EEE205 – Digital Electronics (II)

Lecture 6

Dr. Ming Xu

Dept of Electrical & Electronic Engineering

XJTLU

In This Session

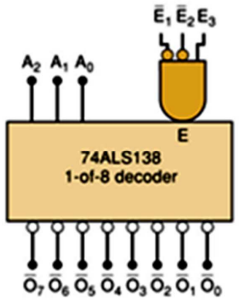
Combinational Logic in AHDL

- AHDL Decoders and Encoders
- AHDL MUX and DEMUX
- AHDL Comparators
- ADHL Adders

AHDL Decoders

74LS138 — a 3-to-8 decoder or a 1-in-8 decoder:

- The outputs are active-LOW.
- The decoder responds to the input code only when $\overline{E}_1 = \overline{E}_2 = 0$ and $E_3 = 1$.



$\overline{E}_1$	$\overline{E}_2$	E_3	Outputs
0	0	1	Respond to input code $A_2A_1A_0$
1	X	X	Disabled – all HIGH
X	1	X	Disabled – all HIGH
X	X	0	Disabled – all HIGH

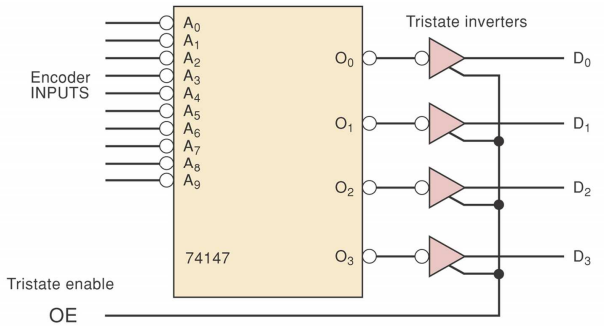

```
1   SUBDESIGN fig9_52
2   (
3       a[2..0]                  :INPUT;     -- binary inputs
4       e3, e2bar, e1bar         :INPUT;     -- enable inputs
5       y7,y6,y5,y4,y3,y2,y1,y0  :OUTPUT;    -- decoded outputs
6   )
7   VARIABLE
8       enable                   :NODE;
9   BEGIN
10      DEFAULTS
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 = e3 & !e2bar & !e1bar;    -- 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 Decoders

- The **DEFAULTS** keyword is to establish a value for variable that are not specified elsewhere in the code.
- This allows each case to force one bit LOW without stating that the others must go HIGH.

AHDL Encoders

- When one of the inputs is activated, it produces a binary number corresponding to that input.
- When more than one of its inputs is activated, it ignores the input of lower significance.




```
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 Encoders

- AHDL has a library **primitive** called **TRI** for tristate buffers.

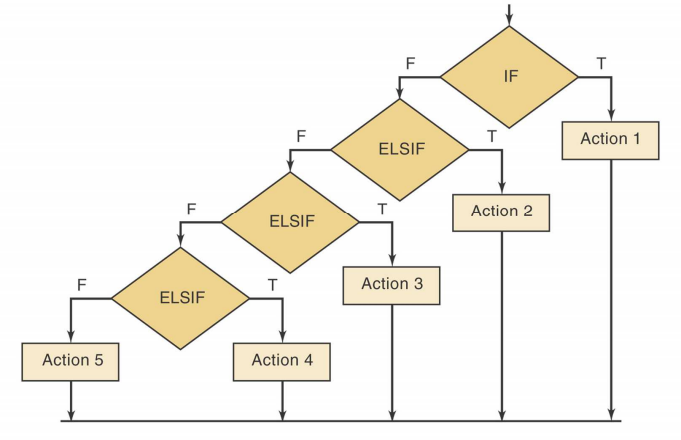
```
VARIABLE  buffer[3..0]    :TRI;
```

- A primitive can be imagined as a component in a store and is similar to a structure in C.
- This makes the implementation easier — just to connect the primitive's ports to the appropriate signals.

```
buffer[].in = B"0000";
buffer[].oe = oe;
d[] = buffer[].out;
```

AHDL Encoders

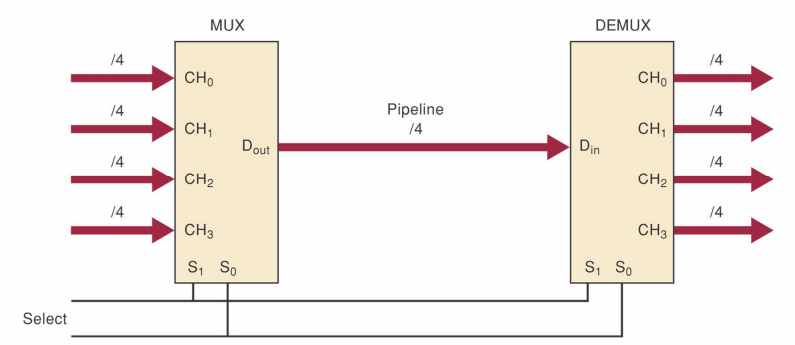
- The AHDL encoder can also be implemented using **IF/ELSIF**, which choose one of many possible outcomes.




```
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 and DEMUX

- A multiplexer selects and connects one of the inputs to the output.
- A demultiplexer distributes an input to one of its outputs.



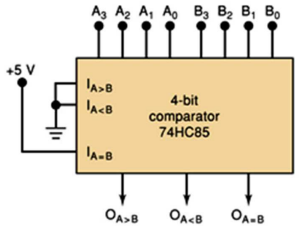
AHDL MUX and DEMUX

```
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 MUX and 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 Comparators

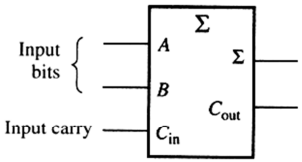


If we need to compare bigger values, we can simply adjust the size of the input ports, rather than cascade MSI chips.

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


AHDL Adders

A	B	C_{in}	C_{out}	Σ
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



$$\Sigma = (A \oplus B) \oplus C_{in}$$

$$C_{out} = AB + (A \oplus B)C_{in}$$

$$C_{out} = ABC_{in} + AB\overline{C}_{in} + A\overline{B}C_{in} + \overline{A}BC_{in}$$
$$= \left(ABC_{in} + AB\overline{C}_{in}\right) + \left(ABC_{in} + A\overline{B}C_{in}\right) + \left(ABC_{in} + \overline{A}BC_{in}\right)$$
$$= AB + AC_{in} + BC_{in}$$

AHDL Adders

```
SUBDESIGN fig6_21
(
      cin         :INPUT;     -- carry in
      a[3..0]     :INPUT;     -- augend
      b[3..0]     :INPUT;     -- addend
      s[3..0]     :OUTPUT;    -- sum
      cout        :OUTPUT;    -- carry OUT
)
VARIABLE
      c[4..0]     :NODE;      -- carry array is 5 bits long!

BEGIN
      c[0] = cin;
      s[] = a[] $ b[] $ c[3..0];   -- generate sum
      c[4..1] = (a[] & b[]) # (a[] & c[3..0]) # (b[] & c[3..0]);
      cout = c[4];                 -- carry out
END;
```