ECE 331 – Digital System Design

Multiplexers, Decoders and Encoders

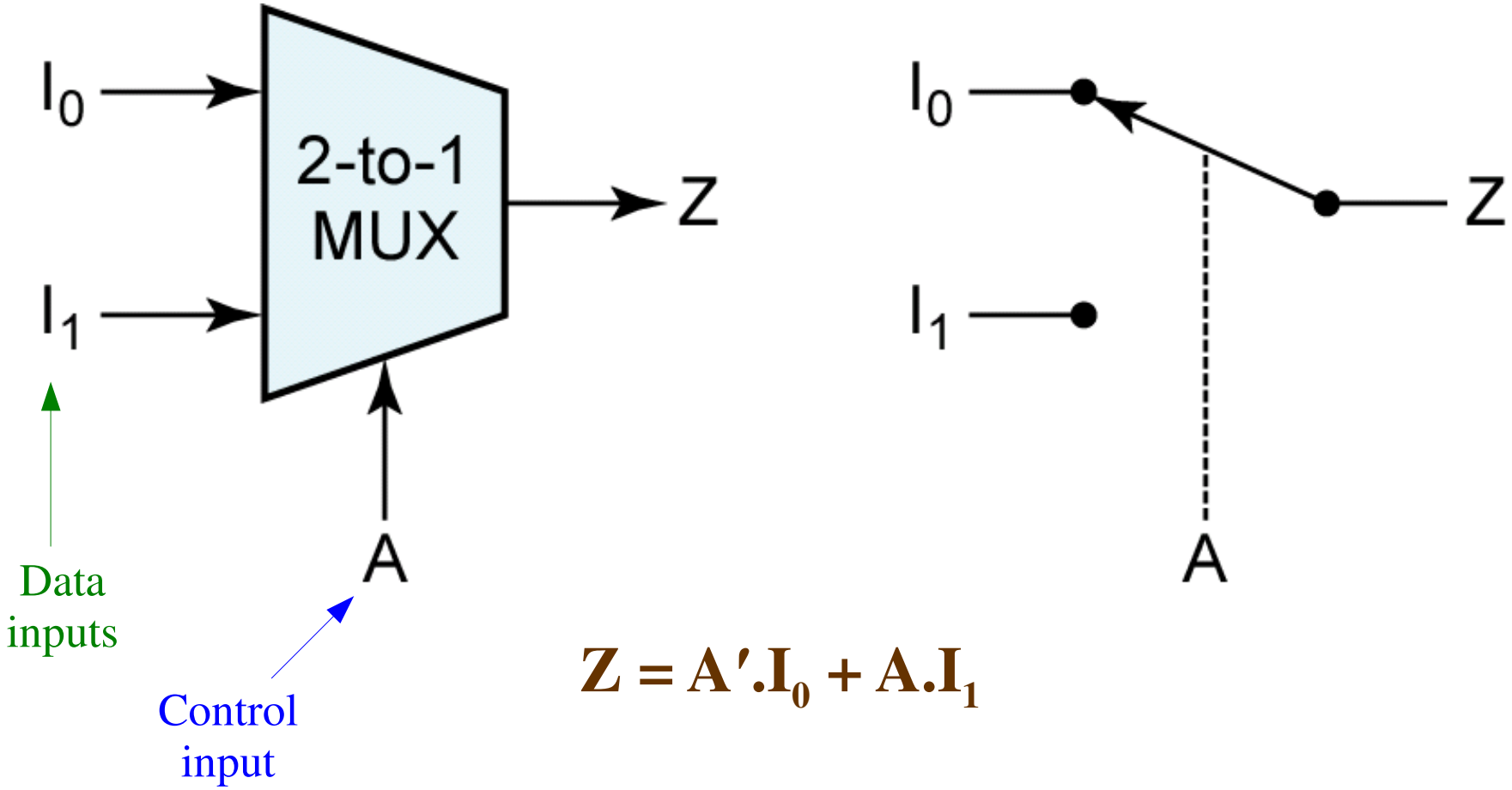
(Lecture #16)

Multiplexers

Multiplexers

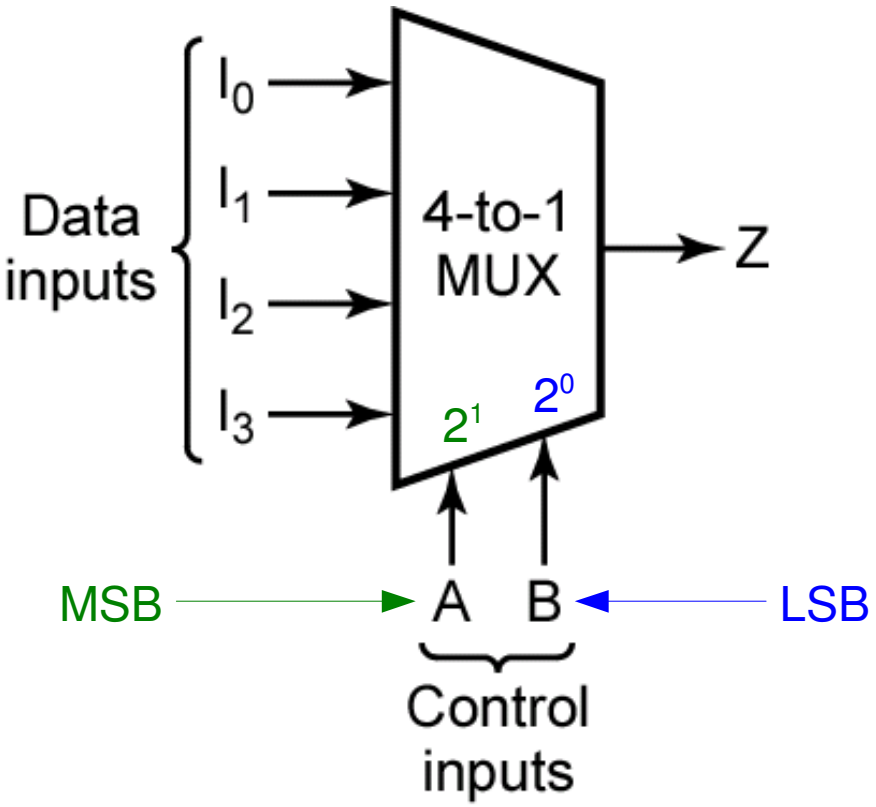
- A multiplexer has
  - 2^n data inputs
  - *n* control inputs
  - 1 output
- A multiplexer routes (or connects) the selected data input to the output.
  - The value of the control inputs determines the data input that is selected.

Multiplexers



$$Z = A'.I_0 + A.I_1$$

Multiplexers



A	B	Z
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

$m_0 = A'.B'$

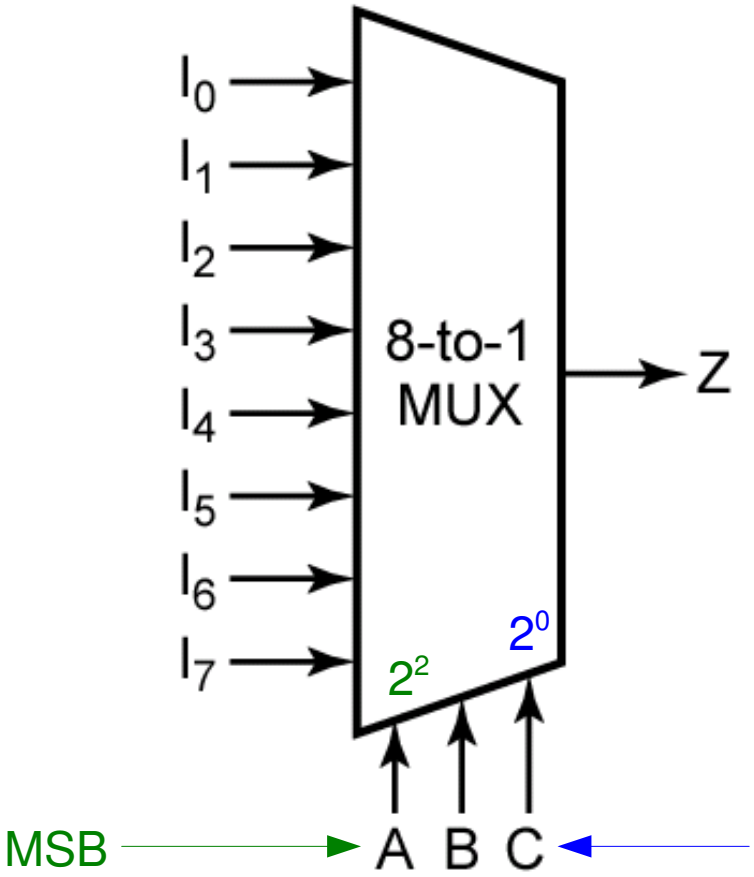
$m_1 = A'.B$

$m_2 = A.B'$

$m_3 = A.B$

$$Z = A'.B'.I_0 + A'.B.I_1 + A.B'.I_2 + A.B.I_3$$

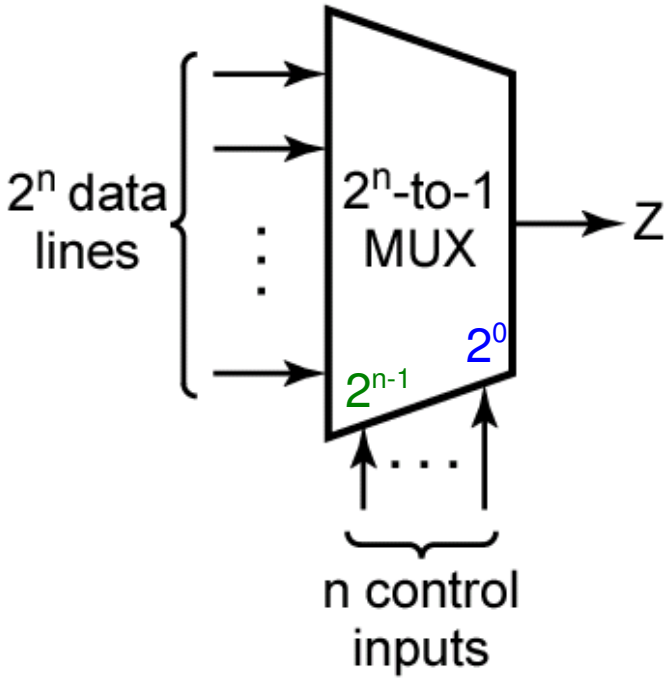
Multiplexers



A	B	C	Z	
0	0	0	I_0	m_0
0	0	1	I_1	m_1
0	1	0	I_2	m_2
0	1	1	I_3	m_3
1	0	0	I_4	m_4
1	0	1	I_5	m_5
1	1	0	I_6	m_6
1	1	1	I_7	m_7

$$Z = A'.B'.C'.I_0 + A'.B'.C.I_1 + A'.B.C'.I_2 + A'.B.C.I_3 + A.B'.C'.I_0 + A.B'.C.I_1 + A'.B.C'.I_2 + A.B.C.I_3$$

Multiplexers



$$Z = \Sigma\, m_i.I_i$$

Multiplexers in VHDL

8-to-1 Multiplexer

```
entity Multiplexer_8to1 is
    Port ( I : in  STD_LOGIC_VECTOR (7 downto 0);
           sel : in  STD_LOGIC_VECTOR (2 downto 0);
           Z : out  STD_LOGIC);
end Multiplexer_8to1;

architecture Bool_Exp of Multiplexer_8to1 is

begin

   Z <= ( not(sel(2)) and not(sel(1)) and not(sel(0)) and I(0) ) or
        ( not(sel(2)) and not(sel(1)) and sel(0) and I(1) ) or
        ( not(sel(2)) and sel(1) and not(sel(0)) and I(2) ) or
        ( not(sel(2)) and sel(1) and sel(0) and I(3) ) or
        ( sel(2) and not(sel(1)) and not(sel(0)) and I(4) ) or
        ( sel(2) and not(sel(1)) and sel(0) and I(5) ) or
        ( sel(2) and sel(1) and not(sel(0)) and I(6) ) or
        ( sel(2) and sel(1) and sel(0) and I(7) );

end Bool_Exp;
```

8-to-1 Multiplexer

```
entity Multiplexer_8to1_2 is
    Port ( I : in  STD_LOGIC_VECTOR (7 downto 0);
           sel : in  STD_LOGIC_VECTOR (2 downto 0);
           Z : out  STD_LOGIC);
end Multiplexer_8to1_2;

architecture Truth_Table of Multiplexer_8to1_2 is

begin

   with sel select

      Z <=  I(7)  when "111",
            I(6)  when "110",
            I(5)  when "101",
            I(4)  when "100",
            I(3)  when "011",
            I(2)  when "010",
            I(1)  when "001",
            I(0)  when "000",
            I(0)  when others;

end Truth_Table;
```

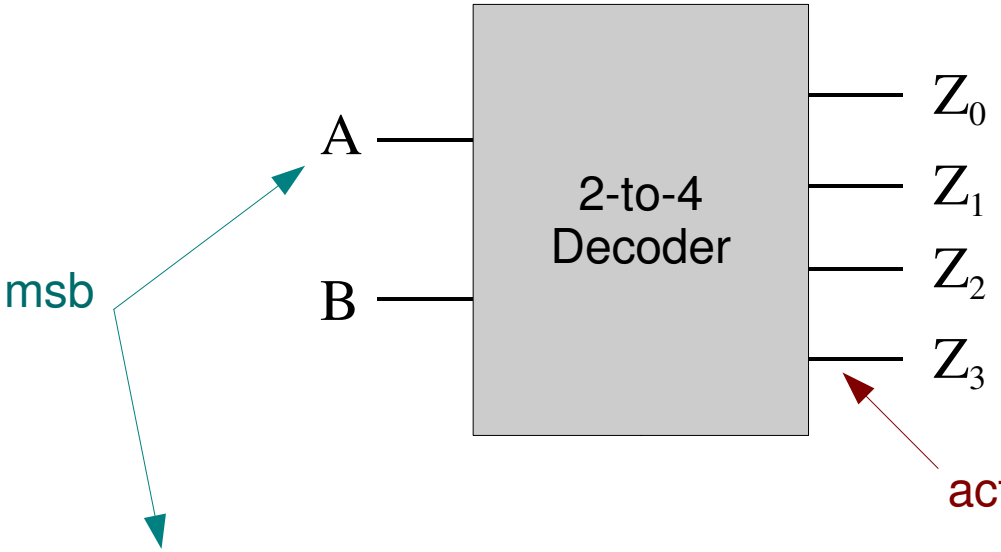
Decoders

Decoders

- A decoder has
  - *n* inputs
  - 2^n outputs
- A decoder selects one of 2^n outputs by decoding the binary value on the *n* inputs.
- The decoder generates all of the minterms of the *n* input variables.
  - Exactly one output will be active for each combination of the inputs.

What does “active” mean?

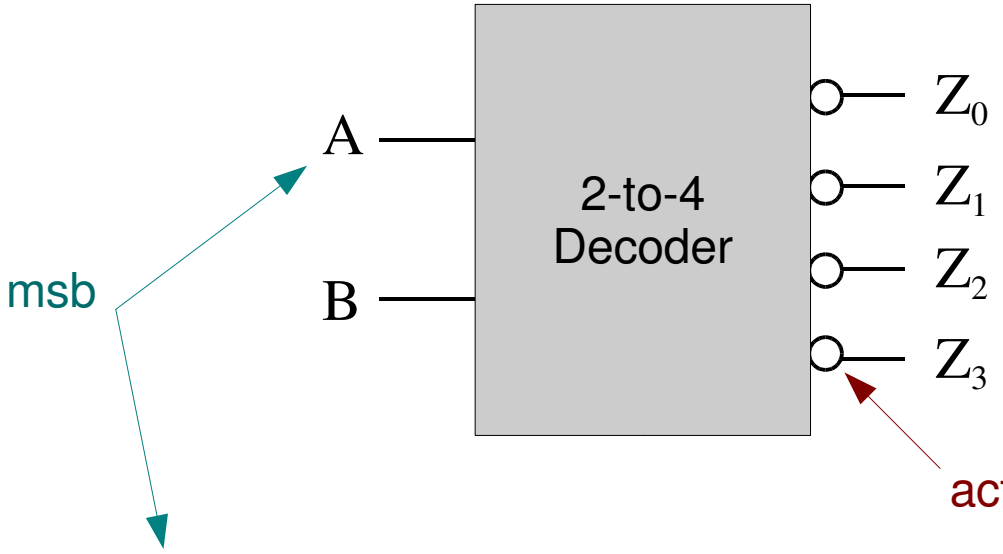
Decoders



$$Z_i = m_i$$

A	B	Z_0	Z_1	Z_2	Z_3	
0	0	1	0	0	0	m_0
0	1	0	1	0	0	m_1
1	0	0	0	1	0	m_2
1	1	0	0	0	1	m_3

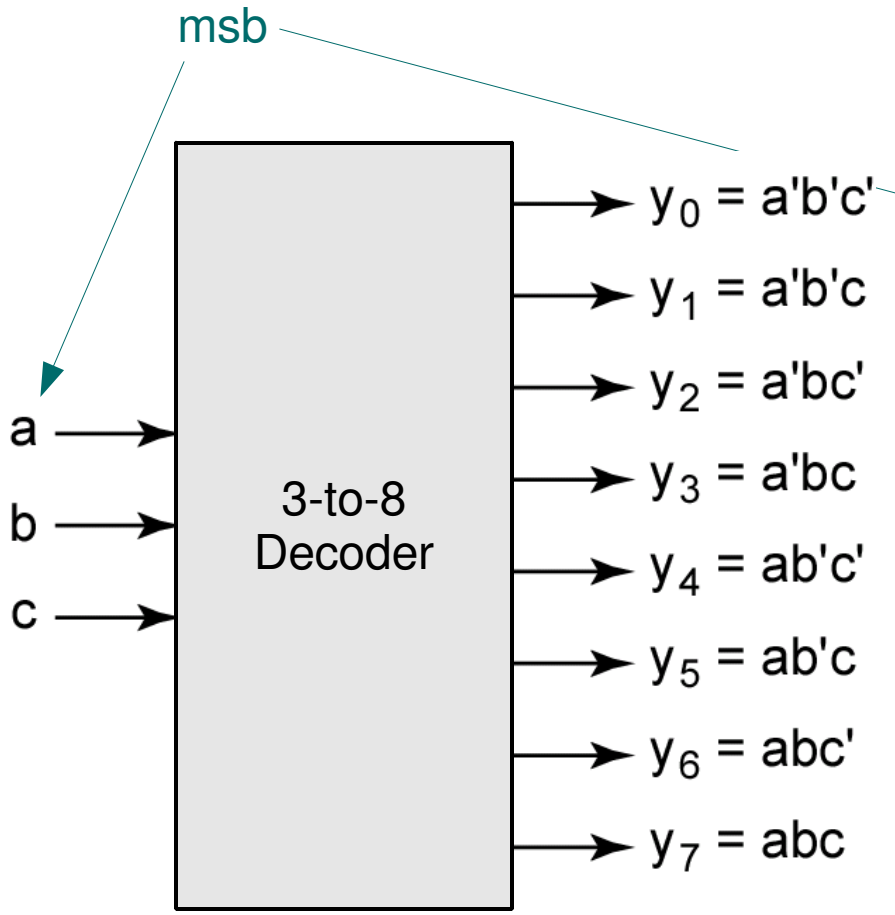
Decoders



$$Z_i = (m_i)' = M_i$$

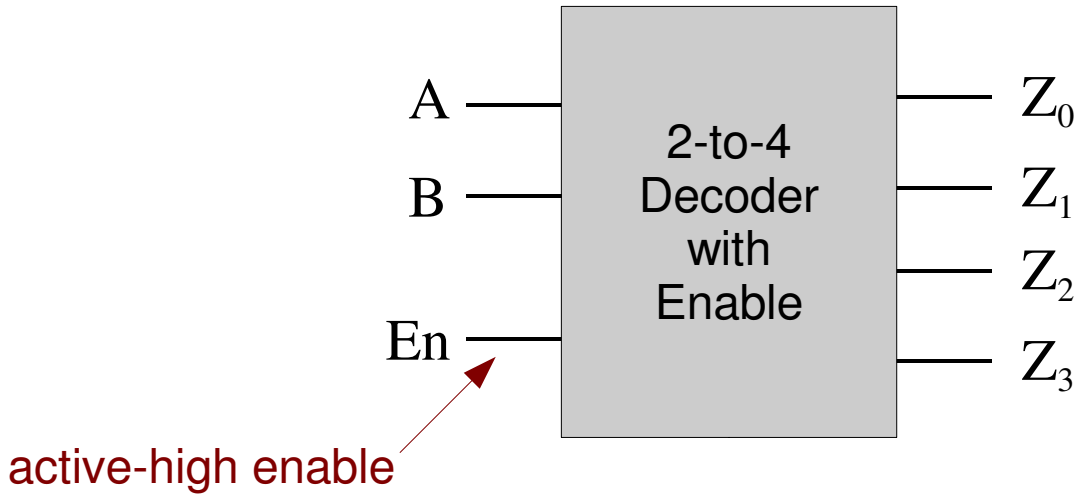
A	B	Z_0	Z_1	Z_2	Z_3	
0	0	0	1	1	1	M_0
0	1	1	0	1	1	M_1
1	0	1	1	0	1	M_2
1	1	1	1	1	0	M_3

Decoders



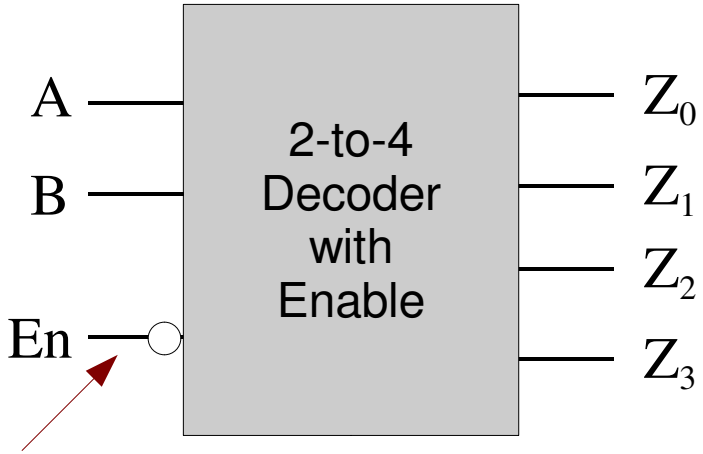
a	b	c	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Decoder with Enable



	En	A	B	Z_0	Z_1	Z_2	Z_3
enabled	1	0	0	1	0	0	0
	1	0	1	0	1	0	0
	1	1	0	0	0	1	0
	1	1	1	0	0	0	1
disabled	0	x	x	0	0	0	0

Decoder with Enable



active-low enable

	En	A	B	Z_0	Z_1	Z_2	Z_3
enabled	0	0	0	1	0	0	0
	0	0	1	0	1	0	0
	0	1	0	0	0	1	0
	0	1	1	0	0	0	1
disabled	1	x	x	0	0	0	0

Decoders in VHDL

3-to-8 Decoder

```
entity Decoder_3to8_1 is
    Port ( I : in  STD_LOGIC_VECTOR (2 downto 0);
           Z : out  STD_LOGIC_VECTOR (7 downto 0) );
end Decoder_3to8_1;

architecture Boolean_Expression of Decoder_3to8_1 is

begin

   Z(0) <= ( not(I(2)) and not(I(1)) and not(I(0)) );
   Z(1) <= ( not(I(2)) and not(I(1)) and I(0) );
   Z(2) <= ( not(I(2)) and I(1) and not(I(0)) );
   Z(3) <= ( not(I(2)) and I(1) and I(0) );
   Z(4) <= ( I(2) and not(I(1)) and not(I(0)) );
   Z(5) <= ( I(2) and not(I(1)) and I(0) );
   Z(6) <= ( I(2) and I(1) and not(I(0)) );
   Z(7) <= ( I(2) and I(1) and I(0) );

end Boolean_Expression;
```

3-to-8 Decoder

```
entity Decoder_3to8_2 is
    Port ( I : in  STD_LOGIC_VECTOR (2 downto 0);
           Z : out  STD_LOGIC_VECTOR (7 downto 0));
end Decoder_3to8_2;

architecture Truth_Table of Decoder_3to8_2 is

begin

   with I select

      Z <=  "10000000"  when "111",
            "01000000"  when "110",
            "00100000"  when "101",
            "00010000"  when "100",
            "00001000"  when "011",
            "00000100"  when "010",
            "00000010"  when "001",
            "00000001"  when "000",
            "00000000"  when others;

end Truth_Table;
```

2-to-4 Decoder with Enable

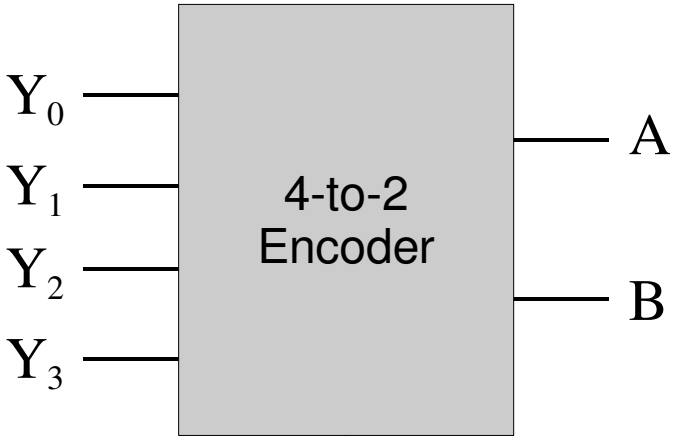
```
entity Decoder_2to4_withEN is
    Port ( I : in  STD_LOGIC_VECTOR (1 downto 0);
           Z : out  STD_LOGIC_VECTOR (3 downto 0);
           En : in  STD_LOGIC);
end Decoder_2to4_withEN;

architecture Truth_Table of Decoder_2to4_withEN is
signal EnabledInput : STD_LOGIC_VECTOR (2 downto 0);

begin

   EnabledInput <= En & I;     -- concatenate the En signal
                               -- with the data inputs (I)

   with EnabledInput select

      Z <=  "1000"   when "111",
            "0100"   when "110",
            "0010"   when "101",
            "0001"   when "100",
            "0000"   when "000",
            "0000"   when others;

end Truth_Table;
```

Encoders

Encoders

- An encoder has
  - 2^n inputs
  - *n* outputs
- Outputs the binary value of the selected (or active) input.
- Performs the inverse operation of a decoder.
- Issues
  - What if more than one input is active?
  - What if no inputs are active?

Encoders



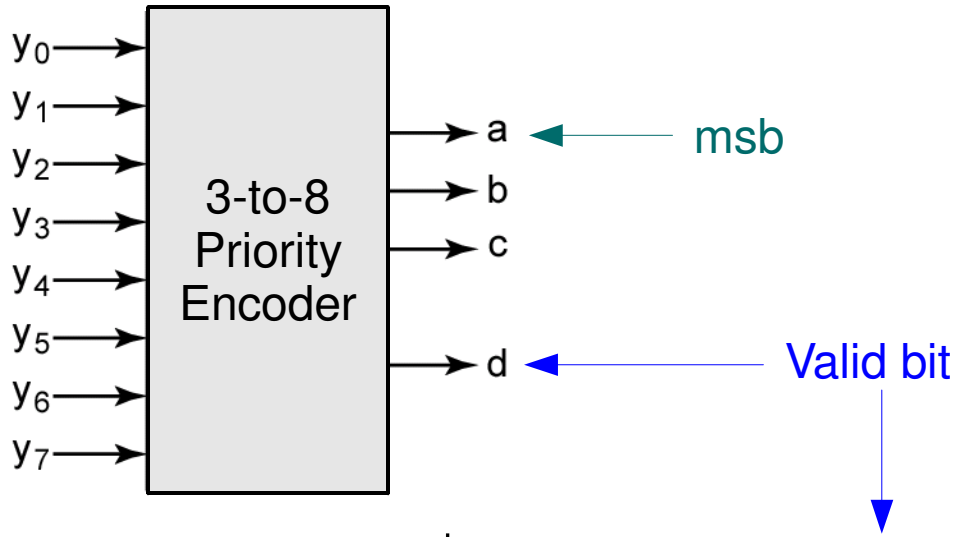
Y_0	Y_1	Y_2	Y_3	A	B
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

Priority Encoders

- If more than one input is active, the higher-order input has priority over the lower-order input.
  - The higher value is encoded on the output
- A valid indicator, d, is included to indicate whether or not the output is valid.
  - Output is invalid when no inputs are active
    - d = 0
  - Output is valid when at least one input is active
    - d = 1

Why is the valid indicator needed?

Priority Encoders



y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	a	b	c	d
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
X	1	0	0	0	0	0	0	0	0	1	1
X	X	1	0	0	0	0	0	0	1	0	1
X	X	X	1	0	0	0	0	0	1	1	1
X	X	X	X	1	0	0	0	1	0	0	1
X	X	X	X	X	1	0	0	1	0	1	1
X	X	X	X	X	X	1	0	1	1	0	1
X	X	X	X	X	X	X	1	1	1	1	1

Encoders in VHDL

4-to-2 Priority Encoder

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY priority IS
    PORT ( w   : IN    STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           y   : OUT   STD_LOGIC_VECTOR(1 DOWNTO 0) ;
           z   : OUT   STD_LOGIC ) ;
END priority ;

ARCHITECTURE Behavior OF priority IS
BEGIN
    y <=  "11" WHEN w(3) = '1' ELSE
          "10" WHEN w(2) = '1' ELSE
          "01" WHEN w(1) = '1' ELSE
          "00" ;
    z <= '0' WHEN w = "0000" ELSE '1' ;
END Behavior ;
```

4 input bits

valid indicator

2 output bits

Active-high
inputs and outputs

Circuit Design using Multiplexers

Using a 2^n-input Multiplexer

- Use a 2^n-input multiplexer to realize a logic circuit for a function with 2^n minterms.
  - n = # of control inputs = # of variables in the function
- Each minterm of the function can be mapped to a data input of the multiplexer.
- For each row in the truth table, for the function, where the output is 1, set the corresponding data input of the multiplexer to 1.
  - That is, for each minterm in the minterm expansion of the function, set the corresponding input of the multiplexer to 1.
- Set the remaining inputs of the multiplexer to 0.

Using an 2^n-input Mux

Example:

Using an 8-to-1 multiplexer, design a logic circuit to realize the following Boolean function

$$F(A,B,C) = \Sigma m(2, 3, 5, 6, 7)$$

Using an 2^n-input Mux

Example:

Using an 8-to-1 multiplexer, design a logic circuit to realize the following Boolean function

$$F(A,B,C) = \Sigma m(1, 2, 4)$$

Using an $2^{(n-1)}$-input Multiplexer

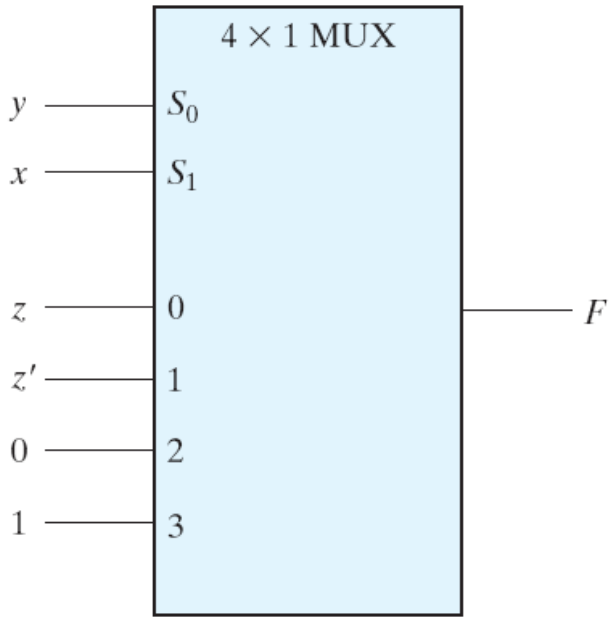
- Use a $2^{(n-1)}$-input multiplexer to realize a logic circuit for a function with 2^n minterms.
  - n – 1 = # of control inputs; n = # of variables in function
- Group the rows of the truth table, for the function, into $2^{(n-1)}$ pairs of rows.
  - Each pair of rows represents a product term of *(n – 1)* variables.
  - Each pair of rows is mapped to one data input of the mux.
- Determine the logical function of each pair of rows in terms of the remaining variable.
  - If the remaining variable, for example, is x, then the

Using an $2^{(n-1)}$-input Mux

Example: F(x,y,z) = Σm(1, 2, 6, 7)

x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

(a) Truth table

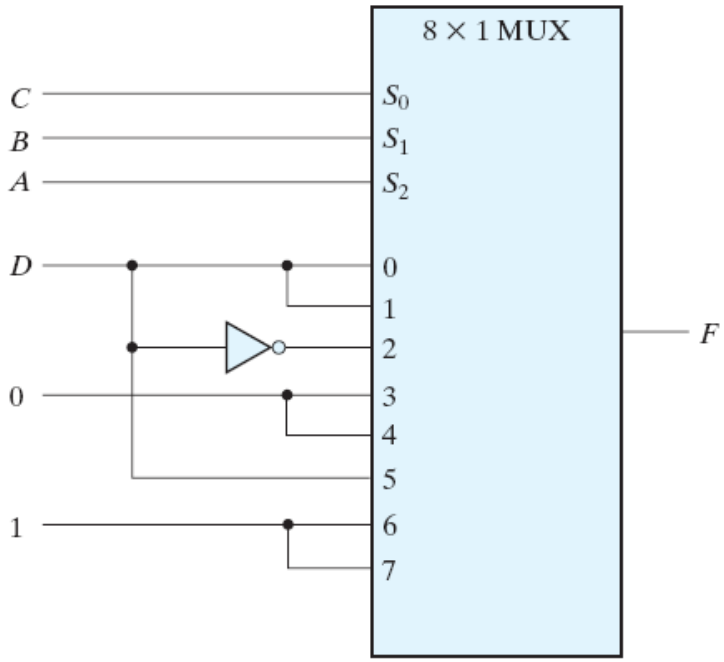


(b) Multiplexer implementation

Using an $2^{(n-1)}$-input Mux

Example: F(A,B,C,D) = Σm(1,3,4,11,12–15)

A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	



Using a $2^{(n-2)}$-input Mux

A similar design approach can be implemented using a $2^{(n-2)}$-input multiplexer.

Circuit Design using Decoders

Using an *n*-output Decoder

- Use an *n*-output decoder to realize a logic circuit for a function with *n* minterms.
- Each minterm of the function can be mapped to an output of the decoder.
- For each row in the truth table, for the function, where the output is 1, sum (or “OR”) the corresponding outputs of the decoder.
  - That is, for each minterm in the minterm expansion of the function, OR the corresponding outputs of the decoder.
- Leave remaining outputs of the decoder unconnected.

Using an *n*-output Decoder

Example:

Using a 3-to-8 decoder, design a logic circuit to realize the following Boolean function

$$F(A,B,C) = \Sigma m(2, 3, 5, 6, 7)$$

Using an *n*-output Decoder

Example:

Using two 2-to-4 decoders, design a logic circuit to realize the following Boolean function

$$F(A,B,C) = \Sigma m(0, 1, 4, 6, 7)$$

Hierarchical Design

Hierarchical Design

- Several issues arise when designing large multiplexers and decoders (as 2-level circuits).
  - Number of logic gates gets prohibitively large
  - Number of inputs to each logic gate (i.e. fan-in) gets prohibitively large
- Instead, design both hierarchically
  - Use smaller elements as building blocks
  - Interconnect building blocks in a multi-tier structure

Hierarchical Design

Exercise:

Design an 8-to-1 multiplexer using 4-to-1 and 2-to-1 multiplexers only.

Hierarchical Design

Exercise:

Design a 16-to-1 multiplexer using 4-to-1 multiplexers only.

Hierarchical Design

Exercise:

Design a 4-to-16 decoder using
2-to-4 decoders only.

Questions?