LECTURE 6: DIGITAL LOGIC CIRCUIT DESIGN

EXCLUSIVE-OR FUNCTION

The exclusive-OR (XOR), denoted by the symbol ⊕, is a logical operation that performs the following Boolean operation:

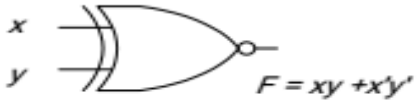
$$x \oplus y = xy' + x'y$$

x	y	x ⊕ y
0	0	0
0	1	1
1	0	1
1	1	0

XOR Truth Table

The exclusive-NOR, also known as equivalence function, performs the following Boolean operation:

$$x \odot y = xy + x'y'$$



x	y	x ⊙ y
0	0	1
0	1	0
1	0	0
1	1	1

XNOR Truth Table

The following identities apply to the XOR operation:

$$x \oplus 0 = x$$
$$x \oplus 1 = x'$$
$$x \oplus x = 0$$
$$x \oplus x' = 1$$
$$x \oplus y' = x' \oplus y = (x \oplus y)'$$
$$x \oplus y = y \oplus x$$
$$(x \oplus y) \oplus z = x \oplus (y \oplus z)$$

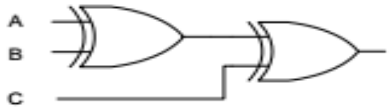
The exclusive-OR operation with three and four variables can be expressed as

$$A \oplus B \oplus C = \sum(1,2,4,7)$$
$$A \oplus B \oplus C \oplus D = \sum(1,2,4,7,8,11,13,14)$$

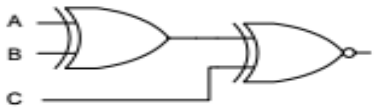
We can see from the above expressions that the XOR function is 1 only when an odd number of variables are equal to1. Hence, in general, the multi-variable XOR operation is defined as the odd function.

The 3-input odd function is implemented by means of 2-input XOR gates as shown below. The complement of an odd function is obtained by replacing the output gate with an XNOR gate.

LECTURE 6: DIGITAL LOGIC CIRCUIT DESIGN



3-input Odd function



3-input even function

Parity Generation and Checking

Exclusive-OR gates are useful for generating and checking a parity bit that is used for detecting/correcting errors during transmission of binary data over communication channels.

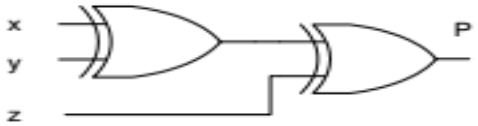
Example

Transmitting a 3-bit message with even parity bit. The three bits – x, y, and z constitute the message and are the inputs to the circuit. The parity bit *P* is the output, which is an odd function and can be expressed as:

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

The truth table and the logic diagram for the parity generator is shown below.

Message			***Parity***
x	*y*	*z*	
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

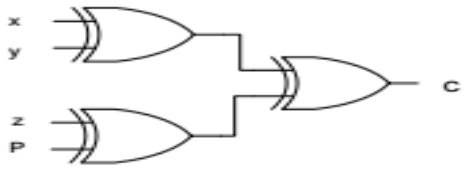


3-bit even parity generator

The three bits in the message together with the even parity pit P are transmitted. The receiver at the destination checks for even number of 1's in the 4-bit message and generates an error C equal to 1 if the number of 1's in the message is odd. Here, again, we can use the odd function property of the XOR gate that produces an output of 1 if odd number of inputs is equal to 1. The truth table and the logic diagram for the parity checker is shown below.

LECTURE 6: DIGITAL LOGIC CIRCUIT DESIGN

4-bit Received Message				Error
x	y	z	P	C
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



4-bit even parity checker

Hardware Description Language (HDL)

Def. HDL is defined as a language that describes the hardware of digital systems in a textual form.

HDL is used to represent:

- ⇒ Logic diagrams
- ⇒ Boolean expressions
- ⇒ More complex digital circuits

There are two applications of HDL processing:

1. Logic simulation

   Can be defined as, the representation of the structure and behaviour of a digital logic system through the use of a computer.

2. Logic synthesis

   Is defined as, the process of deriving a list of components and their interconnections (a netlist) from the model of a digital system described in HDL.

There are many proprietary HDLs in industry, two types are supported by IEEE, these are VHDL and Verilog HDL. The second type is the one adopted and used in our textbook. It is supported by a consortium of companies and Universities known as Open Verilog International (OVI).

LECTURE 6: DIGITAL LOGIC CIRCUIT DESIGN

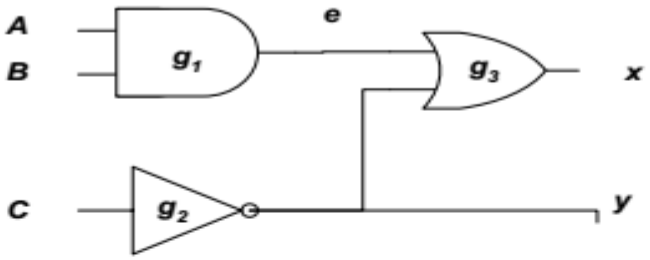
Module Representation

Verilog uses about 100 predefined keywords. e.g. module, endmodule, input, output, wire, …etc.

Text between // and the end of a line is interpreted as a comment.

The module is the building block in Verilog HDL. It is declared by the keyword module and is terminated by the keyword endmodule.

Example of a simple circuit defined in Verilog HDL will be given next.




```
// Description of simple circuit
module smpl_circuit(|A,B,C,x,y);
        input A,B,C;
        output x,y;
        wire e;
        and g1(e,A,B);
        not g2 (y,C);
        or   g3 (x,e,y);
endmodule
```

Gate Delay

Delay in Verilog HDL is specified in terms of time units and the symbol **#.**

LECTURE 6: DIGITAL LOGIC CIRCUIT DESIGN

Association of a time unit and physical time is made using the 'timescale compiler directive.

Example: → 'timescale 1 ns/100 ps

The first number specifies the unit of measurements and the second number specifies the precision. 1 ns time unit is the default if the time scale is not specified.

The previous example may be rewritten with gate time delays specified as shown below"

```
// Description of simple circuit with delay
module circuit_with_delay(|A,B,C,x,y);
    input A,B,C;
    output x,y;
    wire e;
    and #(3) g1(e,A,B);
    not #(20) g2 (y,C);
    or #(10)   g3 (x,e,y);
endmodule
```

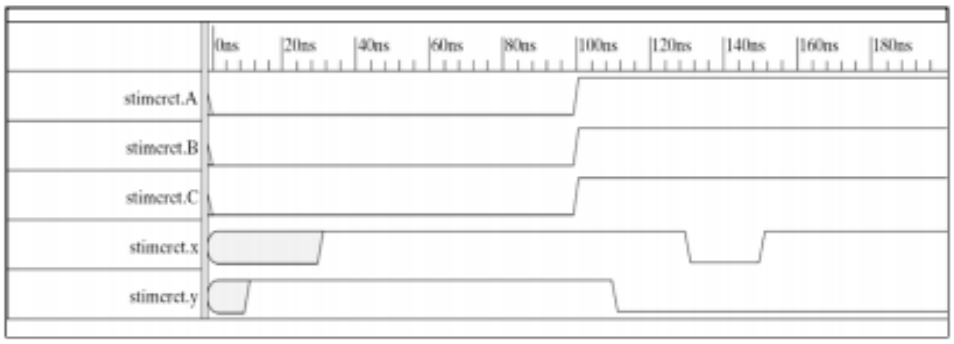
Outputs of Gates after Delay

Time Units	Inputs			Outputs		
ns	**A**	**B**	**C**	**y**	**e**	**x**
-	**0**	**0**	**0**	**1**	**0**	**1**
-	**1**	**1**	**1**	**1**	**0**	**1**
10	**1**	**1**	**1**	**0**	**0**	**1**
20	**1**	**1**	**1**	**0**	**0**	**1**
30	**1**	**1**	**1**	**0**	**1**	**0**
40	**1**	**1**	**1**	**0**	**1**	**0**
50	**1**	**1**	**1**	**0**	**1**	**1**

Circuit simulation in HDL requires the application of inputs to the circuit to generate outputs. This is done through the test bench. An example of a test bench for simulating the circuit with delay is given here:

```
// Stimulus for simple circuit with delay
module stimcrct;
reg A,B,C;
wire x,y;
circuit_with_delay cwd (A,B,C,x,y);
initial
    begin
        A = 1'b0; B = 1'b0; C = 1'b0;
        #100
        A = 1'b0; B = 1'b0; C = 1'b0;
        #100
end
endmodule

// Description of simple circuit with delay
module circuit_with_delay(|A,B,C,x,y);
    input A,B,C;
    output x,y;
    wire e;
    and #(30) g1(e,A,B);
    not #(20) g2 (y,C);
    or  #(10)  g3 (x,e,y);
endmodule
```



LECTURE 6: DIGITAL LOGIC CIRCUIT DESIGN

Combinational Circuits

Logic circuits are either combinational or sequential. Combinational logic circuits consists of logic gates whose outputs at any time are determined from the present combination of the inputs. Sequential circuits consist of memory elements and logic gates.

A block Diagram of Combinational Circuit

Analysis Procedure

1. Obtain Boolean expression from logic diagram

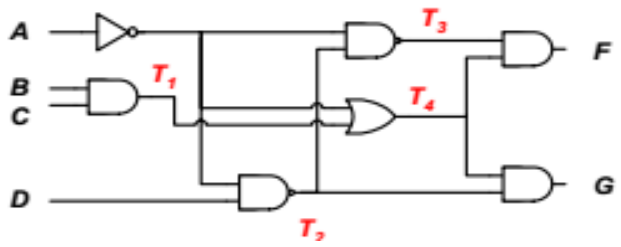
    a. Label all gate outputs that are a function of input variables. Obtain Boolean function for each gate.
    b. Label all gate outputs that are a function of input variables and previously labeled gates. Obtain Boolean function for each of these gates.
    c. Repeat step (b) until the outputs of the circuit are obtained.

2. Obtain the truth table from the logic diagram

    a. Prepare the truth table for n input variables and 2n input combinations.
    b. Label all gate outputs that are a function of input variables. Fill in the truth table for these outputs.
    c. Label all gate outputs that are functions of input variables and previously labeled gates. Fill in the truth table columns for these outputs.
    d. Repeat step (c) until the columns for all the outputs are obtained.

LECTURE 6: DIGITAL LOGIC CIRCUIT DESIGN

Example 1: **Determine the Boolean functions for the outputs F and G as a function of the four inputs A, B, C, and D.**



$$T_1 = BC$$

$$T_2 = (A'D)' = A + D'$$

$$T_3 = (A'T_2)' = A + T_2' = A + A'D = A + D$$

$$T_4 = A' + T_1 = A' + BC$$

$$F = T_3T_4 = (A + D)(A' + BC) = AA' + ABC + A'D + BCD$$
$$= ABC + A'D + BCD = ABC + A'D$$

$$G = T_2T_4 = (A + D')(A' + BC) = AA' + ABC + A'D' + BCD'$$
$$= ABC + A'D' + BCD' = ABC + A'D'$$

Example 2: **Analyze the previous logic circuit by establishing the truth table for F and G.**

Inputs								Outputs	
A	*B*	C	*D*	T_1	T_2	*T3*	T_4	*F*	*G*
0	0	0	0	0	1	0	1	0	1
0	0	0	1	0	0	1	1	1	0
0	0	1	0	0	1	0	1	0	1
0	0	1	1	0	0	1	1	1	0
0	1	0	0	0	1	0	1	0	1
0	1	0	1	0	0	1	1	1	0
0	1	1	0	1	1	0	1	0	1
0	1	1	1	1	0	1	1	1	0
1	0	0	0	0	1	1	0	0	0
1	0	0	1	0	1	1	0	0	0
1	0	1	0	0	1	1	0	0	0
1	0	1	1	0	1	1	0	0	0
1	1	0	0	0	1	1	0	0	0
1	1	0	1	0	1	1	0	0	0
1	1	1	0	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1

LECTURE 6: DIGITAL LOGIC CIRCUIT DESIGN

Design Procedure

1. Describe the problem (i.e., the problem statement).
2. Determine the available number of input variables and required output variables.
3. Assign letter symbols to the input and output variables.
4. Derive the truth table that defines the required relationships between inputs and outputs.
5. Obtain the simplified Boolean function for each output.
6. Draw the logic diagram.

Design of a Code Converter

Design a combinational logic circuit that will convert from a BCD code to Excess-3 code.

Design → 4 inputs → A, B, C, and C

4 outputs → w, x, y, and z

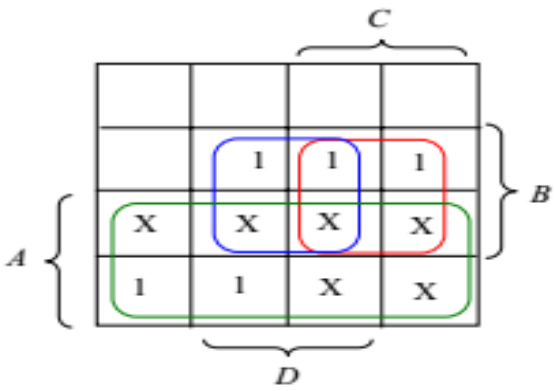
Truth Table:

Inputs (BCD Code)				Outputs(Excess-3)			
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

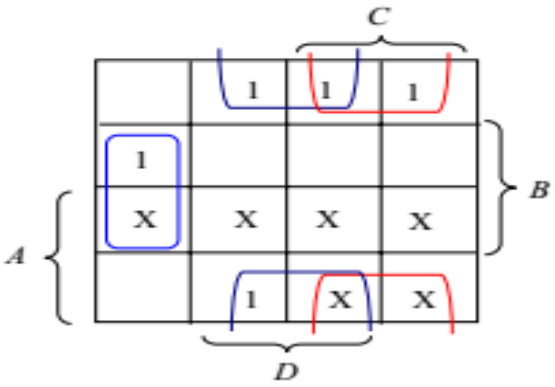
The six unused combinations are considered don't care conditions. These correspond to 10, 11, 12, 13, 14, and 15.

Simplification of the output functions is made using Karnaugh maps and making use of the don't care conditions.

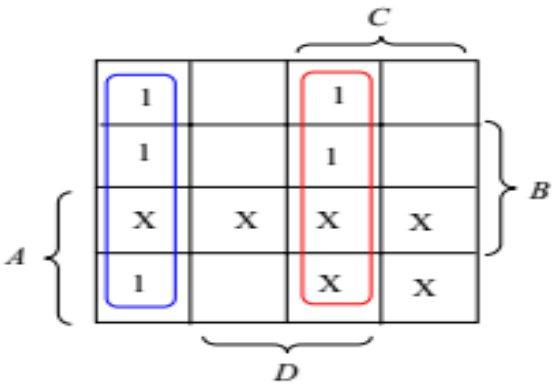
LECTURE 6: DIGITAL LOGIC CIRCUIT DESIGN



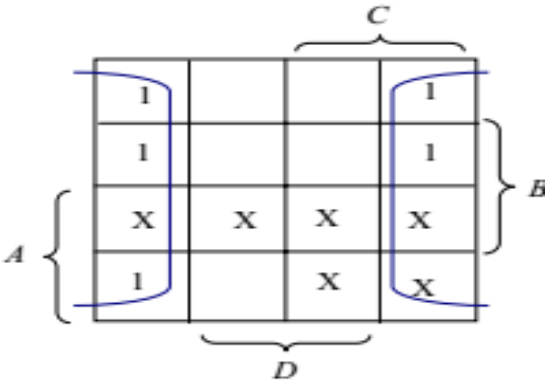
$w = A + BC + BD$



$x = B'C + B'D + BC'D'$



$y = CD + C'D'$



$z = D'$

A two level logic diagram can be obtained directly from the logic expressions. Also the logic expressions may be arranged such that:

$z = D'$ $\quad y = CD + (C + D)'$ $\quad x = B'(C + D) + B(C + D)'$

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