KAUNAS UNIVERSITY OF TECHNOLOGY

THE FIRST PRPICNICPLE OF DIGITAL LOGIC

LAB 1: Digital Feedbackless Circuits

Student: Zalal Youssef
Group: IFU-1
Lecturer: Kazimeras Bagdonas
Date: 05 March 2022

Table of Contents

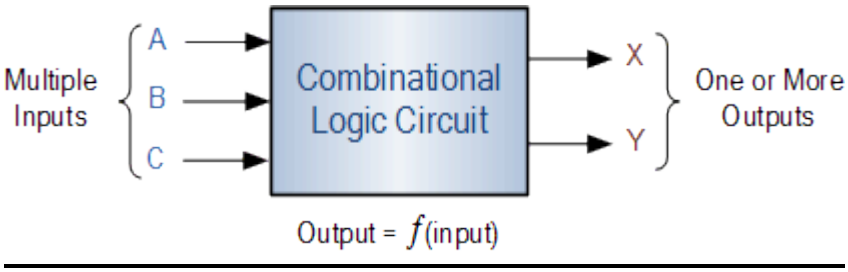
Introductory Theory:

Unlike Sequential Logic Circuits whose outputs are dependent on both their present inputs and their previous output state giving them some form of memory. The outputs of **Combinational Logic Circuits** are only determined by the logical function of their current input state, logic "0" or logic "1", at any given instant in time.

The result is that combinational logic circuits have no feedback, and any changes to the signals being applied to their inputs will immediately have an effect at the output. In other words, in a **Combinational Logic Circuit**, the output is dependent at all times on the combination of its inputs. Thus a combinational circuit is memory-less.

So if one of its inputs condition changes state, from 0-1 or 1-0, so too will the resulting output as by default combinational logic circuits have "no memory", "timing" or "feedback loops" within their design.

Combinational Logic: Fig1



Combinational Logic Circuits are made up from basic logic NAND, NOR or NOT gates that are "combined" or connected together to produce more complicated switching circuits. These logic gates are the building blocks of combinational logic circuits. An example of a combinational circuit is a decoder, which converts the binary code data present at its input into a number of different output lines, one at a time producing an equivalent decimal code at its output.

Combinational logic circuits can be very simple or very complicated and any combinational circuit can be implemented with only NAND and NOR gates as these are classed as "universal" gates.

The three main ways of specifying the function of a combinational logic circuit are:

- **Boolean Algebra**. This forms the algebraic expression showing the operation of the logic circuit for each input variable either True or False that results in a logic "1" output.
- **Truth Table** – A truth table defines the function of a logic gate by providing a concise list that shows all the output states in tabular form for each possible combination of input variable that the gate could encounter.
- **Logic Diagram** – This is a graphical representation of a logic circuit that shows the wiring and connections of each individual logic gate, represented by a specific graphical symbol, that implements the logic circuit.

And all three of these logic circuit representations are shown below. Fig2

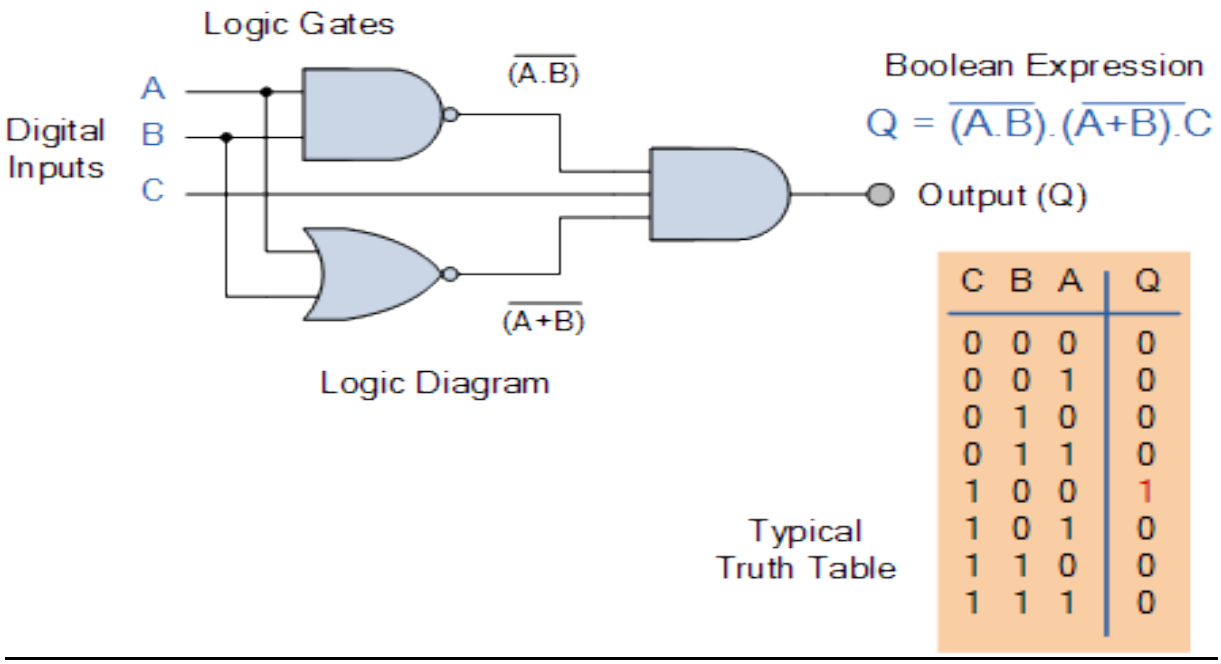


Fig2.

As combinational logic circuits are made up from individual logic gates only, they can also be considered as "decision making circuits" and combinational logic is about combining logic gates together to process two or more signals in order to produce at least one output signal according to the logical function of each logic gate. Common combinational circuits made up from individual logic gates that carry out a desired application.

Implementation:

Design process for a combinational logic circuit.

- Statement of the problem
- Identification of input and output variables
- Expressing the relationship between the input and output variables.
- Construction of a truth table to meet the input-output requirements.
- Writing Boolean expressions for various output variables in terms of input variables. Here either the Sum-of-Products approach or the Product-of-Sums approach may be used to come up with the Boolean expression.
- Simplification of the resultant Boolean expression. The number of gates and the number of input terminals for the gates required for the realization of a logical expression, in general, get reduced considerably if the expression can be simplified. Thus, simplification of logical expressions is very important as it saves the hardware required to design a specific system. Different techniques may be used to simplify a Boolean expression such as:

    1. Using Boolean algebra theorems
    2. Using Karnaugh maps

- Implementation of the simplified Boolean expression using logic gates

Assignment Number #333: 0,3,4,8,10,11,20,21,22,27,33,34,37,38,44,46,50,62

Truth table, where a boolean function with n variables is described with all possible 2 n combinations of these variables;

A function is given: y = f(x0, x1, x2, x3, x4, x5). Let us write the truth table for this function (**Fig3**)

	x5	x4	x3	x2	x1	x0
	32	16	8	4	2	1
	5	4	3	2	1	0
0	0	0	0	0	0	0
3	0	0	0	0	1	1
4	0	0	0	1	0	0
8	0	0	1	0	0	0
10	0	0	1	0	1	0
11	0	0	1	0	1	1
20	0	1	0	1	0	0
21	0	1	0	1	0	1
22	0	1	0	1	1	0
27	0	1	1	0	1	1
33	1	0	0	0	0	1
34	1	0	0	0	1	0
37	1	0	0	1	0	1
38	1	0	0	1	1	0
44	1	0	1	1	0	0
46	1	0	1	1	1	0
50	1	1	0	0	1	0
62	1	1	1	1	1	0

Fig3

Karnaugh map: Creation of the map and filling it with function values. Fig4

		x2x1x0						
	000'	001'	011'	010'	110'	111'	101'	100'
000'	1		1					1
001'	1		1	1				
011'			1					
010'					1		1	1
110'				1				
111'					1			
101'					1			1
100'		1		1	1		1	

(Fig4)

Once the Karnaugh map has been constructed and the adjacent 1s linked by rectangular and square boxes, the algebraic min-terms can be found by examining which variables stay the same within each box.

Min-terms: Fig5

!x5!x4!x3!x1!x0
!x5!x4x3!x2!x0
!x5!x4!x2x1x0
!x5x3!x2x1x0
!x5x4!x3x2!x0
!x5x4!x3x2!x1
x5!x3!x2x1!x0
x5!x4!x3!x1x0
x5!x4x3x2!x0
x5x3x2x1!x0
x5!x4x2x1!x0

(Fig5)

It is also possible to derive this simplification by carefully applying the axioms of Boolean algebra. See Fig 6 & 7 below:

	x5	x4	x3	x2	x1	x0
1	0	0	0		0	0
2	0	0	1	0		0
3	0	0		0	1	1
4	0		1	0	1	1
5	0	1	0	1		0
6	0	1	0	1	0	
7	1		0	0	1	0
8	1	0	0		0	1
9	1	0	1	1		0
10	1		1	1	1	0
11	1	0		1	1	0

(Fig6)

!x5(!x4!x3!x1!x0+!x4x3!x2!x0+!x4!x2x1x0+x3!x2x1x0

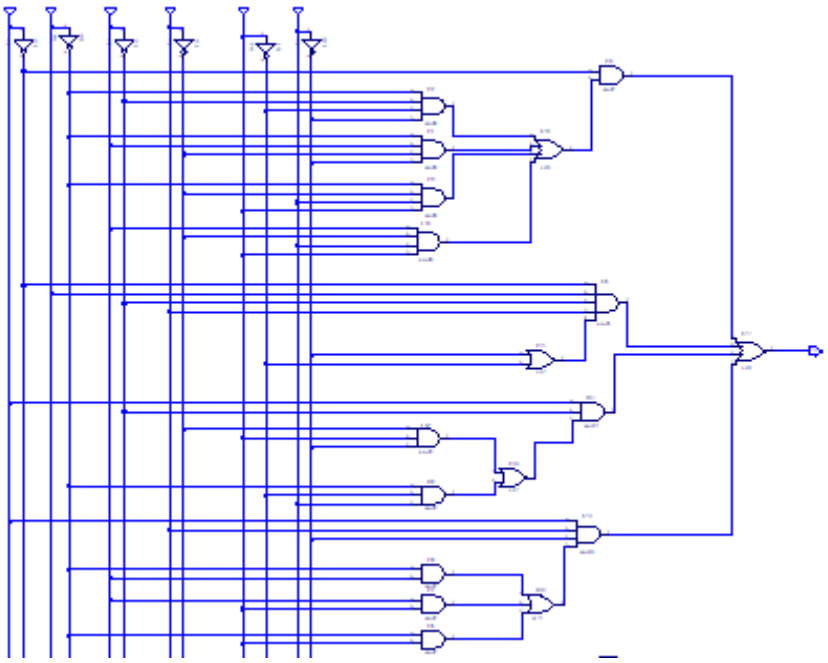
!x5x4!x3x2(!x0+!x1)

x5!x3(!x2x1!x0+!x4!x1x0)

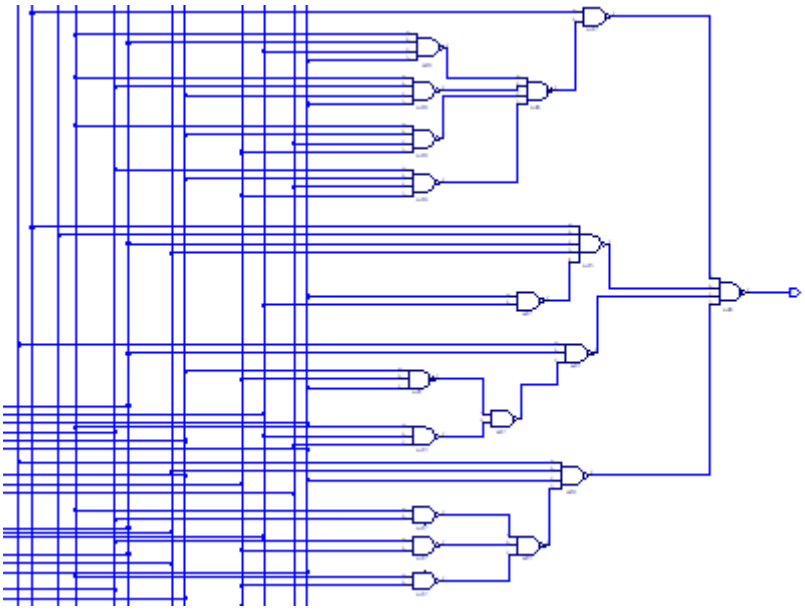
x5x2!x0(!x4x3+x3x1+!x4x1)

(Fig7)

1. **And now implementing the simplified expression using AND, OR, NOT gates only. We get:**



2. **Using only NAND Gates:**



3. **Using a multiplexer mux41.**

Implementing a schematic with a multiplexer requires:

• extract the address variables from the disjunctive normal form. The number of variables depends on the multiplexer itself. In this example, we will use x1 and x2 signals as address variables;

• By extracting these variables, we get:

		00	01	11	10
	F=00	x1x0			
00	x3x2	1		1	
01		1			
11					
10		1		1	1

F-00=!x3!x1!x0+x3!x2!x0+!x2x1x0

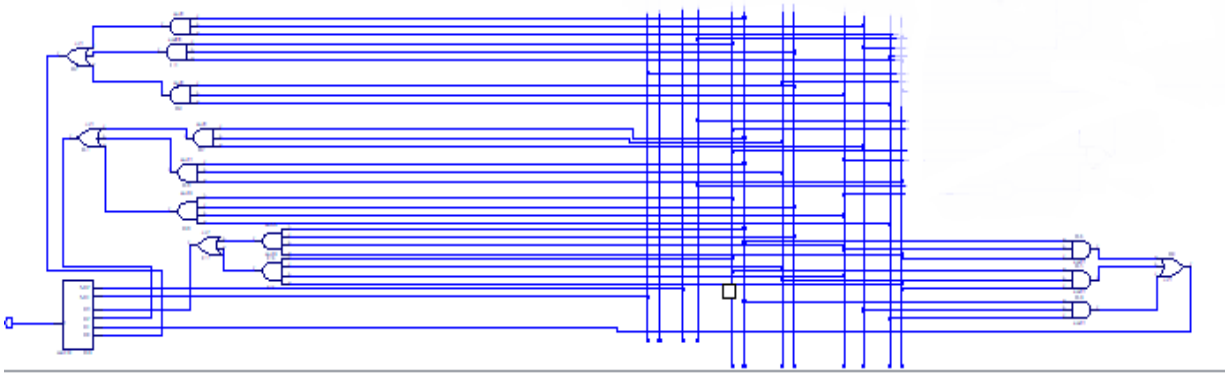
		00	01	11	10
	F=01	x1x0			
00	x3x2				
01		1	1		1
11					
10				1	

F-01=!x3x2!x1+!x3x2!x0+x3!x2x1x0

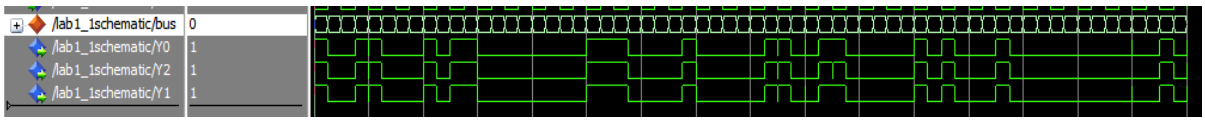
		00	01	11	10
	F=11	x1x0			
00	x3x2				1
01					
11					1
10					
		F-11=!x3!x2x1!x0+x3x2x1!x0			

		00	01	11	10
	F=10	x1x0			
00	x3x2		1		1
01			1		1
11		1			1
10					
		F-10=!x3x1!x0+x3x2!x0+!x3!x1x0			

The implementation of the function with a multiplexer is shown in **fig A**



Result:



Our 5 variables x0,x1,x2,x3,x4 and x5 are the inputs, while Y0, Y1 and Y2 are the outputs.
All three results are the same, which shows the sequesnce of numbers of the assignment and our problem.

Conclusion :
To conclude, there are many types and methods of minimization which can be applied in order to reduce the function by getting min-terms and without affecting the final result. It depends on the experiment, which minimization method is easier to use and reduce. There are different types of minimization such as Tabular method minimization of Boolean functions.