Maggie Johnson

Handout #10
CS103A

Logic Gates

Key Topics

* Logic Gates
* Half Adders
* Redundancy of Logic Gates
* Minterm Expansions and Circuit Diagrams

As mentioned earlier, computers are built of circuits that have only two states: on/off or 1/0. Therefore, many of the functions performed by a computer are just complex Boolean functions of many variables. In most cases, these complex functions can be built up from simpler functions; in fact, there are only a handful of very simple one or two variable Boolean functions that provide all the required building blocks for the complex functions that are the basis of a computer. The seven most important functions, or **logic gates** are defined below:

Name	Verbal Condition for z = 1	Truth table	Formula
AND	x = 1 and y = 1	x y z 0 0 0 0 1 0 1 0 0 1 1 1	z = xy
OR	x = 1 or y = 1	x y z 0 0 0 0 1 1 1 0 1 1 1 1	z = x + y
NOT (inverter)	x = 1 is not true	x z 0 1 1 0	z = x'
NAND	(x=1 and y = 1) is not true	x y z 0 0 1 0 1 1 1 0 1 1 1 0	z = (xy)' z= x | y
NOR	(x=1 or y = 1) is not true	x y z 0 0 1 0 1 0 1 0 0 1 1 0	z=(x + y)' z= x ↓ y
XOR (exclusive or)	(x=1 or y = 1) but not both	x y z 0 0 0 0 1 1 1 0 1 1 1 0	z = x ⊕ y z=x'y+xy'
XNOR y)' (exclusive nor)	(x=1 or y = 1 but not both) is not true	x y z 0 0 1 0 1 0 1 0 0 1 1 1	z=(x ⊕ z=xy+x'y'

These seven gates are all easily fabricated, readily available electronic devices, They are most often seen in highly miniaturized integrated circuits (ICs). The NAND gate is the most common gate.

Half Adders

One of the simplest and most important arithmetic operations which can be done by a computer is binary addition with carry, The simplest case of this operation is adding together two one-bit numbers x_1 and x_2. There are four possibilities:

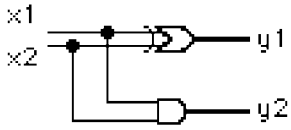
	0		0		1		1
	0		1		0		1
0	0	0	1	0	1	1	0

The truth table for the sum values y_1 and y_2 is given below:

x_1	x_2	y_1
0	0	0
0	1	1
1	0	1
1	1	0

x_1	x_2	y_2
0	0	0
0	1	0
1	0	0
1	1	1

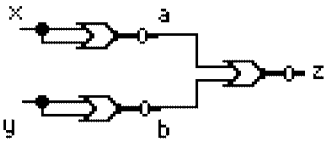
We can build a half adder as follows based on these truth tables:

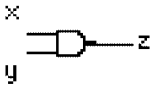


This shows that the truth tables for y_1 and y_2 are just XOR and AND functions. Incidentally, it's called a half adder because it does not consider a carry from a previous addition.

Redundancy of Logic Gates

The two circuits below are equivalent:





The logic diagram on the left represents:

a = x NOR x b = y NOR y z = a NOR b

The truth tables for a and b are as follows:

x	a
0	1
1	0

y	b
0	1
1	0

The truth table for z:

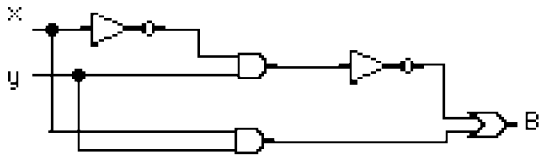
x	y	(ab)	z
0	0	11	0
0	1	10	0
1	0	01	0
1	1	00	1

Thus, an AND gate can be represented with 3 NOR gates. This should not be a surprise since you know the set {• + '}

is functionally complete. There are lots of other examples of redundancy between the seven gates: NAND gate using NOT and AND; NOR using OR and NOT; NOT using one NAND. Any Boolean function can be represented with AND, OR and NOT, but sometimes we can build a function with less gates if we use the other ones. We also have shown that NAND is functionally complete, which is an explanation for why this gate is so commonly used.

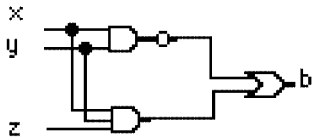
Minterm Expansions and Circuit Diagrams

We have been discussing how Boolean functions can be represented using circuit diagrams. It naturally follows that Boolean expressions can also. For example, the following diagram

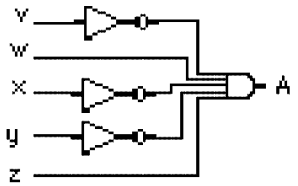


represents the function B = (x'y)' + (xy).

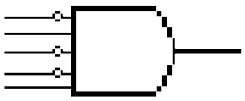
A diagram for b = (xy)' + xyz:



Notice that gates can have more than two inputs. In fact, a diagram for A = v'wx'y'z:



can be written using a shorthand:



This circuit accepts only one input: 01001. This means the circuit outputs a 1 for only the specified combination of inputs. So, this circuit is an **acceptor** for 01001. We can also build **rejector** circuits (output a 0 for some combination of inputs). The following circuit is a rejector for 01011.

Notice the symmetry between acceptor and rejector circuits.

Returning to the half adder discussed earlier, we built that circuit with an XOR and AND gate because we recognized

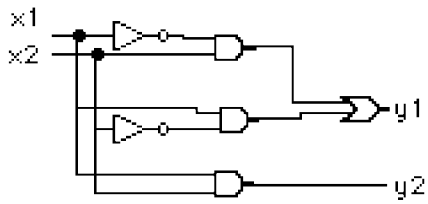
the truth tables. An equivalent circuit uses the minterm expansion from the Boolean functions for y1 and y2:

x_1	x_2	y_1
0	0	0
0	1	1
1	0	1
1	1	0

x_1	x_2	y_2
0	0	0
0	1	0
1	0	0
1	1	1

$y_1 = x_1'x_2 + x_1x_2'$

$y_2 = x_1x_2$



This illustrates an important point, which we will discuss at length next time. There may be several ways to build a circuit. We want to find the simplest representation of a Boolean expression; simplest meaning least number of terms. This way, we will build a circuit using the least number of gates, which is the least expensive.

Historical Notes

Claude Shannon (born 1916) was the first to observe that Boolean algebra can be used to describe the behavior of circuits:

C. Shannon, "Symbolic Analysis of Relay and Switching Circuits," Transactions of AIEE, 57 (1938), 713-723.