

Digital System Design

Module 3 - COMBINATIONAL LOGIC

Dr. Deepthi Sasidharan

Assistant Professor, Department of Information Technology
GEC Barton Hill, Thiruvananthapuram

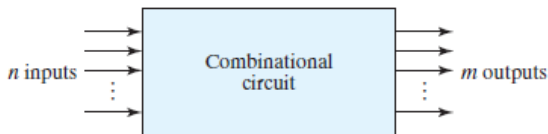
28.09.2020

Module 3 : COMBINATIONAL LOGIC

- ▶ Combinational Circuits Analysis and Design Procedures
- ▶ Binary Adder-Subtractor (Half & Full)
- ▶ Carry look ahead adder, BCD adder, code converter,
- ▶ Magnitude Comparator
- ▶ Decoders, Encoders, Parity Generator
- ▶ Multiplexers, DE-multiplexers, Implementation of Boolean functions using MUX.

Combinational Circuits

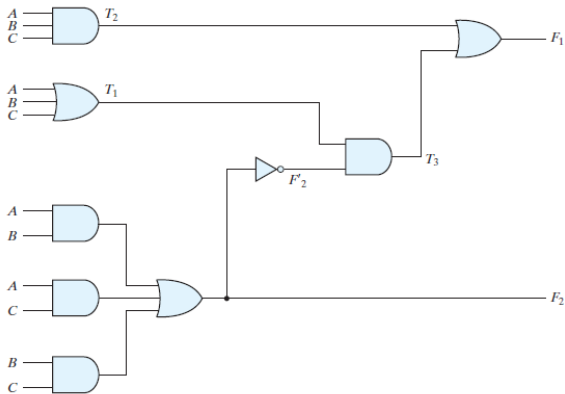
- ▶ A combinational circuit consists of an interconnection of logic gates.
- ▶ Combinational logic gates react to the values of the signals at their inputs and produce the value of the output signal, transforming binary information from the given input data to a required output data.



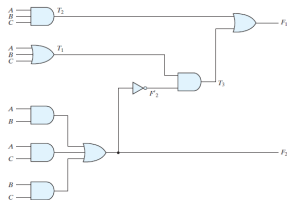
Block diagram of combinational circuit

Analysis of a combinational circuit

- Step 1** Label all gate outputs that are a function of input variables with arbitrary symbols— but with meaningful names. Determine the Boolean functions for each gate output.
- Step 2** Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols. Find the Boolean functions for these gates.
- Step 3** Repeat the process outlined in step 2 until the outputs of the circuit are obtained.
- Step 4** By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables.



Logic diagram for analysis example



Logic diagram for analysis example

- ▶ $F_2 = AB + AC + BC$
- ▶ $T_1 = A + B + C$
- ▶ $T_2 = ABC$

- ▶ $T_3 = F_2' T_1$
- ▶ $F_1 = T_3 + T_2$
- ▶ $F_1 = T_3 + T_2 = F_2' T_1 + ABC$

$$= (AB + AC + BC)'(A + B + C) + ABC$$

$$= (A' + B')(A' + C')(B' + C')(A + B + C) + ABC$$

$$= (A' + B')(A' + C')(B' + C')(A + B + C) + ABC$$

$$= (A' + B' C')(AB' + AC' + BC' + B' C) + ABC$$

$$= A' BC' + A' B' C + AB' C' + ABC$$

Digital System Design

Module 3 - COMBINATIONAL LOGIC

Dr. Deepthi Sasidharan

Assistant Professor, Department of Information Technology
GEC Barton Hill, Thiruvananthapuram

September 29, 2020

DESIGN PROCEDURE

The design of combinational circuits starts from the specification of the design objective and culminates in a logic circuit diagram or a set of Boolean functions from which the logic diagram can be obtained.

1. From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each.
2. Derive the truth table that defines the required relationship between inputs and outputs.
3. Obtain the simplified Boolean functions for each output as a function of the input variables.
4. Draw the logic diagram and verify the correctness of the design (manually or by simulation).

Code Conversion Example

A code converter is a circuit that makes the two systems compatible even though each uses a different binary code.

Code Conversion Example

A code converter is a circuit that makes the two systems compatible even though each uses a different binary code.

Convert binary coded decimal (BCD) to the excess-3 code for the decimal digits.

Code Conversion Example

A code converter is a circuit that makes the two systems compatible even though each uses a different binary code.

Convert binary coded decimal (BCD) to the excess-3 code for the decimal digits.

Input: BCD - 4 binary digits

► A, B, C, D

Code Conversion Example

A code converter is a circuit that makes the two systems compatible even though each uses a different binary code.

Convert binary coded decimal (BCD) to the excess-3 code for the decimal digits.

Input: BCD - 4 binary digits

► A, B, C, D

Output: Excess-3 code - 4 binary digits

► w, x, y, z

The steps involved in design procedure

1. From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each.
2. Derive the truth table that defines the required relationship between inputs and outputs.
3. Obtain the simplified Boolean functions for each output as a function of the input variables.
4. Draw the logic diagram and verify the correctness of the design (manually or by simulation).

Truth Table for Code Conversion Example

Input BCD			
<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1

Truth Table for Code Conversion Example

Input BCD				Output Excess-3 Code			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

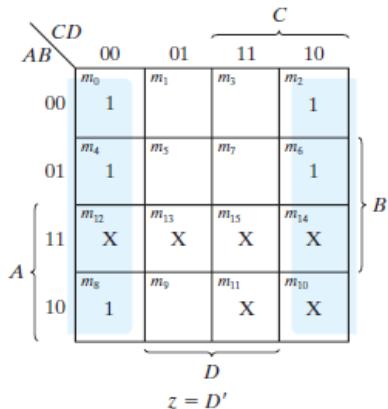
The steps involved in design procedure

1. From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each.
2. Derive the truth table that defines the required relationship between inputs and outputs.
3. Obtain the simplified Boolean functions for each output as a function of the input variables.
4. Draw the logic diagram and verify the correctness of the design (manually or by simulation).

Simplified Boolean Functions

Truth Table for Code Conversion Example

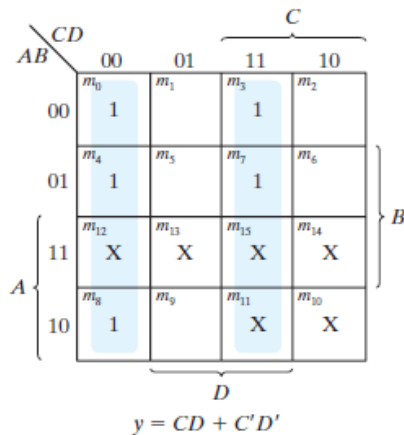
Input BCD				Output Excess-3 Code			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0



Simplified Boolean Functions

Truth Table for Code Conversion Example

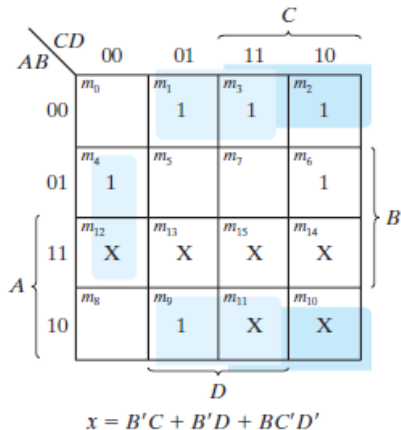
Input BCD				Output Excess-3 Code			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0



Simplified Boolean Functions

Truth Table for Code Conversion Example

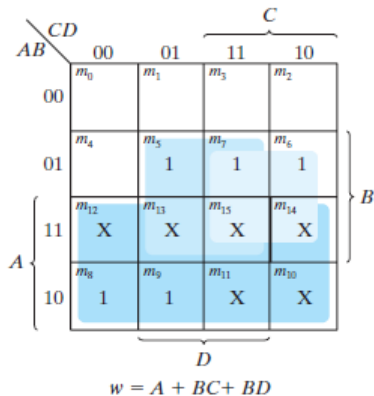
Input BCD				Output Excess-3 Code			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0



Simplified Boolean Functions

Truth Table for Code Conversion Example

Input BCD				Output Excess-3 Code			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0



Simplified Boolean Functions

$$z = D'$$

$$y = CD + C'D' = CD + (C + D)'$$

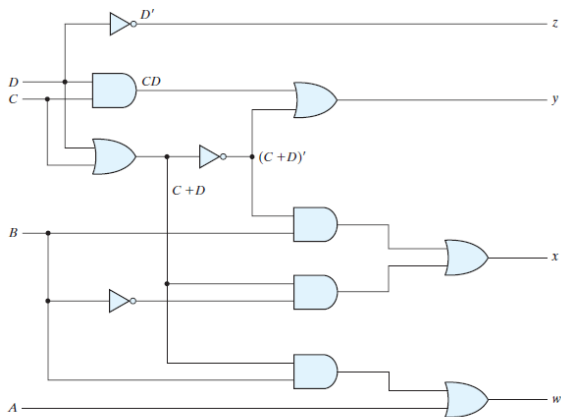
$$\begin{aligned} x &= B'C + B'D + BC'D' = B'(C + D) + BC'D' \\ &= B'(C + D) + B(C + D)' \end{aligned}$$

$$w = A + BC + BD = A + B(C + D)$$

The steps involved in design procedure

1. From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each.
2. Derive the truth table that defines the required relationship between inputs and outputs.
3. Obtain the simplified Boolean functions for each output as a function of the input variables.
4. Draw the logic diagram and verify the correctness of the design (manually or by simulation).

Circuit Diagram



Logic diagram for BCD-to-excess-3 code converter

Digital System Design

Module 3 - COMBINATIONAL LOGIC

Dr. Deepthi Sasidharan

Assistant Professor, Department of Information Technology
GEC Barton Hill, Thiruvananthapuram

October 1, 2020

DESIGN PROCEDURE

The design of combinational circuits starts from the specification of the design objective and culminates in a logic circuit diagram or a set of Boolean functions from which the logic diagram can be obtained.

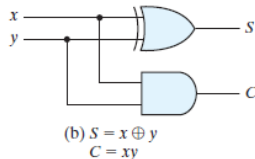
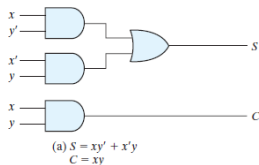
1. From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each.
2. Derive the truth table that defines the required relationship between inputs and outputs.
3. Obtain the simplified Boolean functions for each output as a function of the input variables.
4. Draw the logic diagram and verify the correctness of the design (manually or by simulation).

Binary Addition - Half Adder

- ▶ A combinational circuit that performs the addition of two bits is called a **half adder**.
- ▶ One that performs the addition of three bits (two significant bits and a previous carry) is a **full adder**.

Half Adder

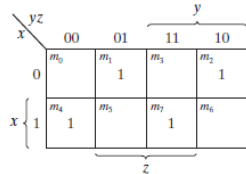
x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



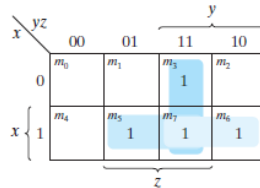
Binary Addition - Full Adder

Full Adder

x	y	z	C	S
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



$$(a) S = x'y'z + x'yz' + xy'z' + xyz$$

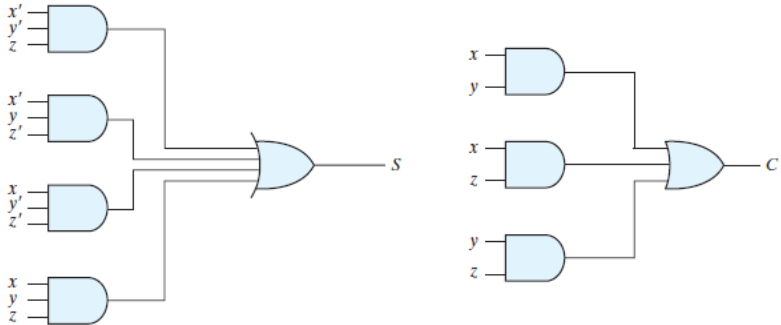


$$(b) C = xy + xz + yz$$

Full Adder

$$S = x'y'z + x'yz' + xy'z' + xyz$$

$$C = xy + yz + xz$$



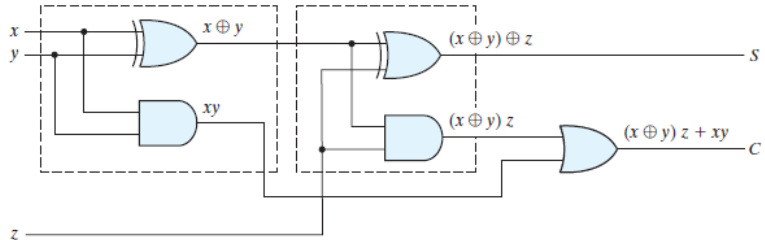
Implementation of full adder in sum-of-products form

Full Adder

$$\begin{aligned}S &= x'y'z + x'yz' + xy'z' + xyz \\&= z(x'y' + xy) + z'(x'y + xy') \\&= z(x \oplus y)' + z'(x \oplus y) \\&= z \oplus (x \oplus y)\end{aligned}$$

$$\begin{aligned}C &= xy + yz + xz = xy + yz(x + x') + xz(y + y') \\&= xy + xyz + x'yz + xyz + xy'z \\&= xy + z(x'y + xy') = xy + z(x \oplus y)\end{aligned}$$

Full Adder

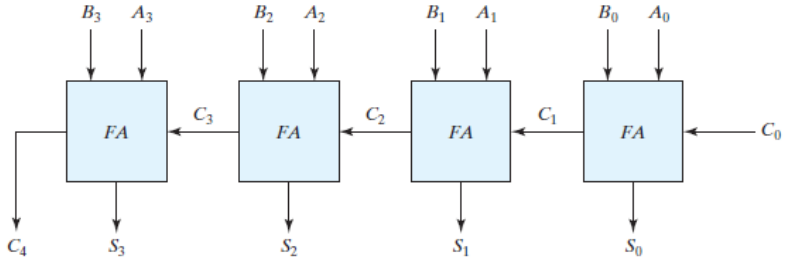


Implementation of full adder with two half adders and an OR gate

4-bit Adder

Subscript i:	3	2	1	0	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}

4-bit Adder



Digital System Design

Module 3 - COMBINATIONAL LOGIC

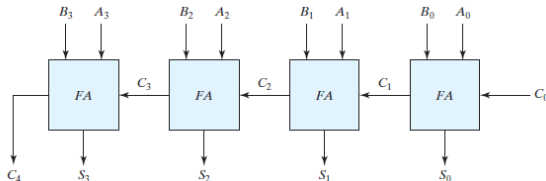
Dr. Deepthi Sasidharan

Assistant Professor, Department of Information Technology
GEC Barton Hill, Thiruvananthapuram

October 1, 2020

Carry Propagation

- ▶ The addition of two binary numbers in parallel implies that all the bits of the augend and addend are available for computation at the same time.
- ▶ The signal must propagate through the gates before the correct output sum is available in the output terminals.
- ▶ The total propagation time is equal to the propagation delay of a typical gate, times the number of gate levels in the circuit.



Carry Propagation

$$C_0 = \text{input_carry}$$

$$C_1 = A_0B_0 + C_0(A_0 \oplus B_0)$$

$$\begin{aligned} C_2 &= A_1B_1 + C_1(A_1 \oplus B_1) \\ &= A_1B_1 + (A_0B_0 + C_0(A_0 \oplus B_0))(A_1 \oplus B_1) \end{aligned}$$

Now, let

$$P_i = A_i \oplus B_i$$

$$G_i = A_iB_i$$

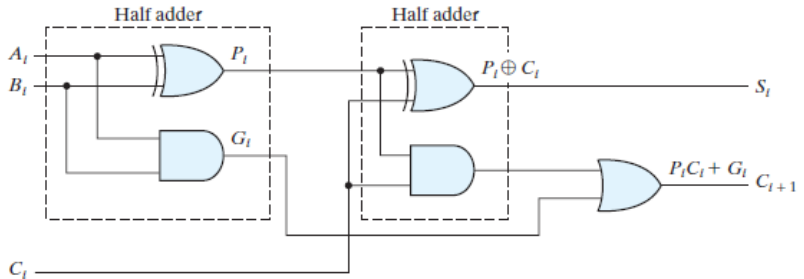
Then,

$$S_i = P_i \oplus C_i$$

$$C_{i+1} = G_i + P_iC_i$$

Carry Propagation

Carry lookahead logic



Full adder with P and G shown

Carry Propagation

$$C_0 = \text{input_carry}$$

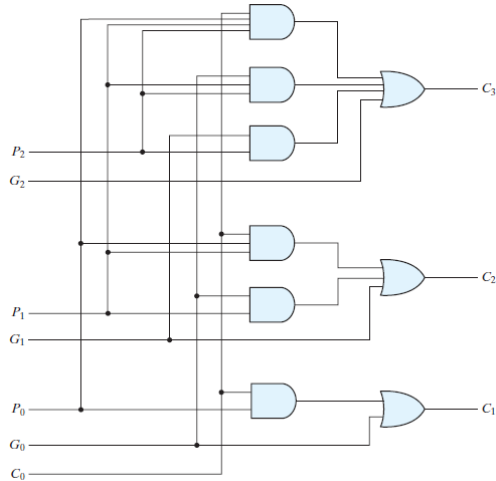
$$C_1 = G_0 + C_0P_0$$

$$\begin{aligned} C_2 &= G_1 + C_1P_1 = G_1 + (G_0 + C_0P_0)P_1 \\ &= G_1 + G_0P_1 + C_0P_0P_1 \end{aligned}$$

$$\begin{aligned} C_3 &= G_2 + C_2P_2 = G_2 + (G_1 + G_0P_1 + C_0P_0P_1)P_2 \\ &= G_2 + G_1P_2 + G_0P_1P_2 + C_0P_0P_1P_2 \end{aligned}$$

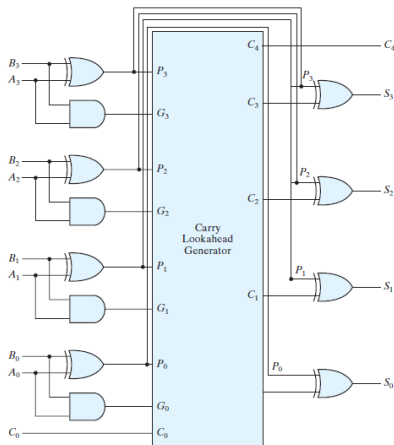
$$C_4 = G_3 + G_2P_3 + G_1P_2P_3 + G_0P_1P_2P_3 + C_0P_0P_1P_2P_3$$

Carry Propagation Generator



Logic diagram of carry lookahead generator

Four-bit adder with carry lookahead



Binary Subtractor

- ▶ The subtraction $A - B$ can be done by taking the 2's complement of B and adding it to A .

Binary Subtractor

- ▶ The subtraction $A - B$ can be done by taking the 2's complement of B and adding it to A .
- ▶ The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant pair of bits.

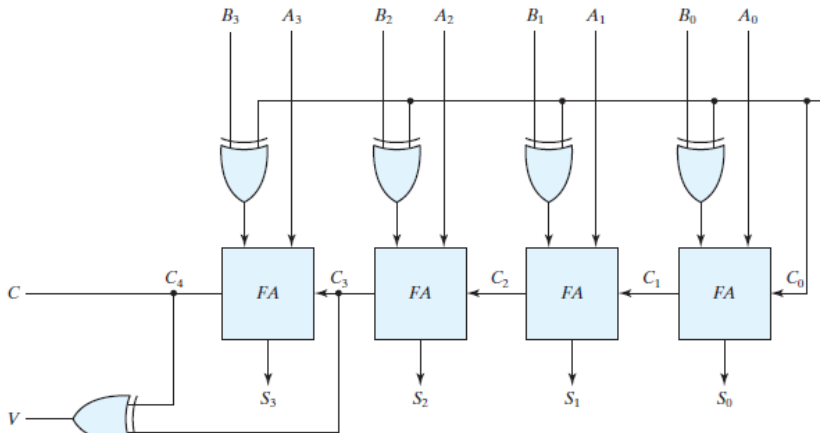
Binary Subtractor

- ▶ The subtraction $A - B$ can be done by taking the 2's complement of B and adding it to A .
- ▶ The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant pair of bits.
- ▶ The 1's complement can be implemented with inverters, and a 1 can be added to the sum through the input carry.

Binary Subtractor

- ▶ The subtraction $A - B$ can be done by taking the 2's complement of B and adding it to A .
- ▶ The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant pair of bits.
- ▶ The 1's complement can be implemented with inverters, and a 1 can be added to the sum through the input carry.
- ▶ The circuit for subtracting $A - B$ consists of an adder with inverters placed between each data input B and the corresponding input of the full adder.
- ▶ The input carry C_0 must be equal to 1 when subtraction is performed.

Four-bit adder-subtractor



Four-bit adder-subtractor (with overflow detection)

Digital System Design

Module 3 - COMBINATIONAL LOGIC

Dr. Deepthi Sasidharan

Assistant Professor, Department of Information Technology
GEC Barton Hill, Thiruvananthapuram

October 19, 2020

Magnitude Comparator

- ▶ The comparison of two numbers is an operation that determines whether one number is greater than, less than, or equal to the other number.
- ▶ A magnitude comparator is a combinational circuit that compares two numbers A and B and determines their relative magnitudes.
- ▶ The outcome of the comparison is specified by three binary variables that indicate whether
 - ▶ $A > B$
 - ▶ $A = B$
 - ▶ $A < B$

Magnitude Comparator

- ▶ Question: Compare two 2-bit binary numbers.
- ▶ Inputs: Numbers $A = (A_1A_2)$ and $B = (B_1B_2)$
- ▶ Outputs:
 - ▶ $G \rightarrow (A \text{ is greater than } B)$
 - ▶ $E \rightarrow (A \text{ is equal to } B)$
 - ▶ $L \rightarrow (A \text{ is less than } B)$

Magnitude Comparator

A_1	A_2	B_1	B_2	G	E	L
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	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	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

A_1A_2	B_1B_2			
	00	01	11	10
00				
01	1			
11	1	1		1
10	1	1		

$$G = A_1B'_1 + A_2B'_1B'_2 + A_1A_2B'_2$$

A_1A_2	B_1B_2			
	00	01	11	10
00	1			
01		1		
11			1	
10				1

$$E = A'_1A'_2B'_1B'_2 + A'_1A_2B'_1B_2 + A_1A_2B_1B_2 + A_1A'_2B_1B'_2$$

Similarly,

$$L = A'_1B_1 + A'_2B_1B_2 + A'_1A'_2B_2$$

Magnitude Comparator

$$\begin{aligned}E &= A_1' A_2' B_1' B_2' + A_1' A_2 B_1' B_2 + A_1 A_2 B_1 B_2 + A_1 A_2' B_1 B_2' \\&= A_1' B_1' (A_2' B_2' + A_2 B_2) + A_1 B_1 (A_2' B_2' + A_2 B_2) \\&= (A_1' B_1' + A_1 B_1) (A_2' B_2' + A_2 B_2)\end{aligned}$$

Magnitude Comparator

- ▶ For comparing two n -bit numbers has 2^{2n} entries in the truth table
- ▶ Digital functions that possess an inherent well-defined regularity can usually be designed by means of an algorithm

Magnitude Comparator

- ▶ Consider two numbers, A and B , with four digit each.
 - ▶ $A = A_3A_2A_1A_0$ and
 - ▶ $B = B_3B_2B_1B_0$
- ▶ The equality of the two numbers A and B is E
- ▶ The two numbers are equal if all pair of significant digits are equal

$$A_0 = B_0$$

$$A_1 = B_1$$

$$A_2 = B_2$$

$$A_3 = B_3$$

Magnitude Comparator

- ▶ The equality of the two numbers A and B is E
- ▶ The equality of each pair of bits can be expressed logically with an exclusive-NOR function

$$x_i = A_i B_i + A'_i B'_i$$

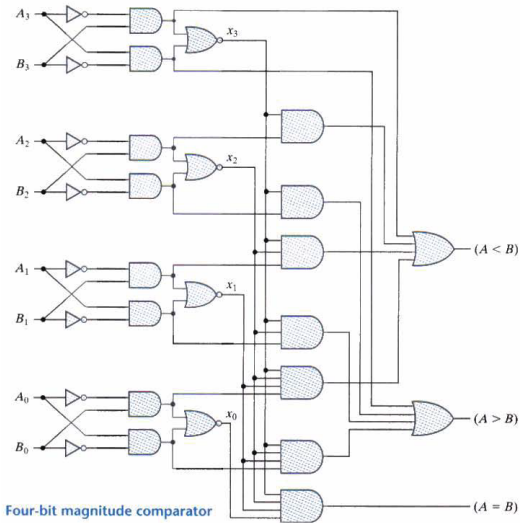
- ▶ where $x_i = 1$ only if the pair of bits in position i are equal

$$E = x_3 x_2 x_1 x_0$$

Magnitude Comparator

Then, to determine whether A is greater or less than B

- ▶ Inspect the relative magnitude of pairs of significant digits starting from MSB
- ▶ If the two digits of a pair are equal, we compare the next lower significant pair of digits
- ▶ The comparison continues until a pair of unequal digits is reached
- ▶ If the corresponding digit of A is 1 and that of B is 0, we conclude that $A > B$
- ▶ If the corresponding digit of A is 0 and that of B is 1, we have $A < B$



Digital System Design

Module 3 - COMBINATIONAL LOGIC

Dr. Deepthi Sasidharan

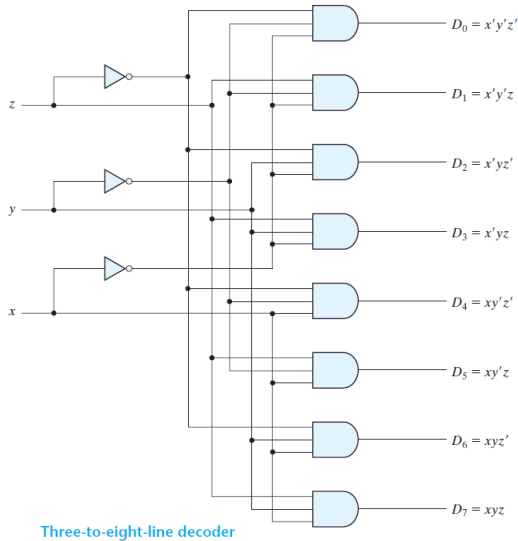
Assistant Professor, Department of Information Technology
GEC Barton Hill, Thiruvananthapuram

October 19, 2020

Decoder

- ▶ A *decoder* is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines
- ▶ If the n -bit coded information has unused combinations, the decoder may have fewer than 2^n outputs
- ▶ n – to – m -line decoders, where $m \leq 2^n$
- ▶ Each combination of inputs will assert a unique output

Decoder

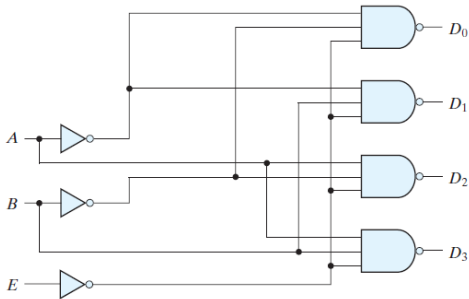


Decoder

Truth Table of a Three-to-Eight-Line Decoder

Inputs			Outputs							
<i>x</i>	<i>y</i>	<i>z</i>	<i>D</i> ₀	<i>D</i> ₁	<i>D</i> ₂	<i>D</i> ₃	<i>D</i> ₄	<i>D</i> ₅	<i>D</i> ₆	<i>D</i> 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



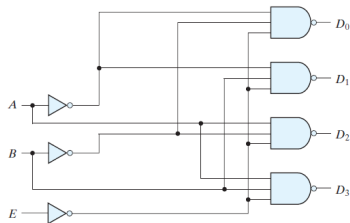
(a) Logic diagram

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	1

(b) Truth table

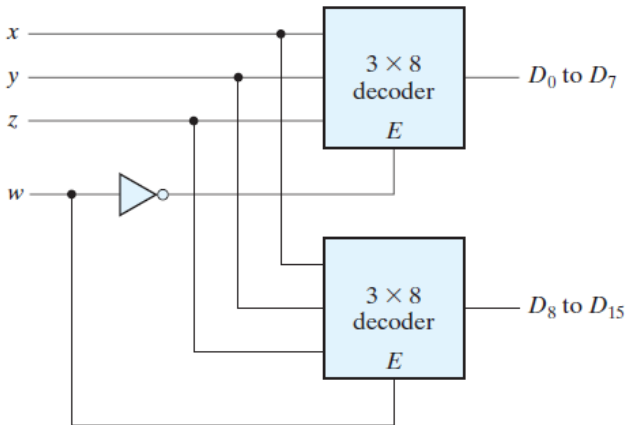
Two-to-four-line decoder with enable input

- ▶ A decoder with enable input can function as a **demultiplexer**— a circuit that receives information from a single line and directs it to one of 2^n possible output lines
- ▶ The selection of a specific output is controlled by the bit combination of n selection lines



A decoder – demultiplexer

Decoder



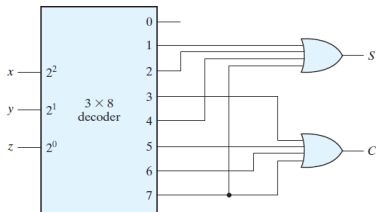
4 × 16 decoder constructed with two 3 × 8 decoders

Decoder

- ▶ Implementation of Combinational circuit
- ▶ Example: Implement a full adder using 3 X 8 decoder

$$S = \Sigma(1, 2, 3, 7)$$

$$C = \Sigma(3, 5, 6, 7)$$



Encoder

- ▶ An **encoder** is a digital circuit that performs the inverse operation of a decoder

Truth Table of an Octal-to-Binary Encoder

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Encoder

Boolean Output Functions:

$$x = D_1 + D_3 + D_5 + D_7$$

$$y = D_2 + D_3 + D_6 + D_7$$

$$z = D_4 + D_5 + D_6 + D_7$$

Priority Encoder

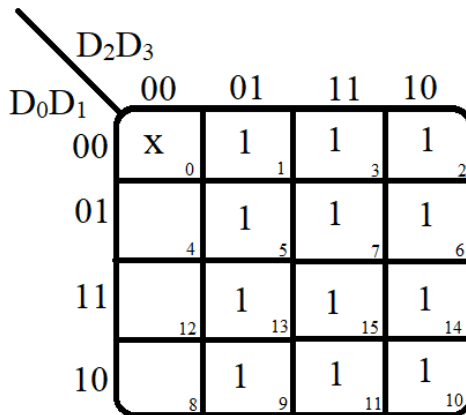
- ▶ A priority encoder is an encoder circuit that includes the priority function

Truth Table of a Priority Encoder

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Priority Encoder

Solving for x



		D_2D_3			
		00	01	11	10
D_0D_1	00	x 0	1 1	1 3	1 2
	01	4	1 5	1 7	1 6
	11	12	1 13	1 15	1 14
	10	8	1 9	1 11	1 10

Priority Encoder

Solving for y

D_2D_3

D_0D_1

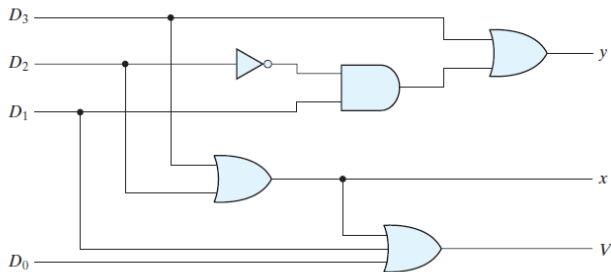
	00	01	11	10
00	X ₀	1 ₁	1 ₃	₂
01	1 ₄	1 ₅	1 ₇	₆
11	1 ₁₂	1 ₁₃	1 ₁₅	₁₄
10	₈	1 ₉	1 ₁₁	₁₀

Priority Encoder

$$x = D_2 + D_3$$

$$y = D_3 + D_1 D_2'$$

$$V = D_0 + D_1 + D_2 + D_3$$



Four-input priority encoder

Digital System Design

Module 3 - COMBINATIONAL LOGIC

Dr. Deepthi Sasidharan

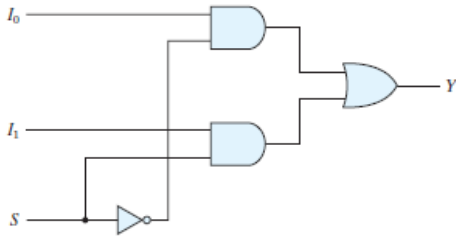
Assistant Professor, Department of Information Technology
GEC Barton Hill, Thiruvananthapuram

October 21, 2020

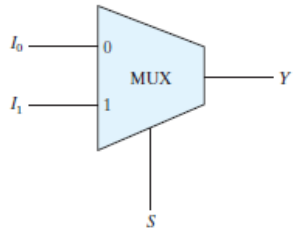
Multiplexer

- ▶ A *multiplexer* is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line
- ▶ The selection of a particular input line is controlled by a set of selection lines
- ▶ Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected
- ▶ A multiplexer is also called a **data selector**, since it selects one of many inputs and steers the binary information to the output line

Multiplexer



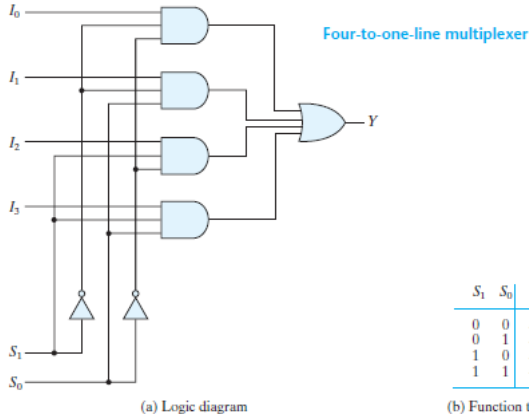
(a) Logic diagram



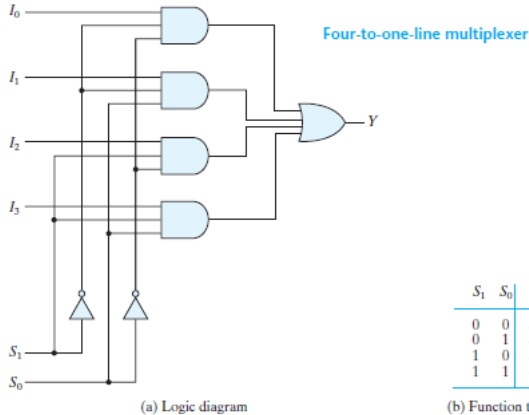
(b) Block diagram

Two-to-one-line multiplexer

Multiplexer



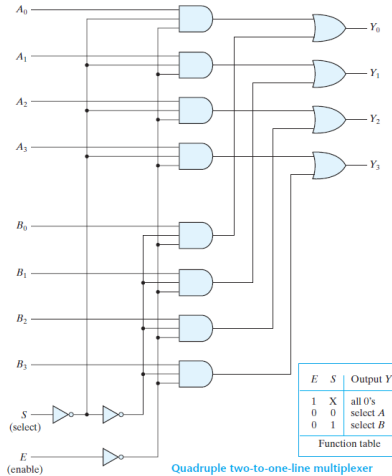
Multiplexer



Multiplexer

- ▶ As in decoders, multiplexers may have an enable input to control the operation of the unit
- ▶ When the enable input is in the inactive state, the outputs are disabled, and when it is in the active state, the circuit functions as a normal multiplexer
- ▶ Multiplexer circuits can be combined with common selection inputs to provide multiple-bit selection logic

Multiplexer



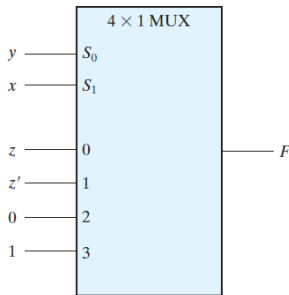
Multiplexer

Boolean Function Implementation:

Implement the Function $F(x, y, z) = \Sigma(1, 2, 6, 7)$

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



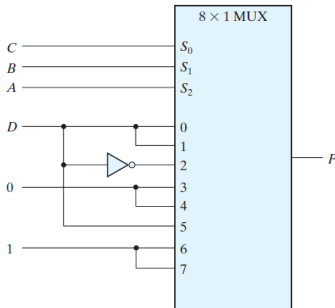
(b) Multiplexer implementation

Implementing a Boolean function with a multiplexer

Multiplexer

Boolean Function Implementation:
Implement the Function $F(A, B, C, D) = \Sigma(1, 3, 4, 11, 12, 13, 14, 15)$

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



Implementing a four-input function with a multiplexer

Parity Generator and Checker

- ▶ Electrical noise in the transmission of binary information can cause errors.
- ▶ To detect such error parity is used
- ▶ Parity systems
 - ▶ Odd parity
 - ▶ Even parity
- ▶ Adds a bit to the binary information

Example: Generate even parity for three bit of information.

- ▶ Inputs: x, y, z
- ▶ Output: P

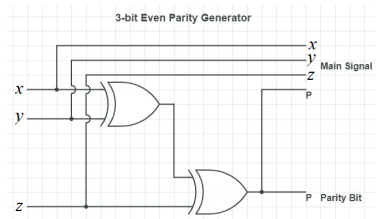
Parity Generator

x	y	z	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Truth Table

	yz	00	01	11	10
x	0		1		1
	1	1		1	

$$P = x \oplus y \oplus z$$



x	y	z	P	E
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
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	0

Parity Checker

		zP			
		00	01	11	10
xy	00	<div>0</div>	<div>1</div>	<div>3</div>	<div>2</div>
	01	<div>4</div>	<div>5</div>	<div>7</div>	<div>6</div>
	11	<div>12</div>	<div>13</div>	<div>15</div>	<div>14</div>
	10	<div>8</div>	<div>9</div>	<div>11</div>	<div>10</div>

$$E = x \oplus y \oplus z \oplus P$$

Module 3

- ▶ Combinational Circuits Analysis and Design Procedures
- ▶ Binary Adder-Subtractor (Half & Full)
- ▶ Carry look ahead adder
- ▶ BCD adder
- ▶ Code converter
- ▶ Magnitude Comparator
- ▶ Decoders Encoders
- ▶ Parity Generator
- ▶ Multiplexers DE multiplexers Implementation of Boolean functions using MUX.