ECE 321L – Introduction to Software Engineering
Lab assignment

1 Introduction

Logic gates are the heart of digital electronics. A gate is an electronic device which is used to compute a function on a two valued signal. Logic gates are the basic building block of digital circuits. We can connect any number of logic gates to design a required digital circuit. Practically, we implement the large number of logic gates in *Integrated Circuits (ICs)*, by which we perform complicated operations at high speeds. By combining logic gates, we can design many specific circuits like flip flops, latches, multiplexers, shift registers etc.

1.1 Boolean Expressions and Truth Tables

In basic algebra, every function has its own equation. Similarly in the field of digital electronics, every gate logic function has its own equation called a *Boolean expression.* The nineteenth century British mathematician, George Boole, invented a type of algebra that uses only two conditions or states. The two states are true and false. This type of algebra using only two states is called *Boolean algebra* in honor of Boole.

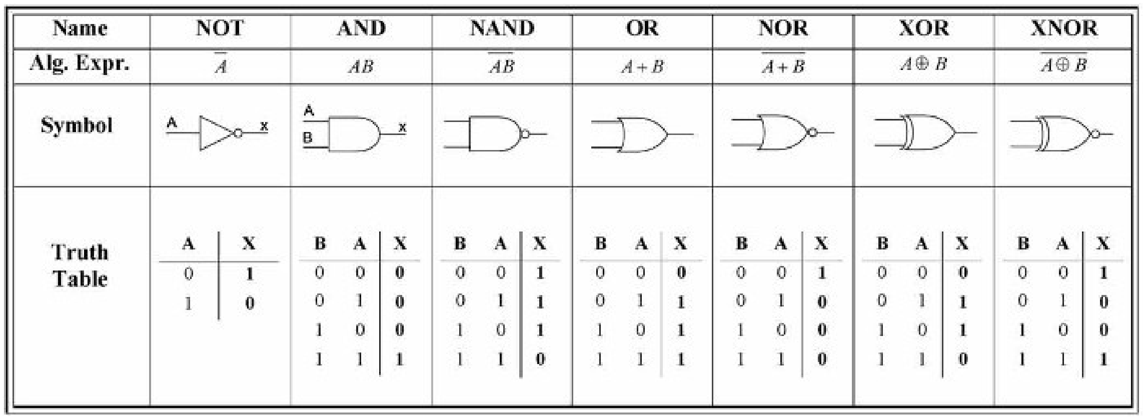
In Boolean algebra, the *true state* is represented by the number one, called *logic high* or *logic one.* The false state is represented by the number zero, called *logic low* or *logic zero.* In the field of digital electronics, logic high is represented by the presence of a voltage potential. Logic low is represented by the absence of a voltage potential. The logic high in a program mable logic controller is represented with five volts ($+5V$), and the logic low is represented with zero volts ($0V$).

By applying conventional algebra, you can plot a function's input and output points to create the characteristic of the function as a graph. This graph represents the function pictorially in an x–y coordinate system. The x-axis is for the input points and the y-axis is for the output points. In Boolean algebra, a table contains the digital input and output points. This table is called a *truth table.*

1.2 Combinational and Sequential Logic Gate Circuits

Combinational logic gates do not require clock pulses to operate. Their outputs depend only on their inputs. This means that the outputs of combinational logic gates are generated instantaneously. Generally, the combinational logic gates are simply called logic gates. Seven logic gates exist: NOT, AND, OR, NAND, NOR, XOR (exclusive OR), and XNOR (exclusive NOR). The gates in a circuit represent a simple Boolean expression. For example, two-input AND gates with inputs A and B and output X graphically represent the expression $X = A \cdot B$. Figure 1 presents the output (X) of all the seven gates for two inputs (A, B). By combining the aforementioned gates, more complicated circuits can be developed. Since the output depends only on the input, the final values depends on the combination of the logical gates. Figure 2 displays a logic gate circuit which shows the connection of logic gates for a Boolean expression.

Sequential logic devices have outputs that depend on their inputs as well as time. They require clock pulses. Therefore, an inherent delay time is always present for the sequential logic circuits. Flipflop devices such as reset-set (RS), JK, delay (D), and toggle (T) are sequential logic devices. Figure 3 displays a sequential logic circuit.



Name	NOT	AND	NAND	OR	NOR	XOR	XNOR
Alg. Expr.	$\overline{A}$	AB	$\overline{AB}$	$A+B$	$\overline{A+B}$	$A\oplus B$	$\overline{A\oplus B}$
Symbol	A x	A B x					
Truth Table	A X 0 1 1 0	B A X 0 0 0 0 1 0 1 0 0 1 1 1	B A X 0 0 1 0 1 1 1 0 1 1 1 0	B A X 0 0 0 0 1 1 1 0 1 1 1 1	B A X 0 0 1 0 1 0 1 0 0 1 1 0	B A X 0 0 0 0 1 1 1 0 1 1 1 0	B A X 0 0 1 0 1 0 1 0 0 1 1 1

Figure 1: Truth tables for seven gates.

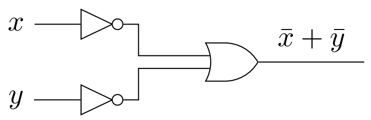


Figure 2: A two-input logic gate circuit

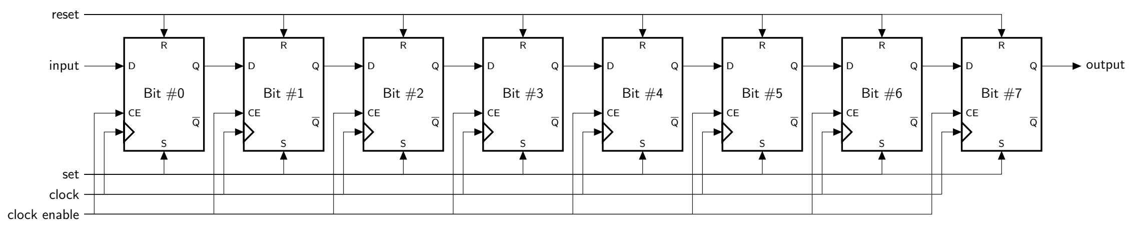


Figure 3: Sequential logic circuit. These circuits require clock pulses.

1.3 Logic Gate Circuits

By combining the aforementioned logic gates, we can design more complex circuits. Figure 4 displays a Boolean expression and its truth table. Note that the upper bar symbol indicates the inverse value of the input. Table 1 presents the truth table of the Boolean expression $Y = A \cdot B + \overline{A} \cdot C$.

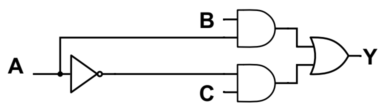


Figure 4: Logic circuit for Boolean expression $Y = A \cdot B + \overline{A} \cdot C$

2 Lab assignment

The focus of this lab series is to create a logic circuit simulator in `C` programming language. Specifically, there will be multiple milestones and students have to present the correct functionality of their implementation

Table 1: Truth table of the Boolean expression $Y = A \cdot B + \overline{A} \cdot C$

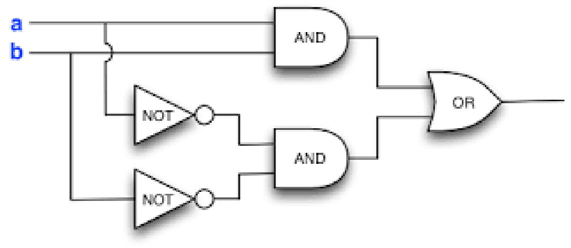
A	**B**	**C**	**Y**
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

2.1 Milestone 1: Logic gates library

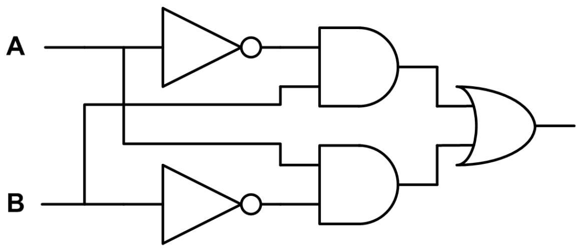
The *first step* of this milestone is to create a file that contains the implementations in `C` programming language of the seven logical gates. Specifically, each gate, must be represented as a separate function with two integer arguments (except for the NOT gate that will have one argument). The return value should be the output of the gate. The body of the functions must be located in a separate file named `gates.c` and the corresponding prototypes in the file named `gates.h`.

The *second step* of this milestone is to write a program in a file named `logic_simulator.c` that utilizes the already developed logic gates. The program should ask from the user which gate to simulate and then ask for the appropriate input. After that, the program should print the output of the logical gate and verify it by printing the truth table as well. Each gate can be indexed according to Figure 1. For example, when the program asks for the gate to simulate, the user can enter 1 for NOT, 2 for AND, 3 for NAND etc. Last, the program should be continuous until the user types 'Q'.

The *third step* of this milestone is to combine the developed logical gates in order to create more complicated circuits. Write a program in a file named `logic_simulator_circuit.c` which simulates the two circuits presented in Figure 5. The program should ask form the user the values for the input for both circuits and then print the final output. Extra points if the calculation is performed without extra variables (i.e. only use variables for the input and the final output).



(a) Circuit 1



(b) Circuit 2

Figure 5: Custom circuits