October 27, 2024 Math 8

Handout 7: Logic gates and circuits

Basic logic operations

- NOT (for example, NOT A): true if A is false, and false if A is true. Commonly denoted by $\neg A$, $\sim A$, or (in computer science) $!A$.
- OR (for example A OR B): true if at least one of A, B is true, and false otherwise. Sometimes also called "inclusive or" to distinguish it from the "exclusive or" described in problem 4 below. Commonly denoted by $A \vee B$.
- AND (for example A AND B): true if both A, B are true, and false otherwise (i.e., if at least one of them is false). Commonly denoted by $A \wedge B$.

Basic derived logic operations

- NOR (NOT OR, for example NOT (A OR B), A NOR $B \Leftrightarrow \neg(A \vee B)$): true if both A and B are false, and false if either A or B is true.
- NAND (NOT AND, for example NOT (A AND B), A NAND $B \Leftrightarrow \neg(A \wedge B)$): true if at least one of A, B is false, and false otherwise (i.e., if both A, B are true).
- XOR (exclusive OR gate, for example A XOR $B \Leftrightarrow (\neg A \wedge B) \vee (A \wedge \neg B)$): true if and only if exactly one of A, B is true and false otherwise.
- XNOR (equivalence gate, NOT exclusive OR, for example A XNOR $B \Leftrightarrow (\neg A \vee B) \wedge (A \vee \neg B)$): true if and only if both A, B are either true or false, and false otherwise, if A and B are different.

Recap: selected Logic Laws

- Double negation:

$$\neg(\neg A) \Leftrightarrow A$$

- Idempotency:

$$A \vee A \Leftrightarrow A$$

$$A \wedge A \Leftrightarrow A$$

- De Morgan (disjunction and conjunction negation):

$$\neg(A \vee B) \Leftrightarrow \neg A \wedge \neg B$$

$$\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B$$

- Distributive:

$$A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$$

$$A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$$

Exercise 1. Write the truth tables for NOR, NAND, XOR, and XNOR operations.

Exercise 2. Using truth tables show that

- A XOR $B \Leftrightarrow (\neg A \wedge B) \vee (A \wedge \neg B)$
- A XNOR $B \Leftrightarrow (\neg A \vee B) \wedge (A \vee \neg B)$

Exercise 3. Using (i) truth tables and (ii) logic laws show that

- A XOR $B \Leftrightarrow (A \vee B) \wedge (\neg A \vee \neg B)$
- A XNOR $B \Leftrightarrow (A \wedge B) \vee (\neg A \wedge \neg B)$

Exercise 4. Using logic laws, express in the simplest possible way using only basic operators, ¬, ∨, ∧ ,

- (A XOR B) XOR C
- (A XNOR B) XNOR C

Exercise 5. Using (i) truth tables and (ii) logic laws show that XOR and XNOR are associative,

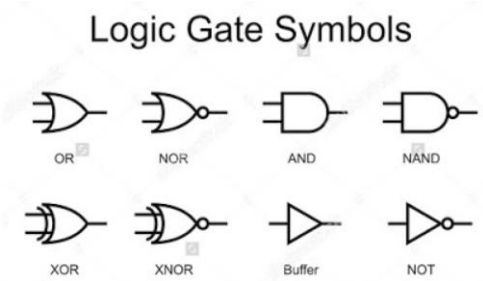
- (A XOR B) XOR $C \Leftrightarrow A$ XOR (B XOR C)
- (A XNOR B) XNOR $C \Leftrightarrow A$ XNOR (B XNOR C)

Logic gates

In electronics, it is common to have digital signals which take two values: LOW and HIGH (e.g., in certain microcontrollers, voltage can range between 0–5V, and anything over 2.5V is considered HIGH, and anything below, LOW). These two values can also be considered as two binary digits: LOW= 0, HIGH= 1, or as Boolean values: LOW= 0 =false, HIGH= 1 =true.

The basis of all modern computers are "logic gates", chips that take two (or more) such inputs and produce an output described by some truth table. (These chips contain transistors and diodes, but this is irrelevant for us here). For example, below is a typical such chip, containing four AND gates, each with two inputs:

In electronics, there are standard notations for different kinds of gates (AND, OR, NAND...):

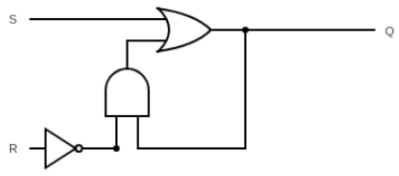


Combining such simple gates, one can create more complicated ones — and use that to create circuits which take as input a collection of binary digits and produce as output some function such as sum or product of inputs (interpreting n binary inputs as an n-digit binary number).

There is a number of online simulators that allow you to create and test such circuits virtually; in particular, we can use https://logic.ly/demo/ (Demo version is enough for exercises).

Homework problems

1. Similarly to what we did with NAND, show that any logic operation can be expressed using NOR, which is defined as: A NOR B = NOT (A OR B). Use the online circuit simulator to show that your formulas indeed work.
2. **Half adder**: Adding two one-digit binary numbers is an operation which has two inputs, A and B, and produces two outputs, X_0, X_1, which are just two digits of the sum $A + B$ written in binary. For example, $1_b + 1_b = 10_b$ (we use subscript b to indicate that this is a binary number, not a decimal one). Thus, when $A = 1, B = 1$, we have X_0=0 (the last digit of the sum) and $X_1 = 1$.
    a. Write a table of values for this operation, listing for each possible combination of A, B the values of outputs X_0, X_1.
    b. Can you get X_0 from A, B by using one of the standard logic gates listed above?
    c. Same question for X_1?
    d. Construct a circuit with two inputs and two outputs, consisting of the basic logic gates, which implements this addition operation. [This is commonly known as half-adder.] Test your circuit in http://logic.ly.
3. **Full adder**: Can you construct a circuit which does binary addition of three inputs A, B, C (each input being a 1-digit binary number)? This circuit is known as full adder. [Hint: use the half-adder you constructed in the previous problem as one of the building blocks: you can use it to first add A and B. You only need one half-adder!] Test your final circuit in http://logic.ly.
4. **Ripple-carry adder**: Using the full adder constructed in the previous problem as a building block, construct a circuit which adds two 3-digit binary numbers. Your circuit must have six inputs A_0, A_1, A_2, B_0, B_1, B_2 (representing digits of A, B respectively) and four outputs X_0, X_1, X_2, X_4 representing digits of the sum. This is known as ripple-carry adder (carry because it implements usual addition with carry digits; word "ripple" can be ignored for now).
5. Consider the circuit shown below.



Can you make a table of values, describing for each combination of values of the inputs S, R the value of the output? [Hint: there is a catch there. . .]. Test it on http://logic.ly.