Combinational Circuit Building Blocks

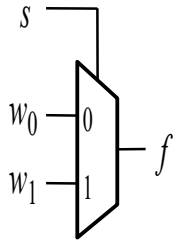
VOLKAN KURSUN

Some material from McGraw Hill

Outline

- Multiplexers
- Synthesis Using Multiplexers
- Decoders
- Synthesis Using Decoders
- Encoders
- Code Converter Example
- Comparators

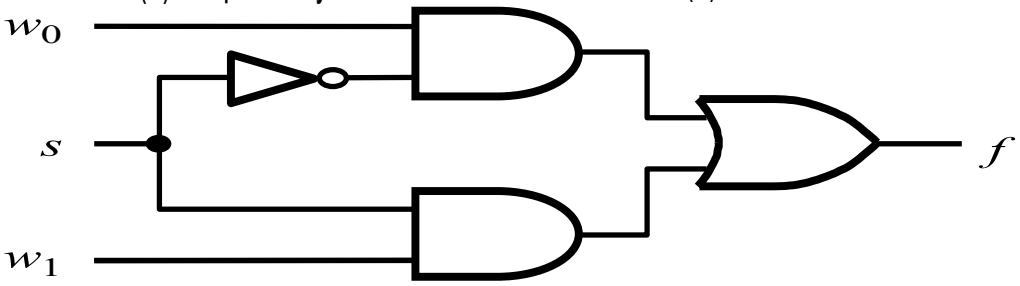
2-to-1 Multiplexer



(a) Graphical symbol

s	f
0	w_0
1	w_1

(b) Truth table



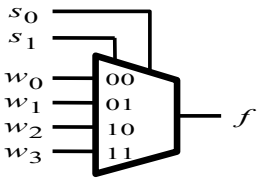
(c) Sum-of-products circuit

Larger Multiplexers

- A multiplexer with n data inputs needs $\log_2 n$ select inputs (or 2^n data inputs require n select signals)
- Larger multiplexers can be built using smaller multiplexers
- Example-1: a 4-to-1 multiplexer can be built with three 2-to-1 multiplexers
- Example-2: a 16-to-1 multiplexer can be built with five 4-to-1 multiplexers

4-to-1 Multiplexer

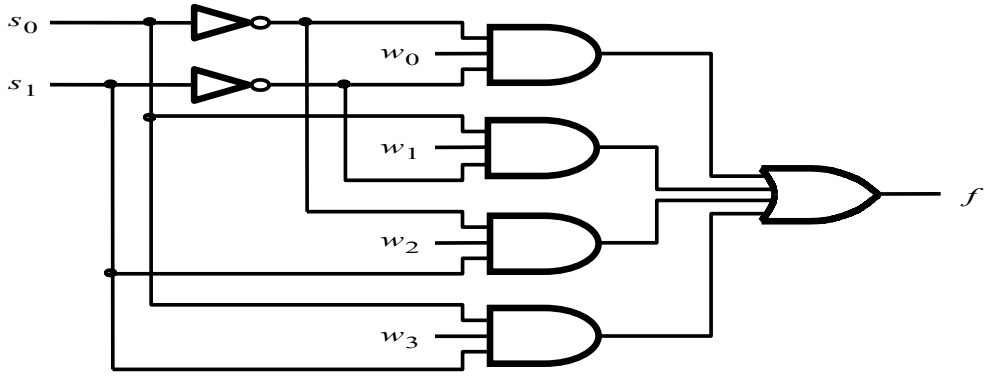
$f = s_1's_0'w_0 + s_1's_0w_1 + s_1s_0'w_2 + s_1s_0w_3$



(a) Graphic symbol

s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

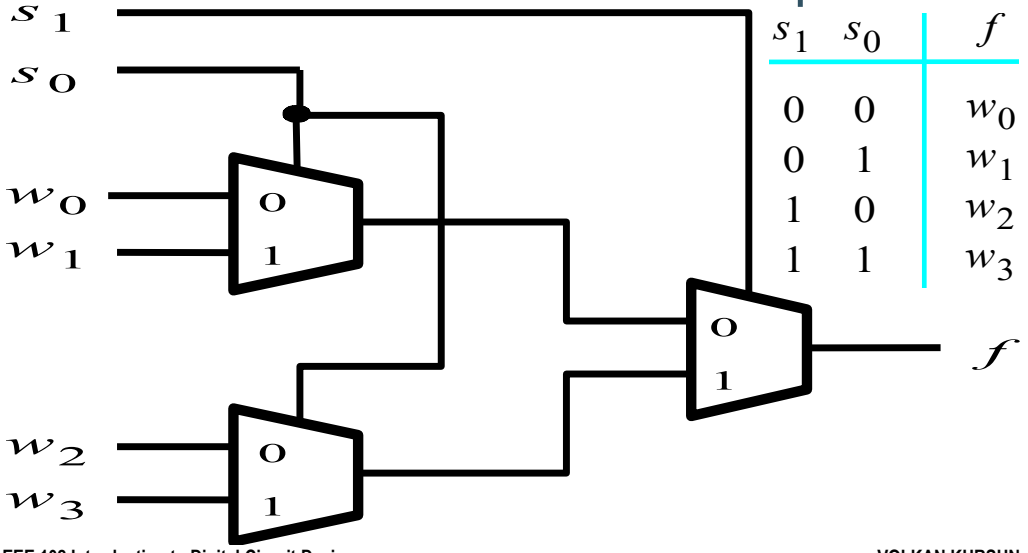
(b) Truth table



(c) Circuit

4-to-1 Multiplexer

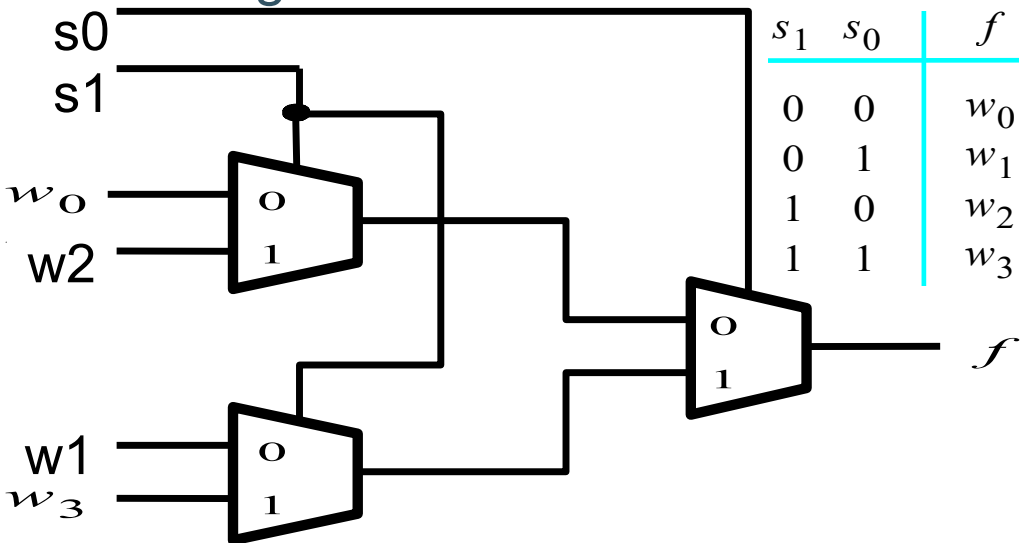
- Example-1: a 4-to-1 multiplexer can be built with three 2-to-1 multiplexers



s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

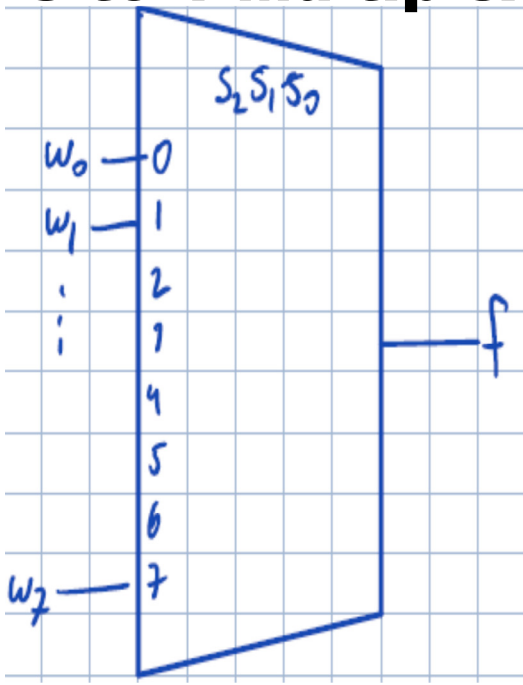
4-to-1 Multiplexer

- Example-1: alternative design with the select signals reversed



s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

8-to-1 Multiplexer VHDL Code



S_2	S_1	S_0	f
0	0	0	w_0
0	0	1	w_1
0	1	0	w_2
0	1	1	w_3
1	0	0	w_4
1	0	1	w_5
1	1	0	w_6
1	1	1	w_7

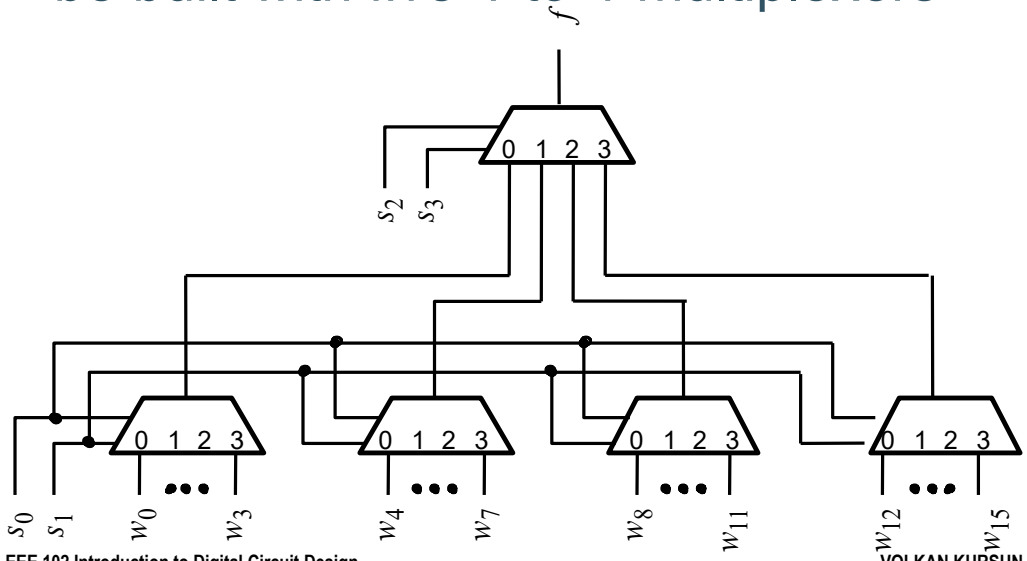
8-to-1 Multiplexer VHDL Code

```
entity mux8to1 is
    port(w: in std_logic_vector(7 downto 0);
         s: in std_logic_vector(2 downto 0);
         f: out std_logic);
end mux8to1;
architecture mux8to1_arch of mux8to1 is
begin
   with s select
      f <= w(0) when "000",
           w(1) when "001",
           w(2) when "010",
           w(3) when "011",
           w(4) when "100",
           w(7) when others;
end mux8to1_arch;
```

→ accounts for all non-listed logic values

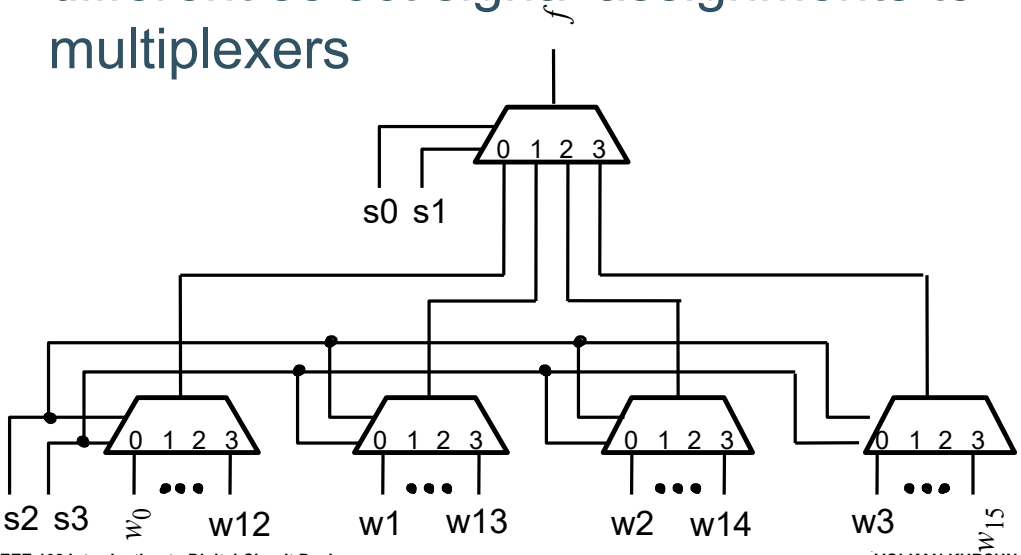
16-to-1 Multiplexer

- Example-2: a 16-to-1 multiplexer can be built with five 4-to-1 multiplexers



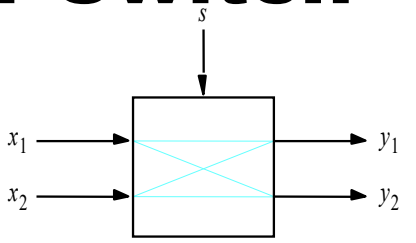
16-to-1 Multiplexer

- Example-2: alternative design with different select signal assignments to multiplexers

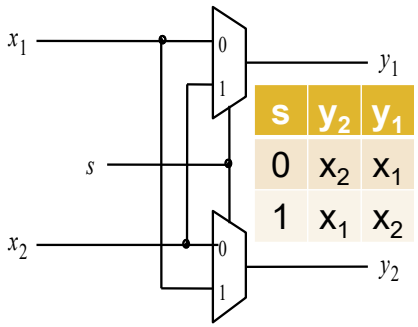


2x2 Crossbar Switch

- 2x2 crossbar switch: either of the two inputs x_1 and x_2 can be connected to either of the two outputs y_1 and y_2 under the control of a select signal s



(a) A 2x2 crossbar switch



s	y_2	y_1
0	x_2	x_1
1	x_1	x_2

(b) Implementation using multiplexers

nxk Crossbar Switch

- nxk crossbar switch: a circuit with n data inputs and k outputs with capability to connect any input to any output based on select signals
- k multiplexers and each multiplexer must be n-to-1
- Example-2: 8x10 crossbar switch, 10*8-to-1 multiplexers
- Example-3: 16x7 crossbar switch, 7*16-to-1 multiplexers

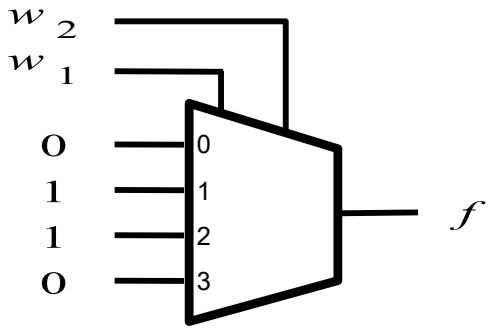
Outline

- Multiplexers
- Synthesis Using Multiplexers
- Decoders
- Synthesis Using Decoders
- Encoders
- Code Converter Example
- Comparators

Synthesis of Logic Functions

- Multiplexers can be used to synthesize logic functions: apply the inputs as select signals and the output corresponding to each row in the truth table as constant data inputs to the multiplexer
- Example: Consider the following XOR function with two inputs $\mathbf{f = w_1 \oplus w_2}$

w_1	w_2	f
0	0	0
0	1	1
1	0	1
1	1	0



Implementation using a 4-to-1 multiplexer

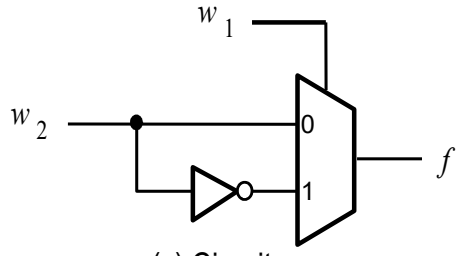
Synthesis of Logic Functions

- Example: Consider the following XOR function with two inputs $\mathbf{f = w_1 \oplus w_2}$
- A better implementation with only one 2-to-1 multiplexer: w_1 input is applied as the select signal to the multiplexer

w_1	w_2	f
0	0	0
0	1	1
1	0	1
1	1	0

w_1	f
0	w_2
1	$\bar{w}_2$

(b) Modified truth table



(c) Circuit

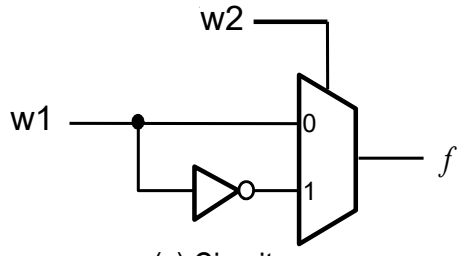
Synthesis of Logic Functions

- Example: Consider the following XOR function with two inputs $f = w_1 \oplus w_2$
- An alternative implementation with w2 driving the select signal: w_1 and w_1' are applied to the data inputs of multiplexer

w_1	w_2	f
0	0	0
0	1	1
1	0	1
1	1	0

w2	f
0	w1
1	w1'

(b) Modified truth table



(c) Circuit

3-Input Majority Function

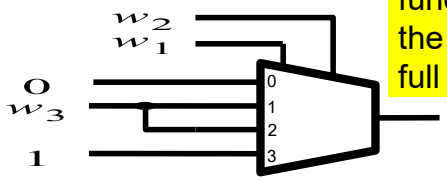
- Implementation with a 4-to-1 multiplexer: any two of the three inputs can be applied to the select inputs of the multiplexer

w_1	w_2	w_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

w_1	w_2	f
0	0	0
0	1	w_3
1	0	w_3
1	1	1

Modified truth table

Note: 3-input majority function implements the carry output of a full adder



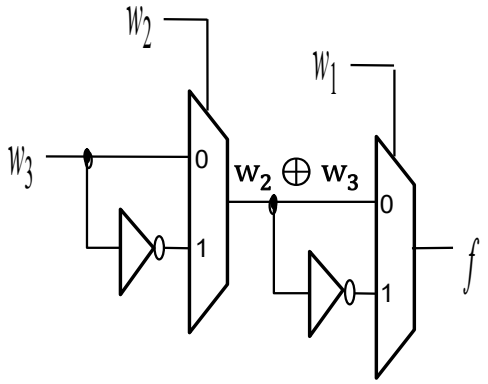
(b) Circuit

3-Input XOR with Multiplexers

- $f = w_1 \oplus w_2 \oplus w_3$
- Can be implemented with two 2-to-1 multiplexers. When $w_1 = 0$, $f = w_2 \oplus w_3$. Alternatively, when $w_1 = 1$, $f = \overline{w_2 \oplus w_3}$.

w_1	w_2	w_3	f	
0	0	0	0	$w_2 \oplus w_3$
0	0	1	1	
0	1	0	1	
0	1	1	0	
1	0	0	1	$\overline{w_2 \oplus w_3}$
1	0	1	0	
1	1	0	0	
1	1	1	1	

Truth table



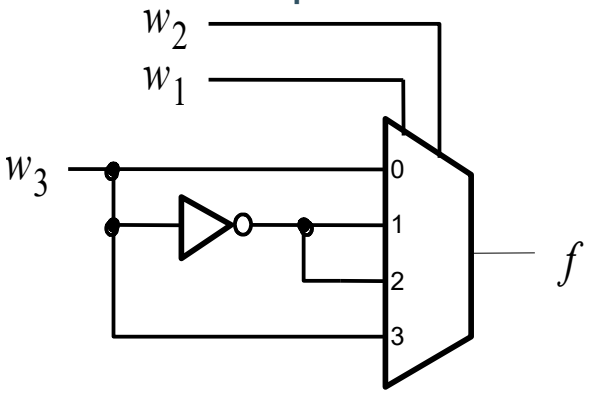
Circuit

3-Input XOR with 1 Multiplexer

- $f = w_1 \oplus w_2 \oplus w_3$
- Can be implemented with one 4-to-1 multiplexer. Use w_1 and w_2 to drive the select inputs of the 4-to-1 multiplexer

w_1	w_2	w_3	f	
0	0	0	0	w_3
0	0	1	1	
0	1	0	1	$\overline{w}_3$
0	1	1	0	
1	0	0	1	$\overline{w}_3$
1	0	1	0	
1	1	0	0	w_3
1	1	1	1	

Truth table



Circuit

Shannon's Expansion Theorem

Any Boolean function can be written in the following form:

$f(w_1, w_2, \ldots, w_n) = w_1'.f(0, w_2, \ldots, w_n) + w_1.f(1, w_2, \ldots, w_n)$

- The expansion is done with w_1 above. In general, any of the n variables can be used for the expansion.
- Example: revisit the three-variable majority function $f(w_1, w_2, w_3) = w_1w_2 + w_1w_3 + w_2w_3$

Let us expand f using w_1:

$f = w_1'(0.w_2 + 0.w_3 + w_2w_3) + w_1(1.w_2 + 1.w_3 + w_2w_3)$

$= w_1'(w_2w_3) + w_1(w_2 + w_3)$

3-Input Majority Function Revisited

- Example: revisit the three-variable majority function $f(w_1, w_2, w_3) = w_1w_2 + w_1w_3 + w_2w_3$

Let us expand f with w_1 using Shannon's theorem:

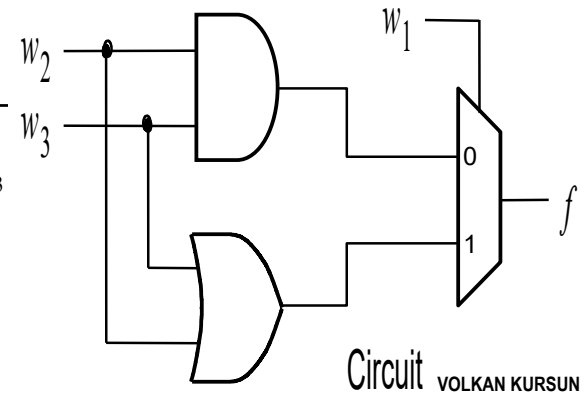
$f = w_1'(0.w_2 + 0.w_3 + w_2w_3) + w_1(1.w_2 + 1.w_3 + w_2w_3)$

$= w_1'(w_2w_3) + w_1(w_2 + w_3)$

w_1	w_2	w_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

w_1	f
0	w_2w_3
1	$w_2 + w_3$

Truth table



3-Input XOR Function Revisited

- Example: revisit the 3-input XOR function $f(w_1, w_2, w_3) = w_1 \oplus w_2 \oplus w_3$

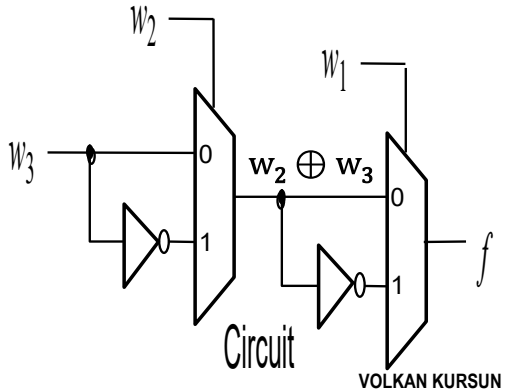
Let us expand f with w_1 using Shannon's theorem:

$f = w_1'(0 \oplus w_2 \oplus w_3) + w_1(1 \oplus w_2 \oplus w_3)$

$= w_1'(w_2 \oplus w_3) + w_1(\overline{w_2 \oplus w_3})$

w_1	w_2	w_3	f	
0	0	0	0	$w_2 \oplus w_3$
0	0	1	1	
0	1	0	1	
0	1	1	0	
1	0	0	1	$\overline{w_2 \oplus w_3}$
1	0	1	0	
1	1	0	0	
1	1	1	1	

Truth table



Choice of Variable for Shannon's Expansion

- The complexity of the logic expression may vary depending on which variable w_i is used for expansion: **try different variables and choose the expansion with the lowest cost**
- Example: Consider the following function

$f(w_1, w_2, w_3) = w_1'w_3 + w_2w_3'$

Shannon's expansion with w_1 yields:

$f = w_1'(1.w_3 + w_2w_3') + w_1(0.w_3 + w_2w_3')$

$= w_1'(w_3 + w_2) + w_1(w_2w_3')$

Shannon's expansion with w_2 yields:

$f = w_2'(w_1'w_3 + 0.w_3') + w_2(w_1'w_3 + 1.w_3')$

$= w_2'w_1'w_3 + w_2(w_1' + w_3')$

Shannon's expansion with w_3 yields the **lowest cost**:

$f = w_3'(w_1'.0 + w_2.1) + w_3(w_1'.1 + w_2.0) = w_3'w_2 + w_3w_1'$

Shannon's Multi-Variable Expansion

- Shannon's expansion can be done with more than one variable
- Expansion with w_1 and w_2 for implementation with a 4-to-1 multiplexer:

$f(w_1, w_2, \dots, w_n) = w_1'w_2'.f(0, 0, \dots, w_n) + w_1'w_2.f(0, 1, \dots, w_n) + w_1w_2'.f(1, 0, \dots, w_n) + w_1w_2.f(1, 1, \dots, w_n)$

- Expansion with w_1, w_2, and w_3 for implementation with an 8-to-1 multiplexer:

$f(w_1, w_2, \dots, w_n) = w_1'w_2'w_3'.f(0, 0, 0, \dots, w_n) + w_1'w_2'w_3.f(0, 0, 1, \dots, w_n) + w_1'w_2w_3'.f(0, 1, 0, \dots, w_n) + w_1'w_2w_3.f(0, 1, 1, \dots, w_n) + w_1w_2'w_3'.f(1, 0, 0, \dots, w_n) + w_1w_2'w_3.f(1, 0, 1, \dots, w_n) + w_1w_2w_3'.f(1, 1, 0, \dots, w_n) + w_1w_2w_3.f(1, 1, 1, \dots, w_n)$

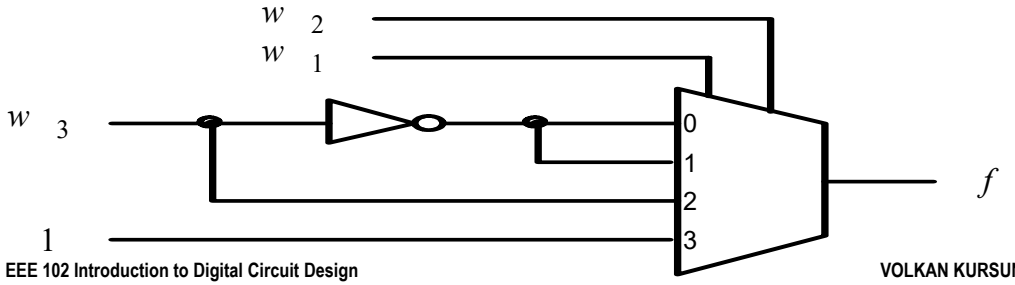
Shannon's Multi-Variable Expansion

- Example: consider the following function

$f(w_1, w_2, w_3) = w_1'w_3' + w_1w_2 + w_1w_3$

- Expansion with w_1 and w_2 for implementation with a 4-to-1 multiplexer:

$f(w_1, w_2, w_3) = w_1'w_2'f(0, 0, w_3) + w_1'w_2f(0, 1, w_3) + w_1w_2'f(1, 0, w_3) + w_1w_2.f(1, 1, w_3) = w_1'w_2'w_3' + w_1'w_2w_3' + w_1w_2'w_3 + w_1w_2 1$



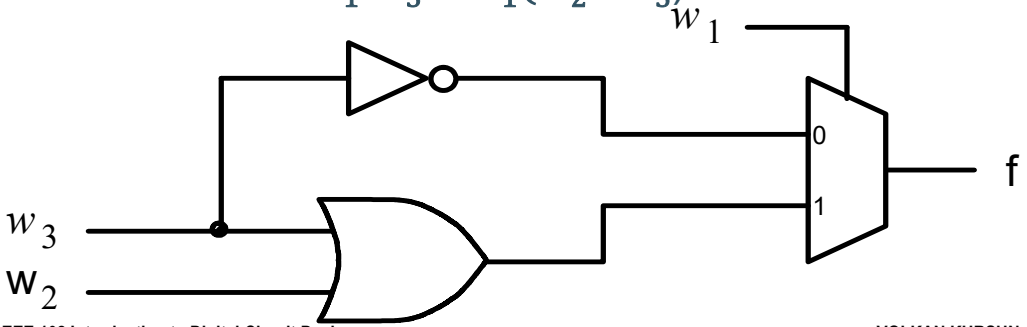
Shannon's Single-Variable Expansion

- Example: implement the same function with 2-to-1 multiplexer

$f(w_1, w_2, w_3) = w_1'w_3' + w_1w_2 + w_1w_3$

- Expansion with w_1 yields:

$f(w_1, w_2, w_3) = w_1'f(0, w_2, w_3) + w_1f(1, w_2, w_3)$
$= w_1'w_3' + w_1(w_2 + w_3)$



3-Input Majority Function Revisited

- Example: the three-variable majority function

$f(w_1, w_2, w_3) = w_1w_2 + w_1w_3 + w_2w_3$

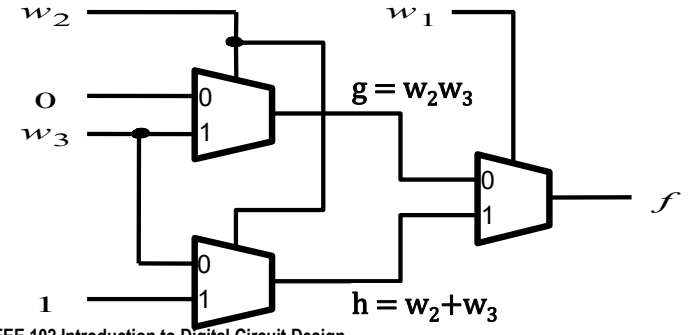
We had expanded f with w_1 using Shannon's theorem:

$f = w_1'(w_2w_3) + w_1(w_2 + w_3)$ Let $g = w_2w_3$ and $h = w_2 + w_3$

Let us expand g and h with w_2:

$g = w_2'(0.w_3) + w_2(1.w_3) = w_2'0 + w_2w_3$

$h = w_2'(0 + w_3) + w_2(1 + w_3) = w_2'w_3 + w_2 1$



w_1	w_2	w_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

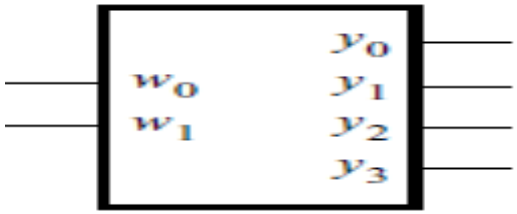
Outline

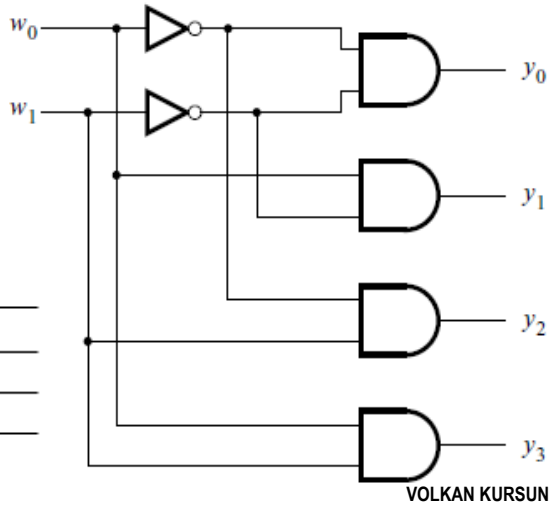
- Multiplexers
- Synthesis Using Multiplexers
- Decoders
- Synthesis Using Decoders
- Encoders
- Code Converter Example
- Comparators

Binary Decoder

- A binary number applied as input is decoded and the corresponding output is asserted. Only one output is asserted at a time (**one-hot encoding**) and the **asserted output corresponds to the valuation** of the binary input

w_1	w_0	y_0	y_1	y_2	y_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

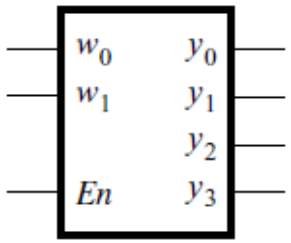


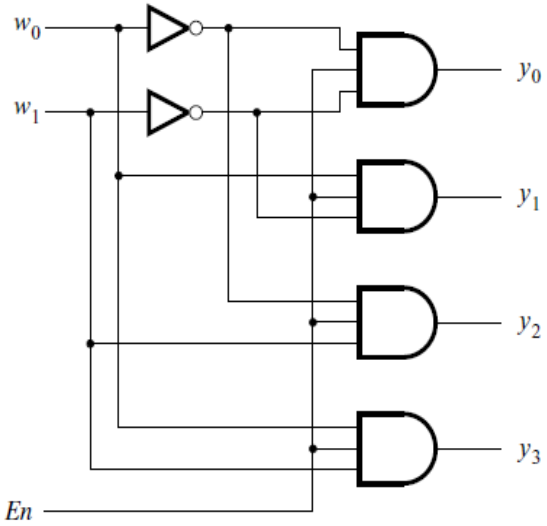


Binary Decoder with Enable

- When Enable = 1, the decoder behaves as before
- When Enable = 0, all outputs are forced to 0: when the decoder is disabled, no output is asserted regardless of the values of w_0 and w_1

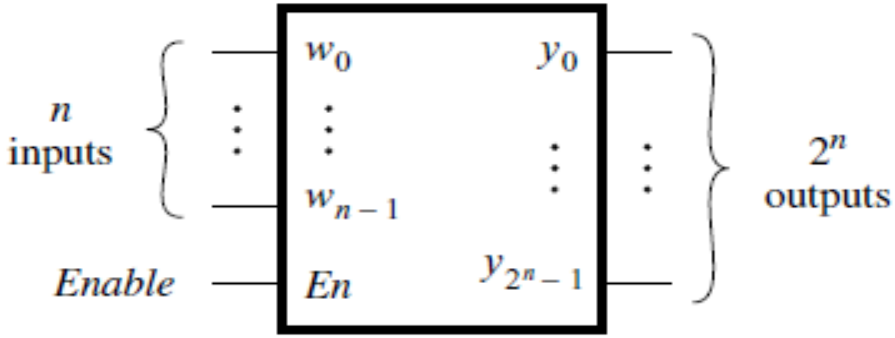
En	w_1	w_0	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0





Larger Decoders

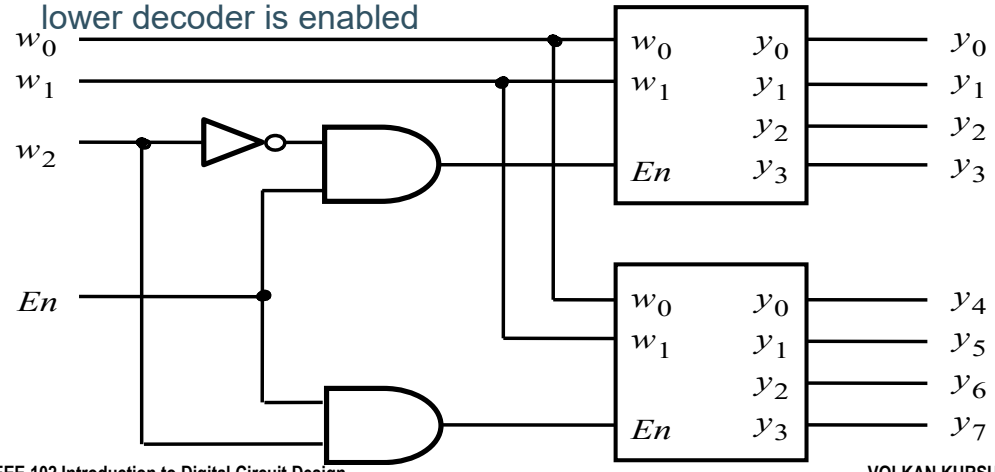
- n-to-2^n decoder: a binary decoder with n inputs has 2^n outputs



- Large decoders can be built as an array of inverters and AND gates. Alternatively, larger decoders can be constructed from smaller decoders.

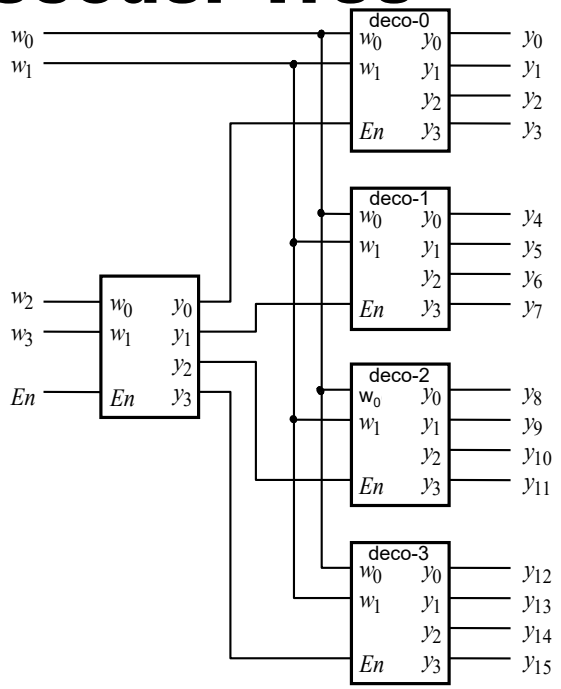
3-to-8 Decoder

- Example: a 3-to-8 decoder built from two 2-to-4 decoders where the w_2 input drives the enable inputs of the two decoders
- w_2 = 0 and enable = 1, upper decoder is enabled while the lower decoder is disabled
- w_2 = 1 and enable = 1, upper decoder is disabled while the lower decoder is enabled



4-to-16 Decoder Tree

- Example: a 4-to-16 decoder built from five 2-to-4 decoders where the first decoder drives the enable inputs of the four decoders in the second stage
- w_3w_2 = 0b00 and en = 1, deco-0 is enabled while the other decoders in the second stage are disabled
- w_3w_2 = 0b01 and en = 1, deco-1 is enabled while the other decoders in the second stage are disabled
- w_3w_2 = 0b10 and en = 1, deco-2 is enabled while the other decoders in the second stage are disabled
- w_3w_2 = 0b11 and en = 1, deco-3 is enabled while the other decoders in the second stage are disabled
- en = 0, all decoders are disabled and all outputs are forced to 0

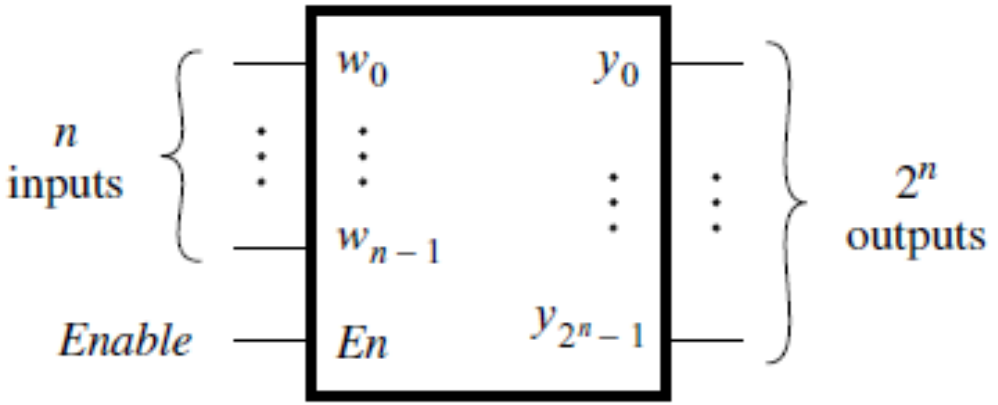


Outline

- Multiplexers
- Synthesis Using Multiplexers
- Decoders
- Synthesis Using Decoders
- Encoders
- Code Converter Example
- Comparators

Synthesis Using Decoders

- **Any logic function** with n-inputs can be implemented using an n-to-2^n decoder and OR gates (**OR the minterms for which the function = 1**)



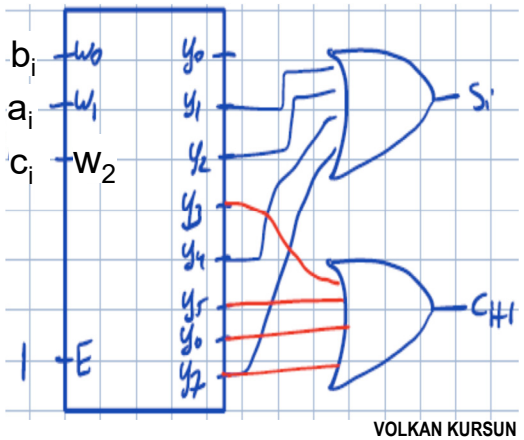
Synthesis Using Decoders

- Example: 3-input full adder designed with a 3-to-8 decoder and two OR gates

c_i	a_i	b_i	c_{i+1}	s_i
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

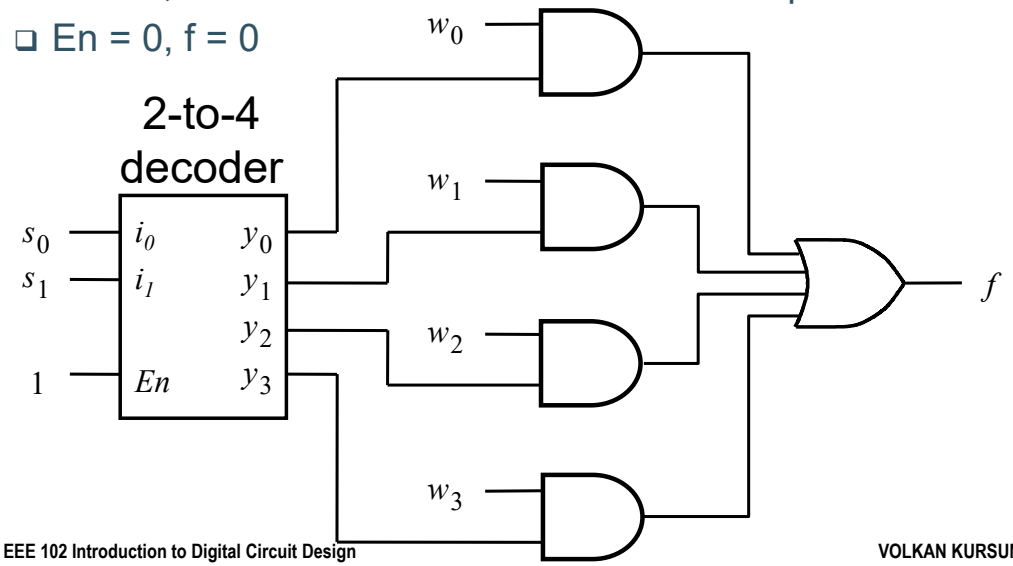
$$s_i = \sum m(1,2,4,7)$$

$$c_{i+1} = \sum m(3,5,6,7)$$



4-to-1 Multiplexer Using a Decoder

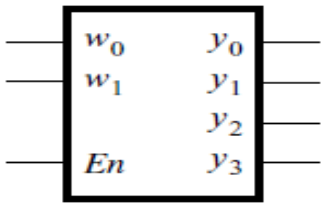
- The one-hot encoded outputs of a decoder can be used to select among the input channels of a multiplexer
- En = 1, the circuit behaves as a 4-to-1 multiplexer
- En = 0, f = 0



Decoders Act As Demultiplexers

- Demultiplexer does the opposite of a multiplexer: places the data from a single input channel onto one of multiple output channels
- Example: a 2-to-4 decoder can be used as a 1-to-4 demultiplexer. The **enable input of the decoder serves as the data input of the demultiplexer**. The actual data inputs w_1 and w_0 of the decoder serve as the select signals for the demultiplexer. The valuation of w_1w_0 determines which of the four outputs the enable signal is transferred to when en = 1. Alternatively, when en = 0, all outputs are 0

En	w_1	w_0	y_0	y_1	y_2	y_3
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
0	x	x	0	0	0	0



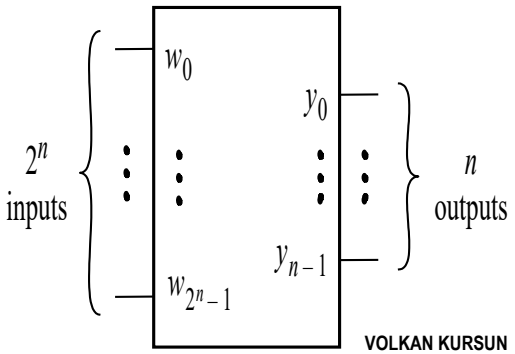
Outline

- Multiplexers
- Synthesis Using Multiplexers
- Decoders
- Synthesis Using Decoders
- Encoders
- Code Converter Example
- Comparators

Binary Encoders

- 2^n-to-n encoder: performs the opposite function of a decoder. Encodes information from 2^n inputs into an n-bit code
- Only one input is supposed to have a value of 1 and the **outputs present the binary number that identifies which input is asserted**

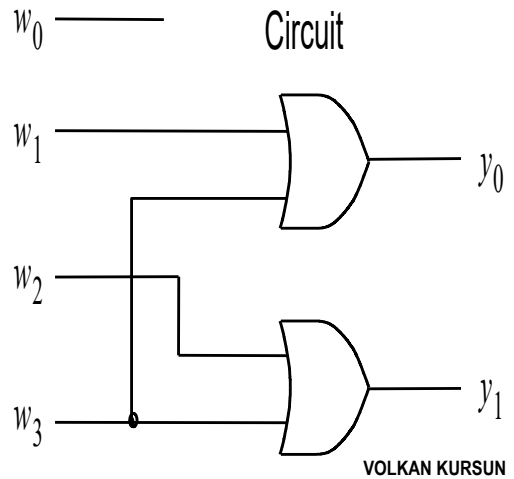
w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1



4-to-2 Binary Encoder

- Encodes information from 4 inputs into a 2-bit code: output y_1 is 1 when w_3 or w_2 is 1 and output y_0 is 1 when w_1 or w_3 is 1. Therefore, the outputs can be generated with two OR gates.

w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1



What Are Encoders Used For?

- Practical use: more **economical** (**2^n-to-n compaction**) transmission of information in digital systems
- Encoders are used to reduce the number of bits to represent information: encoding the information allows the **transmission link to be built with fewer wires**
- Encoding also **reduces the needed memory capacity for storing information**: **fewer bits are stored**

Priority Encoder

- Each input has a priority level associated with it and the encoder **output indicates the active input with the highest priority**
- When an **input with higher priority is asserted**, the other **inputs with lower priority are ignored**
- Example: 4-to-2 priority encoder where w_3 has the highest priority and w_0 has the lowest priority. The **outputs y_1 and y_0 represent the binary number that corresponds to the highest priority input set to 1**. A third output z is provided to check if at least one input is asserted. If at least one of the inputs is 1, z = 1. If none of the inputs is 1 (all inputs are 0), z = 0.

4-to-2 Priority Encoder

- Example: 4-to-2 priority encoder where w_3 has the highest priority and w_0 has the lowest priority

w_3	w_2	w_1	w_0	y_1	y_0	z
0	0	0	0	d	d	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

- Define a set of intermediate signals i_0, i_1, i_2, and i_3

$i_0 = w_3'w_2'w_1'w_0$, $i_1 = w_3'w_2'w_1$, $i_2 = w_3'w_2$, $i_3 = w_3$

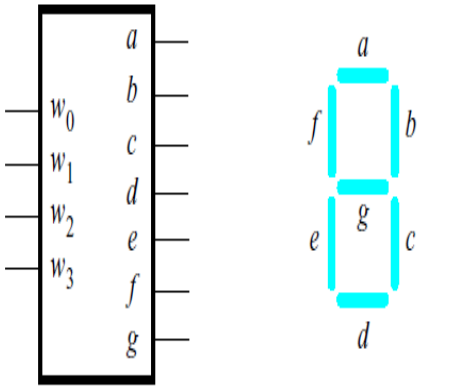
$y_0 = i_1 + i_3$, $y_1 = i_2 + i_3$, $z = i_0 + i_1 + i_2 + i_3$

Outline

- Multiplexers
- Synthesis Using Multiplexers
- Decoders
- Synthesis Using Decoders
- Encoders
- Code Converter Example
- Comparators

Hexadecimal Display

- Design a decoder to display hexadecimal digits that correspond to 4-bit binary inputs on a 7-segment display



(a) Code converter (b) 7-segment display

w_3	w_2	w_1	w_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	1	1	1	0	1	1	1
1	0	1	1	0	0	1	1	1	1	1
1	1	0	0	1	0	0	1	1	1	0
1	1	0	1	0	1	1	1	1	0	1
1	1	1	0	1	0	0	1	1	1	1
1	1	1	1	1	0	0	0	1	1	1

Hexadecimal Display VHDL

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

ENTITY seg7 IS
    PORT ( hex  : IN  STD_LOGIC_VECTOR(3 DOWNTO 0) ;
           leds : OUT STD_LOGIC_VECTOR(1 TO 7) ) ;
END seg7 ;

ARCHITECTURE Behavior OF seg7 IS
BEGIN
    PROCESS ( hex )
    BEGIN
        CASE hex IS                          --  abcdefg
            WHEN "0000" => leds <= "1111110" ;
            WHEN "0001" => leds <= "0110000" ;
            WHEN "0010" => leds <= "1101101" ;
            WHEN "0011" => leds <= "1111001" ;
            WHEN "0100" => leds <= "0110011" ;
            WHEN "0101" => leds <= "1011011" ;
            WHEN "0110" => leds <= "1011111" ;
            WHEN "0111" => leds <= "1110000" ;
            WHEN "1000" => leds <= "1111111" ;
            WHEN "1001" => leds <= "1111011" ;
            WHEN "1010" => leds <= "1110111" ;
            WHEN "1011" => leds <= "0011111" ;
            WHEN "1100" => leds <= "1001110" ;
            WHEN "1101" => leds <= "0111101" ;
            WHEN "1110" => leds <= "1001111" ;
            WHEN OTHERS => leds <= "1000111" ;
        END CASE ;
    END PROCESS ;
END Behavior ;
```

Association operator

Case statement must include a WHEN clause for all possible valuations of the select signal: OTHERS keyword covers all remaining valuations

Outline

- Multiplexers
- Synthesis Using Multiplexers
- Decoders
- Synthesis Using Decoders
- Encoders
- Code Converter Example
- Comparators

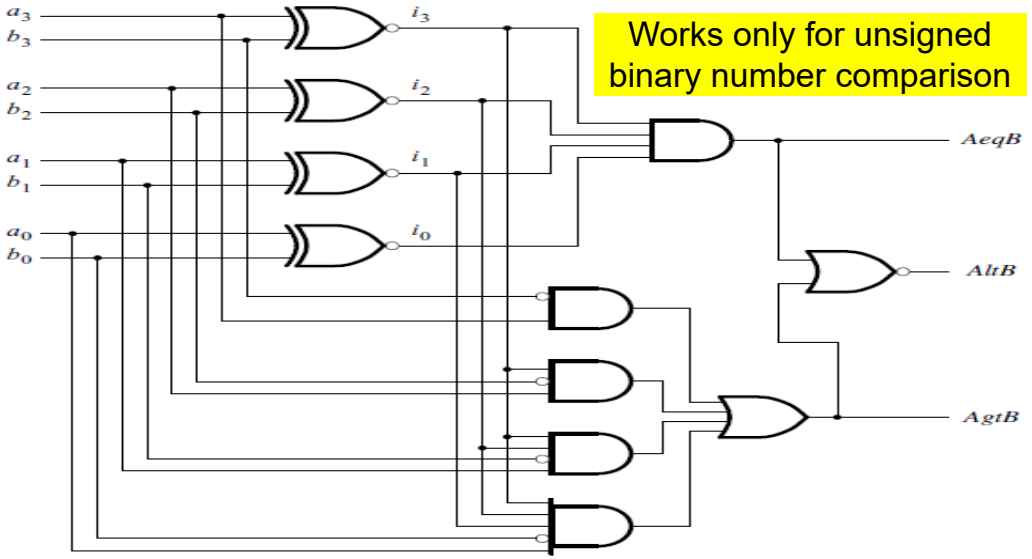
Unsigned Comparator

- Compare the values of two unsigned binary numbers
- Example: design a 4-bit comparator to compare the values of two unsigned inputs A = $a_3a_2a_1a_0$ and B = $b_3b_2b_1b_0$
- Define intermediate signals i_0, i_1, i_2, and i_3 to determine if the corresponding bit positions are equal: $i_0 = \overline{a_0 \oplus b_0}$, $i_1 = \overline{a_1 \oplus b_1}$, $i_2 = \overline{a_2 \oplus b_2}$, $i_3 = \overline{a_3 \oplus b_3}$
- AeqB = $i_3i_2i_1i_0$
- AgtB = $a_3b_3' + i_3a_2b_2' + i_3i_2a_1b_1' + i_3i_2i_1a_0b_0'$
- AltB = (AeqB + AgtB)' = AeqB'AgtB'

4-Bit Unsigned Comparator

AeqB = $i_3i_2i_1i_0$, AgtB = $a_3b_3' + i_3a_2b_2' + i_3i_2a_1b_1' + i_3i_2i_1a_0b_0'$

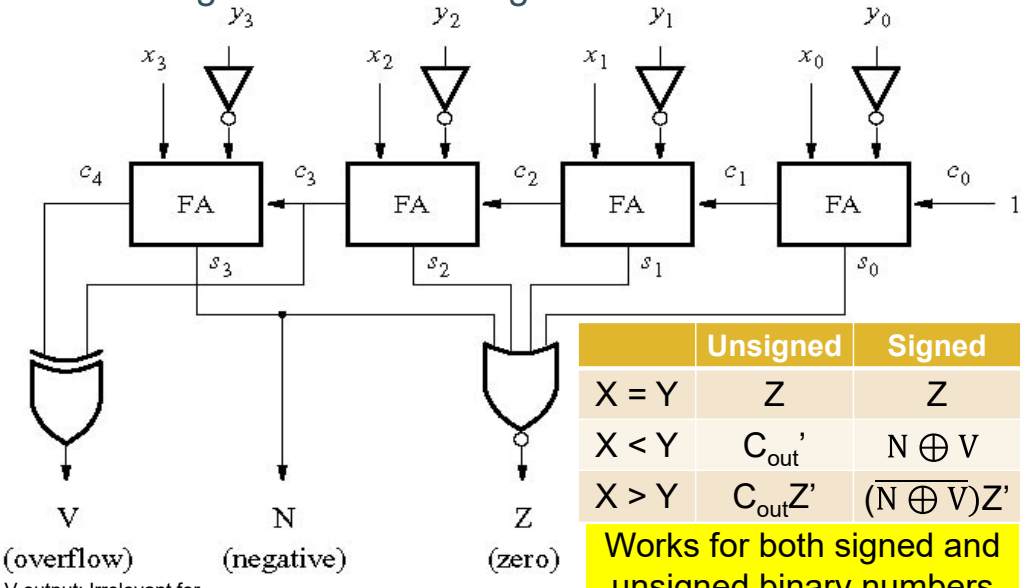
AltB = (AeqB + AgtB)' = AeqB'AgtB'



Works only for unsigned binary number comparison

4-Bit Comparator

Use the negative and zero flags and Cout of a subtractor:



V output: Irrelevant for unsigned subtraction

	Unsigned	Signed
X = Y	Z	Z
X < Y	C_{out}'	N ⊕ V
X > Y	$C_{out}Z'$	$\overline{(N \oplus V)}Z'$

Works for both signed and unsigned binary numbers