

# 1 Basic Digital Concepts

By converting continuous analog signals into a finite number of discrete states, a process called *digitization*, then to the extent that the states are sufficiently well separated so that noise does not create errors, the resulting digital signals allow the following (slightly idealized):

- storage over arbitrary periods of time
- flawless retrieval and reproduction of the stored information
- flawless transmission of the information

Some information is intrinsically digital, so it is natural to process and manipulate it using purely digital techniques. Examples are numbers and words.

The drawback to digitization is that a single analog signal (*e.g.* a voltage which is a function of time, like a stereo signal) needs many discrete states, or *bits*, in order to give a satisfactory reproduction. For example, it requires a minimum of 10 bits to determine a voltage at any given time to an accuracy of  $\approx 0.1\%$ . For transmission, one now requires 10 lines instead of the one original analog line.

The explosion in digital techniques and technology has been made possible by the incredible increase in the density of digital circuitry, its robust performance, its relatively low cost, and its speed. The requirement of using many bits in reproduction is no longer an issue: The more the better.

This circuitry is based upon the transistor, which can be operated as a switch with two states. Hence, the digital information is intrinsically *binary*. So in practice, the terms digital and binary are used interchangeably. In the following sections we summarize some conventions for defining the binary states and for doing binary arithmetic.

## 1.1 Binary Logic States

The following table attempts to make correspondences between conventions for defining binary logic states. In the case of the TTL logic gates we will be using in the lab, the Low voltage state is roughly 0–1 Volt and the High state is roughly 2.5–5 Volts. See page 475 of the text for the exact conventions for TTL as well as other hardware gate technologies.

Boolean Logic	Boolean Algebra	Voltage State (positive true)	Voltage State (negative true)
True (T)	1	High (H)	Low (L)
False (F)	0	L	H

The convention for naming these states is illustrated in Fig. 1. The “positive true” case is illustrated. The relationship between the logic state and label (in this case “switch open”) at some point in the circuit can be summarized with the following:

*The labelled voltage is High (Low) when the label’s stated function is True (False).*

In the figure, the stated function is certainly true (switch open), and this does correspond to a high voltage at the labelled point. (Recall that with the switch open, Ohm’s Law implies that with zero current, the voltage difference across the “pull up” resistor is zero, so that

the labelled point is at +5 Volts. With a closed switch, the labelled point is connected to ground, with a 5 Volt drop across the resistor and a current of  $I = V/R = 5 \text{ mA}$  through it.)

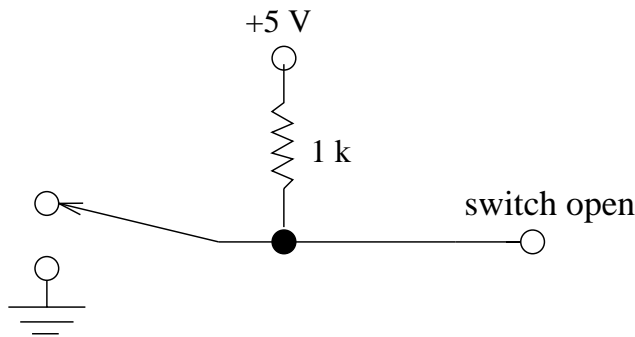


Figure 1: Illustration for labelling logic states (“positive true”).

With the convention known as “negative true”, the label would be changed to “switch closed” with a bar over it:  $\overline{\text{switch closed}}$ . Our statement becomes:

*The labelled voltage is Low (High) when the label’s stated function is True (False).*

So in the figure, the stated function (switch closed) is true when the voltage is low. The bar is meant to invoke the boolean inversion operation:  $\overline{\overline{T}} = T$ ,  $\overline{\overline{F}} = F$ ,  $\overline{\overline{T}} = T$ , and so forth.

## 1.2 Binary Arithmetic

Each digit in binary is a 0 or a 1 and is called a *bit*, which is an abbreviation of *binary digit*. There are several common conventions for representation of numbers in binary.

The most familiar is *unsigned binary*. An example of a 8-bit number in this case is

$$01001111_2 = 0 \times 2^7 + 1 \times 2^6 + \dots + 1 \times 2^0 = 64 + 8 + 4 + 2 + 1 = 79_{10}$$

(Generally the subscripts will be omitted, since it will be clear from the context.) To convert from base 10 to binary, one can use a decomposition like above, or use the following algorithm illustrated by 79:  $79/2 = 39$ , remainder 1, then  $39/2 = 19$  r 1, and so forth. Then assemble all the remainders in reverse order.

The largest number which can be represented by  $n$  bits is  $2^n - 1$ . For example, with 4 bits the largest number is  $1111_2 = 15$ .

The most significant bit (MSB) is the bit representing the highest power of 2, and the LSB represents the lowest power of 2.

Arithmetic with unsigned binary is analogous to decimal. For example 1-bit addition and multiplication are as follows:  $0 + 0 = 0$ ,  $0 + 1 = 1$ ,  $1 + 1 = 0$ ,  $0 \times 0 = 0$ ,  $0 \times 1 = 0$ , and  $1 \times 1 = 1$ . Note that this is *different from Boolean algebra*, as we shall see shortly, where  $1 + 1 = 1$ .

Another convention is called *BCD* (“binary coded decimal”). In this case each decimal digit is separately converted to binary. Therefore, since  $7 = 0111_2$  and  $9 = 1001_2$ , then  $79 = 01111001$  (BCD). Note that this is *different* than our previous result. We will use BCD quite often in this course. It is quite convenient, for example, when decimal numerical displays are used.

Yet another convention is *Gray code*. You have a homework problem to practice this. This is less commonly used.

### 1.2.1 Representation of Negative Numbers

There are two commonly used conventions for representing negative numbers.

With *sign magnitude*, the MSB is used to flag a negative number. So for example with 4-bit numbers we would have  $0011 = 3$  and  $1011 = -3$ . This is simple to see, but is not good for doing arithmetic.

With *2's complement*, negative numbers are designed so that the sum of a number and its 2's complement is zero. Using the 4-bit example again, we have  $0101 = 5$  and its 2's complement  $-5 = 1011$ . Adding (remember to carry) gives  $10000 = 0$ . (The 5th bit doesn't count!) Both addition and multiplication work as you would expect using 2's complement. There are two methods for forming the 2's complement:

1. Make the transformation  $0 \rightarrow 1$  and  $1 \rightarrow 0$ , then add 1.
2. Add some number to  $-2^{\text{MSB}}$  to get the number you want. For 4-bit numbers an example of finding the 2's complement of 5 is  $-5 = -8 + 3 = 1000 + 0011 = 1011$ .

### 1.2.2 Hexadecimal Representation

It is very often quite useful to represent blocks of 4 bits by a single digit. Thus in base 16 there is a convention for using one digit for the numbers 0,1,2,...,15 which is called *hexadecimal*. It follows decimal for 0–9, then uses letters A–F.

Decimal	Binary	Hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

## 2 Logic Gates and Combinational Logic

### 2.1 Gate Types and Truth Tables

The basic logic gates are AND, OR, NAND, NOR, XOR, INV, and BUF. The last two are not standard terms; they stand for “inverter” and “buffer”, respectively. The symbols for these gates and their corresponding Boolean expressions are given in Table 8.2 of the text which, for convenience, is reproduced (in part) in Fig. 2.

















Name	Expression	Symbol	Negative true symbol
AND	$AB$		
NAND	$\overline{AB}$		
OR	$A + B$		
NOR	$\overline{A + B}$		
INVERT	$\bar{A}$		
BUFFER	$A$		
XOR	$A \oplus B$		
XNOR	$\overline{A \oplus B}$		

Figure 2: Table 8.2 from the text.

All of the logical gate functions, as well as the Boolean relations discussed in the next section, follow from the truth tables for the AND and OR gates. We reproduce these below. We also show the XOR truth table, because it comes up quite often, although, as we shall see, it is not elemental.

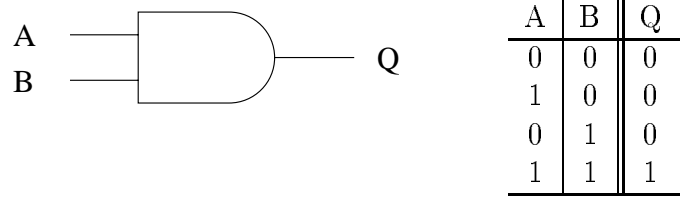


Figure 3: AND gate.

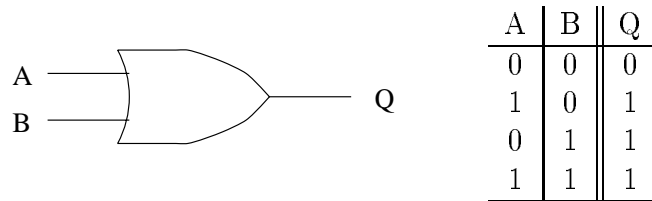


Figure 4: OR gate.

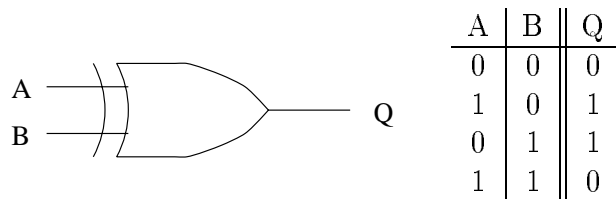


Figure 5: XOR (exclusive OR) gate.

## 2.2 Boolean Algebra and DeMorgan's Theorems

Boolean algebra can be used to formalize the combinations of binary logic states. The fundamental relations are given in Table 8.3 of the text. In these relations,  $A$  and  $B$  are binary quantities, that is, they can be either logical true (T or 1) or logical false (F or 0). Most of these relations are obvious. Here are a few of them:

$$AA = A; \quad A + A = A; \quad A + \bar{A} = 1; \quad A\bar{A} = 0; \quad \bar{\bar{A}} = A$$

Recall that the text sometimes uses an apostrophe for inversion ( $A'$ ). We use the standard overbar notation ( $\bar{A}$ ).

We can use algebraic expressions to complete our definitions of the basic logic gates we began above. Note that the Boolean operations of “multiplication” and “addition” are defined by the truth tables for the AND and OR gates given above in Figs. 3 and 4. Using these definitions, we can define all of the logic gates algebraically. The truth tables can also be constructed from these relations, if necessary. See Fig. 2 for the gate symbols.

- AND:  $Q = AB$  (see Fig. 3)
- OR:  $Q = A + B$  (see Fig. 4)
- NAND:  $Q = \overline{AB}$
- NOR:  $Q = \overline{A + B}$
- XOR:  $Q = A \oplus B$  (defined by truth table Fig. 5)
- INV:  $Q = \bar{A}$
- BUF:  $Q = A$

### 2.2.1 Example: Combining Gates

Let's re-express the XOR operation in terms of standard Boolean operations. The following truth table evaluates the expression  $Q = \bar{A}B + A\bar{B}$ .

$A$	$B$	$\bar{A}B$	$A\bar{B}$	$Q$
0	0	0	0	0
1	0	0	1	1
0	1	1	0	1
1	1	0	0	0

We see that this truth table is identical to the one for the XOR operation. Therefore, we can write

$$A \oplus B = \bar{A}B + A\bar{B} \quad (1)$$

A schematic of this expression in terms of gates is given in Fig. 6 (as well as Fig. 8.25 of the text). Recall that the open circles at the output or input of a gate represent inversion.

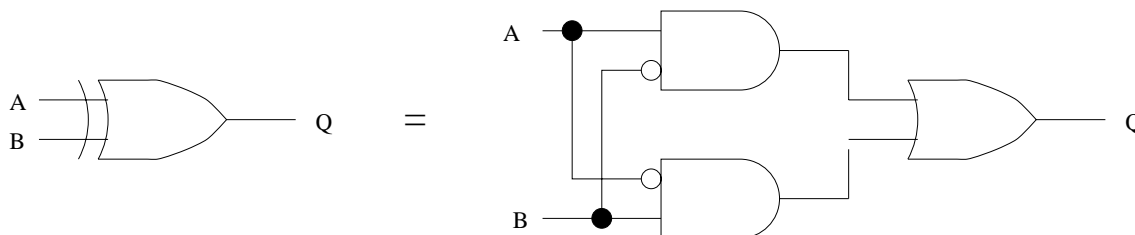


Figure 6: Realization of the XOR gate in terms of AND and OR gates.

### 2.2.2 Gate Interchangeability

In an example from the homework, we can make an INV gate from a 2-input NOR gate. Simply connect the two inputs of the NOR gate together. Algebraically, if the two original NOR gate inputs are labelled  $B$  and  $C$ , and they are combined to form  $A$ , then we have  $Q = \overline{B + C} = \overline{A + A} = \overline{A}$ , which is the INV operation.

Note that an INV gate can not be made from OR or AND gates. For this reason the OR and AND gates are not universal. So for example, no combination of AND gates can be combined to substitute for a NOR gate. However, the NAND and NOR gates *are universal*.

### 2.2.3 DeMorgan

Perhaps the most interesting of the Boolean identities are the two known as DeMorgan's Theorems:

$$\overline{A + B} = \overline{A} \overline{B} \quad (\text{or, } A + B = \overline{\overline{A} \overline{B}}) \quad (2)$$

$$\overline{AB} = \overline{A} + \overline{B} \quad (\text{or, } AB = \overline{\overline{A} + \overline{B}}) \quad (3)$$

These expressions turn out to be quite useful, and we shall use them often.

An example of algebraic logic manipulation follows. It is the one mentioned at the end of Lab 1. One is to show that an XOR gate can be composed of 4 NAND gates. From the section above we know  $A \oplus B = \overline{A}B + A\overline{B}$ . Since  $A\overline{A} = 0$  and  $B\overline{B} = 0$ , we can add these, rearrange, and apply the two DeMorgan relations to give

$$A \oplus B = A(\overline{A} + \overline{B}) + B(\overline{A} + \overline{B}) = A(\overline{AB}) + B(\overline{AB}) = \overline{\overline{A(\overline{AB})}} \overline{\overline{B(\overline{AB})}}$$

## 2.3 Symbolic Logic

The two DeMorgan expressions above can be evoked using gate symbols by following this prescription: *Change gate shape (AND $\leftrightarrow$ OR) and invert all inputs and outputs.*

By examining the two rightmost columns of Fig. 2, one sees that the transformation between 3rd and 4th columns for the gates involving AND/OR gates works exactly in this way. For example, the DeMorgan expression  $\overline{AB} = \overline{A} + \overline{B}$  is represented symbolically by the equivalence between the 3rd and 4th columns of the 2nd row ("NAND") of Fig. 2. We will go over how this works, and some more examples, in class.