Part I

Introduction

1 The digital abstraction

Digital systems are pervasive in modern society. Some uses of digital technology are obvious – such as a personal computer or a network switch. However, there are also many other applications of digital technology. When you speak on the phone, in almost all cases your voice is being digitized and transmitted via digital communications equipment. When you listen to an audio file, the music, recorded in digital form, is processed by digital logic to correct errors and improve the audio quality. When you watch TV, the image is transmitted in a digital format and processed by digital electronics. If you have a DVR (digital video recorder) you are recording video in digital form. DVDs are compressed digital video recordings. When you play a DVD or stream a movie, you are digitally decompressing and processing the video. Most communication radios, such as cell phones and wireless networks, use digital signal processing to implement their modems. The list goes on.

Most modern electronics uses analog circuitry only at the edge – to interface to a physical sensor or actuator. As quickly as possible, signals from a sensor (e.g., a microphone) are converted into digital form. All real processing, storage, and transmission of information is done digitally. The signals are converted back to analog form only at the output – to drive an actuator (e.g., a speaker) or control other analog systems.

Not so long ago, the world was not as digital. In the 1960s digital logic was found only in expensive computer systems and a few other niche applications. All TVs, radios, music recordings, and telephones were analog.

The shift to digital was enabled by the scaling of integrated circuits. As integrated circuits became more complex, more sophisticated signal processing became possible. Complex techniques such as modulation, error correction, and compression were not feasible in analog technology. Only digital logic, with its ability to perform computations without accumulating noise and its ability to represent signals with arbitrary precision, could implement these signal processing algorithms.

In this book we will look at how the digital systems that form such a large part of our lives function and how they are designed.

1.1 DIGITAL SIGNALS

Digital systems store, process, and transport information in digital form. Digital information is represented as discrete symbols that are encoded into ranges of a physical quantity. Most often we represent information with just two symbols, “0” and “1,” and encode these symbols into voltage ranges as shown in Figure 1.1. Any voltage in the ranges labeled “0” and “1” represents a “0” or “1” symbol, respectively. Voltages between these two ranges, in the region labeled “?,”

Table 1.1. **Encoding of binary signals for 2.5 V LVCMOS logic**

Signals with voltage in $[-0.3, 0.7]$ are considered to be a 0. Signals with voltage in $[1.7, 2.8]$ are considered to be a 1. Voltages in $[0.7, 1.7]$ are undefined. Voltages outside $[-0.3, 2.8]$ may cause permanent damage.

Parameter	Value	Description
V_{min}	−0.3 V	absolute minimum voltage below which damage occurs
V_0	0.0 V	nominal voltage representing logic “0”
V_{OL}	0.2 V	maximum output voltage representing logic “0”
V_{IL}	0.7 V	maximum voltage considered to be a logic “0” by a module input
V_{IH}	1.7 V	minimum voltage considered to be a logic “1” by a module input
V_{OH}	2.1 V	minimum output voltage representing logic “1”
V_1	2.5 V	nominal voltage representing logic “1”
V_{max}	2.8 V	absolute maximum voltage above which damage occurs

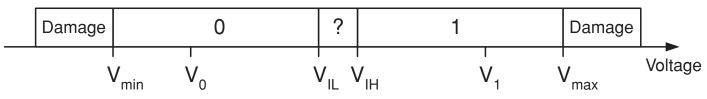


Figure 1.1. Encoding of two symbols, 0 and 1, into voltage ranges. Any voltage in the range labeled 0 is considered to be a 0 symbol. Any voltage in the range labeled 1 is considered to be a 1 symbol. Voltages between the 0 and 1 ranges (the ? range) are undefined and represent neither symbol. Voltages outside the 0 and 1 ranges may cause permanent damage to the equipment receiving the signals.

are undefined and represent neither symbol. Voltages outside the ranges, below the “0” range, or above the “1” range are not allowed and may permanently damage the system if they occur. We call a signal encoded in the manner shown in Figure 1.1 a *binary* signal because it has two valid states.

Table 1.1 shows the JEDEC JESD8-5 standard [62] for encoding a binary digital signal in a system with a 2.5 V power supply. Using this standard, any signal with a voltage between −0.3 V and 0.7 V is considered to be a “0,” and a signal with a voltage between 1.7 V and 2.8 V is considered to be a “1.” Signals that do not fall into these two ranges are undefined. If a signal is below −0.3 V or above 2.8 V, it may cause damage.[1]

Digital systems are not restricted to binary signals. One can generate a digital signal that can take on three, four, or any finite number of discrete values. However, there are few advantages to using more than two values, and the circuits that store and operate on binary signals are simpler and more robust than their multi-valued counterparts. Thus, except for a few niche applications, binary signals are universal in digital systems today.

Digital signals can also be encoded using physical quantities other than voltage. Almost any physical quantity that can be easily manipulated and sensed can be used to represent a digital

[1] The actual specification for V_{max} is $V_{DD} + 0.3$, where V_{DD}, the power supply, is allowed to vary between 2.3 and 2.7 V.

signal. Systems have been built using electrical current, air or fluid pressure, and physical position to represent digital signals. However, the tremendous capability of manufacturing complex systems as low-cost CMOS integrated circuits has made voltage signals universal.

1.2 DIGITAL SIGNALS TOLERATE NOISE

The main reason why digital systems have become so pervasive, and what distinguishes them from *analog* systems, is that they can process, transport, and store information without it being distorted by noise. This is possible because of the discrete nature of digital information. A binary signal represents either a 0 or a 1. If you take the voltage that represents a 1, V_1, and disturb it with a small amount of noise, ϵ, it still represents a 1. There is no loss of information with the addition of noise, until the noise becomes large enough to push the signal out of the 1 range. In most systems it is easy to bound the noise to be less than this value.

Figure 1.2 compares the effect of noise on an analog system (Figure 1.2(a)) and on a digital system (Figure 1.2(b)). In an analog system, information is represented by an analog voltage, V. For example, we might represent temperature (in degrees Fahrenheit) with voltage according to the relation $V = 0.2(T - 68)$. So a temperature of 72.5 °F is represented by a voltage of 900 mV. This representation is continuous; every voltage corresponds to a different temperature. Thus, if we disturb the signal V with a noise voltage ϵ, the resulting signal $V + \epsilon$ corresponds to a different temperature. If $\epsilon = 100$ mV, for example, the new signal $V + \epsilon = 1$ V corresponds to a temperature of 73 °F ($T = 5V + 68$), which is different from the original temperature of 72.5 °F.

In a digital system, each bit of the signal is represented by a voltage, V_1 or V_0 depending on whether the bit is 1 or 0. If a noise source perturbs a digital 1 signal V_1, for example, as shown in Figure 1.2(b), the resulting voltage $V_1 + \epsilon$ still represents a 1 and applying a function to this noisy signal gives the same result as applying a function to the original signal. Moreover, if a temperature of 72 °F is represented by a three-bit digital signal with value 010 (see Figure 1.7(c)), the signal still represents a temperature of 72 °F even after all three bits of the signal have been disturbed by noise – as long as the noise is not so great as to push any bit of the signal out of the valid range.

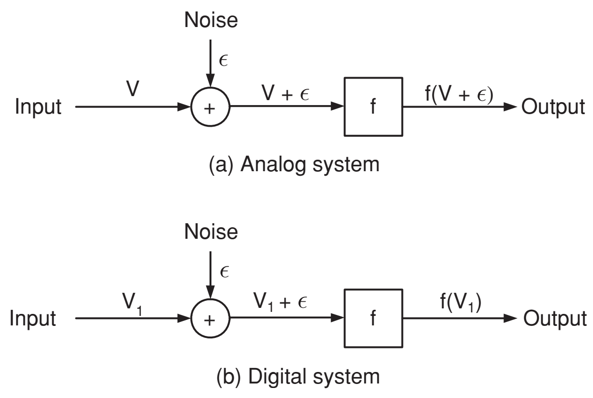


Figure 1.2. Effects of noise in analog and digital systems. (a) In an analog system, perturbing a signal V by noise ϵ results in a degraded signal $V + \epsilon$. Operating on this degraded signal with a function f gives a result $f(V + \epsilon)$ that is different from the result of operating on the signal without noise. (b) In a digital system, adding noise ϵ to a signal V_1 representing a symbol, 1, gives a signal $V_1 + \epsilon$ that still represents the symbol 1. Operating on this signal with a function f gives the same result $f(V_1)$ as operating on the signal without the noise.

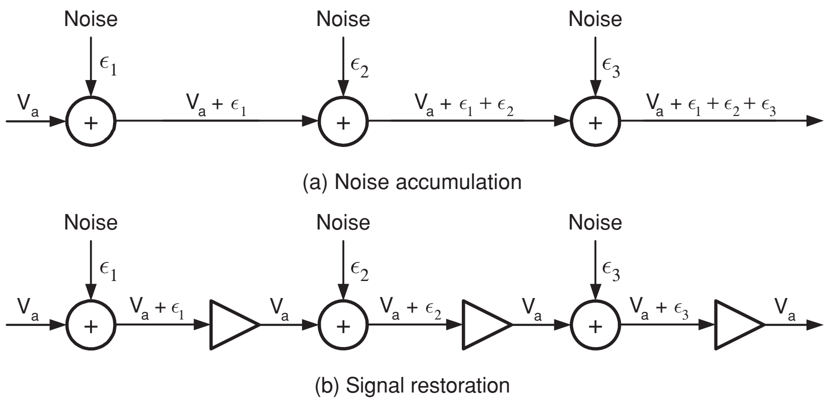


Figure 1.3. Restoration of digital signals. (a) Without restoration, signals accumulate noise and will eventually accumulate enough noise to cause an error. (b) By restoring the signal to its proper value after each operation, noise is prevented from accumulating.

To prevent noise from accumulating to the point where it pushes a digital signal out of the valid 1 or 0 range, we periodically restore digital signals as illustrated in Figure 1.3. After transmitting, storing, and retrieving, or operating on a digital signal, it may be disturbed from its nominal value V_a (where a is 0 or 1) by some noise ϵ_i. Without restoration (Figure 1.3(a)), the noise accumulates after each operation and eventually will overwhelm the signal. To prevent accumulation, we restore the signal after each operation. The restoring device, which we call a *buffer*, outputs V_0 if its input lies in the 0 range and V_1 if its output lies in the 1 range. The buffer, in effect, restores the signal to be a pristine 0 or 1, removing any additive noise.

This capability to restore a signal to its noiseless state after each operation enables digital systems to carry out complex high-precision processing. Analog systems are limited to performing a small number of operations on relatively low-precision signals because noise is accumulated during each operation. After a large number of operations, the signal is swamped by noise. Since all voltages are valid analog signals, there is no way to restore the signal to a noiseless state between operations. Analog systems are also limited in precision. They cannot represent a signal with an accuracy finer than the background noise level. Digital systems can perform an indefinite number of operations, and, as long as the signal is restored after each operation, no noise is accumulated. Digital systems can also represent signals of arbitrary precision without corruption by noise.[2]

In practice, buffers and other restoring logic devices do not guarantee output voltages of exactly V_0 or V_1. Variations in power supplies, device parameters, and other factors lead the outputs to vary slightly from these nominal values. As illustrated in Figure 1.4(b), all restoring logic devices guarantee that their 0 (1) outputs fall into a 0 (1) range that is narrower than the input 0 (1) range. Specifically, all 0 signals are guaranteed to be less than V_{OL} and all 1 signals are guaranteed to be greater than V_{OH}. To ensure that the signal is able to tolerate some amount of noise, we insist that $V_{OL} < V_{IL}$ and that $V_{IH} < V_{OH}$. For example, the values of V_{OL} and V_{OH} for 2.5 V LVCMOS are shown in Table 1.1. We can quantify the amount of noise that can

[2] Of course, one is limited by analog input devices in acquiring real-world signals of high precision.

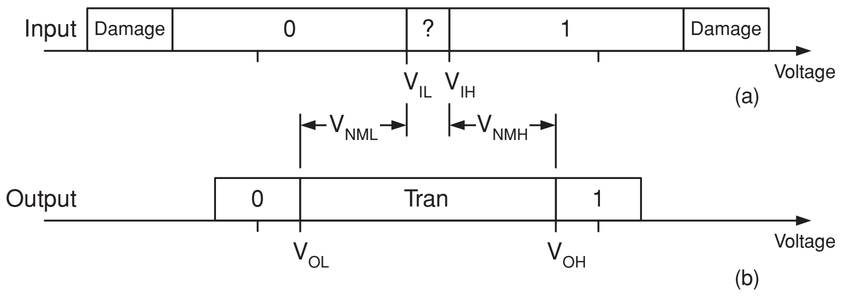


Figure 1.4. Input and output voltage ranges. (a) Inputs of logic modules interpret signals as shown in Figure 1.1. (b) Outputs of logic modules restore signals to narrower ranges of valid voltages.

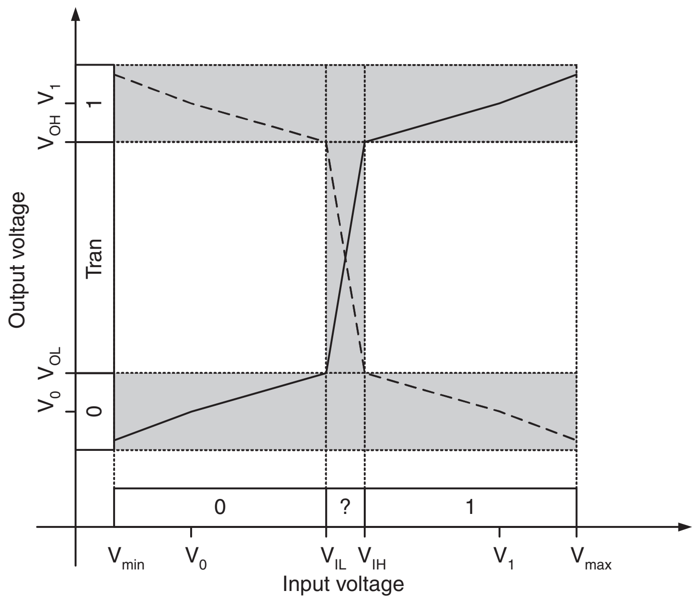


Figure 1.5. DC transfer curve for a logic module. For an input in the valid ranges, $V_{\text{min}} \leq V_{\text{in}} \leq V_{IL}$ or $V_{IH} \leq V_{\text{in}} \leq V_{\text{max}}$, the output must be in the valid output ranges $V_{\text{out}} \leq V_{OL}$ or $V_{OH} \leq V_{\text{out}}$. Thus, all valid curves must stay in the shaded region. This requires that the module have gain > 1 in the invalid input region. The solid curve shows a typical transfer function for a non-inverting module. The dashed curve shows a typical transfer function for an inverting module.

be tolerated as the *noise margins* of the signal:

$$\begin{aligned} V_{NMH} &= V_{OH} - V_{IH}, \\ V_{NML} &= V_{IL} - V_{OL}. \end{aligned} \tag{1.1}$$

While one might assume that a bigger noise margin would be better, this is not necessarily the case. Most noise in digital systems is induced by signal transitions, and hence tends to be proportional to the signal swing. Thus, what is really important is the *ratio* of the noise margin to the signal swing, $V_{NM}/(V_1 - V_0)$, rather than the absolute magnitude of the noise margin.

Figure 1.5 shows the relationship between DC input voltage and output voltage for a logic module. The horizontal axis shows the module input voltage and the vertical axis shows the module output voltage. To conform to our definition of *restoring*, the transfer curve for all modules must lie entirely within the shaded region of the figure so that an input signal in the valid 0 or 1 range will result in an output signal in the narrower output 0 or 1 range. Non-inverting modules, like the buffer of Figure 1.3, have transfer curves similar to the solid line. Inverting modules have transfer curves similar to the dashed line. In either case, gain is required to implement a restoring logic module. The absolute value of the maximum slope of the signal is bounded by

$$\max \left| \frac{dV_{\text{out}}}{dV_{\text{in}}} \right| \geq \frac{V_{OH} - V_{OL}}{V_{IH} - V_{IL}}. \tag{1.2}$$

From this we conclude that restoring logic modules must be active elements capable of providing gain.

EXAMPLE 1.1 Noise toleration

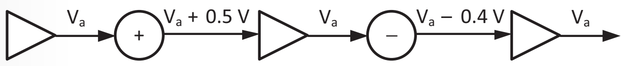


Figure 1.6. Noise model for Example 1.1.

Figure 1.6 illustrates a noise model where noise (modeled by the two voltage sources) can displace the voltage on the output of a buffer V_a by up to 0.5 V in the positive direction and up to 0.4 V in the negative direction. That is, the additive noise voltage $V_n \in [-0.4, 0.5]$. For an output voltage of V_a, the voltage on the input of the following buffer is $V_a + V_n \in [V_a - 0.4, V_a + 0.5]$. Using this noise model and the input and damage constraints of Table 1.1, calculate the range of legal values both for low and for high outputs.

There are four constraints that must be satisfied when calculating the output voltage. A low output (V_{OL}) must give an input voltage that is detected as low ($V_{OL}+V_n \leq V_{IL}$) and not damage the chip ($V_{OL}-V_n \geq V_{\min}$). The high voltage (V_{OH}) cannot damage the chip ($V_{OH}+V_n \leq V_{\max}$) and must be sensed as high ($V_{OH} - V_n \geq V_{IH}$). So we have

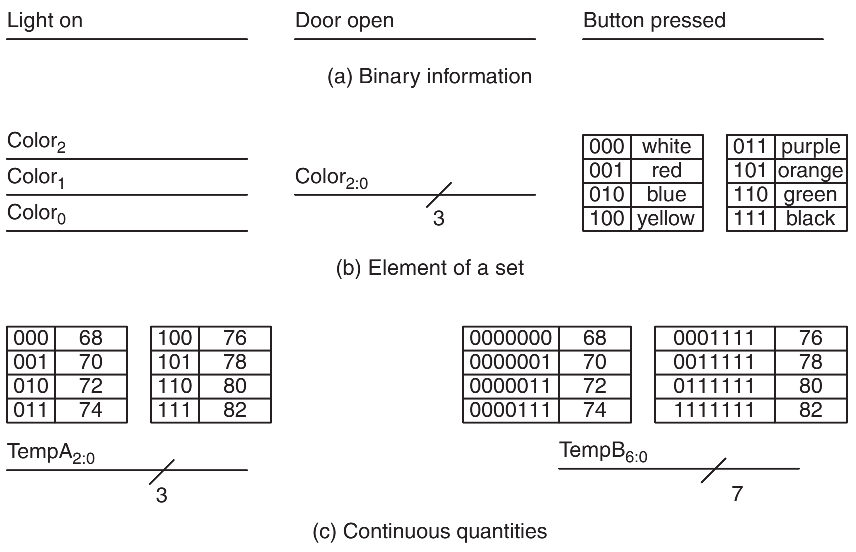
$$\begin{aligned} V_{OL} + 0.5\text{ V} &\leq 0.7\text{ V}, \\ V_{OL} - 0.4\text{ V} &\geq -0.3\text{ V}, \\ 0.1\text{ V} \leq V_{OL} &\leq 0.2\text{ V}, \end{aligned}$$

$$\begin{aligned} V_{OH} + 0.5\text{ V} &\leq 2.8\text{ V}, \\ V_{OH} - 0.4\text{ V} &\geq 1.7\text{ V}, \\ 2.1\text{ V} \leq V_{OH} &\leq 2.3\text{ V}. \end{aligned}$$

The output voltage from each buffer must be between 0.1 V and 0.2 V to represent "0" and between 2.1 V and 2.3 V to represent "1." Although not acceptable with this noise model, almost all circuits will run with nominal output voltages of 0 V and V_{DD} (the supply) to represent 0 and 1.

1.3 DIGITAL SIGNALS REPRESENT COMPLEX DATA

Some information is naturally binary in nature and can be represented with a single binary digital signal (Figure 1.7(a)). Truth propositions or predicates fall into this category. For example, a single signal can indicate that a door is open, a light is on, a seatbelt is buckled, or a button is pressed.



000	white	011	purple
001	red	101	orange
010	blue	110	green
100	yellow	111	black

000	68	100	76
001	70	101	78
010	72	110	80
011	74	111	82

0000000	68	0001111	76
0000001	70	0011111	78
0000011	72	0111111	80
0000111	74	1111111	82

Figure 1.7. Representing information with digital signals. (a) Binary-valued predicates are represented by a single-bit signal. (b) Elements of sets with more than two elements are represented by a group of signals. In this case one of eight colors is denoted by a three-bit signal $Color_{2:0}$. (c) A continuous quantity, like temperature, is *quantized*, and the resulting set of values is encoded by a group of signals. Here one of eight temperatures can be encoded as a three-bit signal $TempA_{2:0}$ or as a seven-bit *thermometer-coded* signal $TempB_{6:0}$ with at most one transition from 0 to 1.

By convention, we often consider a signal to be "true" when the voltage is high. This need not be the case, as nothing precludes using low voltages to represent the above conditions. Throughout the book we will try to make it clear when we are using this low–true convention. The signal description in such cases will often be changed instead, e.g., "a seatbelt is unbuckled."

Often we need to represent information that is not binary in nature: a day of the year, the value and suit of a playing card, the temperature in a room, or a color. We encode information with more than two natural states using a group of binary signals (Figure 1.7(b)). The elements of a set with N elements can be represented by a signal with $n = \lceil \log_2 N \rceil$ bits. For example, the eight colors shown in Figure 1.7(b) can be represented by three one-bit signals, $Color_0$, $Color_1$, and $Color_2$. For convenience we refer to this group of three signals as a single multi-bit signal $Color_{2:0}$. In a circuit or schematic diagram, rather than drawing three lines for these three signals, we draw a single line with a slash indicating that it is a multi-bit signal and the number "3" near the slash to indicate that it is composed of three bits.[3]

Continuous quantities, such as voltage, temperature, and pressure, are encoded as digital signals by *quantizing* them. This reduces the problem to one of representing elements of a set. Suppose, for example, that we need to represent temperatures between 68 °F and 82 °F and that it suffices to resolve the temperature to an accuracy of 2 °F. We quantize this temperature range into eight discrete values as shown in Figure 1.7(c). We can represent this range with binary

[3] This notation for multi-bit signals is discussed in more detail in Section 8.1.

weighted signals $TempA_{2:0}$, where the temperature represented is

$$T = 68 + 2\sum_{i=0}^{2} 2^i TempA_i. \tag{1.3}$$

Alternatively, we can represent this range with a seven-bit *thermometer-coded* signal $TempB_{6:0}$:

$$T = 68 + 2\sum_{i=0}^{6} TempB_i. \tag{1.4}$$

Many other encodings of this set are possible. A designer chooses a representation depending on the task at hand. Some sensors (e.g., thermometers) naturally generate thermometer-coded signals. In some applications it is important that adjacent codes differ in only a single bit. At other times, cost and complexity are reduced by minimizing the number of bits needed to represent an element of the set. We will revisit digital representations of continuous quantities when we discuss numbers and arithmetic in Chapter 10.

1.3.1 Representing the day of the year

Suppose we wish to represent the day of the year with a digital signal. (We will ignore for now the problem of leap years.) The signal is to be used for operations that include determining the next day (i.e., given the representation of today, compute the representation of tomorrow), testing whether two days are in the same month, determining whether one day comes before another, and whether a day is a particular day of the week.

One approach is to use a $\lceil \log_2 365 \rceil = 9$-bit signal that represents the integers from 0 to 364, where 0 represents January 1 and 364 represents December 31. This representation is compact (you cannot do it in fewer than nine bits), and it makes it easy to determine whether one day comes before another. However, it does not facilitate the other two operations we need to perform. To determine the month a day corresponds to requires comparing the signal with ranges for each month (January is 0–30, February is 31–58, etc.). Determining the day of the week requires taking the date integer modulo 7.

A better approach, for our purposes, is to represent the signal as a four-bit month field (January $= 1$, December $= 12$) and a five-bit day field (1–31, leaving 0 unused). With this representation, for example, July 4 (US Independence Day) is $0111\ 00100_2$. The $0111_2 = 7$ represents July and $00100_2 = 4$ represents the day. With this representation we can still directly compare whether one day comes before another and also easily test whether two days are in the same month by comparing the upper four bits. However, it is even more difficult with this representation to determine the day of the week.

To solve the problem of the day of the week, we use a redundant representation that consists of a four-bit month field (1–12), a five-bit day of the month field (1–31), and a three-bit day of the week field (Sunday $= 1, \ldots,$ Saturday $= 7$). With this representation, July 4 (which is a Monday in 2016) would be represented as the 12-bit binary number 0111 00100 100. The 0111 means month 7, or July, 00100 means day 4 of the month, and 100 means day 4 of the week, or Wednesday.