1. Introduction to the digital computer

Logic circuits are the basis for modern digital computer systems. To appreciate how computer systems operate you will need to understand digital logic and Boolean algebra. This chapter provides only a basic introduction to Boolean algebra – describing it in its entirety would take up an entire textbook. I chose to concentrate on the basics of Boolean algebra, rather than on optimizing concepts like Karnaugh Maps.

1.1 Digital vs. Analog

The term *digital* refers to the fact that the signal is limited to only a few possible values. In general, digits signals are represented by only two possible voltages on a wire - 0 volts (which we called "binary 0", or just "0") and 5 volts (which we call "binary 1", or just "1"). We sometimes call these values "low" and "high", or "false" and "true". More complicated signals can be constructed from 1s and 0s by stringing them end-to-end, like a necklace. If we put three binary digits end-to-end, we have eight possible combinations: 000, 001, 010, 011, 100, 101, 110 and 111. In principle, there is no limit to how many binary digits we can use in a signal, so signals can be as complicated as you like. The figure below shows a typical digital signal, firstly represented as a series of voltage levels that change as time goes on, and then as a series of 1s and 0s.

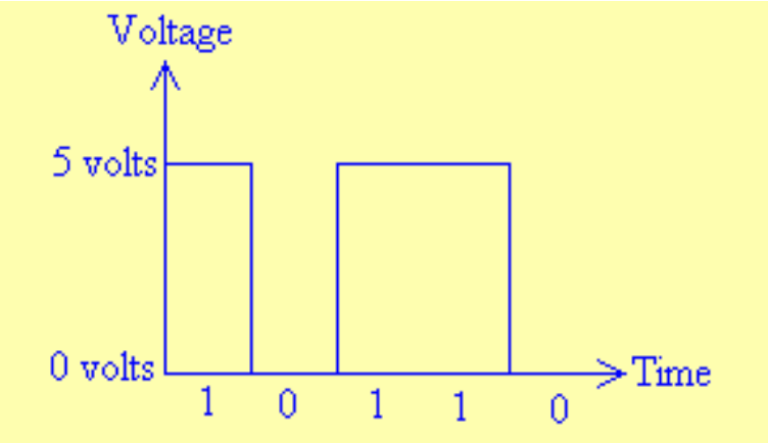


Figure 1. A digital signal

Analog electronics uses voltages that can be any value (within limits, of course - it's difficult to imagine a radio with voltages of a million volts!) The voltages often change smoothly from one value to the next, like gradually turning a light dimmer switch up or down. The figure below shows an analog signal that changes with time.

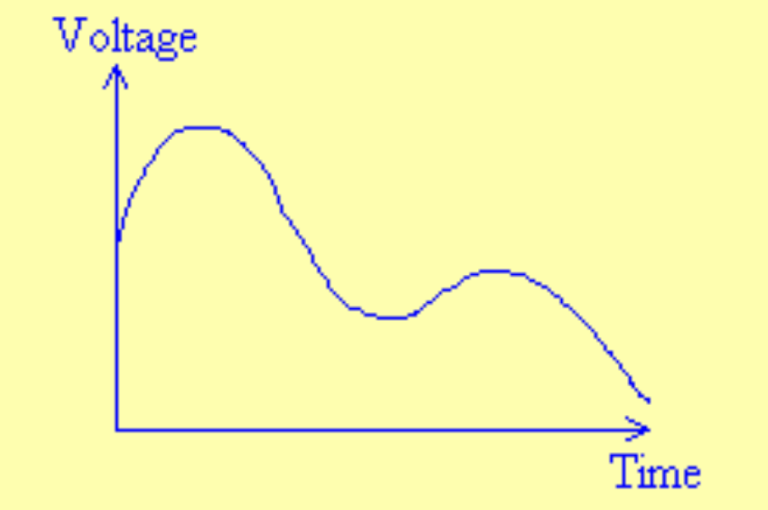


Figure 2. An analog signal

1.2 Number Systems

The ***binary number*** system is a natural choice for representing the behavior of circuits that operate in one of two states (on or off, 1 or 0). For instance, we studied a diode logic gate (refer to the Diodes and Transistors handout online) when we discussed diode circuits. But before we study logic gates, you need to be intimately familiar with the binary number system – the system used by computers for counting. Let's count from zero to twenty using decimal, binary and hexadecimal to contrast these systems of numeration:

Number	Decimal	Binary	Hexadecimal
Zero	0	0	0
One	1	1	1
Two	2	10	2
Three	3	11	3
Four	4	100	4
Five	5	101	5
Six	6	110	6
Seven	7	111	7
Eight	8	1000	8
Nine	9	1001	9
Ten	10	1010	A
Eleven	11	1011	B
Twelve	12	1100	C
Thirteen	13	1101	D
Fourteen	14	1110	E
Fifteen	15	1111	F

1.3 Bit Groupings

The singular reason for learning and using the binary numeration system in electronics is to understand how to design, build, and troubleshoot circuits that represent and process numerical quantities in digital form. Since the bivalent (two-valued) system of binary bit numeration lends itself so easily to representation by "on" and "off" transistor states (recall saturation and cutoff, respectively in the case of the BJT), it makes sense to design and build circuits leveraging this principle to perform binary calculations. If we were to build a circuit to represent a binary number, we would have to allocate enough transistor circuits to represent as many bits as we desire. In other words, in designing a digital circuit, we must first decide how many bits (maximum) we would like to be able to represent, since each bit requires one on/off circuit to represent it.

In digital, electronic computer design, it is common to design the system for a common "***bit width***" a maximum number of bits allocated to represent numerical quantities. Early digital computers handled bits in groups of four or eight. More modern systems handle numbers in clusters of 32 bits or more. To more conveniently express the "bit width" of such clusters in a digital computer, specific labels were applied to the more common groupings. Eight bits, grouped together to form a single binary quantity, is known as a byte. Four bits, grouped together as one binary number, is known by the humorous title of ***nibble***, often spelled as ***nybble***.

A multitude of terms have followed byte and nibble for labeling specific groupings of binary bits. Most of the terms shown here are informal, and have not been made "authoritative" by any standards group or other sanctioning body. However, their inclusion into this chapter is warranted by their occasional appearance in technical literature, as well as the levity they add to an otherwise dry subject:

- **Bit:** A single, bivalent unit of binary notation. Equivalent to a decimal "digit."
- **Crumb, Tydbit, or Tayste:** Two bits.
- **Nibble, or Nybble:** Four bits.
- **Nickle:** Five bits.
- **Byte:** Eight bits.
- **Deckle:** Ten bits.
- **Playte:** Sixteen bits.
- **Dynner:** Thirty-two bits.
- **Word:** (system dependent).

The most ambiguous term by far is ***word***, referring to the standard bit-grouping within a particular digital system. For a computer system using a 32 bit-wide "data path," a "word" would mean 32 bits. If the system used 16 bits as the standard grouping for binary quantities,

a "word" would mean 16 bits. The terms ***playte*** and ***dynner*** (hardly used), by contrast, always refer to 16 and 32 bits, respectively, regardless of the system context in which they are used.

1.4 Digital Logic Gates

While the binary numeration system is an interesting mathematical abstraction, we haven't yet seen its practical application to electronics. This section is devoted to just that: practically applying the concept of binary bits to circuits. What makes binary numeration so important to the application of digital electronics is the ease in which bits may be represented in physical terms. Because a binary bit can only have one of two different values, either 0 or 1, any physical medium capable of switching between two saturated states may be used to represent a bit. Consequently, any physical system capable of representing binary bits is able to represent numerical quantities, and potentially has the ability to manipulate those numbers. This is the basic concept underlying digital computing.

1.4.1 The NOT gate

The gate shown here is known as an inverter, or NOT gate, because it outputs the exact opposite digital signal as what is input. For convenience, gate circuits are generally represented by their own symbols rather than by their constituent transistors and resistors. The following is the symbol for an inverter:

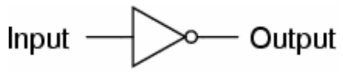


Figure 4. The NOT gate

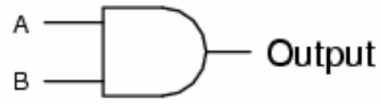
One common way to express the particular function of a gate circuit is called a truth table. Truth tables show all combinations of input conditions in terms of logic level states (either "high" or "low," "1" or "0," for each input terminal of the gate), along with the corresponding output logic level, either "high" or "low." For the inverter, or NOT, circuit just illustrated, the truth table is very simple indeed:

Input	Output
0	1
1	0

Figure 5. NOT gate truth table

1.4.2 The AND gate

Figure 6 shows the schematic symbol and the truth table for a 2-input AND gate.



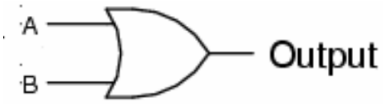
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

Figure 6. AND gate truth table

You can see the reason why it is called “AND” gate: the output will be 1 if and only if all inputs are 1. AND gates can be made with more than three inputs, but they are far less common.

1.4.3 The OR gate

Our next gate to investigate is the OR gate, so-called because the output of this gate will be "high" (1) if any of the inputs (first input or the second input or . . .) are "high" (1). The output of an OR gate goes "low" (0) if and only if all inputs are "low" (0). Figure 7 shows the schematic symbol and truth table of an OR gate.



A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

Figure 7. AND gate truth table