Electronic Engineering Part 1 Laboratory Experiment

Digital Circuit Design 1
Combinational Logic

(3 hours)

1. Introduction

These days most signal processing is done digitally. Electronic signals (representing physical quantities such sound, brightness, position, speed) are stored and processed as numbers. Although numbers could be processed in decimal form this is inconvenient because ten different voltage or current levels would be required for each digit. Instead binary codes are used; these use binary digits (bits) which have two values: 0 or 1.

Binary values 0 and 1 are represented in digital circuits by distinct voltage levels. Normally low voltages represent 0, and high voltages represent 1. The exact voltage ranges used depend on the electronic logic family being used, and different logic families are not normally compatible.

2. Boolean Algebra

Boolean Algebra is an algebra of 2-state quantities, and is there appropriate for describing operations on binary digits. In Boolean algebra there are three basic operators: NOT, AND and OR.

The NOT function operates on a single Boolean variable. The symbol for NOT is a line over the variable:

$$\text{NOT } X = \overline{X}$$

The effect of the NOT operator is defined by the truth table:

X	$\overline{X}$
0	1
1	0

A truth table can be used to define any function of any number of Boolean variables. There is a row for every combination of values of the variables, and in every row the value of the function is specified. A truth table for n variables has 2^n rows.
The AND and OR functions operate between 2 or more variables and are represented by . and + respectively:

$$X \text{ AND } Y = X.Y$$
$$X \text{ OR } Y = X + Y$$

The effect of the AND and OR operators for 2 variables is defined by the truth table:

X	Y	$X.Y$	$X+Y$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

In other words $X.Y$ is 1 if X=1 **and** Y=1, and is 0 otherwise. $X+Y$ is 1 if X=1 **or** Y=1, and is 0 otherwise.

The effect of the AND and OR operators for 3 variables is defined by the truth table:

X	Y	Z	$X.Y.Z$	$X+Y+Z$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Any Boolean function can be expressed using these three operators. For example:

$$F = Z.(X + Y)$$

Truth tables and Boolean expressions are equivalent ways of defining Boolean functions. For example the Boolean function F of the four variables A, B, C and D might be defined by the expression:

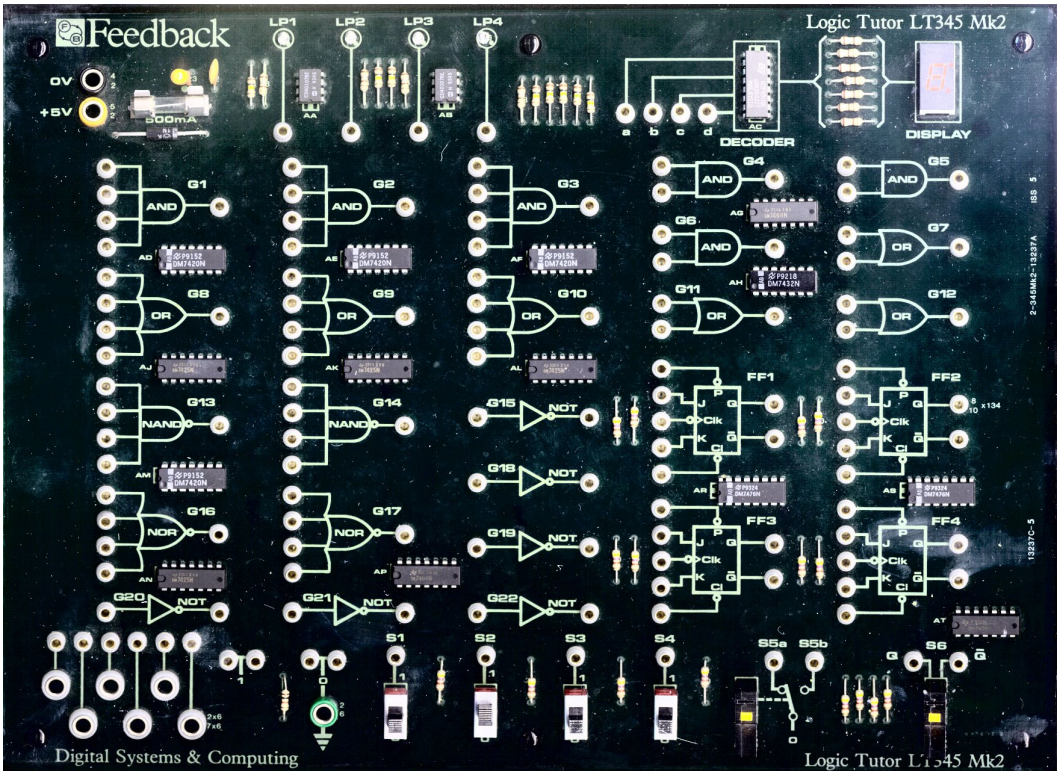
$$F = \overline{A}.(B + C + \overline{B}.\overline{C}) + A.B.\overline{(C + D)}$$

The equivalent truth table is:

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

3. Logic Tutor

You will be doing experiments on logic gates using a logic tutor. This is a board on which there is a selection of electronic logic gates, flip-flops and input and output devices. Patch wires are provided to make interconnections between different devices.



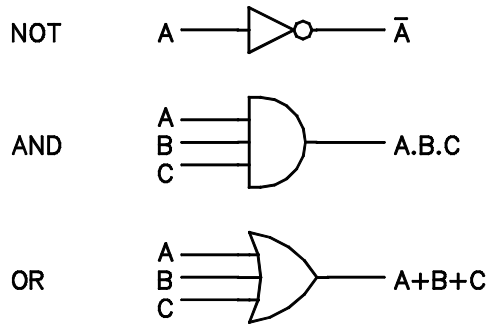
Before the board can be used power must be provided. Connect 0V and +5V from a power supply to the connectors at the top-left of the board; the 7-segment indicator at the top-right should now display 0.

The primary input devices are the switches S1 to S4. These generate a 0 or 1 depending on the position of the switch slider. The primary output devices are the light-emitting diodes (leds) LP1 to LP4. These light up if their input is a logical 1, otherwise they remain off.

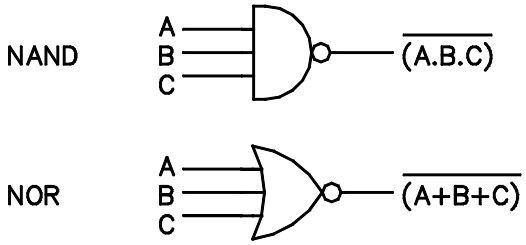
Connect S1 to LP1 and confirm that the led indicator responds to the logic value from the switch.

4. Gates

Digital (binary) signals are processed by gates. Gates have one output terminal, and one or more input terminals.



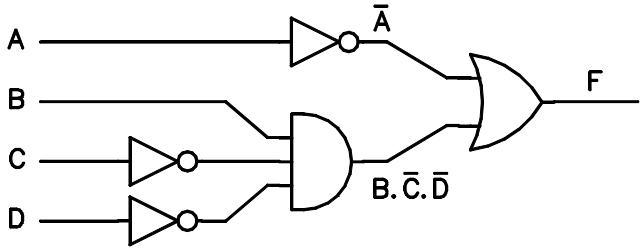
In addition to the NOT, AND and OR gates, there are two other important gate types: NAND and NOR. These are universal gates and any Boolean function can be generated using only NAND gates or only NOR gates.



Gates can be interconnected to generate more complex functions. For example the function:

$$F = \overline{A} + B.\overline{C}.\overline{D}$$

can be generated by the digital circuit:



Digital circuits which do not have feedback generate a Boolean function of the input variables and do not have the property of memory. Such digital circuits are termed combinational. Digital circuits with feedback can have memory, and are termed sequential.

4.1. The NOT Gate

Connect S1 to the input of NOT gate G15, and connect the output of the gate to LP1. Draw a truth table for the gate.

It is bad practice to leave any gate input unconnected. For many logic families the logic value at an unconnected input is undefined, and may change unpredictably; certainly it is wrong to assume that an unconnected input will assume logic 0.

The logic family used in the logic tutor is TTL (transistor-transistor logic), which was once the most popular logic family but is now obsolete and not used in new designs. Disconnect the NOT gate input and observe the output value. Deduce the logic value at the unconnected input.

4.2. The AND Gate

Connect S1-S4 to the inputs of AND gate G2, and connect the output of the gate to LP1. Draw a truth table for this gate by selecting all 16 combinations of input values and observing the output. Is it in accordance with the definition of the AND function give above?

4.3. The OR Gate

Draw a truth table for the OR gate G9. Is it in accordance with the definition of the OR function give above?

4.4. The NAND Gate

NAND gates are termed universal because any Boolean Function can be implemented using only gates of this type.

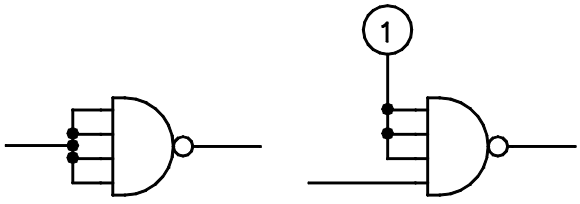
Draw a truth table for NAND gate G14.

To show that this type of gate is universal is simply necessary to prove that it can be used to perform the operations NOT, AND and OR. Since any Boolean function can be expressed using these operators it follows that NAND gate alone can implement any Boolean function.

There are two ways of connecting a NAND gate to perform the NOT operation.

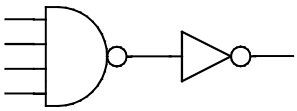
(a) Apply the input signal to all the gate inputs connected together.

(b) Apply the input signal to any gate input and connect all the other gate inputs to logic 1.

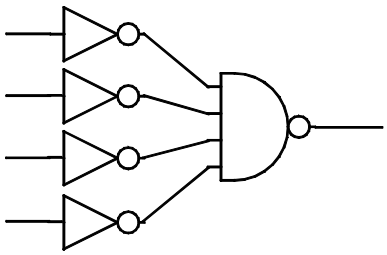


Logic 1 signals are available from the bottom-left of the logic tutor. Confirm that both of these methods implement the NOT operation.

Connect the output of the NAND gate to the input of a NOT gate, and draw a truth table for the combination. You should find that the truth table corresponds to the AND operation.



Now remove the output NOT, and connect four NOT gates in the input lines of the NAND gate. Draw a truth table for this combination. You should find that the truth table corresponds to the OR operation.



This proves that the NAND gate is indeed universal.

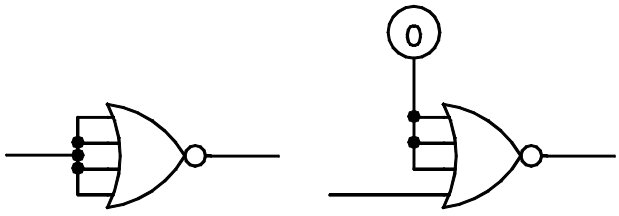
4.5. The NOR Gate

NOR gates are also universal. Draw a truth table for NOR gate G17.

There are two ways of connecting a NOR gate to perform the NOT operation.

(a) Apply the input signal to all the gate inputs connected together.

(b) Apply the input signal to any gate input and connect all the other gate inputs to logic 0.



Confirm that both of these methods implement the NOT operation.

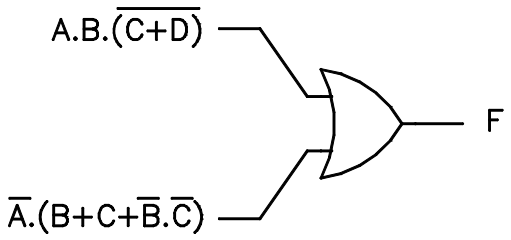
Devise combinations of a NOR gate and NOT gates to perform the operations of AND and OR. Draw truth tables to confirm that your circuits operate as expected.

5. Combinational Logic Systems

Devise a system of logic gates to generate the Boolean function F of the four variables A, B, C and D:

$$F = \overline{A}.(B + C + \overline{B}.\overline{C}) + A.B.\overline{(C + D)}$$

Start by implementing the overall OR function using gate g12:



Then use appropriate gates to generate the rest of the Boolean expression. If you require a 3-input AND or NAND gate, use a 4-input gate and connect the unused input to logic 1; if you require a 3-input OR or NOR gate, use a 4-input gate and connect the unused input to logic 0.

Connect the inputs A, B, C and D to the switches S1-S4 and draw a truth table for this circuit. It should correspond to the truth table for this function shown in the section on Boolean Algebra above.

This Boolean function can be simplified (either by a graphical method based on K-maps or by using the Quine-McCluskey method). Its simplified form is:

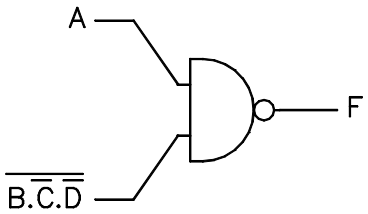
$$F = \overline{A} + B.\overline{C}.\overline{D}$$

Implement this expression on the logic tutor. When you have built the circuit draw a truth table and confirm that it does indeed generate the same function as the circuit for the more complex expression.

This function can be manipulated algebraically into a form suitable for implementing using universal gates. For NAND gates:

$$F = \overline{(A.\overline{(B.\overline{C}.\overline{D})})}$$

Start by implementing the overall NAND function:



Then use appropriate NAND gates to generate the rest of the Boolean expression. You will be unable to implement the NOT functions using NAND gates because of the limited number of NAND gates on the logic tutor. Instead use the NOT gates provided. When you have built the circuit draw a truth table and confirm that it generates the correct function.

6. The Exclusive-OR Function

A function that is commonly required in digital circuits is the exclusive-OR (XOR) function. This has the symbol $\oplus$:

$$X \ \ X\text{OR } Y = X \oplus Y$$

Exclusive OR is used, for example, in binary adders, parity generators and Gray-to-binary code converters. For two variables XOR is similar to OR, except when both variables are 1 then the result is 0:

X	Y	$X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

The effect of the XOR operator for 3 variables is shown in the truth table:

X	Y	Z	$X \oplus Y \oplus Z$
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

The result of XOR-ing any number of variables is 1 if an odd number of the variables is 1; otherwise the result is 0.

Devise a combination of NOR, AND and OR gates to generate the XOR function between two variables. Construct the circuit and draw a truth table. Confirm that the truth table corresponds with that given above.

The XOR function between any number variables can be generated using 2-input XOR gates. For example a 3-input XOR function can be generated using two 2-input XORs:

$$X \oplus Y \oplus Z = (X \oplus Y) \oplus Z$$

Construct a second 2-input XOR circuit, and connect this with the original 2-input XOR to create a 3-input XOR function. Draw a truth table. Confirm that the truth table corresponds with that of the 3-input XOR function given above.