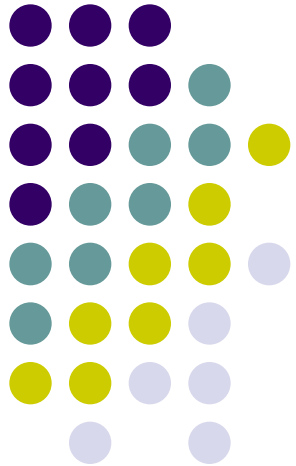
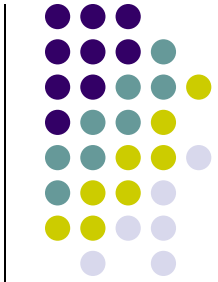
AHDL (Chapter 9)

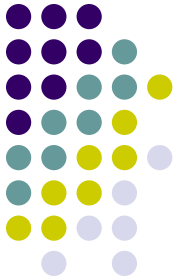


AHDL Decoder using CASE



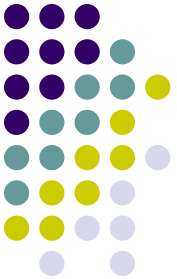
```
1   SUBDESIGN fig9_52
2   (
3       a[2..0]                    :INPUT;      -- binary inputs
4       e1, e2bar, e3bar           :INPUT;      -- enable inputs
5       y7,y6,y5,y4,y3,y2,y1,y0    :OUTPUT;     -- decoded outputs
6   )
7   VARIABLE
8       enable                     :NODE;
9   BEGIN
10      DEFAULTS
11         y7=VCC;y6=VCC;y5=VCC;y4=VCC;
12         y3=VCC;y2=VCC;y1=VCC;y0=VCC;     -- defaults all HIGH out
13      END DEFAULTS;
14      enable = e1 & !e2bar & !e3bar;     -- all enables activated
15      IF enable    THEN
16         CASE a[] IS
17            WHEN 0     =>     y0 = GND;
18            WHEN 1     =>     y1 = GND;
19            WHEN 2     =>     y2 = GND;
20            WHEN 3     =>     y3 = GND;
21            WHEN 4     =>     y4 = GND;
22            WHEN 5     =>     y5 = GND;
23            WHEN 6     =>     y6 = GND;
24            WHEN 7     =>     y7 = GND;
25         END CASE;
26      END IF;
27  END;
```

AHDL Decoder using a TABLE



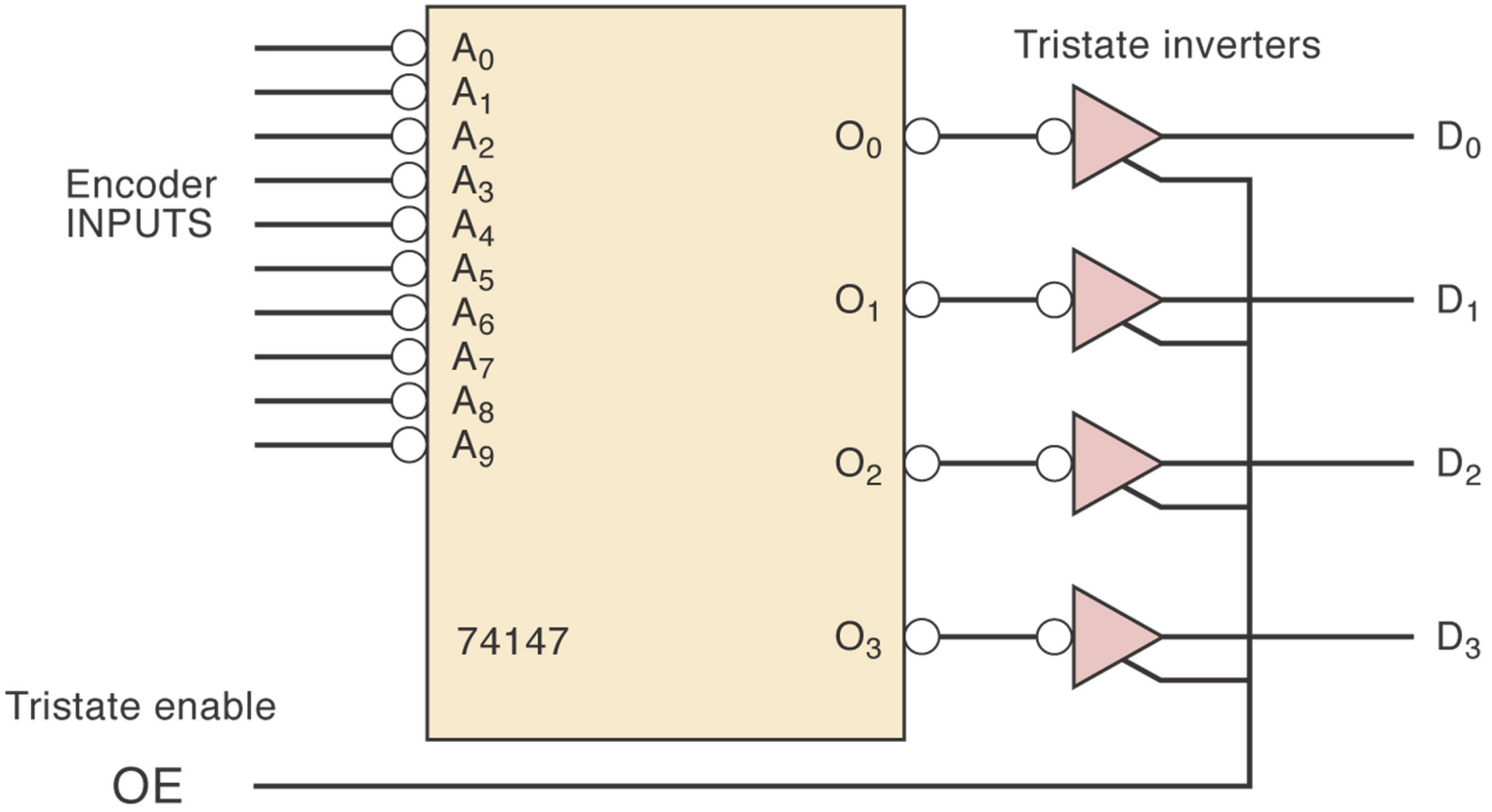
```
1   SUBDESIGN fig9_53
2   (
3       a[2..0]            :INPUT;     -- decoder inputs
4       e1, e2bar, e3bar   :INPUT;     -- enable inputs
5       y[7..0]            :OUTPUT;    -- decoded outputs
6   )
7   VARIABLE
8      inputs[5..0]        :NODE;      -- all 6 inputs combined
9
10  BEGIN
11     inputs[] = (e1, e2bar, e3bar, a[]); -- concatenate the inputs
12     TABLE
13        inputs[]         =>     y[];
14        B"0XXXXX"        =>     B"11111111";    -- e1 not enabled
15        B"X1XXXX"        =>     B"11111111";    -- e2bar disabled
16        B"XX1XXX"        =>     B"11111111";    -- e3bar disabled
17        B"100000"        =>     B"11111110";    -- Y0 active
18        B"100001"        =>     B"11111101";    -- Y1 active
19        B"100010"        =>     B"11111011";    -- Y2 active
20        B"100011"        =>     B"11110111";    -- Y3 active
21        B"100100"        =>     B"11101111";    -- Y4 active
22        B"100101"        =>     B"11011111";    -- Y5 active
23        B"100110"        =>     B"10111111";    -- Y6 active
24        B"100111"        =>     B"01111111";    -- Y7 active
25     END TABLE;
26  END;
```

AHDL 7-Segment Decoder/Driver

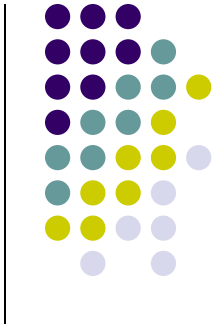


```
1   SUBDESIGN fig9_55
2   (
3      bcd[3..0]          :INPUT;        -- 4-bit number
4      lt, bi, rbi        :INPUT;        -- 3 independent controls
5      a,b,c,d,e,f,g,rbo  :OUTPUT;       -- individual outputs
6   )
7   BEGIN
8       IF !bi THEN
9           (a,b,c,d,e,f,g,rbo) = (1,1,1,1,1,1,1,0);    % blank all %
10      ELSIF     !lt THEN
11          (a,b,c,d,e,f,g,rbo) = (0,0,0,0,0,0,0,1);    % test segments %
12      ELSIF !rbi & bcd[] == 0  THEN
13          (a,b,c,d,e,f,g,rbo) = (1,1,1,1,1,1,1,0);    % blank leading 0's %
14      ELSIF bcd[] > 9 THEN
15          (a,b,c,d,e,f,g,rbo) = (1,1,1,1,1,1,1,1);    % blank non BCD input %
16      ELSE
17          TABLE                    % display 7 segment Common Anode pattern %
18          bcd[]     =>    a,b,c,d,e,f,g,rbo;
19          0         =>    0,0,0,0,0,0,1,1;
20          1         =>    1,0,0,1,1,1,1,1;
21          2         =>    0,0,1,0,0,1,0,1;
22          3         =>    0,0,0,0,1,1,0,1;
23          4         =>    1,0,0,1,1,0,0,1;
24          5         =>    0,1,0,0,1,0,0,1;
25          6         =>    1,1,0,0,0,0,0,1;
26          7         =>    0,0,0,1,1,1,1,1;
27          8         =>    0,0,0,0,0,0,0,1;
28          9         =>    0,0,0,1,1,0,0,1;
29          END TABLE;
30      END IF;
31  END;
```

Encoder with Tristate Outputs

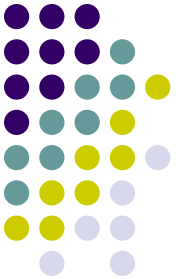


AHDL Priority Encoder (TABLE)



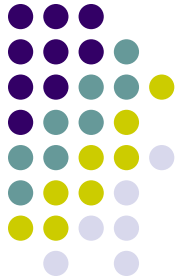
```
1   SUBDESIGN fig9_58
2   (
3      a[9..0], oe          :INPUT;
4      d[3..0]              :OUTPUT;
5   )
6   VARIABLE buffer[3..0]   :TRI;
7   BEGIN
8      TABLE
9         a[]             => buffer[].in;
10        B"1111111111"   => B"1111";    -- no input active
11        B"1111111110"   => B"0000";    -- 0
12        B"111111110X"   => B"0001";    -- 1
13        B"11111110XX"   => B"0010";    -- 2
14        B"1111110XXX"   => B"0011";    -- 3
15        B"111110XXXX"   => B"0100";    -- 4
16        B"11110XXXXX"   => B"0101";    -- 5
17        B"1110XXXXXX"   => B"0110";    -- 6
18        B"110XXXXXXX"   => B"0111";    -- 7
19        B"10XXXXXXXX"   => B"1000";    -- 8
20        B"0XXXXXXXXX"   => B"1001";    -- 9
21     END TABLE;
22     buffer[].oe = oe;     -- hook up enable line
23     d[] = buffer[].out;   -- hook up outputs
24  END;
```

AHDL Priority Encoder (IF/ELSE)



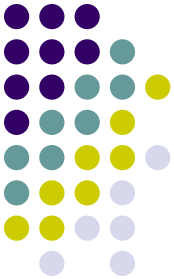
```
1   SUBDESIGN fig9_59
2   (
3      sw[9..0], oe   :INPUT;
4      d[3..0]        :OUTPUT;
5   )
6   VARIABLE
7      buffers[3..0]  :TRI;
8   BEGIN
9      IF    !sw[9]   THEN  buffers[].in = 9;
10     ELSIF !sw[8]   THEN  buffers[].in = 8;
11     ELSIF !sw[7]   THEN  buffers[].in = 7;
12     ELSIF !sw[6]   THEN  buffers[].in = 6;
13     ELSIF !sw[5]   THEN  buffers[].in = 5;
14     ELSIF !sw[4]   THEN  buffers[].in = 4;
15     ELSIF !sw[3]   THEN  buffers[].in = 3;
16     ELSIF !sw[2]   THEN  buffers[].in = 2;
17     ELSIF !sw[1]   THEN  buffers[].in = 1;
18     ELSE                 buffers[].in = 0;
19     END IF;
20     buffers[].oe = oe & sw[]!=b"1111111111";  -- enable on any input
21     d[] = buffers[].out;                      -- connect to outputs
22  END;
```

AHDL MUX



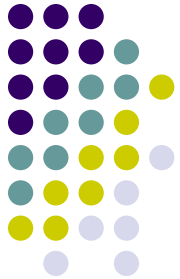
```
1   SUBDESIGN fig9_62
2   (
3     ch0[3..0], ch1[3..0], ch2[3..0], ch3[3..0]:INPUT;
4     s[1..0]                                    :INPUT; -- select inputs
5     dout[3..0]                                 :OUTPUT;
6   )
7   BEGIN
8      CASE s[] IS
9            WHEN 0 =>     dout[] = ch0[];
10           WHEN 1 =>     dout[] = ch1[];
11           WHEN 2 =>     dout[] = ch2[];
12           WHEN 3 =>     dout[] = ch3[];
13       END CASE;
14  END;
```

AHDL DEMUX



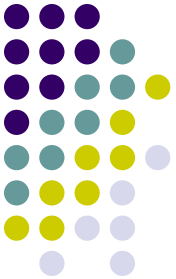
```
1   SUBDESIGN fig9_63
2   (
3      ch0[3..0], ch1[3..0], ch2[3..0], ch3[3..0]   :OUTPUT;
4      s[1..0]                                      :INPUT;
5      din[3..0]                                    :INPUT;
6   )
7   BEGIN
8      DEFAULTS
9         ch0[] = B"1111";
10        ch1[] = B"1111";
11        ch2[] = B"1111";
12        ch3[] = B"1111";
13     END DEFAULTS;
14
15     CASE S[] IS
16           WHEN 0 =>     ch0[] = din[];
17           WHEN 1 =>     ch1[] = din[];
18           WHEN 2 =>     ch2[] = din[];
19           WHEN 3 =>     ch3[] = din[];
20      END CASE;
21  END;
```

AHDL Magnitude Comparator



```
1   SUBDESIGN fig9_66
2   (
3      a[3..0], b[3..0]   :INPUT;
4      gtin, ltin, eqin   :INPUT;       % cascade inputs %
5      agtb, altb, aeqb   :OUTPUT;
6   )
7      % standard cascade inputs:  gtin = ltin = GND eqin = VCC %
8
9   BEGIN
10     IF    a[] > b[] THEN
11           agtb = VCC;    altb = GND;         aeqb = GND;
12     ELSIF a[] < b[] THEN
13           agtb = GND;    altb = VCC;         aeqb = GND;
14     ELSE  agtb = gtin;   altb = ltin;        aeqb = eqin;
15     END IF;
16  END;
```

BCD to Binary Code Converter



```
1   SUBDESIGN fig9_68
2   (  ones[3..0], tens[3..0]     :INPUT;
3      binary[6..0]               :OUTPUT;  )
4
5   VARIABLE times10[6..0]        :NODE;    % variable for tens digit times 10%
6
7   BEGIN
8      times10[] = (tens[],B"000") + (B"00",tens[],B"0");
9            % shift left 3X (times 8) + shift left 1X (times 2) %
10     binary[] = times10[] + (B"000",ones[]);
11           % tens digit times 10 + ones digit %
12  END;
```

numx10=numx8 + numx2

numx8 === shift left 3x

numx2 === shift left 1x