0	1	0	X	X	1	1	X
0	1	1	0	1	0	0	X	X	0	X	1
1	0	0	0	1	1	X	1	1	X	1	X
1	0	1	X	X	X	X	X	X	X	X	X
1	1	0	X	X	X	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X	X	X	X

QY QZ

QX	00	01	11	10
0	1	0	0	0
1	X	X	X	X

JX

QY QZ

QX	00	01	11	10
0	X	X	X	X
1	1	X	X	X

KX

QY QZ

QX	00	01	11	10
0	0	0	X	X
1	1	X	X	X

JY

QY QZ

QX	00	01	11	10
0	X	X	0	1
1	X	X	X	X

KY

QY QZ

QX	00	01	11	10
0	0	X	X	1
1	1	X	X	X

JZ

QY QZ

QX	00	01	11	10
0	X	1	1	X
1	X	X	X	X

KZ

$JX = \overline{QY}\ \overline{QZ}$ $\quad KX = 1$

$JY = QX$ $\quad KY = \overline{QZ}$

$JZ = QY + QX$ $\quad KZ = 1$

QY QZ

QX	00	01	11	10
0	1	0	0	0
1	0	X	X	X

$D_{QX} = \overline{QX}\ \overline{QY}\ \overline{QZ}$

QY QZ

QX	00	01	11	10
0	0	0	1	0
1	1	X	X	X

$D_{QY} = QX + QY\ QZ$

QY QZ

QX	00	01	11	10
0	0	0	0	1
1	1	X	X	X

$D_{QZ} = QX + QY\ \overline{QZ}$

16.3 Synchronous 8421 BCD counter design

n				n+1							
QD	QC	QB	QA	QD	QC	QB	QA	JD	KD	JC KC	JB KB	JA	KA
0	0	0	0	0	0	0	1	0	X	0 X	0 X	1	X
0	0	0	1	0	0	1	0	0	X	0 X	1 X	X	1
0	0	1	0	0	0	1	1	0	X	0 X	X 0	1	X
0	0	1	1	0	1	0	0	0	X	1 X	X 1	X	1
0	1	0	0	0	1	0	1	0	X	X 0	0 X	1	X
0	1	0	1	0	1	1	0	0	X	X 0	1 X	X	1
0	1	1	0	0	1	1	1	0	X	X 0	X 0	1	X
0	1	1	1	1	0	0	0	1	X	X 1	X 1	X	1
1	0	0	0	1	0	0	1	X	0	0 X	0 X	1	X
1	0	0	1	0	0	0	0	X	1	0 X	0 X	X	1
1	0	1	0	X	X	X	X	X	X	X X	X X	X	X
1	0	1	1	X	X	X	X	X	X	X X	X X	X	X
1	1	0	0	X	X	X	X	X	X	X X	X X	X	X
1	1	0	1	X	X	X	X	X	X	X X	X X	X	X
1	1	1	0	X	X	X	X	X	X	X X	X X	X	X
1	1	1	1	X	X	X	X	X	X	X X	X X	X	X

QD QC \ QB QA

	0 0	0 1	1 1	1 0
0 0	0	0	0	0
0 1	0	0	1	0
1 1	X	X	X	X
1 0	X	X	X	X

JD

JD = QC QB QA

QD QC \ QB QA

	0 0	0 1	1 1	1 0
0 0	X	X	X	X
0 1	X	X	X	X
1 1	X	X	X	X
1 0	0	1	X	X

KD

KD = QA

QD QC \ QB QA

	0 0	0 1	1 1	1 0
0 0	0	0	1	0
0 1	X	X	X	X
1 1	X	X	X	X
1 0	0	0	X	X

JC

JC = QB QA

QD QC \ QB QA

	0 0	0 1	1 1	1 0
0 0	X	X	X	X
0 1	0	0	1	0
1 1	X	X	X	X
1 0	X	X	X	X

KC

KC = QB QA

QD QC \ QB QA

	0 0	0 1	1 1	1 0
0 0	0	1	X	X
0 1	0	1	X	X
1 1	X	X	X	X
1 0	0	0	X	X

JB

JB = $\overline{QD}$ QA

QD QC \ QB QA

	0 0	0 1	1 1	1 0
0 0	X	X	1	0
0 1	X	X	1	0
1 1	X	X	X	X
1 0	X	X	X	X

KB

KB = QA

QD QC \ QB QA

	0 0	0 1	1 1	1 0
0 0	1	X	X	1
0 1	1	X	X	1
1 1	X	X	X	X
1 0	1	X	X	X

JA

JA = 1

QD QC \ QB QA

	0 0	0 1	1 1	1 0
0 0	X	1	1	X
0 1	X	1	1	X
1 1	X	X	X	X
1 0	X	1	X	X

KA

KA = 1

QD QC \ QB QA	0 0	0 1	1 1	1 0
0 0	0	0	0	0
0 1	0	0	1	0
1 1	X	X	X	X
1 0	1	0	X	X

$$DQD = QD\,\overline{QA} + QC\,QB\,QA$$

QD QC \ QB QA	0 0	0 1	1 1	1 0
0 0	0	0	1	0
0 1	1	1	0	1
1 1	X	X	X	X
1 0	0	0	X	X

$$DQC = QC\,\overline{QB} + QC\,\overline{QA} + \overline{QC}\,QB\,QA$$

QD QC \ QB QA	0 0	0 1	1 1	1 0
0 0	0	1	0	1
0 1	0	1	0	1
1 1	X	X	X	X
1 0	0	0	X	X

$$DQB = QB\,\overline{QA} + \overline{QD}\,\overline{QB}\,QA$$

QD QC \ QB QA	0 0	0 1	1 1	1 0
0 0	1	0	0	1
0 1	1	0	0	1
1 1	X	X	X	X
1 0	1	0	X	X

$$DQA = \overline{QA}$$

16.4 Synchronous 8421 BCD down counter design

n				n+1							
QD	QC	QB	QA	QD	QC	QB	QA	JD KD	JC KC	JB KB	JA KA
0	0	0	0	1	0	0	1	1 X	0 X	0 X	1 X
0	0	0	1	0	0	0	0	0 X	0 X	0 X	X 1
0	0	1	0	0	0	0	1	0 X	0 X	X 1	1 X
0	0	1	1	0	0	1	0	0 X	0 X	X 0	X 1
0	1	0	0	0	0	1	1	0 X	X 1	1 X	1 X
0	1	0	1	0	1	0	0	0 X	X 0	0 X	X 1
0	1	1	0	0	1	0	1	0 X	X 0	X 1	1 X
0	1	1	1	0	1	1	0	0 X	X 0	X 0	X 1
1	0	0	0	0	1	1	1	X 1	1 X	1 X	1 X
1	0	0	1	1	0	0	0	X 0	0 X	0 X	X 1
1	0	1	0	X	X	X	X	X X	X X	X X	X X
1	0	1	1	X	X	X	X	X X	X X	X X	X X
1	1	0	0	X	X	X	X	X X	X X	X X	X X
1	1	0	1	X	X	X	X	X X	X X	X X	X X
1	1	1	0	X	X	X	X	X X	X X	X X	X X
1	1	1	1	X	X	X	X	X X	X X	X X	X X

QD QC \ QB QA	0 0	0 1	1 1	1 0
0 0	1	0	0	0
0 1	0	0	0	0
1 1	X	X	X	X
1 0	X	X	X	X

JD

$$JD = \overline{QC}\,\overline{QB}\,\overline{QA}$$

QD QC \ QB QA	0 0	0 1	1 1	1 0
0 0	X	X	X	X
0 1	X	X	X	X
1 1	X	X	X	X
1 0	1	0	X	X

KD

$$KD = \overline{QA}$$

QD QC \ QB QA	0 0	0 1	1 1	1 0
0 0	0	0	0	0
0 1	X	X	X	X
1 1	X	X	X	X
1 0	1	0	X	X

JC

$JC = QD\ \overline{QA}$

QD QC \ QB QA	0 0	0 1	1 1	1 0
0 0	X	X	X	X
0 1	1	0	0	0
1 1	X	X	X	X
1 0	X	X	X	X

KC

$KC = \overline{QB}\ \overline{QA}$

QD QC \ QB QA	0 0	0 1	1 1	1 0
0 0	0	0	X	X
0 1	1	0	X	X
1 1	X	X	X	X
1 0	1	0	X	X

JB

$JB = QC\ \overline{QA} + QD\ \overline{QA}$

QD QC \ QB QA	0 0	0 1	1 1	1 0
0 0	X	X	0	1
0 1	X	X	0	1
1 1	X	X	X	X
1 0	X	X	X	X

KB

$KB = \overline{QA}$

QD QC \ QB QA	0 0	0 1	1 1	1 0
0 0	1	X	X	1
0 1	1	X	X	1
1 1	X	X	X	X
1 0	1	X	X	X

JA

$JA = 1$

QD QC \ QB QA	0 0	0 1	1 1	1 0
0 0	X	1	1	X
0 1	X	1	1	X
1 1	X	X	X	X
1 0	X	1	X	X

KA

$KA = 1$

QD QC \ QB QA	0 0	0 1	1 1	1 0
0 0	1	0	0	0
0 1	0	0	0	0
1 1	X	X	X	X
1 0	0	1	X	X

$DQD = QD\ QA + \overline{QD}\ \overline{QC}\ \overline{QB}\ \overline{QA}$

QD QC \ QB QA	0 0	0 1	1 1	1 0
0 0	0	0	0	0
0 1	0	1	1	1
1 1	X	X	X	X
1 0	1	0	X	X

$DQC = QC\ QB + QC\ QA + QD\ \overline{QA}$

QD QC \ QB QA	0 0	0 1	1 1	1 0
0 0	0	0	1	0
0 1	1	0	1	0
1 1	X	X	X	X
1 0	1	0	X	X

$DQB = QB\ QA + QD\ \overline{QA} + QC\ \overline{QB}\ \overline{QA}$

QD QC \ QB QA	0 0	0 1	1 1	1 0
0 0	1	0	0	1
0 1	1	0	0	1
1 1	X	X	X	X
1 0	1	0	X	X

$DQA = \overline{QA}$

16.5 Synchronous mod-11 counter

n				n+1							
QD	QC	QB	QA	QD	QC	QB	QA	JD KD	JC KC	JB KB	JA KA
0	0	0	0	0	0	0	1	0 X	0 X	0 X	1 X
0	0	0	1	0	0	1	0	0 X	0 X	1 X	X 1
0	0	1	0	0	0	1	1	0 X	0 X	X 0	1 X
0	0	1	1	0	1	0	0	0 X	1 X	X 1	X 1
0	1	0	0	0	1	0	1	0 X	X 0	0 X	1 X
0	1	0	1	0	1	1	0	0 X	X 0	1 X	X 1
0	1	1	0	0	1	1	1	0 X	X 0	X 0	1 X
0	1	1	1	1	0	0	0	1 X	X 1	X 1	X 1
1	0	0	0	1	0	0	1	X 0	0 X	0 X	1 X
1	0	0	1	1	0	1	0	X 0	0 X	1 X	X 1
1	0	1	0	0	0	0	0	X 1	0 X	X 1	0 X
1	0	1	1	X	X	X	X	X X	X X	X X	X X
1	1	0	0	X	X	X	X	X X	X X	X X	X X
1	1	0	1	X	X	X	X	X X	X X	X X	X X
1	1	1	0	X	X	X	X	X X	X X	X X	X X
1	1	1	1	X	X	X	X	X X	X X	X X	X X

QB QA / QD QC

	0 0	0 1	1 1	1 0
0 0	0	0	0	0
0 1	0	0	1	0
1 1	X	X	X	X
1 0	X	X	X	X

JD

JD = QC QB QA

QB QA / QD QC

	0 0	0 1	1 1	1 0
0 0	X	X	X	X
0 1	X	X	X	X
1 1	X	X	X	X
1 0	0	0	X	1

KD

KD = QB

QB QA / QD QC

	0 0	0 1	1 1	1 0
0 0	0	0	1	0
0 1	X	X	X	X
1 1	X	X	X	X
1 0	0	0	X	0

JC

JC = QB QA

QB QA / QD QC

	0 0	0 1	1 1	1 0
0 0	X	X	X	X
0 1	0	0	1	0
1 1	X	X	X	X
1 0	X	X	X	X

KC

KC = QB QA

QB QA / QD QC

	0 0	0 1	1 1	1 0
0 0	0	1	X	X
0 1	0	1	X	X
1 1	X	X	X	X
1 0	0	1	X	X

JB

JB = QA

QB QA / QD QC

	0 0	0 1	1 1	1 0
0 0	X	X	1	0
0 1	X	X	1	0
1 1	X	X	X	X
1 0	X	X	X	1

KB

KB = QA + QD

QB QA / QD QC

	0 0	0 1	1 1	1 0
0 0	1	X	X	1
0 1	1	X	X	1
1 1	X	X	X	X
1 0	1	X	X	0

JA

JA = $\overline{QB}$ + $\overline{QD}$

QB QA / QD QC

	0 0	0 1	1 1	1 0
0 0	X	1	1	X
0 1	X	1	1	X
1 1	X	X	X	X
1 0	X	1	X	X

KA

KA = 1

QD QC \ QB QA	0 0	0 1	1 1	1 0
0 0	0	0	0	0
0 1	0	0	1	0
1 1	X	X	X	X
1 0	1	1	X	0

$$DQD = QD\ \overline{QB} + QC\ QB\ QA$$

QD QC \ QB QA	0 0	0 1	1 1	1 0
0 0	0	0	1	0
0 1	1	1	0	1
1 1	X	X	X	X
1 0	0	0	X	0

$$DQC = QC\ \overline{QB} + QC\ \overline{QA} + \overline{QC}\ QB\ QA$$

QD QC \ QB QA	0 0	0 1	1 1	1 0
0 0	0	1	0	1
0 1	0	1	0	1
1 1	X	X	X	X
1 0	0	1	X	0

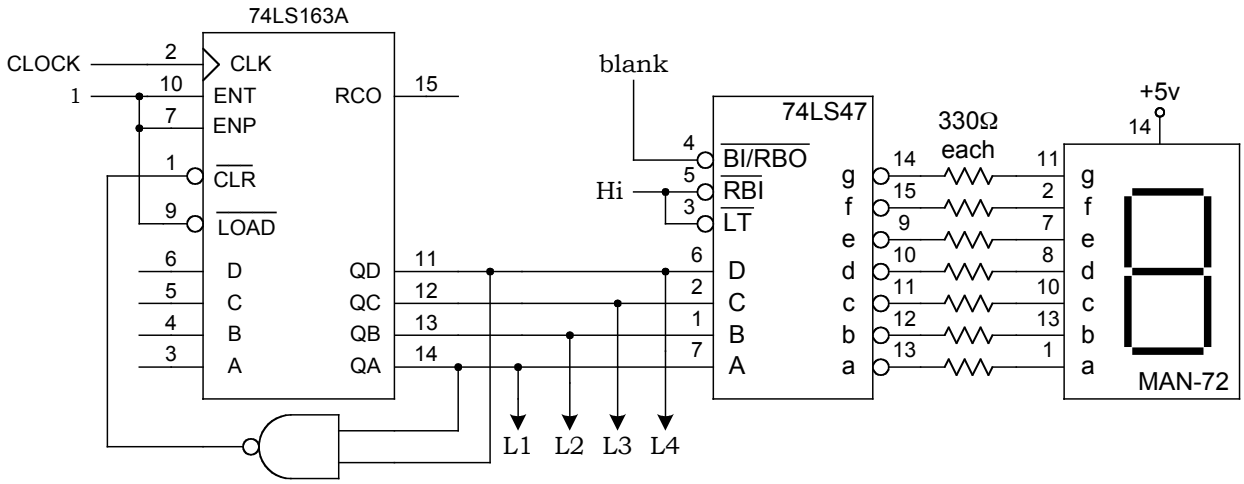
$$DQB = \overline{QB}\ QA + \overline{QD}\ QB\ \overline{QA}$$

QD QC \ QB QA	0 0	0 1	1 1	1 0
0 0	1	0	0	1
0 1	1	0	0	1
1 1	X	X	X	X
1 0	1	0	X	0

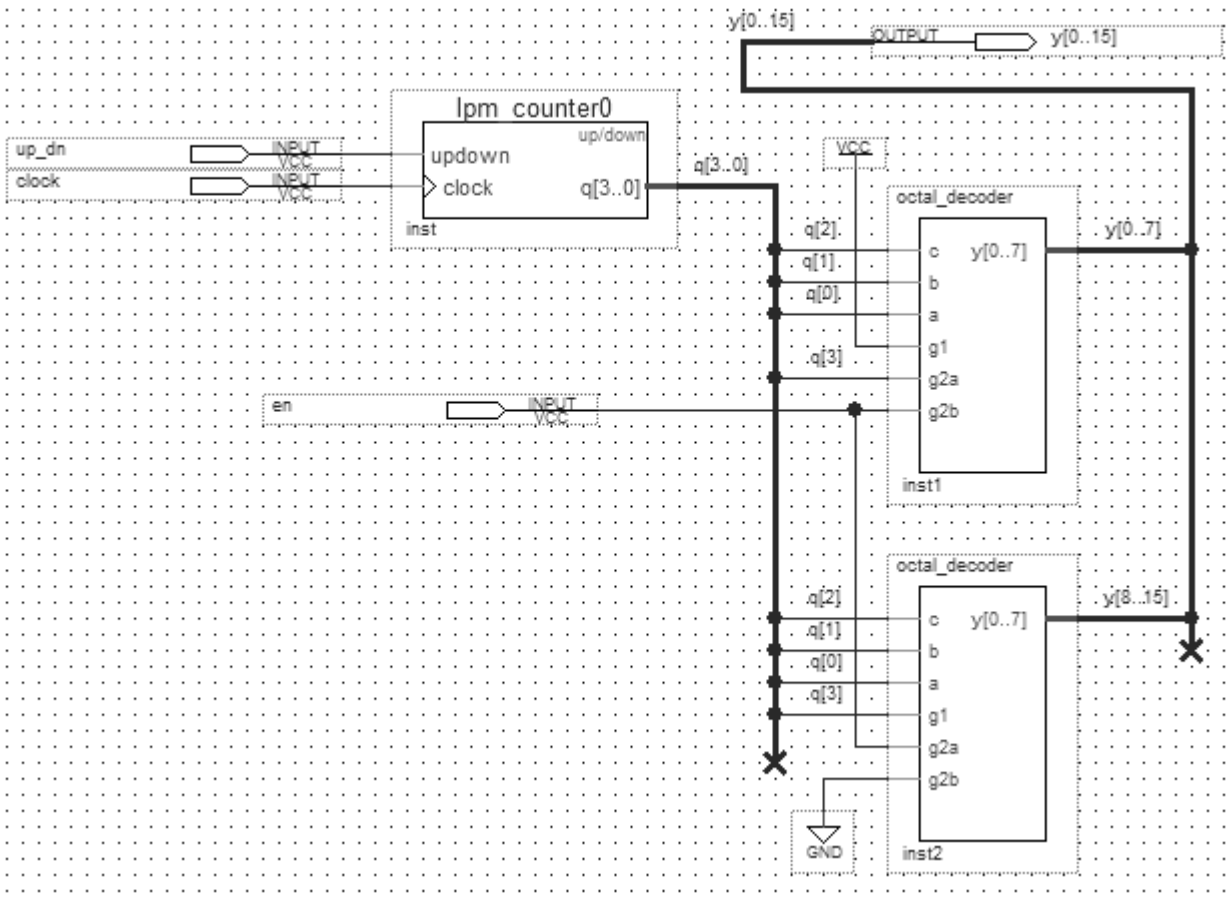
$$DQA = \overline{QB}\ \overline{QA} + \overline{QD}\ \overline{QA}$$

Unit 17 Decoder and Display Applications

17.1 Modified 4-bit binary counter and display



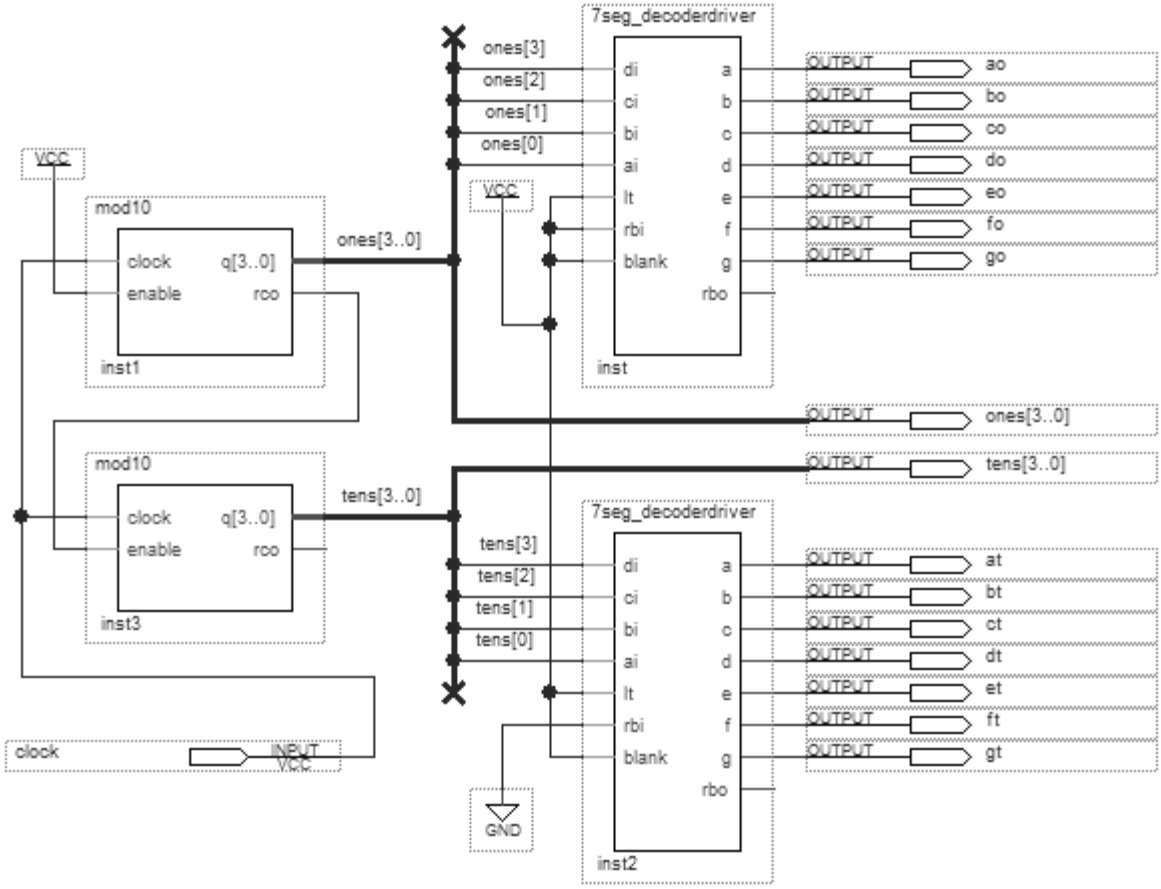
17.2 Mod-16 counter and decoder



```
SUBDESIGN  octal_decoder
(c, b, a, g1, g2a, g2b           :INPUT;
     y[0..7]                     :OUTPUT;)
VARIABLE
     enable[1..3]          :NODE;
     bin[2..0]             :NODE;
BEGIN
     bin[] = (c, b, a);
     enable[] = (g1, g2a, g2b);
     IF enable[] == B"100" THEN
          CASE bin[] IS
               WHEN B"000"      =>    y[] = B"01111111";
               WHEN B"001"      =>    y[] = B"10111111";
               WHEN B"010"      =>    y[] = B"11011111";
               WHEN B"011"      =>    y[] = B"11101111";
               WHEN B"100"      =>    y[] = B"11110111";
               WHEN B"101"      =>    y[] = B"11111011";
               WHEN B"110"      =>    y[] = B"11111101";
               WHEN B"111"      =>    y[] = B"11111110";
          END CASE;
     ELSE                               y[] = B"11111111";
     END IF;
END;
```

```
ENTITY  octal_decoder  IS
PORT (c, b, a, g1, g2a, g2b      :IN BIT;
      y                    :OUT BIT_VECTOR (0 TO 7));
END octal_decoder;
ARCHITECTURE vhdl OF octal_decoder IS
BEGIN
PROCESS (c, b, a, g1, g2a, g2b)
VARIABLE enable       :BIT_VECTOR (1 TO 3);
VARIABLE bin          :BIT_VECTOR (2 DOWNTO 0);
BEGIN
     bin := (c & b & a);
     enable := (g1 & g2a & g2b);
     IF (enable = "100")   THEN
          CASE bin IS
               WHEN "000" =>    y <= B"01111111";
               WHEN "001" =>    y <= B"10111111";
               WHEN "010" =>    y <= B"11011111";
               WHEN "011" =>    y <= B"11101111";
               WHEN "100" =>    y <= B"11110111";
               WHEN "101" =>    y <= B"11111011";
               WHEN "110" =>    y <= B"11111101";
               WHEN "111" =>    y <= B"11111110";
          END CASE;
     ELSE                          y <= B"11111111";
     END IF;
END PROCESS;
END vhdl;
```

17.3 Mod-100 BCD counter and 7-segment display




```
SUBDESIGN  mod10
(clock, enable                    :INPUT;
      q[3..0], rco                :OUTPUT;)
VARIABLE
      counter[3..0]               :DFF;
BEGIN
      DEFAULTS
            rco = GND;
      END DEFAULTS;
      counter[].clk  = clock;
      IF  enable  THEN
            IF counter[].q == 9  THEN
                  counter[].d = B"0000";
                  rco = VCC;
            ELSE
                  counter[].d = counter[].q + 1;
            END IF;
      ELSE
            counter[].d = counter[].q;
      END IF;
      q[] = counter[].q;
END;
```

```
SUBDESIGN  7seg_decoderdriver    -- common anode
(di, ci, bi, ai, lt, rbi, blank             :INPUT;
     a, b, c, d, e, f, g, rbo               :OUTPUT;)
VARIABLE
     seg[0..6]              :NODE;
     bin[3..0]              :NODE;
BEGIN
DEFAULTS
     rbo = VCC;
END DEFAULTS;
     bin[] = (di, ci, bi, ai);
     a = seg0;
     b = seg1;
     c = seg2;
     d = seg3;
     e = seg4;
     f = seg5;
     g = seg6;
     IF    (lt == GND)     THEN  seg[] = B"0000000";
     ELSIF (blank == GND)  THEN  seg[] = B"1111111";
     ELSIF (rbi == GND & bin[] == 0)  THEN  rbo = GND;
                                    seg[] = B"1111111";
     ELSE
          CASE bin[] IS    -- segments   abcdefg
               WHEN 0           =>    seg[] = B"0000001";
               WHEN 1           =>    seg[] = B"1001111";
               WHEN 2           =>    seg[] = B"0010010";
               WHEN 3           =>    seg[] = B"0000110";
               WHEN 4           =>    seg[] = B"1001100";
               WHEN 5           =>    seg[] = B"0100100";
               WHEN 6           =>    seg[] = B"1100000";
               WHEN 7           =>    seg[] = B"0001111";
               WHEN 8           =>    seg[] = B"0000000";
               WHEN 9           =>    seg[] = B"0001100";
               WHEN OTHERS      =>    seg[] = B"1111111";
          END CASE;
     END IF;
END;
```

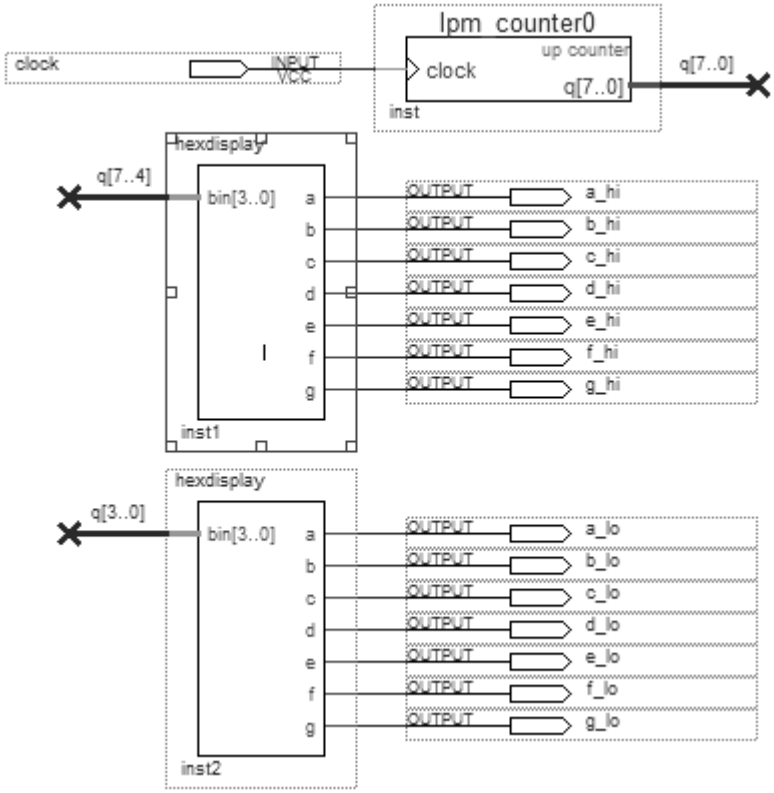
```
ENTITY mod10 IS
PORT (clock                 :IN BIT;
      enable                :IN BIT;
      q                     :OUT INTEGER RANGE 0 TO 15;
      rco                   :OUT BIT);
END mod10;
ARCHITECTURE vhdl OF mod10 IS
BEGIN
     PROCESS (clock, enable)
     VARIABLE  counter     :INTEGER RANGE 0 TO 15;
     BEGIN
```

```
          IF ((counter = 9) AND (enable = '1')) THEN
                    rco <= '1';
          ELSE      rco <= '0';
          END IF;
          IF (clock'EVENT AND clock = '1') THEN
               IF  (enable = '1')  THEN
                    IF (counter = 9)  THEN
                              counter := 0;
                    ELSE      counter := counter + 1;
                    END IF;
               END IF;
          END IF;
          q <= counter;
     END PROCESS;
END vhdl;
```

```
ENTITY  seg_decoderdriver  IS    -- common anode
PORT (di, ci, bi, ai, lt, rbi, blank       :IN BIT;
      a, b, c, d, e, f, g, rbo             :OUT BIT);
END seg_decoderdriver;
ARCHITECTURE vhdl OF seg_decoderdriver IS
BEGIN
PROCESS (di, ci, bi, ai, lt, rbi, blank)
VARIABLE seg          :BIT_VECTOR (0 TO 6);
VARIABLE bin          :BIT_VECTOR (3 DOWNTO 0);
BEGIN
     bin := (di & ci & bi & ai);
     a <= seg(0);
     b <= seg(1);
     c <= seg(2);
     d <= seg(3);
     e <= seg(4);
     f <= seg(5);
     g <= seg(6);
     IF    (lt = '0')      THEN
                           seg := "0000000"; rbo <= '1';
     ELSIF (blank = '0')   THEN
                           seg := "1111111"; rbo <= '1';
     ELSIF (rbi = '0' AND bin = "0000")    THEN
                           seg := "1111111"; rbo <= '0';
     ELSE
        CASE bin IS     -- segments   abcdefg
           WHEN "0000" =>   seg := "0000001"; rbo <= '1';
           WHEN "0001" =>   seg := "1001111"; rbo <= '1';
           WHEN "0010" =>   seg := "0010010"; rbo <= '1';
           WHEN "0011" =>   seg := "0000110"; rbo <= '1';
           WHEN "0100" =>   seg := "1001100"; rbo <= '1';
           WHEN "0101" =>   seg := "0100100"; rbo <= '1';
           WHEN "0110" =>   seg := "1100000"; rbo <= '1';
           WHEN "0111" =>   seg := "0001111"; rbo <= '1';
           WHEN "1000" =>   seg := "0000000"; rbo <= '1';
```

```
            WHEN "1001" =>   seg := "0001100"; rbo <= '1';
            WHEN OTHERS =>   seg := "1111111"; rbo <= '1';
         END CASE;
      END IF;
END PROCESS;
END vhdl;
```

17.4 Mod-256 binary counter and hex decoder/driver




```
SUBDESIGN  hex_display           -- AHDL solution
(bin[3..0]                    :INPUT;
     a,b,c,d,e,f,g            :OUTPUT;)
BEGIN
     TABLE      -- common-anode truth table
          bin[]=>       a ,b,  c,  d,  e,  f,  g;
          H"0" =>       0,  0,  0,  0,  0,  0,  1;
          H"1" =>       1,  0,  0,  1,  1,  1,  1;
          H"2" =>       0,  0,  1,  0,  0,  1,  0;
          H"3" =>       0,  0,  0,  0,  1,  1,  0;
          H"4" =>       1,  0,  0,  1,  1,  0,  0;
          H"5" =>       0,  1,  0,  0,  1,  0,  0;
          H"6" =>       0,  1,  0,  0,  0,  0,  0;
          H"7" =>       0,  0,  0,  1,  1,  1,  1;
          H"8" =>       0,  0,  0,  0,  0,  0,  0;
          H"9" =>       0,  0,  0,  0,  1,  0,  0;
          H"A" =>       0,  0,  0,  1,  0,  0,  0;
          H"B" =>       1,  1,  0,  0,  0,  0,  0;
```

```
           H"C" =>       0, 1, 1, 0, 0, 0, 1;
           H"D" =>       1, 0, 0, 0, 0, 1, 0;
           H"E" =>       0, 1, 1, 0, 0, 0, 0;
           H"F" =>       0, 1, 1, 1, 0, 0, 0;
     END TABLE;
END;
```

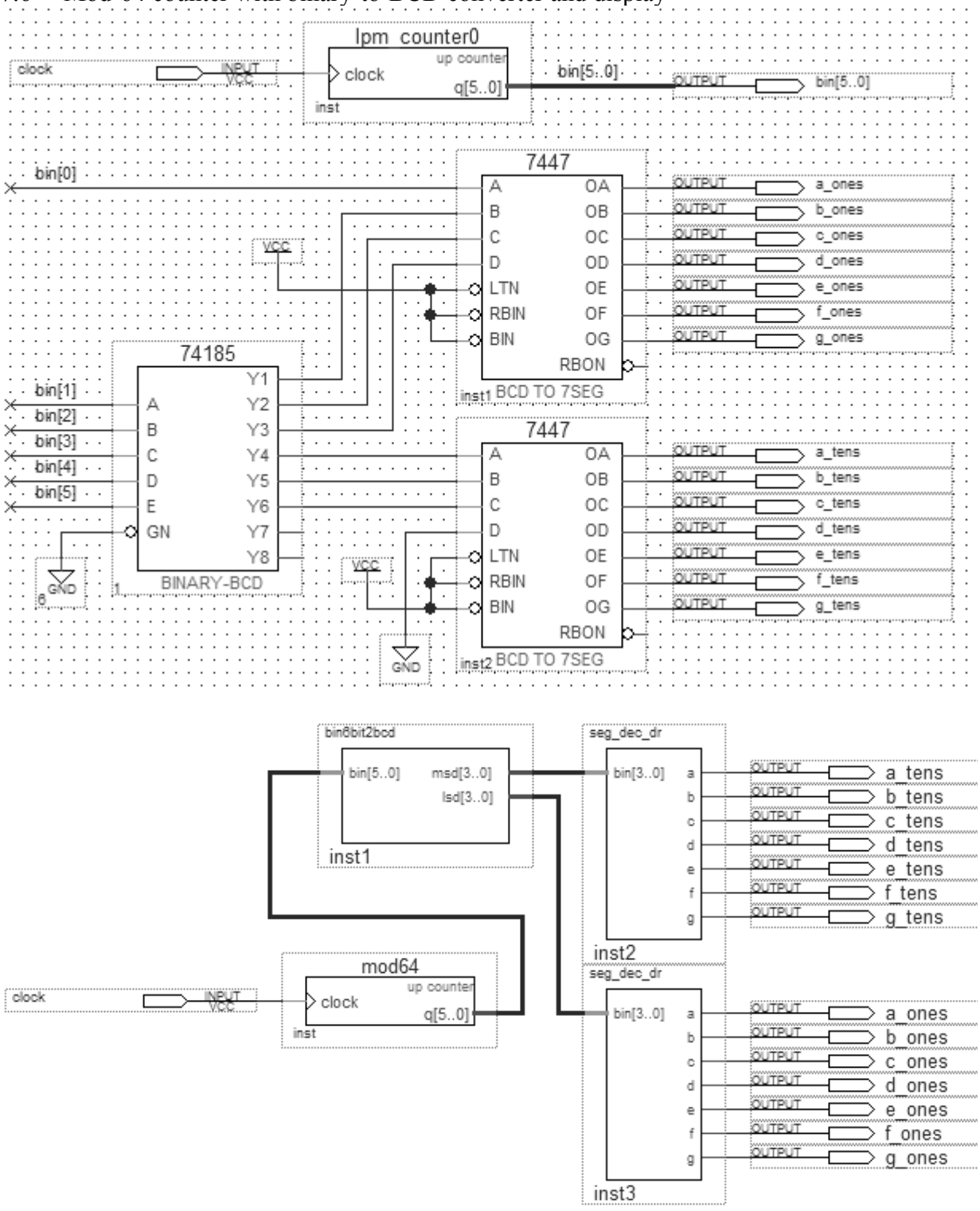
```
ENTITY  hexdisplay  IS
PORT ( bin                  :IN BIT_VECTOR (3 DOWNTO 0);
       a,b,c,d,e,f,g        :OUT BIT );
END hexdisplay;
ARCHITECTURE vhdl OF hexdisplay IS
SIGNAL  segments                :BIT_VECTOR (1 TO 7);
BEGIN
    PROCESS (bin)
    BEGIN
         CASE  bin  IS                  --      abcdefg
              WHEN  X"0"    =>    segments <= "0000001";
              WHEN  X"1"    =>    segments <= "1001111";
              WHEN  X"2"    =>    segments <= "0010010";
              WHEN  X"3"    =>    segments <= "0000110";
              WHEN  X"4"    =>    segments <= "1001100";
              WHEN  X"5"    =>    segments <= "0100100";
              WHEN  X"6"    =>    segments <= "0100000";
              WHEN  X"7"    =>    segments <= "0001111";
              WHEN  X"8"    =>    segments <= "0000000";
              WHEN  X"9"    =>    segments <= "0000100";
              WHEN  X"A"    =>    segments <= "0001000";
              WHEN  X"B"    =>    segments <= "1100000";
              WHEN  X"C"    =>    segments <= "0110001";
              WHEN  X"D"    =>    segments <= "1000010";
              WHEN  X"E"    =>    segments <= "0110000";
              WHEN  X"F"    =>    segments <= "0111000";
         END CASE;
    END PROCESS;
a <= segments(1);
b <= segments(2);
c <= segments(3);
d <= segments(4);
e <= segments(5);
f <= segments(6);
g <= segments(7);
END vhdl;
```

17.5 BCD decoder

```
SUBDESIGN  BCD_decoder
(     bcd[3..0]              :INPUT;
      state[9..0], err :OUTPUT;   )
BEGIN
      DEFAULTS
            state[] = B"0000000000";
            err = VCC;
      END DEFAULTS;
      CASE bcd[] IS
            WHEN 0      =>    state[0] = VCC; err = GND;
            WHEN 1      =>    state[1] = VCC; err = GND;
            WHEN 2      =>    state[2] = VCC; err = GND;
            WHEN 3      =>    state[3] = VCC; err = GND;
            WHEN 4      =>    state[4] = VCC; err = GND;
            WHEN 5      =>    state[5] = VCC; err = GND;
            WHEN 6      =>    state[6] = VCC; err = GND;
            WHEN 7      =>    state[7] = VCC; err = GND;
            WHEN 8      =>    state[8] = VCC; err = GND;
            WHEN 9      =>    state[9] = VCC; err = GND;
      END CASE;
END;
```

```
ENTITY  BCD_decoder  IS
PORT (bcd        :IN INTEGER RANGE 0 TO 15;
       state     :OUT BIT_VECTOR (9 DOWNTO 0);
       err       :OUT BIT);
END BCD_decoder;
ARCHITECTURE vhdl OF BCD_decoder IS
BEGIN
PROCESS (bcd)
BEGIN
   CASE bcd IS
      WHEN 0 => state <= "0000000001"; err <= '0';
      WHEN 1 => state <= "0000000010"; err <= '0';
      WHEN 2 => state <= "0000000100"; err <= '0';
      WHEN 3 => state <= "0000001000"; err <= '0';
      WHEN 4 => state <= "0000010000"; err <= '0';
      WHEN 5 => state <= "0000100000"; err <= '0';
      WHEN 6 => state <= "0001000000"; err <= '0';
      WHEN 7 => state <= "0010000000"; err <= '0';
      WHEN 8 => state <= "0100000000"; err <= '0';
      WHEN 9 => state <= "1000000000"; err <= '0';
      WHEN OTHERS =>
                state <= "0000000000";err <= '1';
   END CASE;
END PROCESS;
END vhdl;
```

17.6 Mod-64 counter with binary-to-BCD converter and display



```
SUBDESIGN  bin6bit2bcd
    (bin[5..0]                    :INPUT;
     msd[3..0], lsd[3..0]         :OUTPUT;)
VARIABLE
     ones[5..0]     :NODE;    -- to capture the ones digit
BEGIN
     lsd[3..0] = ones[3..0];       -- output lsd BCD digit
     IF     bin[] >= 60  THEN
          msd[] = 6;               -- set MSD value
          ones[] = bin[] - 60;     -- calculate LSD value
     ELSIF bin[] >= 50   THEN
          msd[] = 5;          ones[] = bin[] - 50;
     ELSIF bin[] >= 40   THEN
          msd[] = 4;          ones[] = bin[] - 40;
     ELSIF bin[] >= 30   THEN
          msd[] = 3;          ones[] = bin[] - 30;
     ELSIF bin[] >= 20   THEN
          msd[] = 2;          ones[] = bin[] - 20;
     ELSIF bin[] >= 10   THEN
          msd[] = 1;          ones[] = bin[] - 10;
     ELSE
          msd[] = 0;          ones[] = bin[];
     END IF;
END;
```

```
SUBDESIGN  seg_dec_dr    -- common anode
(    bin[3..0]                :INPUT;
     a, b, c, d, e, f, g      :OUTPUT;  )
VARIABLE
     seg[7..1]                :NODE;
BEGIN
     (a, b, c, d, e, f, g) = seg[7..1];
     CASE bin[] IS        -- segments   abcdefg
          WHEN H"0"      =>   seg[] = B"0000001";
          WHEN H"1"      =>   seg[] = B"1001111";
          WHEN H"2"      =>   seg[] = B"0010010";
          WHEN H"3"      =>   seg[] = B"0000110";
          WHEN H"4"      =>   seg[] = B"1001100";
          WHEN H"5"      =>   seg[] = B"0100100";
          WHEN H"6"      =>   seg[] = B"1100000";
          WHEN H"7"      =>   seg[] = B"0001111";
          WHEN H"8"      =>   seg[] = B"0000000";
          WHEN H"9"      =>   seg[] = B"0001100";
          WHEN OTHERS    =>   seg[] = B"1111111";
     END CASE;
END;
```

```
ENTITY bin6bit2bcd IS
PORT (    bin            :IN INTEGER RANGE 0 TO 63;
          msd, lsd  :OUT INTEGER RANGE 0 TO 9);
END bin6bit2bcd;
ARCHITECTURE vhdl OF bin6bit2bcd IS
BEGIN
PROCESS (bin)       -- bin change invokes process
BEGIN
     IF    bin >= 60     THEN
           msd <= 6;           -- set MSD value
           lsd <= bin - 60;    -- calculate LSD value
     ELSIF bin >= 50     THEN
           msd <= 5;
           lsd <= bin - 50;
     ELSIF bin >= 40     THEN
           msd <= 4;
           lsd <= bin - 40;
     ELSIF bin >= 30     THEN
           msd <= 3;
           lsd <= bin - 30;
     ELSIF bin >= 20     THEN
           msd <= 2;
           lsd <= bin - 20;
     ELSIF bin >= 10     THEN
           msd <= 1;
           lsd <= bin - 10;
     ELSE
           msd <= 0;
           lsd <= bin;
     END IF;
END PROCESS;
END vhdl;
```

```
ENTITY  seg_dec_dr  IS    -- common anode
PORT (bin                      :IN BIT_VECTOR (3 DOWNTO 0);
      a, b, c, d, e, f, g      :OUT BIT);
END seg_dec_dr;

ARCHITECTURE vhdl OF seg_dec_dr IS
BEGIN
PROCESS (bin)
VARIABLE seg        :BIT_VECTOR (0 TO 6);
BEGIN
     a <= seg(0);
     b <= seg(1);
     c <= seg(2);
     d <= seg(3);
     e <= seg(4);
     f <= seg(5);
     g <= seg(6);

     CASE bin IS         -- segments   abcdefg
          WHEN "0000"    =>   seg := "0000001";
          WHEN "0001"    =>   seg := "1001111";
          WHEN "0010"    =>   seg := "0010010";
          WHEN "0011"    =>   seg := "0000110";
          WHEN "0100"    =>   seg := "1001100";
          WHEN "0101"    =>   seg := "0100100";
          WHEN "0110"    =>   seg := "1100000";
          WHEN "0111"    =>   seg := "0001111";
          WHEN "1000"    =>   seg := "0000000";
          WHEN "1001"    =>   seg := "0001100";
          WHEN OTHERS    =>   seg := "1111111";
     END CASE;
END PROCESS;
END vhdl;
```

17.7 Memory decoder

```
SUBDESIGN  mem_decode
(a[15..0], memr, memw          :INPUT;
 cs[5..0]                      :OUTPUT;)
BEGIN
If (memr == GND & memw == VCC) # (memr == VCC & memw == GND) THEN
   IF    a[] >= H"0000" & a[] <= H"03FF" THEN  cs[] = B"111110";
   ELSIF a[] >= H"0400" & a[] <= H"07FF" THEN  cs[] = B"111101";
   ELSIF a[] >= H"0800" & a[] <= H"0BFF" THEN  cs[] = B"111011";
   ELSIF a[] >= H"0C00" & a[] <= H"0FFF" THEN  cs[] = B"110111";
   ELSIF a[] >= H"E800" & a[] <= H"EBFF" THEN  cs[] = B"101111";
   ELSIF a[] >= H"EC00" & a[] <= H"EFFF" THEN  cs[] = B"011111";
   ELSE                                        cs[] = B"111111";
   END IF;
ELSE                                           cs[] = B"111111";
END IF;
END;
```

```
ENTITY  mem_decode  IS
PORT (a                        :IN INTEGER RANGE 0 TO 65535;
       memr, memw              :IN BIT;
       cs                      :OUT BIT_VECTOR (5 DOWNTO 0) );
END mem_decode;
ARCHITECTURE vhdl OF mem_decode IS
SIGNAL     mem_en              :BIT;
BEGIN
mem_en <= memr XOR memw;
PROCESS (a, mem_en)
BEGIN
   IF  mem_en = '1' THEN
      IF       a >= 16#0000# AND a <= 16#03FF#     THEN
               cs <= "111110";
      ELSIF    a >= 16#0400# AND a <= 16#07FF#     THEN
               cs <= "111101";
      ELSIF    a >= 16#0800# AND a <= 16#0BFF#     THEN
               cs <= "111011";
      ELSIF    a >= 16#0C00# AND a <= 16#0FFF#     THEN
               cs <= "110111";
      ELSIF    a >= 16#E800# AND a <= 16#EBFF#     THEN
               cs <= "101111";
      ELSIF    a >= 16#EC00# AND a <= 16#EFFF#     THEN
               cs <= "011111";
      ELSE     cs <= "111111";
      END IF;
   ELSE        cs <= "111111";
   END IF;
END PROCESS;
END vhdl;
```

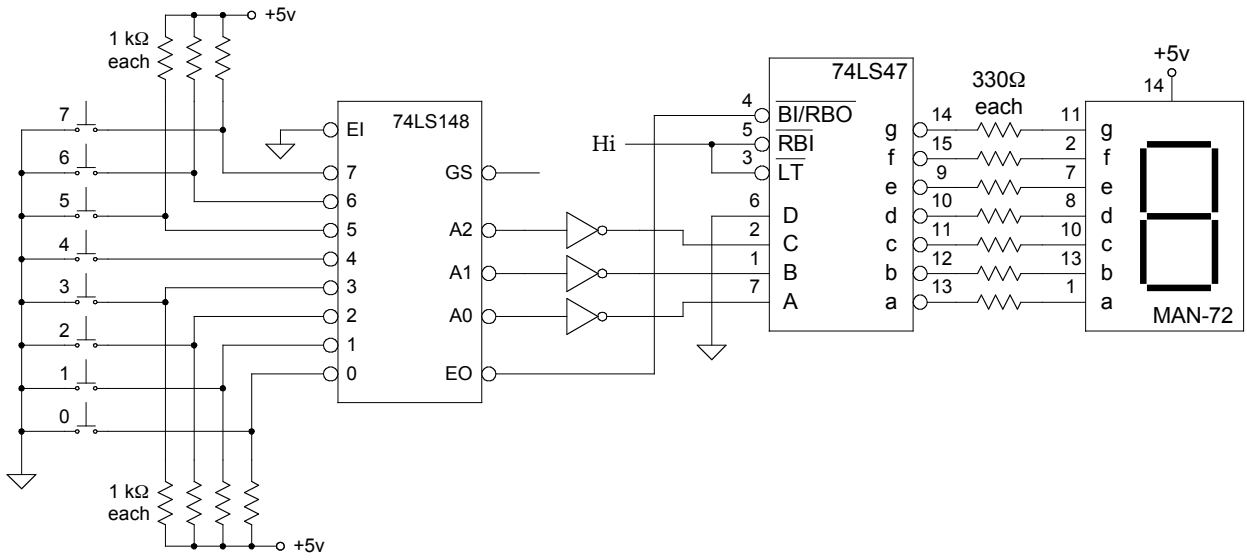
17.8 State decoder

```
SUBDESIGN  state_decoder                -- AHDL solution
(     in[6..0]          :INPUT;
      y[9..1]           :OUTPUT;   )

BEGIN
      DEFAULTS
            y[] = B"000000000";         -- default inactive
      END DEFAULTS;
      TABLE                   -- selected states to decode
            in[]  =>    y[];
            10    =>    B"000000001";
            20    =>    B"000000010";
            30    =>    B"000000100";
            40    =>    B"000001000";
            50    =>    B"000010000";
            60    =>    B"000100000";
            70    =>    B"001000000";
            80    =>    B"010000000";
            90    =>    B"100000000";
      END TABLE;
END;
```

```
ENTITY  state_decoder  IS               -- VHDL solution
PORT (     input      :IN INTEGER RANGE 0 TO 99;
           y          :OUT BIT_VECTOR (9 DOWNTO 1)    );
END state_decoder;

ARCHITECTURE  vhdl  OF  state_decoder  IS
BEGIN
      PROCESS (input)       -- invoke on input change
      BEGIN
            CASE  input  IS             -- decoded states
                  WHEN  10       =>      y <= "000000001";
                  WHEN  20       =>      y <= "000000010";
                  WHEN  30       =>      y <= "000000100";
                  WHEN  40       =>      y <= "000001000";
                  WHEN  50       =>      y <= "000010000";
                  WHEN  60       =>      y <= "000100000";
                  WHEN  70       =>      y <= "001000000";
                  WHEN  80       =>      y <= "010000000";
                  WHEN  90       =>      y <= "100000000";
                  WHEN OTHERS    =>      y <= "000000000";
            END CASE;
      END PROCESS;
END vhdl;
```

Unit 18 Encoder Applications

18.1 Encoder and display

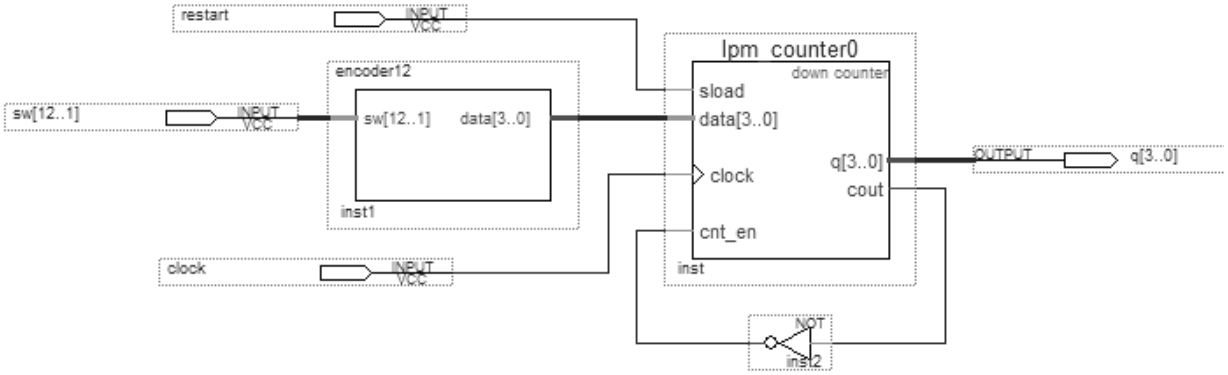


18.2 HDL 74148 encoder

```
SUBDESIGN  encoder_ahdl          -- AHDL solution
(
      in[7..0], ei               :INPUT;
      a[2..0], gs, eo            :OUTPUT;
)

BEGIN
IF !ei THEN                  -- active-low enable
      TABLE
            -- priority encoder with inverted outputs
            in[]             =>    a[], gs, eo;
            B"11111111"      =>    B"111", VCC, GND;
            B"11111110"      =>    B"111", GND, VCC;
            B"1111110X"      =>    B"110", GND, VCC;
            B"111110XX"      =>    B"101", GND, VCC;
            B"11110XXX"      =>    B"100", GND, VCC;
            B"1110XXXX"      =>    B"011", GND, VCC;
            B"110XXXXX"      =>    B"010", GND, VCC;
            B"10XXXXXX"      =>    B"001", GND, VCC;
            B"0XXXXXXX"      =>    B"000", GND, VCC;
      END TABLE;
ELSE a[] = B"111"; gs = VCC; eo = VCC;      -- disabled
END IF;
END;
```

```
ENTITY encoder_vhdl IS           -- VHDL solution
PORT (
     input           :IN BIT_VECTOR (7 DOWNTO 0);
     ei              :IN BIT;
     a               :OUT BIT_VECTOR (2 DOWNTO 0);
     gs, eo          :OUT BIT
);
END encoder_vhdl;

ARCHITECTURE vhdl OF encoder_vhdl IS
BEGIN
PROCESS (input, ei)
BEGIN

     IF ei = '0' THEN           -- active-low enable
               -- check highest priority input first
          IF       input(7) = '0' THEN
                   a <= "000"; gs <= '0'; eo <= '1';
          ELSIF    input(6) = '0' THEN
                   a <= "001"; gs <= '0'; eo <= '1';
          ELSIF    input(5) = '0' THEN
                   a <= "010"; gs <= '0'; eo <= '1';
          ELSIF    input(4) = '0' THEN
                   a <= "011"; gs <= '0'; eo <= '1';
          ELSIF    input(3) = '0' THEN
                   a <= "100"; gs <= '0'; eo <= '1';
          ELSIF    input(2) = '0' THEN
                   a <= "101"; gs <= '0'; eo <= '1';
          ELSIF    input(1) = '0' THEN
                   a <= "110"; gs <= '0'; eo <= '1';
          ELSIF    input(0) = '0' THEN
                   a <= "111"; gs <= '0'; eo <= '1';
          ELSE             -- no input
                   a <= "111"; gs <= '1'; eo <= '0';
          END IF;
     ELSE                  -- disabled
                   a <= "111"; gs <= '1'; eo <= '1';
     END IF;

END PROCESS;
END vhdl;
```

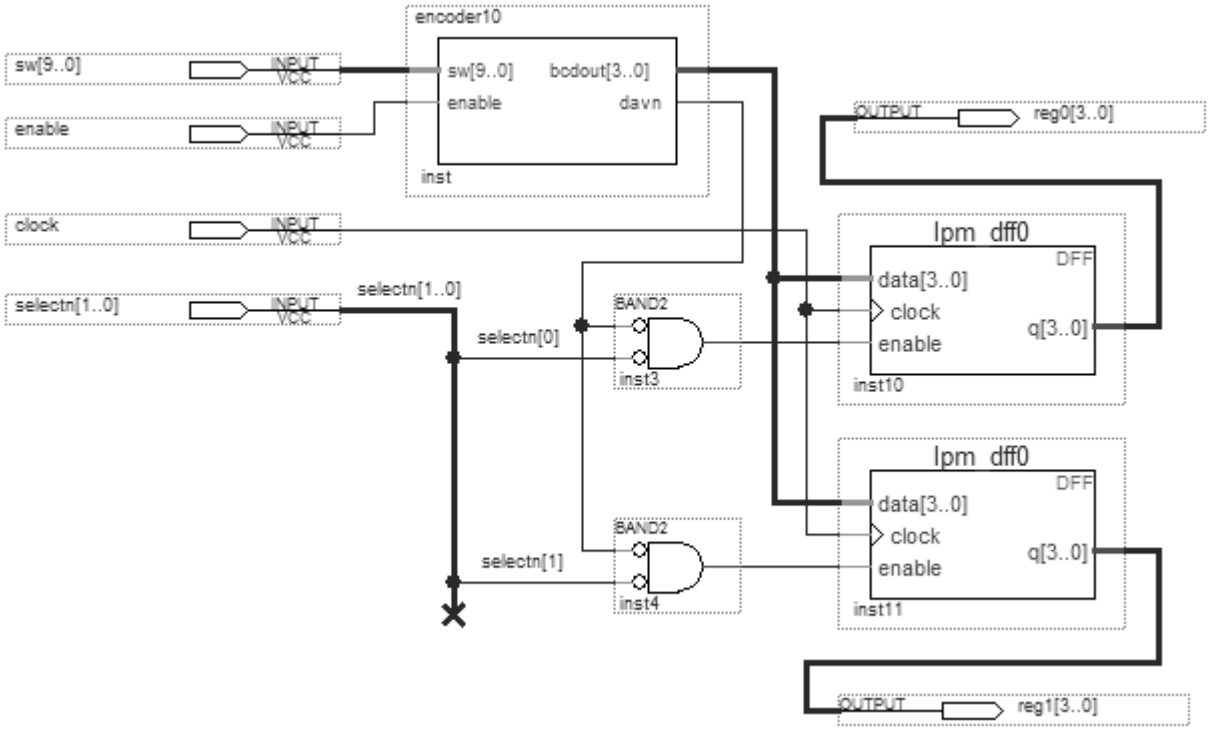
18.3 Variable self-stopping counter




```
SUBDESIGN  encoder12
(
      sw[12..1]         :INPUT;
      data[3..0]        :OUTPUT;
)
BEGIN
      TABLE
            sw[]              =>    data[];
--            CBA987654321
            B"111111111111"   =>    B"0000";
            B"111111111110"   =>    B"0001";
            B"11111111110X"   =>    B"0010";
            B"1111111110XX"   =>    B"0011";
            B"111111110XXX"   =>    B"0100";
            B"11111110XXXX"   =>    B"0101";
            B"1111110XXXXX"   =>    B"0110";
            B"111110XXXXXX"   =>    B"0111";
            B"11110XXXXXXX"   =>    B"1000";
            B"1110XXXXXXXX"   =>    B"1001";
            B"110XXXXXXXXX"   =>    B"1010";
            B"10XXXXXXXXXX"   =>    B"1011";
            B"0XXXXXXXXXXX"   =>    B"1100";
      END TABLE;
END;
```

```
ENTITY  encoder12  IS
PORT (      sw                :IN BIT_VECTOR (12 DOWNTO 1);
            data        :OUT INTEGER RANGE 0 TO 15 );
END encoder12;

ARCHITECTURE vhdl OF encoder12 IS
BEGIN
data <=     12 WHEN sw(12) = '0' ELSE
            11 WHEN sw(11) = '0' ELSE
            10 WHEN sw(10) = '0' ELSE
             9 WHEN sw(9) = '0' ELSE
             8 WHEN sw(8) = '0' ELSE
             7 WHEN sw(7) = '0' ELSE
             6 WHEN sw(6) = '0' ELSE
             5 WHEN sw(5) = '0' ELSE
             4 WHEN sw(4) = '0' ELSE
             3 WHEN sw(3) = '0' ELSE
             2 WHEN sw(2) = '0' ELSE
             1 WHEN sw(1) = '0' ELSE
             0 ;
END vhdl;
```

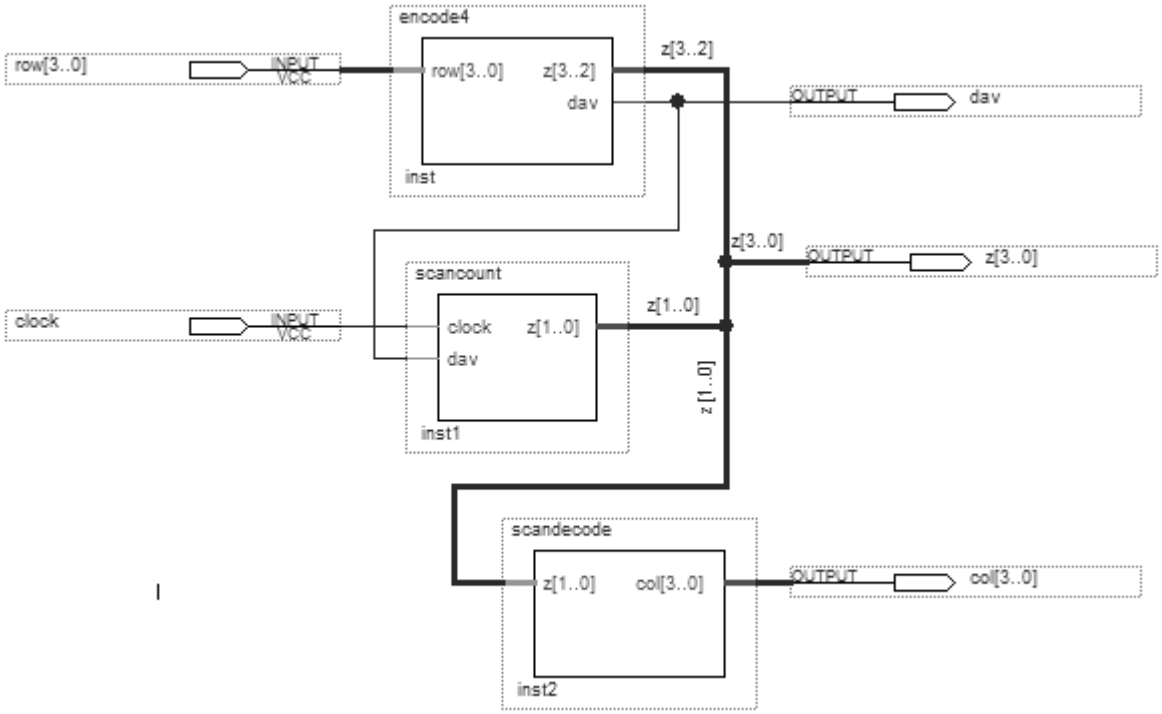
18.4 BCD encoder



```
SUBDESIGN  encoder10
(
      sw[9..0], enable      :INPUT;
      bcdout[3..0], davn          :OUTPUT;
)
BEGIN
      TABLE
            sw[], enable        =>      bcdout[], davn;
            B"XXXXXXXXXX", 0    =>      B"0000", 1;
            B"1111111111", 1    =>      B"0000", 1;
            B"1111111110", 1    =>      B"0000", 0;
            B"111111110X", 1    =>      B"0001", 0;
            B"11111110XX", 1    =>      B"0010", 0;
            B"1111110XXX", 1    =>      B"0011", 0;
            B"111110XXXX", 1    =>      B"0100", 0;
            B"11110XXXXX", 1    =>      B"0101", 0;
            B"1110XXXXXX", 1    =>      B"0110", 0;
            B"110XXXXXXX", 1    =>      B"0111", 0;
            B"10XXXXXXXX", 1    =>      B"1000", 0;
            B"0XXXXXXXXX", 1    =>      B"1001", 0;
      END TABLE;
END;
```

```
ENTITY  encoder10  IS
PORT (      sw          :IN BIT_VECTOR (9 DOWNTO 0);
            enable      :IN BIT;
            bcdout      :OUT BIT_VECTOR (3 DOWNTO 0);
            davn  :OUT BIT );
END encoder10;

ARCHITECTURE vhdl OF encoder10 IS
BEGIN
davn <= '1' WHEN (enable = '0' OR sw = "1111111111")
      ELSE '0';
bcdout  <=        "0000" WHEN enable = '0' ELSE
                  "1001" WHEN sw(9) = '0' ELSE
                  "1000" WHEN sw(8) = '0' ELSE
                  "0111" WHEN sw(7) = '0' ELSE
                  "0110" WHEN sw(6) = '0' ELSE
                  "0101" WHEN sw(5) = '0' ELSE
                  "0100" WHEN sw(4) = '0' ELSE
                  "0011" WHEN sw(3) = '0' ELSE
                  "0010" WHEN sw(2) = '0' ELSE
                  "0001" WHEN sw(1) = '0' ELSE
                  "0000";
END vhdl;
```

18.5 Scanning encoder



```
SUBDESIGN encode4
(
      row[3..0]              :INPUT;
      z[3..2]                :OUTPUT;
      dav                    :OUTPUT;
)
BEGIN
      TABLE
            row[] =>    z[], dav;
            B"1111"     =>    B"00", 0;
            B"1110"     =>    B"00", 1;
            B"110X"     =>    B"01", 1;
            B"10XX"     =>    B"10", 1;
            B"0XXX"     =>    B"11", 1;
      END TABLE;
END;
```

```
SUBDESIGN scancount
(
      clock, dav             :INPUT;
      z[1..0]                :OUTPUT;
)
VARIABLE
      z[1..0]                :DFF;
BEGIN
      z[].clk = clock;

      IF (dav == GND)  THEN
            z[].d = z[].q + 1;
      ELSE
            z[].d = z[].q;
      END IF;
END;
```

```
SUBDESIGN scandecode
(
      z[1..0]                :INPUT;
      col[3..0]              :OUTPUT;
)
BEGIN
      CASE z[] IS
            WHEN 0      =>    col[] = B"1110";
            WHEN 1      =>    col[] = B"1101";
            WHEN 2      =>    col[] = B"1011";
            WHEN 3      =>    col[] = B"0111";
      END CASE;
END;
```

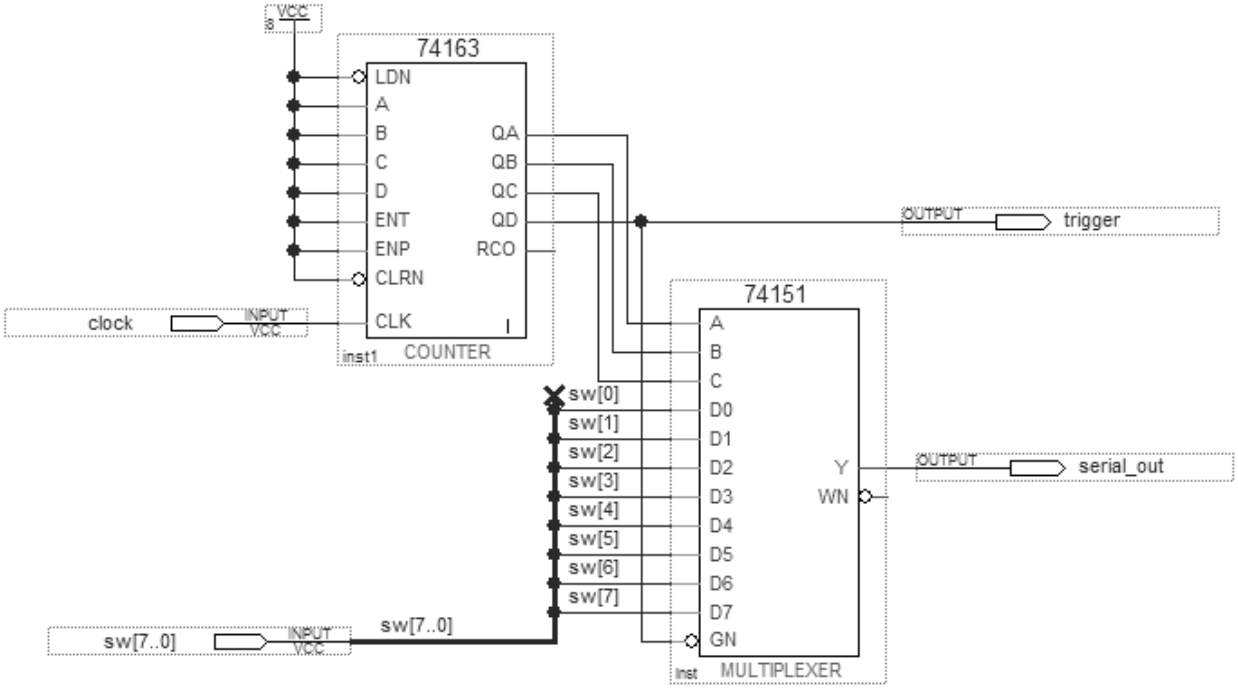
```
ENTITY encode4 IS
PORT (row        :IN BIT_VECTOR (3 DOWNTO 0);
      z          :OUT BIT_VECTOR (3 DOWNTO 2);
      dav        :OUT BIT);
END encode4;
ARCHITECTURE vhdl OF encode4 IS
BEGIN
z <=  "00" WHEN row(0) = '0' ELSE
      "01" WHEN row(1) = '0' ELSE
      "10" WHEN row(2) = '0' ELSE
      "11" WHEN row(3) = '0' ELSE
      "00";
dav <= '0' WHEN row = "1111" ELSE '1';
END;
```

```
ENTITY scancount IS
PORT (clock, dav      :IN BIT;
      z               :OUT INTEGER RANGE 0 TO 3);
END scancount;
ARCHITECTURE vhdl OF scancount IS
BEGIN
PROCESS (clock)
VARIABLE count  :INTEGER RANGE 0 TO 3;
BEGIN
     IF (clock'EVENT AND clock = '1') THEN
          IF (dav = '0')  THEN
               count := count + 1;
          END IF;
     END IF;
     z <= count;
END PROCESS;
END;
```

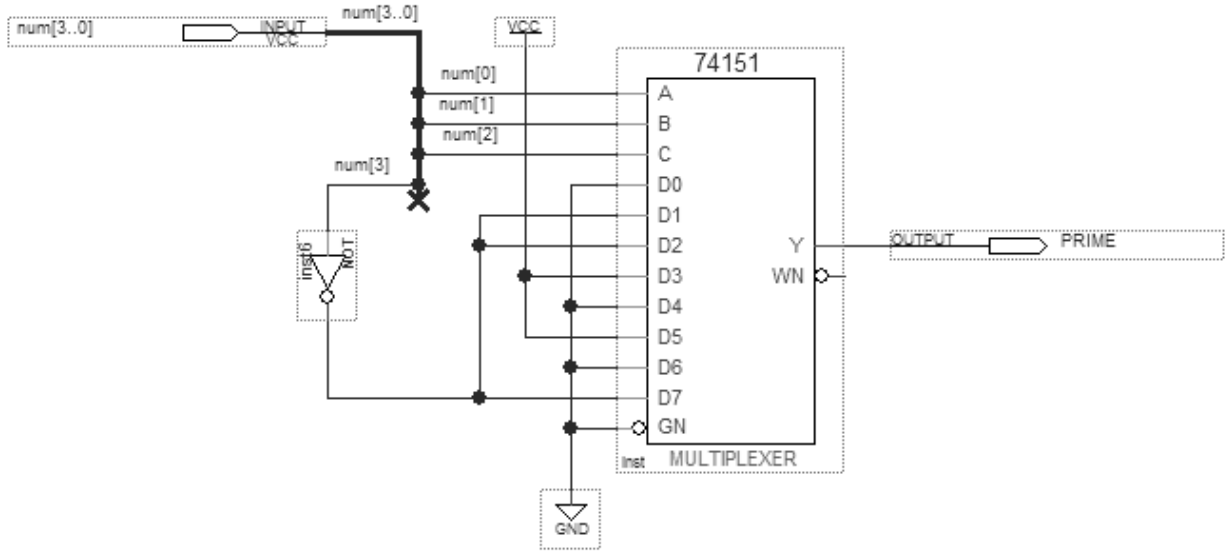
```
ENTITY scandecode IS
PORT (z          :IN INTEGER RANGE 0 TO 3;
      col        :OUT BIT_VECTOR (3 DOWNTO 0));
END scandecode;
ARCHITECTURE vhdl OF scandecode IS
BEGIN
PROCESS (z)
BEGIN
     CASE z IS
          WHEN 0     =>    col <= "1110";
          WHEN 1     =>    col <= "1101";
          WHEN 2     =>    col <= "1011";
          WHEN 3     =>    col <= "0111";
     END CASE;
END PROCESS;
END;
```

Unit 19 Multiplexer Applications

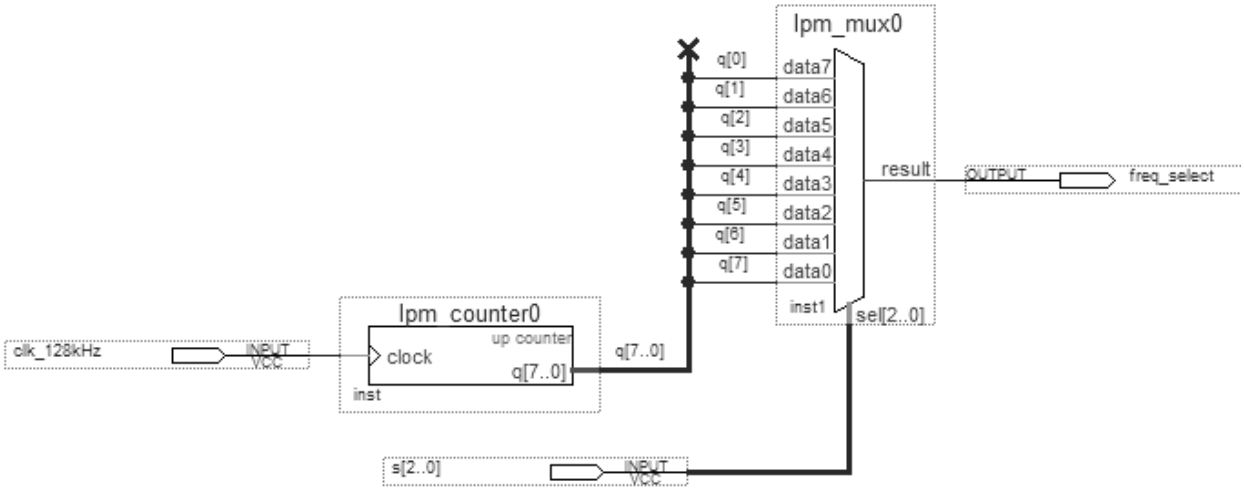
19.1 Serial data word generator



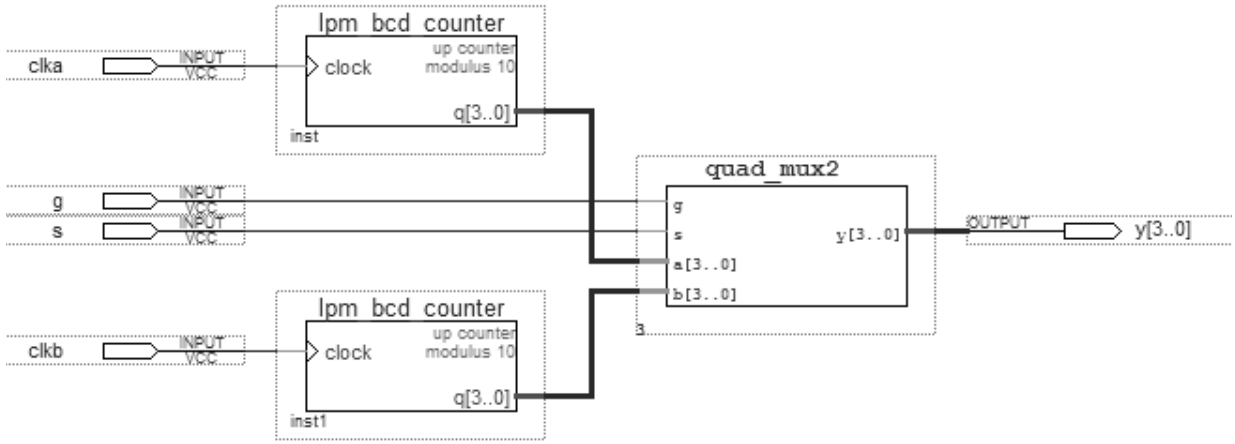
19.2 Prime number detector



19.3 Variable frequency divider



19.4 Multiplexed BCD display




```
SUBDESIGN        quad_mux2
(g, s, a[3..0], b[3..0]                  :INPUT;
 y[3..0]                                 :OUTPUT;)
BEGIN
      IF g  THEN
            CASE  s  IS
                  WHEN 0  =>  y[] = a[];
                  WHEN 1  =>  y[] = b[];
            END CASE;
      ELSE  y[] = H"F";
      END IF;
END;
```

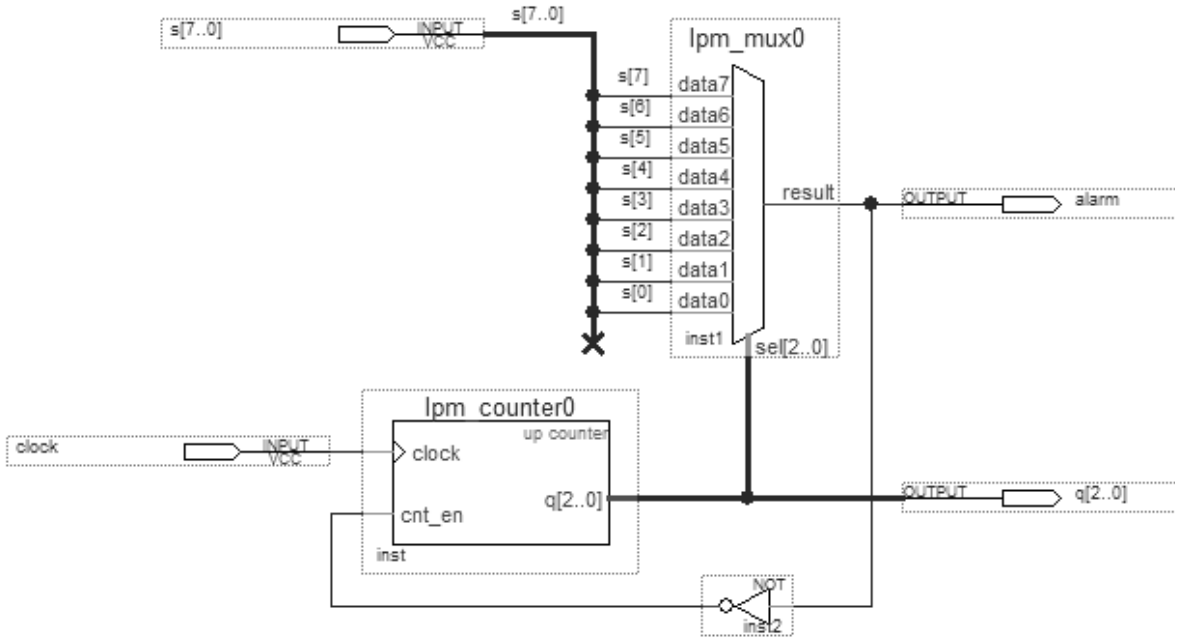
```
ENTITY quad_mux2 IS
PORT (     g, s  : IN BIT;
           a, b  : IN BIT_VECTOR (3 DOWNTO 0);
           y          : OUT BIT_VECTOR (3 DOWNTO 0) );
END quad_mux2;

ARCHITECTURE vhdl OF quad_mux2 IS
BEGIN
      PROCESS (g, s, a, b)
      BEGIN
            IF g = '1' THEN
                  CASE  s  IS
                        WHEN '0'  =>  y <= a;
                        WHEN '1'  =>  y <= b;
                  END CASE;
            ELSE  y <= "1111";
            END IF;
      END PROCESS;
END vhdl;
```

19.5 Quad, 3-channel MUX

```
SUBDESIGN  quad_mux3             -- AHDL solution
(
      s[1..0], a[3..0], b[3..0], c[3..0]    :INPUT;
      y[3..0]                               :OUTPUT;
)

BEGIN
      CASE  s[]  IS               -- input channel select
            WHEN 0  =>  y[] = 0;        -- disabled
            WHEN 1  =>  y[] = a[];      -- channel A
            WHEN 2  =>  y[] = b[];      -- channel B
            WHEN 3  =>  y[] = c[];      -- channel C
      END CASE;
END;
```

```
ENTITY  quad_mux3 IS              -- VHDL solution
PORT (     s                  :IN INTEGER RANGE 0 TO 3;
           a, b, c            :IN BIT_VECTOR (3 DOWNTO 0);
           y                  :OUT BIT_VECTOR (3 DOWNTO 0)
);
END quad_mux3;

ARCHITECTURE vhdl OF quad_mux3 IS
BEGIN
      PROCESS (s, a, b, c)        -- monitor all inputs
      BEGIN
           CASE  s  IS            -- input channel select
                WHEN 0  =>  y <= "0000";    -- disabled
                WHEN 1  =>  y <= a;         -- channel A
                WHEN 2  =>  y <= b;         -- channel B
                WHEN 3  =>  y <= c;         -- channel C
           END CASE;
      END PROCESS;
END vhdl;
```

19.6 Alarm system



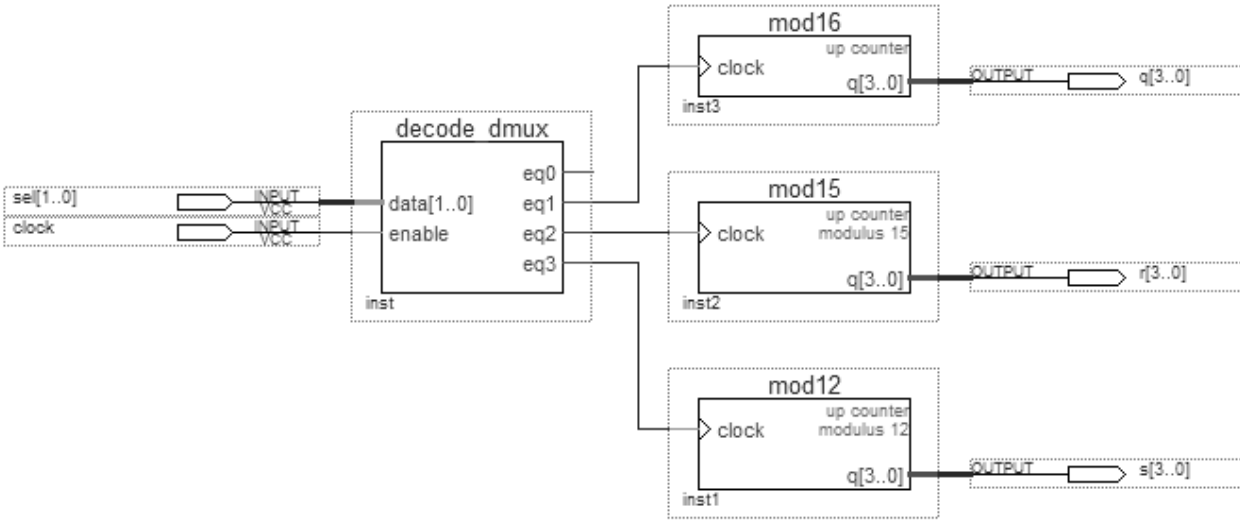
19.7 16-channel, 1-bit MUX

```
SUBDESIGN  mux16chan       -- AHDL solution
(
     d[15..0]         :INPUT;
     sel[3..0]        :INPUT;
     y                :OUTPUT;
)

BEGIN
     CASE  sel[]  IS       -- select input channel
          WHEN 0   =>  y = d[0];
          WHEN 1   =>  y = d[1];
          WHEN 2   =>  y = d[2];
          WHEN 3   =>  y = d[3];
          WHEN 4   =>  y = d[4];
          WHEN 5   =>  y = d[5];
          WHEN 6   =>  y = d[6];
          WHEN 7   =>  y = d[7];
          WHEN 8   =>  y = d[8];
          WHEN 9   =>  y = d[9];
          WHEN 10  =>  y = d[10];
          WHEN 11  =>  y = d[11];
          WHEN 12  =>  y = d[12];
          WHEN 13  =>  y = d[13];
          WHEN 14  =>  y = d[14];
          WHEN 15  =>  y = d[15];
     END CASE;
END;
```

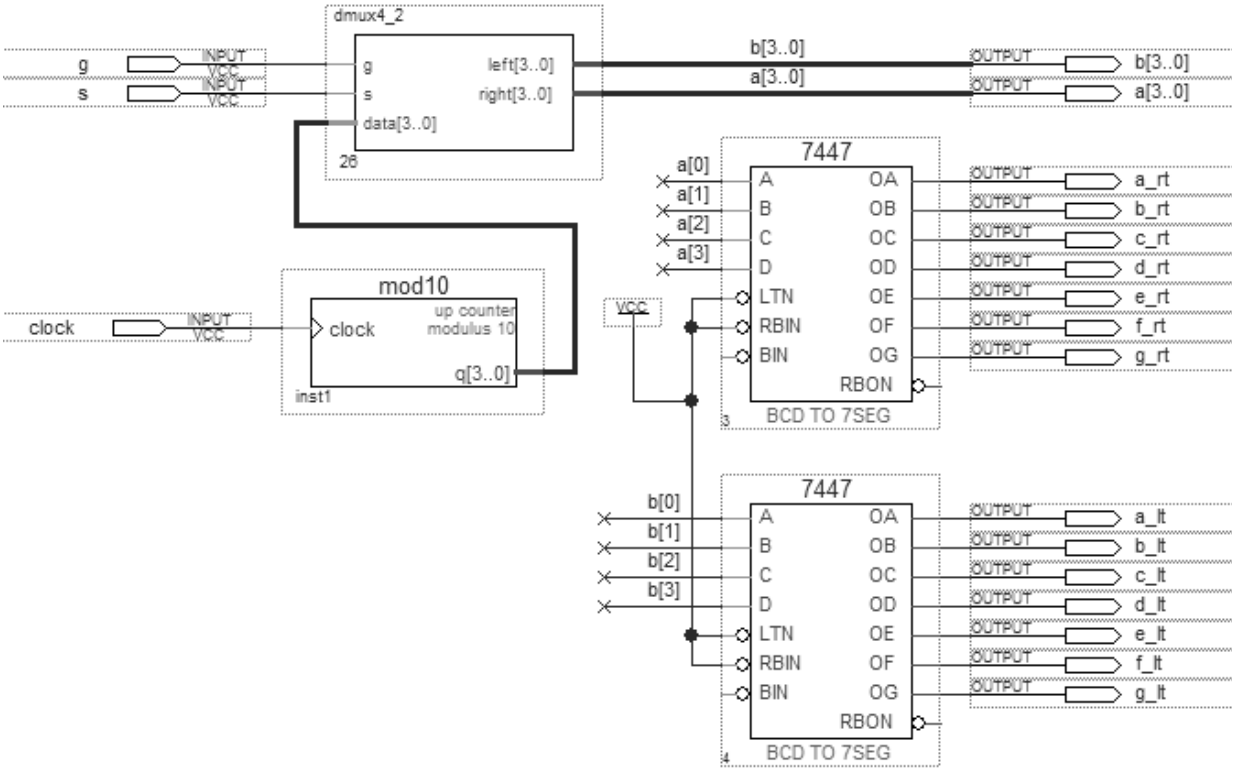
```
ENTITY  mux16chan  IS       -- VHDL solution
PORT (
     d          :IN BIT_VECTOR (15 DOWNTO 0);
     sel        :IN INTEGER RANGE 0 TO 15;
     y          :OUT BIT );
END mux16chan;

ARCHITECTURE vhdl OF mux16chan IS
BEGIN

PROCESS (d, sel)            -- monitor inputs
     BEGIN

     CASE  sel  IS          -- select input channel
          WHEN 0   =>  y <= d(0);
          WHEN 1   =>  y <= d(1);
          WHEN 2   =>  y <= d(2);
          WHEN 3   =>  y <= d(3);
          WHEN 4   =>  y <= d(4);
          WHEN 5   =>  y <= d(5);
          WHEN 6   =>  y <= d(6);
          WHEN 7   =>  y <= d(7);
          WHEN 8   =>  y <= d(8);
          WHEN 9   =>  y <= d(9);
          WHEN 10  =>  y <= d(10);
          WHEN 11  =>  y <= d(11);
          WHEN 12  =>  y <= d(12);
          WHEN 13  =>  y <= d(13);
          WHEN 14  =>  y <= d(14);
          WHEN 15  =>  y <= d(15);
     END CASE;

END PROCESS;
END vhdl;
```

Unit 20 Demultiplexer Applications

20.1 Clock DMUX



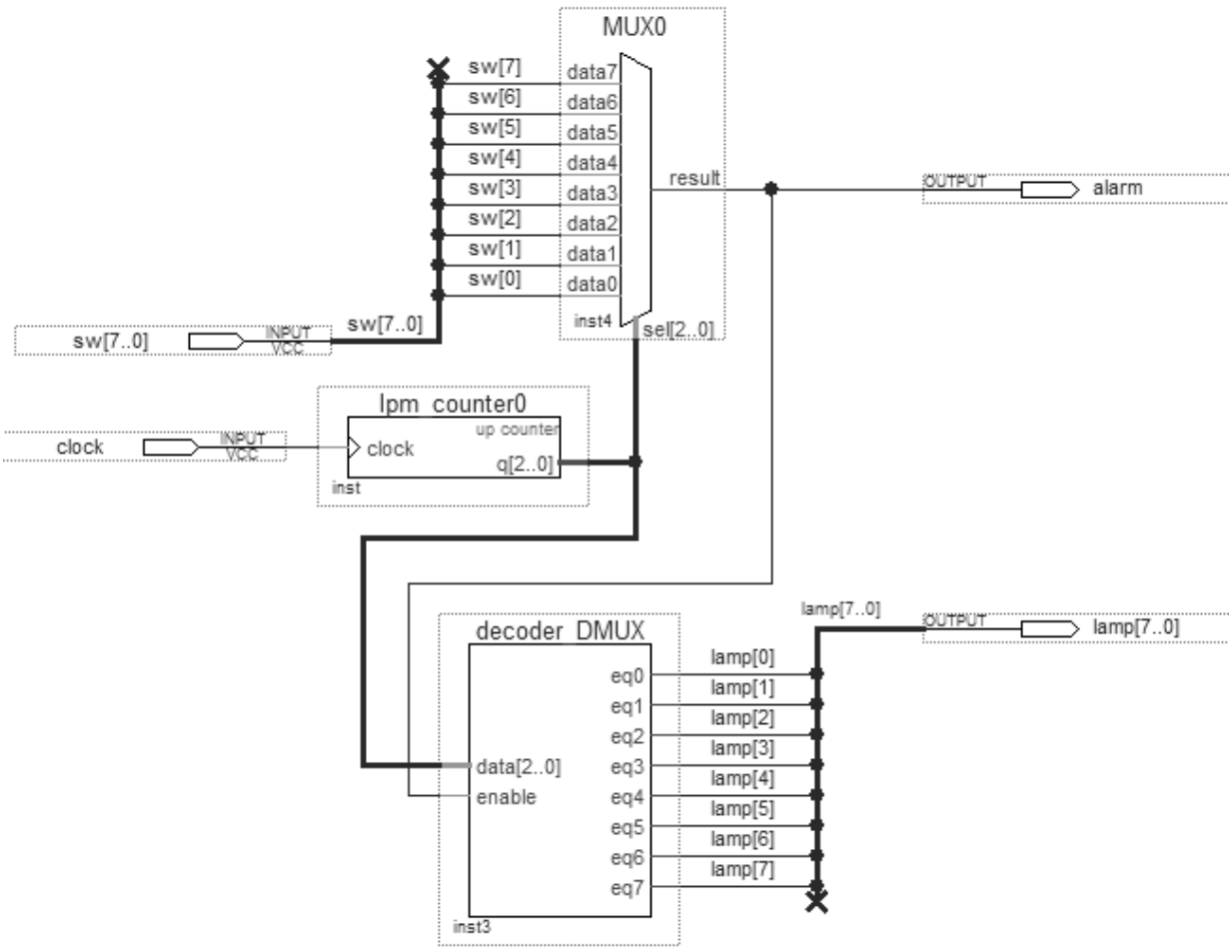
20.2 BCD counter demultiplexer



```
SUBDESIGN  dmux4_2
(
      g, s, data[3..0]                  :INPUT;
      left[3..0], right[3..0]           :OUTPUT;
)
BEGIN
      DEFAULTS
            left[] = H"F";
            right[] = H"F";
      END DEFAULTS;
      IF  !g  THEN
            CASE  s  IS
                  WHEN  0  =>  left[] = data[];
                  WHEN  1  =>  right[] = data[];
            END CASE;
      END IF;
END;
```

```
ENTITY dmux4_2 IS
PORT (     g, s             :IN BIT;
           data             :IN BIT_VECTOR (3 DOWNTO 0);
           left, right            :OUT BIT_VECTOR (3 DOWNTO
0) );
END dmux4_2;

ARCHITECTURE vhdl OF dmux4_2 IS
BEGIN
      PROCESS (g, s, data)
      BEGIN
            IF  g = '1'  THEN
                  left <= "1111";  right <= "1111";
            ELSIF  s = '0'  THEN
                  left <= data;   right <= "1111";
            ELSE
                  left <= "1111";  right <= data;
            END IF;
      END PROCESS;
END vhdl;
```

20.3 Alarm indicator



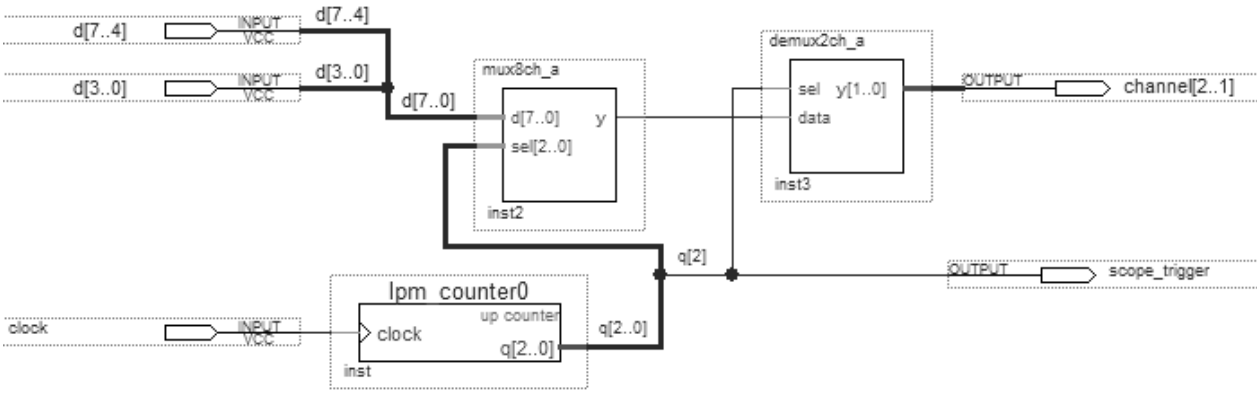
20.4 8-channel, 1-bit DMUX

```
SUBDESIGN  demux_ahdl       -- AHDL solution
(
      sel[2..0]         :INPUT;
      data, en          :INPUT;
      y[7..0]           :OUTPUT;
)

BEGIN
      DEFAULTS
            y[] = B"00000000";
                        -- inactive outputs are low
      END DEFAULTS;
```

```
      IF en  THEN           -- active-high enable
            CASE sel[] IS   -- select output channel
                  WHEN 0    =>   y[0] = data;
                  WHEN 1    =>   y[1] = data;
                  WHEN 2    =>   y[2] = data;
                  WHEN 3    =>   y[3] = data;
                  WHEN 4    =>   y[4] = data;
                  WHEN 5    =>   y[5] = data;
                  WHEN 6    =>   y[6] = data;
                  WHEN 7    =>   y[7] = data;
            END CASE;
      END IF;
END;
```

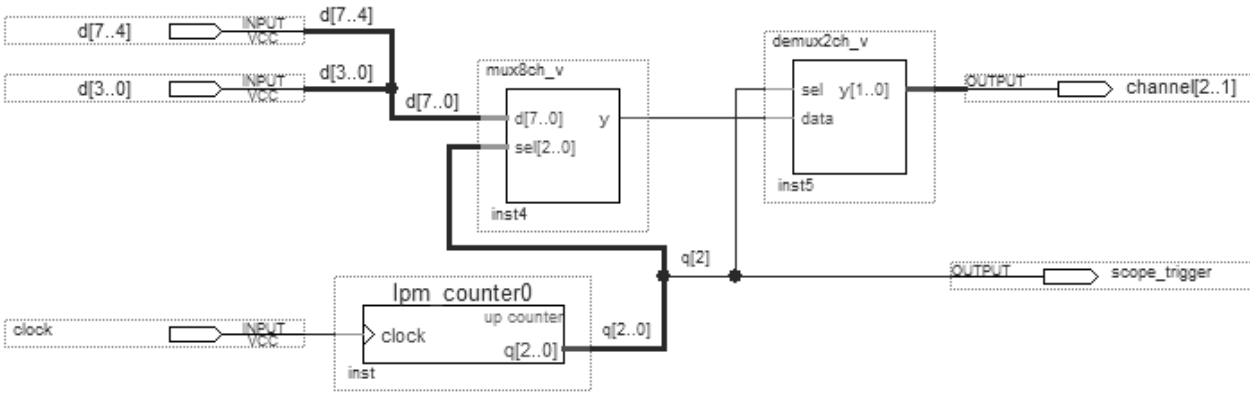
```
ENTITY  demux_vhdl IS       -- VHDL solution
PORT (
      sel              :IN BIT_VECTOR (2 DOWNTO 0);
      data, en         :IN BIT;
      y                :OUT BIT_VECTOR (7 DOWNTO 0)
);
END demux_vhdl;

ARCHITECTURE vhdl OF demux_vhdl IS
BEGIN
PROCESS (data, sel, en)
BEGIN

      IF en = '1' THEN      -- active-high enable

            CASE sel IS     -- select output channel
                  WHEN "000"  =>   y <= "0000000" & data;
                              -- concatenate output bit vector
                  WHEN "001"  =>   y <= "000000" & data & '0';
                  WHEN "010"  =>   y <= "00000" & data & "00";
                  WHEN "011"  =>   y <= "0000" & data & "000";
                  WHEN "100"  =>   y <= "000" & data & "0000";
                  WHEN "101"  =>   y <= "00" & data & "00000";
                  WHEN "110"  =>   y <= '0' & data & "000000";
                  WHEN "111"  =>   y <= data & "0000000";
            END CASE;

      ELSE       y <= "00000000";      -- disabled output
      END IF;

END PROCESS;
END vhdl;
```

20.5 2-channel data transmitter




```
SUBDESIGN  mux8ch_a
(d[7..0]          :INPUT;
 sel[2..0]        :INPUT;
 y                :OUTPUT;)
BEGIN
      CASE  sel[]  IS
            WHEN 0  =>  y = d[0];
            WHEN 1  =>  y = d[1];
            WHEN 2  =>  y = d[2];
            WHEN 3  =>  y = d[3];
            WHEN 4  =>  y = d[4];
            WHEN 5  =>  y = d[5];
            WHEN 6  =>  y = d[6];
            WHEN 7  =>  y = d[7];
      END CASE;
END;
```

```
SUBDESIGN  demux2ch_a
(sel, data        :INPUT;
 y[1..0]          :OUTPUT;)
BEGIN
      DEFAULTS
            y[] = B"00";
      END DEFAULTS;
      CASE sel IS
            WHEN 0      =>    y[0] = data;
            WHEN 1      =>    y[1] = data;
      END CASE;
END;
```

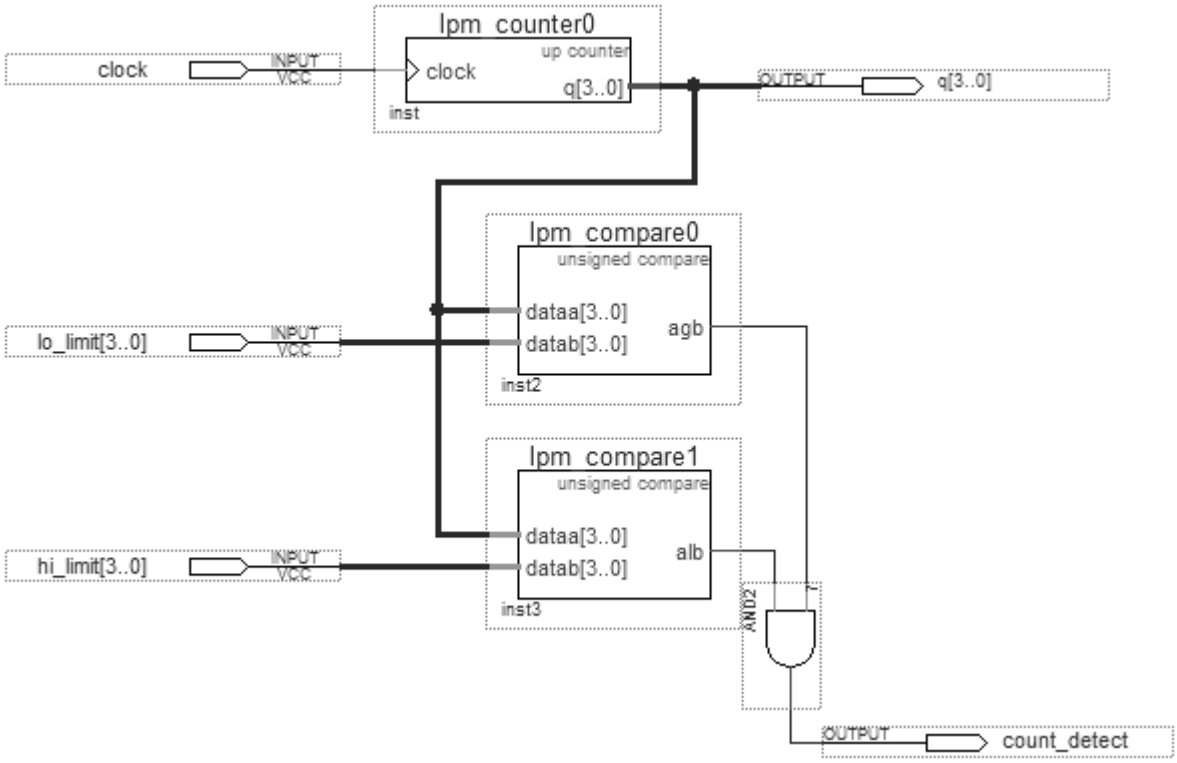



```
ENTITY  mux8ch_v  IS
PORT (d          :IN BIT_VECTOR (7 DOWNTO 0);
      sel        :IN INTEGER RANGE 0 TO 7;
      y          :OUT BIT);
END mux8ch_v;
ARCHITECTURE vhdl OF mux8ch_v IS
BEGIN
      PROCESS (d, sel)
      BEGIN
      CASE  sel  IS
            WHEN 0  =>  y <= d(0);
            WHEN 1  =>  y <= d(1);
            WHEN 2  =>  y <= d(2);
            WHEN 3  =>  y <= d(3);
            WHEN 4  =>  y <= d(4);
            WHEN 5  =>  y <= d(5);
            WHEN 6  =>  y <= d(6);
            WHEN 7  =>  y <= d(7);
      END CASE;
      END PROCESS;
END vhdl;
```

```
ENTITY  demux2ch_v  IS
PORT (sel, data       :IN BIT;
      y               :OUT BIT_VECTOR (1 DOWNTO 0));
END demux2ch_v;
ARCHITECTURE vhdl OF demux2ch_v IS
BEGIN
      PROCESS (sel, data)
      BEGIN
      CASE sel IS
            WHEN '0'   =>    y <= ('0' & data);
            WHEN '1'   =>    y <= (data & '0');
      END CASE;
      END PROCESS;
END vhdl;
```

Unit 21 Magnitude Comparator Applications

21.1 Count detector



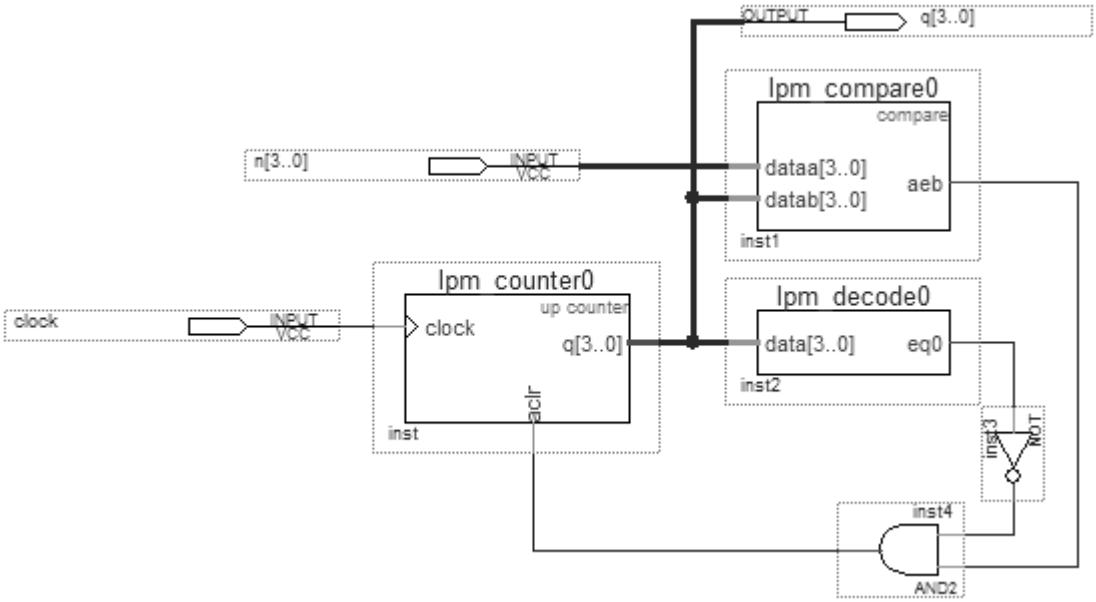
21.2 Adjustable range detector

```
SUBDESIGN  range_detect          -- AHDL solution
(value[4..0], r[1..0]       :INPUT;
 ltr, gtr, rng              :OUTPUT;)
VARIABLE
      lo_value[4..0]        :node;
      hi_value[4..0]        :node;
BEGIN
      DEFAULTS
            ltr = GND;
            gtr = GND;
            rng = GND;
      END DEFAULTS;
      CASE  r[]  IS         -- set range limits
            WHEN  0  =>  lo_value[] = 16;  hi_value[] = 16;
            WHEN  1  =>  lo_value[] = 15;  hi_value[] = 17;
            WHEN  2  =>  lo_value[] = 14;  hi_value[] = 18;
            WHEN  3  =>  lo_value[] = 13;  hi_value[] = 19;
      END CASE;
```

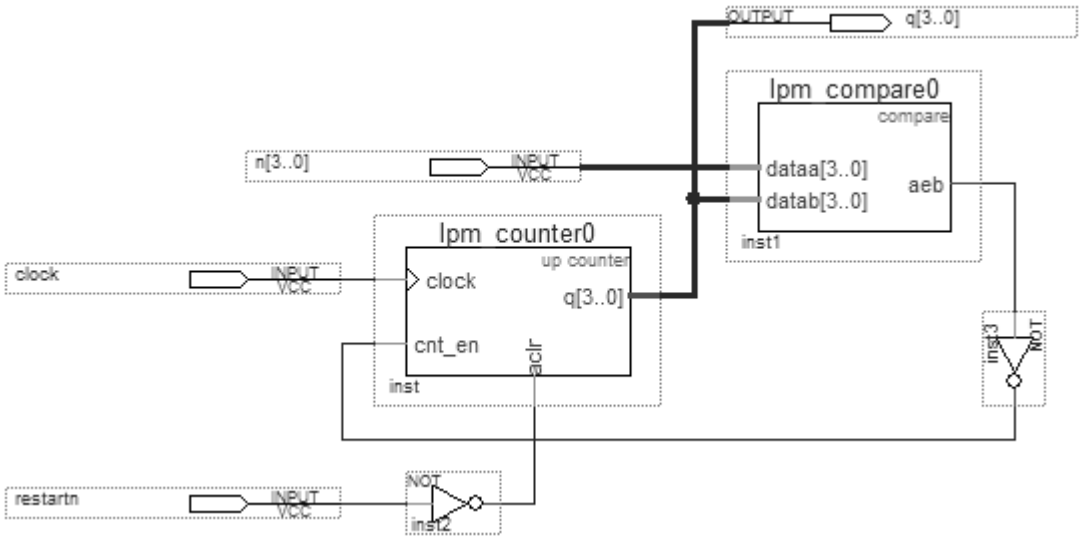
```
                              -- determine active output
     IF     value[] < lo_value[]  THEN  ltr = VCC;
     ELSIF  value[] > hi_value[]  THEN  gtr = VCC;
     ELSE                               rng = VCC;
     END IF;
END;
```

```
ENTITY  range_detect IS          -- VHDL solution
PORT (     value                 :IN INTEGER RANGE 0 TO 31;
           r                     :IN BIT_VECTOR (1 DOWNTO 0);
           ltr, gtr, rng         :OUT BIT    );
END range_detect;

ARCHITECTURE vhdl OF range_detect IS
SIGNAL     lo_value              :INTEGER RANGE 0 TO 31;
SIGNAL     hi_value              :INTEGER RANGE 0 TO 31;
BEGIN
    PROCESS (value, r)
    BEGIN
        CASE  r  IS              -- set range limits
            WHEN  "00"  =>  lo_value <= 16;  hi_value <= 16;
            WHEN  "01"  =>  lo_value <= 15;  hi_value <= 17;
            WHEN  "10"  =>  lo_value <= 14;  hi_value <= 18;
            WHEN  "11"  =>  lo_value <= 13;  hi_value <= 19;
        END CASE;
                                 -- determine active output
        IF      value < lo_value  THEN
                      ltr <= '1';  gtr <= '0';  rng <= '0';
        ELSIF   value > hi_value  THEN
                      ltr <= '0';  gtr <= '1';  rng <= '0';
        ELSE
                      ltr <= '0';  gtr <= '0';  rng <= '1';
        END IF;
    END PROCESS;
END vhdl;
```

21.3 Variable modulus counter



21.4 Variable self-stopping counter



21.5 Number selector

```
SUBDESIGN  num_select            -- AHDL solution
(
      f, a[3..0], b[3..0]        :INPUT;
      y[3..0]                    :OUTPUT;
)

VARIABLE
      agtb                       :NODE;     -- buried node

BEGIN

      agtb = a[] > b[];                     -- hi if a>b

      y[] = a[] & !(f $ agtb)  #  b[] & (f $ agtb);
                -- MUX function to select a or b input
                -- XNOR selects a & XOR selects b

END;
```

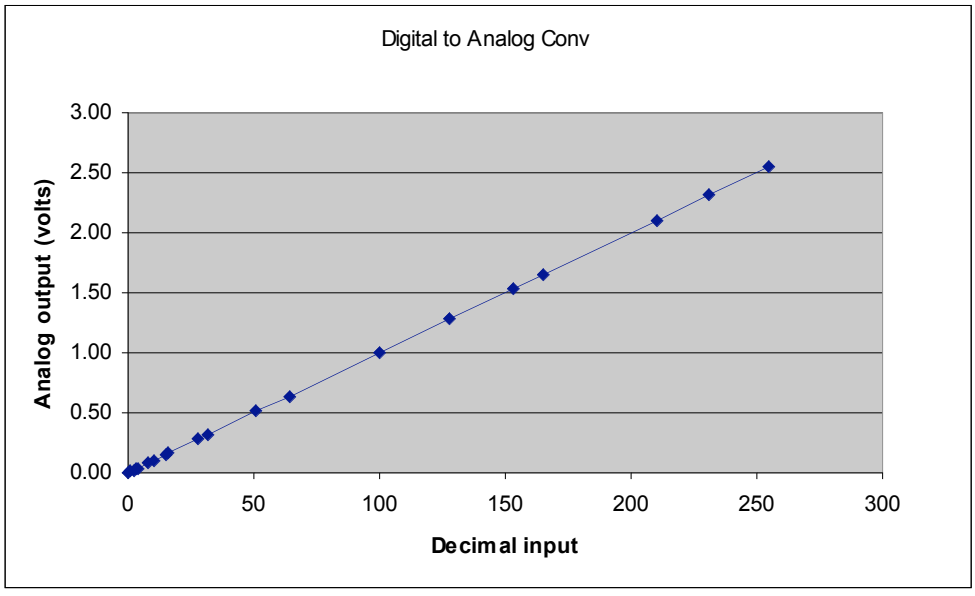
```
ENTITY  num_select IS            -- VHDL solution
PORT (     f          :IN BIT;
           a, b       :IN INTEGER RANGE 0 TO 15;
           y          :OUT INTEGER RANGE 0 TO 15    );
END num_select;

ARCHITECTURE vhdl OF num_select IS
SIGNAL     agtb       :BIT;
SIGNAL     sel_a      :BIT;

BEGIN
      agtb <= '1'      WHEN (a > b) ELSE '0';
                -- hi if a>b

      sel_a <= '1' WHEN ((f XOR agtb) = '0') ELSE '0';
                -- determine when to select input a

      y <= a WHEN (sel_a = '1') ELSE b;
                -- MUX function selects a or b input

END vhdl;
```

Unit 22 Digital/Analog and Analog/Digital Conversion

22.1 Digital-to-analog converter (see Fig. 22-1)

Digital to Analog Conversion

Hex input	Decimal in	Analog out
0	0	0.00
1	1	0.01
2	2	0.02
3	3	0.03
4	4	0.04
8	8	0.08
A	10	0.10
F	15	0.15
10	16	0.16
1C	28	0.28
20	32	0.32
33	51	0.51
40	64	0.64
64	100	1.00
80	128	1.28
99	153	1.53
A5	165	1.65
D2	210	2.10
E7	231	2.31
FF	255	2.55

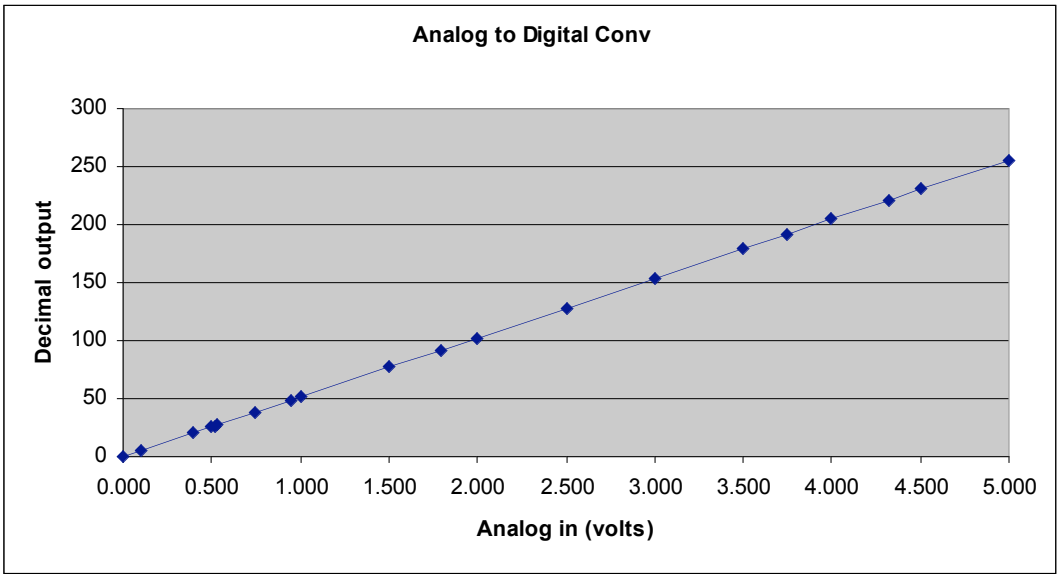


22.2 Analog-to-digital converter (see Fig. 22-2)

WARNING: Do not exceed 5.0 V on analog input!

Analog to Digital Conversion

Analog in	Decimal out	Hex out
0.000	0	0
0.100	5	5
0.400	20	14
0.500	26	19
0.515	26	1A
0.530	27	1B
0.750	38	26
0.950	49	30
1.000	51	33
1.500	77	4C
1.800	92	5C
2.000	102	66
2.500	128	80
3.000	154	99
3.500	179	B3
3.750	192	C0
4.000	205	CC
4.320	221	DD
4.500	230	E6
5.000	255	FF



22.3 Free-running ADC with span adjust (see Fig. 22-3)

resolution = 15 mV

full-scale voltage = 15 mV × 255 = 3.825 V

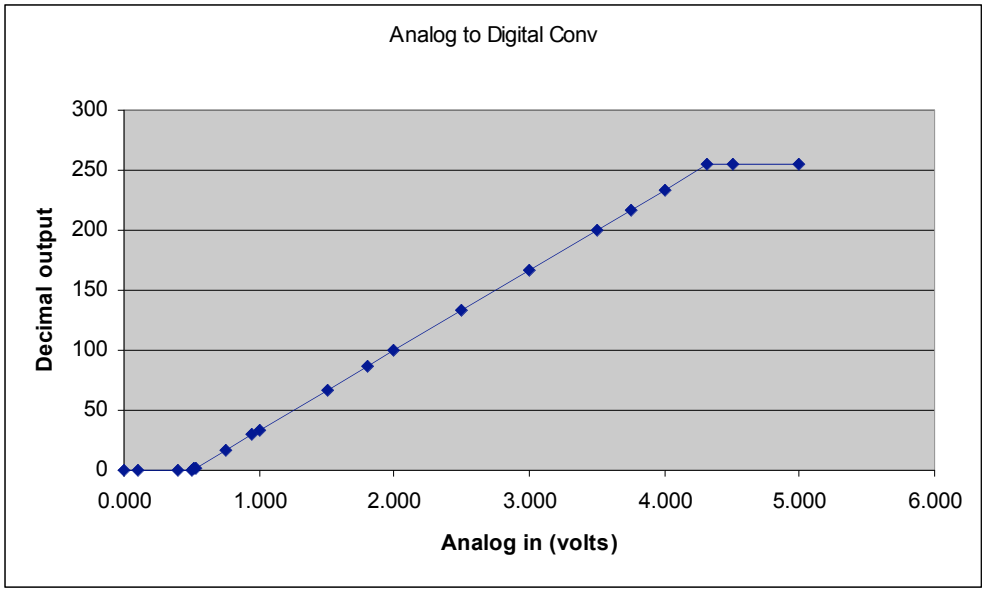
V_{REF} = 256 × 15 mV = 3.84 V

adjust pot to $V_{REF}/2$ = 3.84 V ÷ 2 = 1.92 V

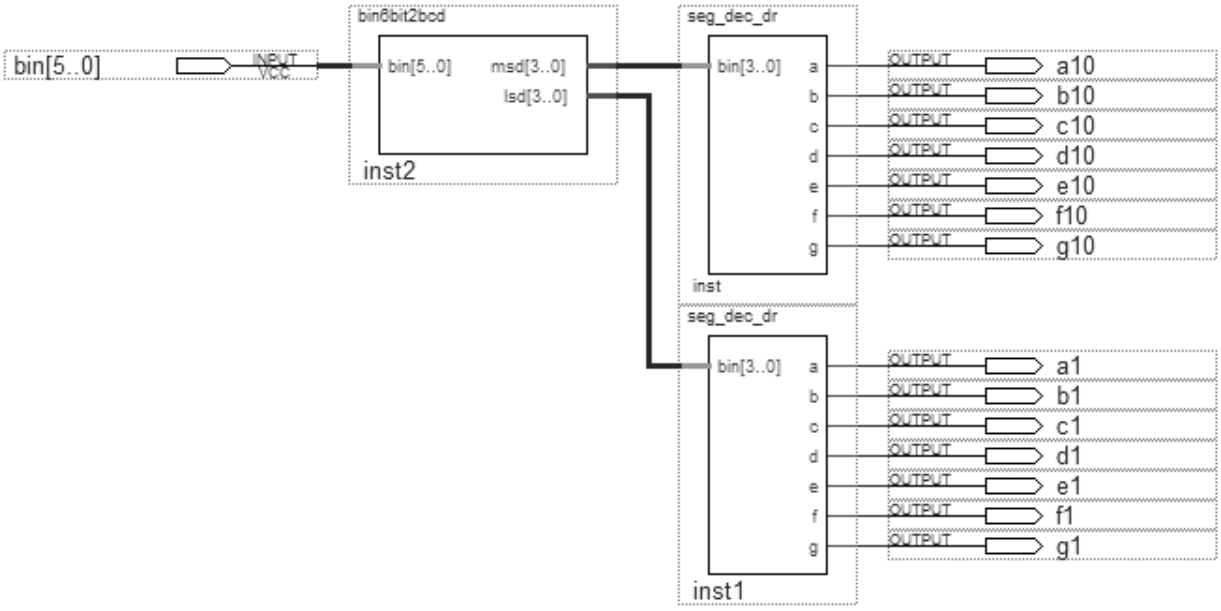
$[R_2/(R_1+R_2)][5\ V] = 0.5\ V \rightarrow$ use R_2=1.1 kΩ & R_1 = 10 kΩ

Analog to Digital Conversion

Analog in	Decimal out	Hex out
0.000	0	0
0.100	0	0
0.400	0	0
0.500	0	0
0.515	1	1
0.530	2	2
0.750	17	10
0.950	30	1D
1.000	33	21
1.500	67	42
1.800	87	56
2.000	100	64
2.500	133	85
3.000	167	A6
3.500	200	C8
3.750	217	D8
4.000	233	E9
4.320	255	FE
4.500	255	FF
5.000	255	FF



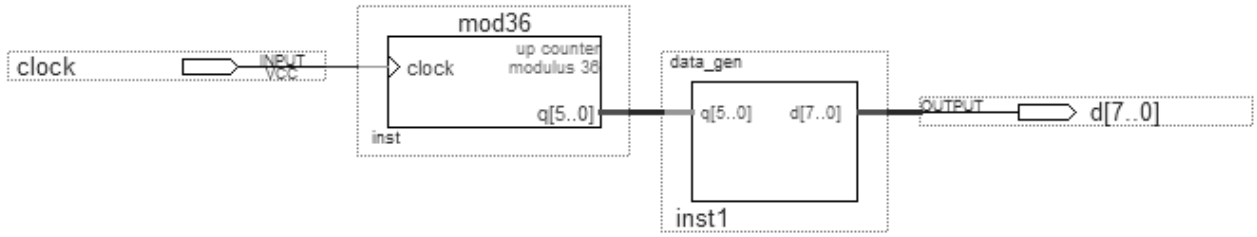
22.4 Digital voltmeter




```
SUBDESIGN  bin6bit2bcd
    (bin[5..0]                          :INPUT;
     msd[3..0], lsd[3..0]               :OUTPUT;)
VARIABLE
     ones[5..0]     :NODE;    -- to capture the ones digit
BEGIN
     lsd[3..0] = ones[3..0];       -- output lsd BCD digit
     IF     bin[] >= 60  THEN
          msd[] = 6;               -- set MSD value
          ones[] = bin[] - 60;     -- calculate LSD value
     ELSIF bin[] >= 50   THEN
          msd[] = 5;          ones[] = bin[] - 50;
     ELSIF bin[] >= 40   THEN
          msd[] = 4;          ones[] = bin[] - 40;
     ELSIF bin[] >= 30   THEN
          msd[] = 3;          ones[] = bin[] - 30;
     ELSIF bin[] >= 20   THEN
          msd[] = 2;          ones[] = bin[] - 20;
     ELSIF bin[] >= 10   THEN
          msd[] = 1;          ones[] = bin[] - 10;
     ELSE
          msd[] = 0;          ones[] = bin[];
     END IF;
END;
```

```
ENTITY bin6bit2bcd IS
PORT (    bin            :IN INTEGER RANGE 0 TO 63;
          msd, lsd  :OUT INTEGER RANGE 0 TO 9);
END bin6bit2bcd;
ARCHITECTURE vhdl OF bin6bit2bcd IS
BEGIN
PROCESS (bin)       -- bin change invokes process
BEGIN
     IF    bin >= 60     THEN
           msd <= 6;           -- set MSD value
           lsd <= bin - 60;    -- calculate LSD value
     ELSIF bin >= 50     THEN
           msd <= 5;
           lsd <= bin - 50;
     ELSIF bin >= 40     THEN
           msd <= 4;
           lsd <= bin - 40;
     ELSIF bin >= 30     THEN
           msd <= 3;
           lsd <= bin - 30;
     ELSIF bin >= 20     THEN
           msd <= 2;
           lsd <= bin - 20;
     ELSIF bin >= 10     THEN
           msd <= 1;
           lsd <= bin - 10;
     ELSE
           msd <= 0;
           lsd <= bin;
     END IF;
END PROCESS;
END vhdl;
```

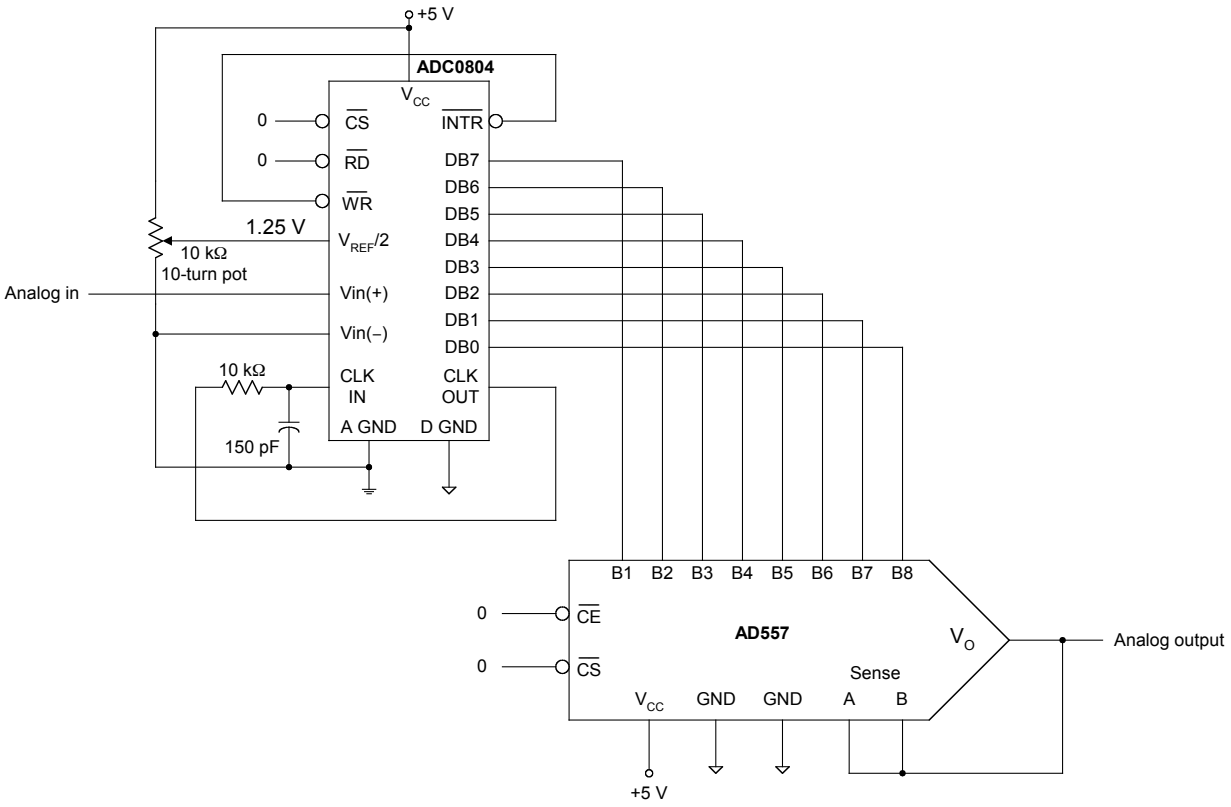
22.5 Digitized sine-wave generator




```
      -- AHDL solution
SUBDESIGN  data_gen
(
      q[5..0]     :INPUT;
      d[7..0]     :OUTPUT;
)

BEGIN
-- D/A input values at
-- 10 degree increments

      TABLE
      q[]   =>    d[];
      0     =>    150;
      1     =>    167;
      2     =>    184;
      3     =>    200;
      4     =>    214;
      5     =>    227;
      6     =>    237;
      7     =>    244;
      8     =>    248;
      9     =>    250;
      10    =>    248;
      11    =>    244;
      12    =>    237;
      13    =>    227;
      14    =>    214;
      15    =>    200;
      16    =>    184;
      17    =>    167;
      18    =>    150;
      19    =>    133;
      20    =>    116;
      21    =>    100;
      22    =>    86;
      23    =>    73;
      24    =>    63;
      25    =>    56;
      26    =>    52;
      27    =>    50;
      28    =>    52;
      29    =>    56;
      30    =>    63;
      31    =>    73;
      32    =>    86;
      33    =>    100;
      34    =>    116;
      35    =>    133;
      END TABLE;
END;
```

```
ENTITY  data_gen IS         -- VHDL solution
PORT (     q          :IN INTEGER RANGE 0 TO 35;
           d          :OUT INTEGER RANGE 0 TO 255    );
END data_gen;

ARCHITECTURE vhdl OF data_gen IS
BEGIN
      -- D/A input values at 10 degree increments

      WITH q     SELECT
      d   <=     150 WHEN 0,
                 167 WHEN 1,
                 184 WHEN 2,
                 200 WHEN 3,
                 214 WHEN 4,
                 227 WHEN 5,
                 237 WHEN 6,
                 244 WHEN 7,
                 248 WHEN 8,
                 250 WHEN 9,
                 248 WHEN 10,
                 244 WHEN 11,
                 237 WHEN 12,
                 227 WHEN 13,
                 214 WHEN 14,
                 200 WHEN 15,
                 184 WHEN 16,
                 167 WHEN 17,
                 150 WHEN 18,
                 133 WHEN 19,
                 116 WHEN 20,
                 100 WHEN 21,
                 86  WHEN 22,
                 73  WHEN 23,
                 63  WHEN 24,
                 56  WHEN 25,
                 52  WHEN 26,
                 50  WHEN 27,
                 52  WHEN 28,
                 56  WHEN 29,
                 63  WHEN 30,
                 73  WHEN 31,
                 86  WHEN 32,
                 100 WHEN 33,
                 116 WHEN 34,
                 133 WHEN 35;
END vhdl;
```

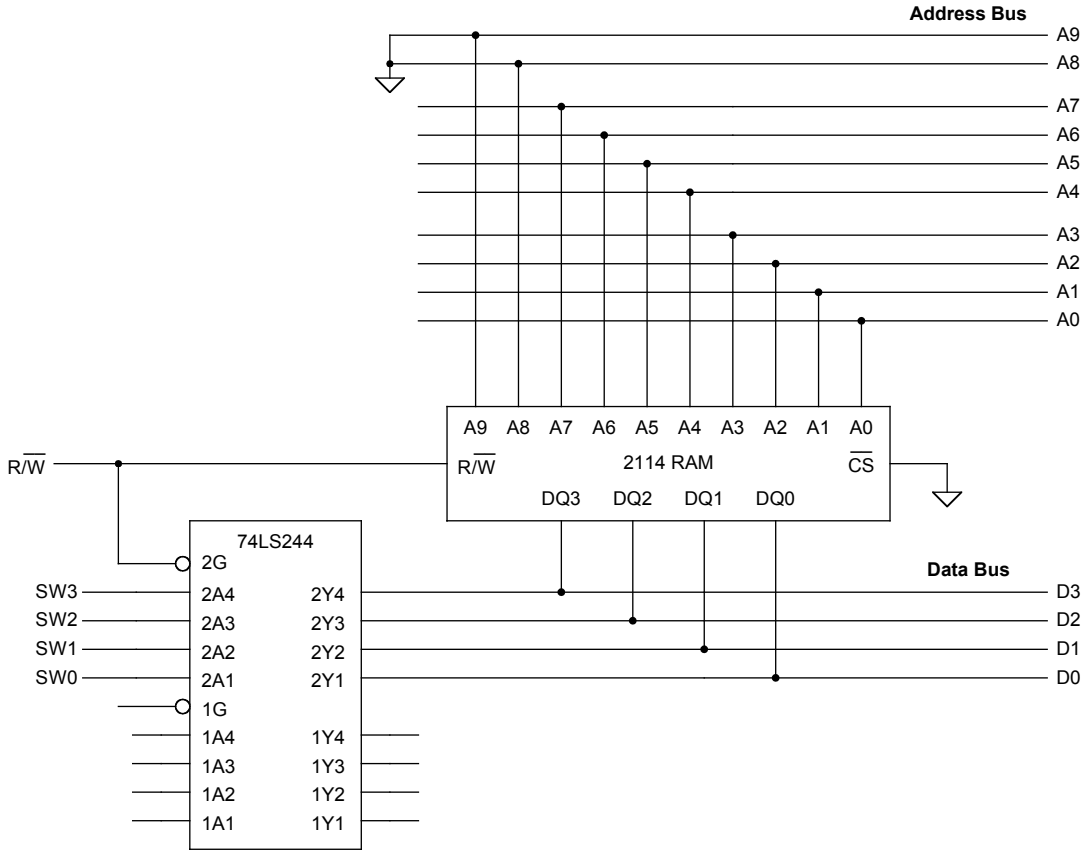
22.6 Reconstructing a digitized signal



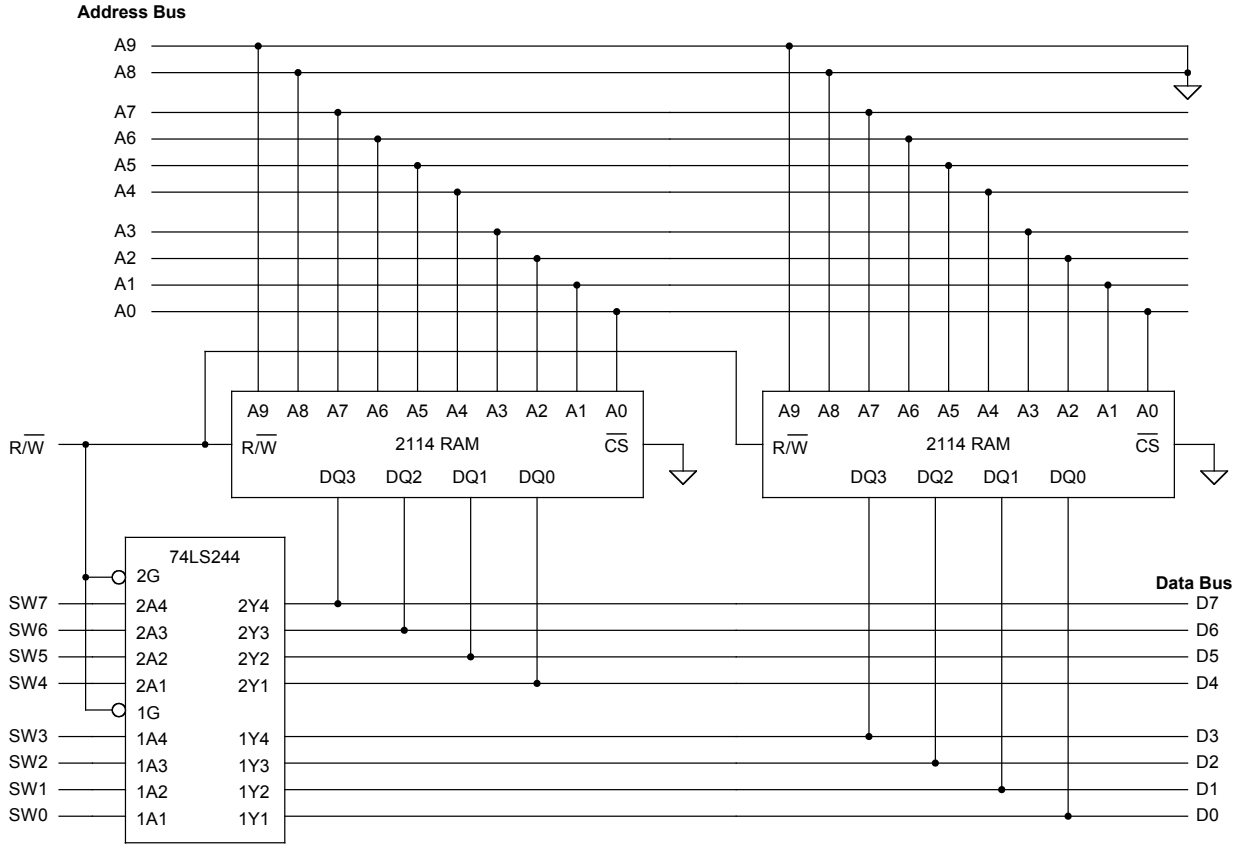
$V_{REF} = 2.5\ V \rightarrow V_{REF}/2 = 1.25\ V$
Analog-out should approximate the analog-in voltage
Analog-out will be ragged with stairstep output
At higher frequencies, analog in will overrun conversion time of A/D converter (aliasing)

Unit 23 Memory Systems

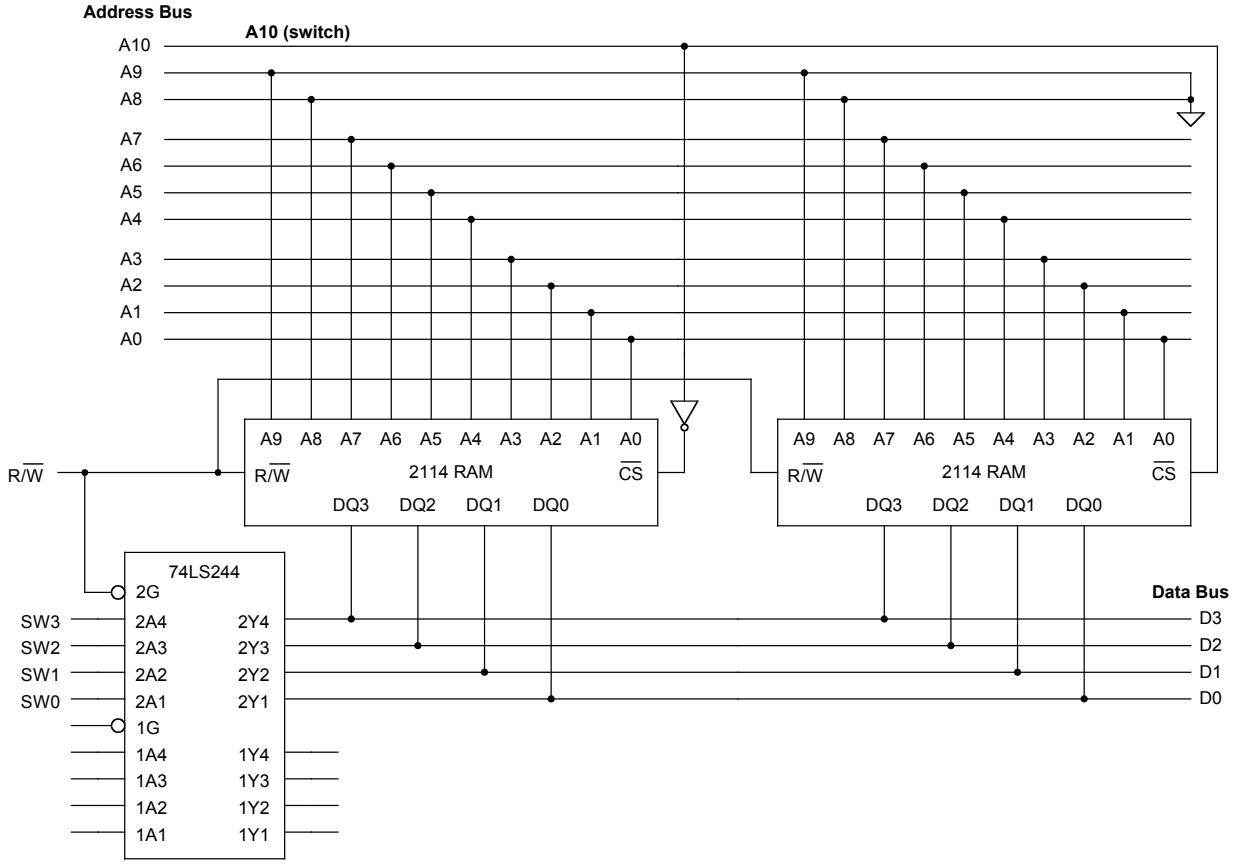
23.1 RAM memory chip



23.2 Memory word-size expansion



23.3 Memory capacity expansion

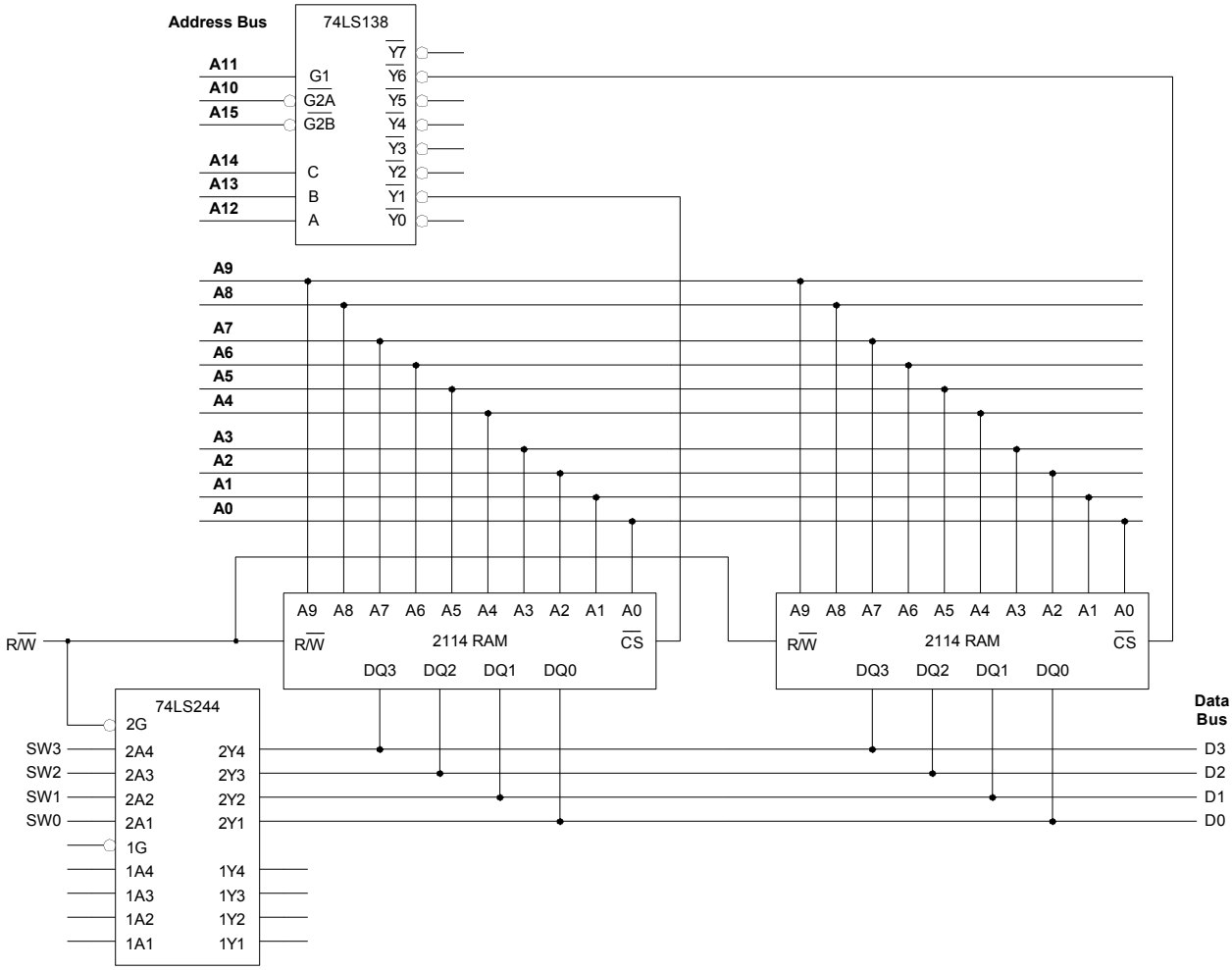


23.4 Memory address decoding

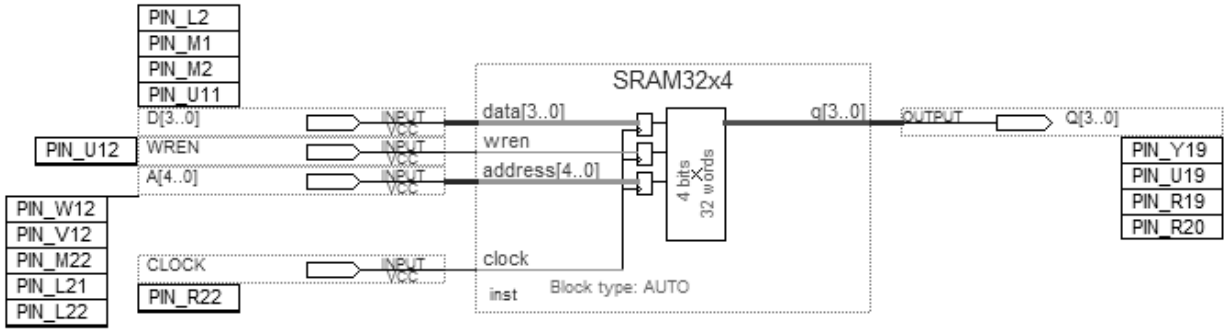
```
SUBDESIGN mem2k_decode
(     a[15..0]          :INPUT;
      cs[1..0]          :OUTPUT;    )
BEGIN
IF     a[] >= H"1800" AND a[] <= H"1BFF"     THEN
      cs[] = B"10";
ELSIF a[] >= H"6800" AND a[] <= H"6BFF"     THEN
      cs[] = B"01";
ELSE
      cs[] = B"11";
END IF;
END;
```

```
ENTITY mem2k_decode IS
PORT (     a           :IN INTEGER RANGE 0 TO 65535;
           cs          :OUT BIT_VECTOR (1 DOWNTO 0)     );
END mem2k_decode;

ARCHITECTURE memory_decoder OF mem2k_decode IS
BEGIN
PROCESS (a)
      BEGIN
      IF     a >= 16#1800# AND a <= 16#1BFF# THEN
            cs <= "10";
      ELSIF a >= 16#6800# AND a <= 16#6BFF# THEN
            cs <= "01";
      ELSE
            cs <= "11";
      END IF;
END PROCESS;
END memory_decoder;
```

Maxplus2 or MSI chip solution:



23.5 Embedded SRAM memory (DE1 pin assignments shown)



23.6 DE1/DE2 SRAM memory chip (DE1 pin assignments shown)