Digital Systems

1

This first chapter divides up into three sections. The first section defines the concept of digital system. For that, the more general concept of physical system is first defined. Then, the particular characteristics of digital physical systems are presented. In the second section, several methods of digital system specification are considered. A correct and unambiguous initial system specification is a key aspect of the development work. Finally, the third section is a brief introduction to digital electronics.

1.1 Definition

As a first step, the more general concept of physical system is introduced. It is not easy to give a complete and rigorous definition of physical system. Nevertheless, this expression has a rather clear intuitive meaning, and some of their more important characteristics can be underlined.

A physical system could be defined as a set of interconnected objects or elements that realize some function and are characterized by a set of input signals, a set of output signals, and a relation between input and output signals. Furthermore, every signal is characterized by

- Its type, for example a voltage, a pressure, a temperature, and a switch state
- A range of values, for example all voltages between 0 and 1.5 V and all temperatures between 15 and 25 °C

Example 1.1 Consider the system of Fig. 1.1.
It controls the working of a boiler that is part of a room heating system and is connected to a mechanical selector that permits to define a reference temperature. A temperature sensor measures the ambient temperature. Thus, the system has two input signals

- *pos*: the selector position that defines the desired ambient temperature (any value between 10 and 30°)
- *temp*: the temperature measured by the sensor

and one output signal

- *onoff*, with two possible values *ON* (start the boiler) and *OFF* (stop the boiler).

J.-P. Deschamps et al., *Digital Systems*, DOI 10.1007/978-3-319-41198-9_1

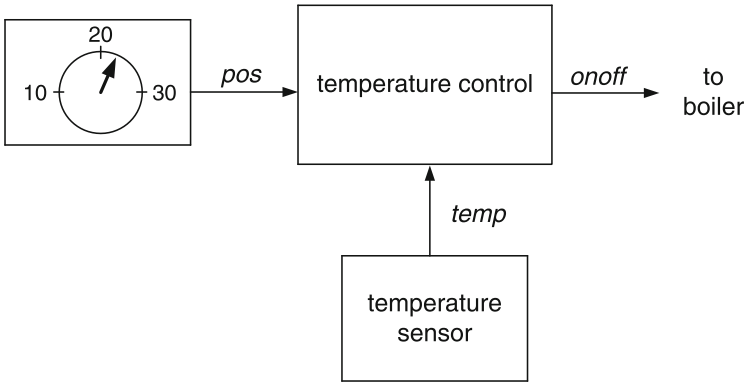


Fig. 1.1 Temperature control

The relation between inputs and output is defined by the following program in which *half_degree* is a previously defined constant equal to 0.5.

Algorithm 1.1 Temperature Control

```
loop
    if temp < pos - half_degree then onoff = on;
    elsif temp > pos + half_degree then onoff = off;
    end if;
    wait for 10 s;
end loop;
```

This is a pseudo-code program. An introduction to pseudo-code is given in Appendix B. However, this piece of program is quite easy to understand, even without any previous knowledge. Actually, the chosen pseudo-code is a simplified (non-executable) version of VHDL (Appendix A).

Algorithm 1.1 is a loop whose body is executed every 10 s: the measured temperature *temp* is compared with the desired temperature *pos* defined by the mechanical selector position; then

- If *temp* is smaller than $pos - 0.5$, then the boiler must get started so that the output signal $onoff = ON$.
- If *temp* is greater than $pos + 0.5$, then the boiler must be stopped so that the output signal $onoff = OFF$.
- If *temp* is included between $pos - 0.5$ and $pos + 0.5$, then no action is undertaken and the signal *onoff* value remains unchanged.

This is a functional specification including some additional characteristics of the final system. For example: The temperature updating is performed every 10 s, so that the arithmetic operations must be executed in less than 10 s, and the accuracy of the control is about $\pm 0.5°$.

As mentioned above, the type and range of the input and output signals must be defined.

- The input signal *temp* represents the ambient temperature measured by a sensor. Assume that the sensor is able to measure temperatures between 0 and 50°. Then *temp* is a signal whose type is "temperature" and whose range is "0 to 50°."
- The input signal *pos* is the position of a mechanical selector. Assume that it permits to choose any temperature between 10 and 30°. Then *pos* is a signal whose type is "position" and whose range is "10–30."
- The output signal *onoff* has only two possible values. Its type is "command" and its range is $\{ON, OFF\}$.

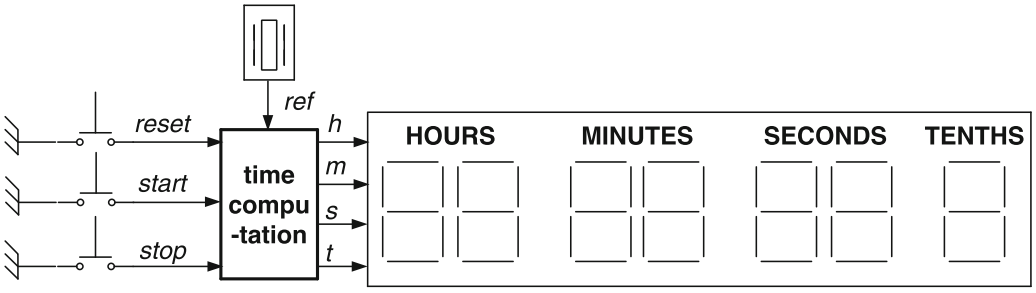
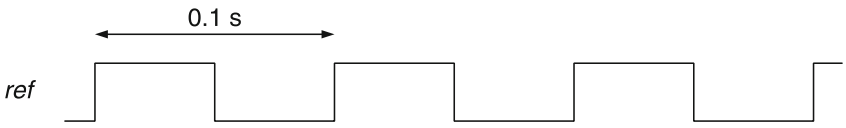


Fig. 1.2 Chronometer

Fig. 1.3 Time reference signal



Assume now that the sensor is an ideal one, able to measure the temperature with an infinite accuracy, and that the selector is a continuous one, able to define the desired temperature with an infinite precision. Then both signals *temp* and *pos* are real numbers whose ranges are [0, 50] and [10, 30], respectively. Those signals, characterized by a continuous and infinite range of values, are called analog signals.

On the contrary, the range of the output signal *onoff* is a finite set {*ON*, *OFF*}. Signals whose range is a finite set (not necessary binary as in the case of *onoff*) are called digital signals or discrete signals.

Example 1.2 Figure 1.2 represents the structure of a chronometer.

- Three push buttons control its working. They generate binary (2-valued) signals *reset*, *start*, and *stop*.
- A crystal oscillator generates a time reference signal *ref* (Fig. 1.3): it is a square wave signal whose period is equal to 0.1 s (10 Hz).
- A *time computation* system computes the value of signals *h* (hours), *m* (minutes), *s* (seconds), and *t* (tenths of second).
- Some graphical interface displays the values of signals *h*, *m*, *s*, and *t*.

Consider the *time computation* block. It is a physical system (a subsystem of the complete chronometer) whose input signals are

- *reset*, *start*, and *stop* that are generated by three push buttons; according to the state of the corresponding switch, their value belongs to the set {*closed*, *open*}.
- *ref* is the signal generated by the crystal oscillator and is assumed to be an ideal square wave equal to either 0 or 1 V

and whose output signals are

- *h* belonging to the set {0, 1, 2, . . ., 23}
- *m* and *s* belonging to the set {0, 1, 2, . . ., 59}
- *t* belonging to the set {0, 1, 2, . . ., 9}

The relation between inputs and outputs can be defined as follows (in natural language):

- When *reset* is pushed down then $h = m = s = t = 0$.
- When *start* is pushed down, the chronometer starts counting; h, m, s, and t represent the elapsed time in tenth of seconds.
- When *stop* is pushed down, the chronometer stops counting; h, m, s, and t represent the latest elapsed time.

In this example, all input and output signal values belong to finite sets. So, according to a previous definition, all input and output signals are digital. Systems whose all input and output signals are digital are called digital system.

1.2 Description Methods

In this section several specification methods are presented.

1.2.1 Functional Description

The relation between inputs and outputs of a digital system can be defined in a functional way, without any information about the internal structure of the system. Furthermore, a distinction can be made between explicit and implicit functional descriptions.

Example 1.3 Consider again the temperature controller of Example 1.1, with two modifications:

- The desired temperature (*pos*) is assumed to be constant and equal to 20° ($pos = 20$).
- The measured temperature has been discretized so that the signal *temp* values belong to the set $\{0, 1, 2, \ldots, 50\}$.

Then, the working of the controller can be described, in a completely explicit way, by Table 1.1 that associates to each value of *temp* the corresponding value of *onoff*: if *temp* is smaller than 20, then *onoff* = *ON*; if *temp* is greater than 20, then *onoff* = *ON*; if *temp* is equal to 20, then *onoff* keeps unchanged.

The same specification could be expressed by the following program.

Table 1.1 Explicit specification

temp	*onoff*
0	*ON*
1	*ON*
...	...
18	*ON*
19	*ON*
20	*unchanged*
21	*OFF*
22	*OFF*
...	...
49	*OFF*
50	*OFF*

Algorithm 1.2 Simplified Temperature Control

```
if temp < 20 then onoff = on;
elsif temp > 20 then onoff = off;
end if;
```

This type of description, by means of an algorithm, will be called "implicit functional description." In such a simple example, the difference between Table 1.1 and Algorithm 1.2 is only formal; in fact it is the same description. In more complex systems, a completely explicit description (a table) could be unmanageable.

Example 1.4 As a second example of functional specification consider a system (Fig. 1.4) that adds two 2-digit numbers.

Its input signals are

- x_1, x_0, y_1, and y_0 whose values belong to $\{0, 1, 2, \ldots, 9\}$

and its output signals are

- z_2 whose values belong to $\{0, 1\}$, and z_1 and z_0 whose values belong to $\{0, 1, 2, \ldots, 9\}$.

Digits x_1 and x_0 represent a number X belonging to the set $\{0, 1, 2, \ldots, 99\}$; digits y_1 and y_0 represent a number Y belonging to the same set $\{0, 1, 2, \ldots, 99\}$, and digits z_2, z_1, and z_0 represent a number Z belonging to the set $\{0, 1, 2, \ldots, 198\}$ where $198 = 99 + 99$ is the maximum value of $X + Y$.

An explicit functional specification is Table 1.2 that contains 10,000 rows!

Another way to specify the function of a 2-digit adder is the following algorithm in which symbol/ stands for the integer division.

Fig. 1.4 2-Digit adder

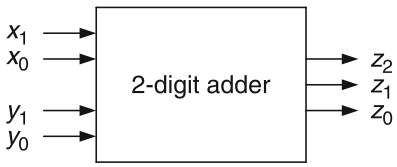


Table 1.2 Explicit specification of a 2-digit adder

$x_1\ x_0$	$y_1\ y_0$	$z_2\ z_1\ z_0$
00	00	000
00	01	001
...	...	...
00	99	099
01	00	001
01	01	002
...	...	...
01	99	100
...	...	...
99	00	099
99	01	100
...	...	...
99	99	198

Algorithm 1.3 2-Digit Adder

```
X = 10·x1 + x0;
Y = 10·y1 + y0;
Z = X + Y;
z2 = Z/100;
z1 = (Z - 100·z2)/10;
z0 = Z - 100·z2 - 10·z1;
```

As an example, if $x_1 = 5$, $x_0 = 7$, $y_1 = 7$, and $y_0 = 1$, then

$X = 10{\cdot}5 + 7 = 57.$
$Y = 10 \cdot 7 + 1 = 71.$
$Z = 57 + 71 = 128.$
$z_2 = 128/100 = 1.$
$z_2 = (128 - 100 \cdot 1)/10 = 28/10 = 2.$
$z_3 = 128 - 100 \cdot 1 - 10.2 = 8.$

At the end of the algorithm execution:

$$X + Y = Z = 100 \cdot z_2 + 10 \cdot z_1 + z_0.$$

Table 1.2 and Algorithm 1.3 are functional specifications. The first is explicit, the second is implicit, and both are directly deduced from the initial unformal definition: x_1 and x_0 represent X, digits y_1 and y_0 represent Y, and z_2, z_1, and z_0 represent $Z = X + Y$.

Another way to define the working of the 2-digit adder is to use the classical pencil and paper algorithm. Given two 2-digit numbers $x_1\ x_0$ and $y_1\ y_0$,

- Compute $s_0 = x_1 + x_0$.
- If $s_0 < 10$ then $z_0 = s_0$ and $carry = 0$; in the contrary case ($s_0 \geq 10$) then $z_0 = s_0 - 10$ and $carry = 1$.
- Compute $s_1 = y_1 + y_0 + carry$.
- If $s_1 < 10$ then $z_1 = s_1$ and $z_2 = 0$; in the contrary case ($s_1 \geq 10$) then $z_1 = s_1 - 10$ and $z_2 = 1$.

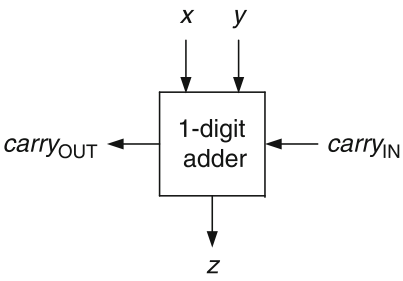
Algorithm 1.4 Pencil and Paper Algorithm

```
s0 = x0 + y0;
if s0 ≥ 10 then z0 = s0 - 10; carry = 1;
else z0 = s0; carry = 0;
end if;
s1 = x1 + y1 + carry;
if s1 ≥ 10 then z1 = s1 - 10; z2 = 1;
else z1 = s1; z2 = 0;
end if;
```

As an example, if $x_1 = 5$, $x_0 = 7$, $y_1 = 7$, and $y_0 = 1$, then

$s_0 = 7 + 1 = 8;$
$s_0 < 10$ so that $z_0 = 8$; $carry = 0$;

Fig. 1.5 1-Digit adder



$s_1 = 5 + 7 + 0 = 12$;
$s_1 \geq 10$ so that $z_1 = 12 - 10 = 2$; $z_2 = 1$;

and thus $57 + 71 = 128$.

Comment 1.1
Algorithm 1.4 is another implicit functional specification. However it is not directly deduced from the initial unformal definition as was the case of Table 1.2 and of Algorithm 1.3. It includes a particular step-by-step addition method and, to some extent, already gives some indication about the structure of the system (the subject of next Sect. 1.2.2). Furthermore, it could easily be generalized to the case of n-digit operands for any $n > 2$.

1.2.2 Structural Description

Another way to specify the relation between inputs and outputs of a digital system is to define its internal structure. For that, a set of previously defined and reusable subsystems called components must be available.

Example 1.5 Assume that a component called 1-digit adder (Fig. 1.5) has been previously defined. Its input signals are

- Digits x and y belonging to $\{0, 1, 2, \ldots, 9\}$
- $carry_{IN} \in \{0, 1\}$

and its output signals are

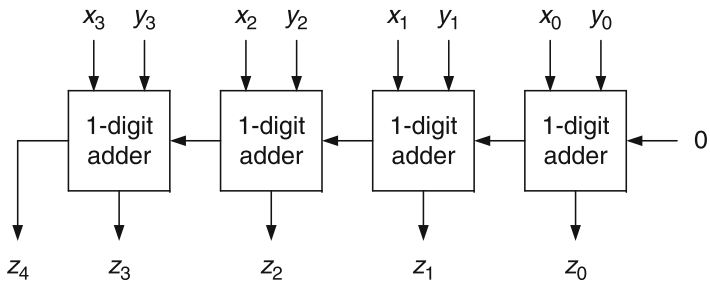
- $z \in \{0, 1, 2, \ldots, 9\}$
- $carry_{OUT} \in \{0, 1\}$

Every 1-digit adder component executes the operations that correspond to a particular step of the pencil and paper addition method (Algorithm 1.4):

- Add two digits and an incoming carry.
- If the obtained sum is greater than or equal to 10, subtract 10 and the outgoing carry is 1; in the contrary case the outgoing carry is 0.

The following algorithm specifies its working.

Fig. 1.6 4-Digit adder



Algorithm 1.5 1-Digit Adder

```
s = x + y + carry_IN;
if s ≥ 10 then z = s - 10; carry_OUT = 1;
else z = s; carry_OUT = 0;
end if;
```

With this component, the structure of a 4-digit adder can be defined (Fig. 1.6).

It computes the sum $Z = X + Y$ where $X = x_3\, x_2\, x_1\, x_0$ and $Y = y_3\, y_2\, y_1\, y_0$ are two 4-digit numbers and $Z = z_4\ \ z_3\ \ z_2\ \ z_1\ \ z_0$ is a 5-digit number whose most significant digit z_4 is 0 or 1 ($X + Y \leq 9999 + 9999 = 19{,}998$).

Comment 1.2
In the previous Example 1.5, four identical components (1-digit adders) are used to define a 4-digit adder by means of its structure (Fig. 1.6). The 1-digit adder in turn has been defined by its function (Algorithm 1.5). This is an example of 2-level hierarchical description. The first level is a diagram that describes the structure of the system, while the second level is the functional description of the components.

1.2.3 Hierarchical Description

Hierarchical descriptions with more than two levels can be considered. The following example describes a 3-level hierarchical description.

Example 1.6 Consider a system that computes the sum $z = w + x + y$ where w, x, and y are 4-digit numbers. The maximum value of z is 9999 + 9999 + 9999 = 29,997 that is a 5-digit number whose most significant digit is equal to 0, 1, or 2. The first hierarchical level (top level) is a block diagram with two different blocks (Fig. 1.7): a 4-digit adder and a 5-digit adder.

The 4-digit adder can be divided into four 1-digit adders (Fig. 1.8) and the 5-digit adder can be divided into five 1-digit adders (Fig. 1.9). Figures 1.8 and 1.9 constitute a second hierarchical level. Finally, a 1-digit adder (Fig. 1.5) can be defined by its functional description (Algorithm 1.5). It constitutes a third hierarchical level (bottom level).

Thus, the description of the system that computes z consists of three levels (Fig. 1.10). The lowest level is the functional description of a 1-digit adder. Assuming that 1-digit adder components are available, the system can be built with nine components.

A hierarchical description could be defined as follows.

- It is a set of interconnected blocks.
- Every block, in turn, is described either by its function or by a set of interconnected blocks, and so on.
- The final blocks correspond to available components defined by their function.

Fig. 1.7 Top level

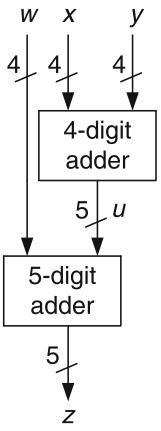


Fig. 1.8 4-Digit adder

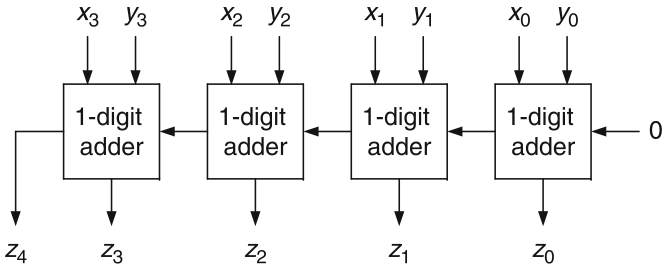
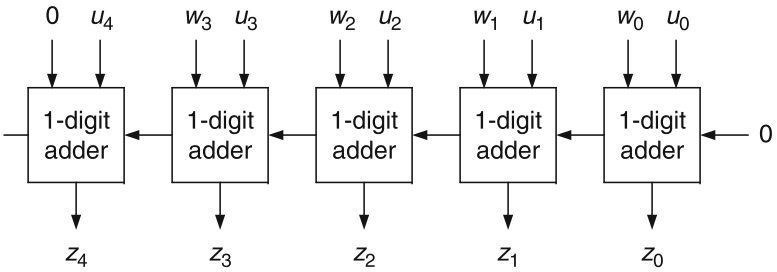


Fig. 1.9 5-Digit adder



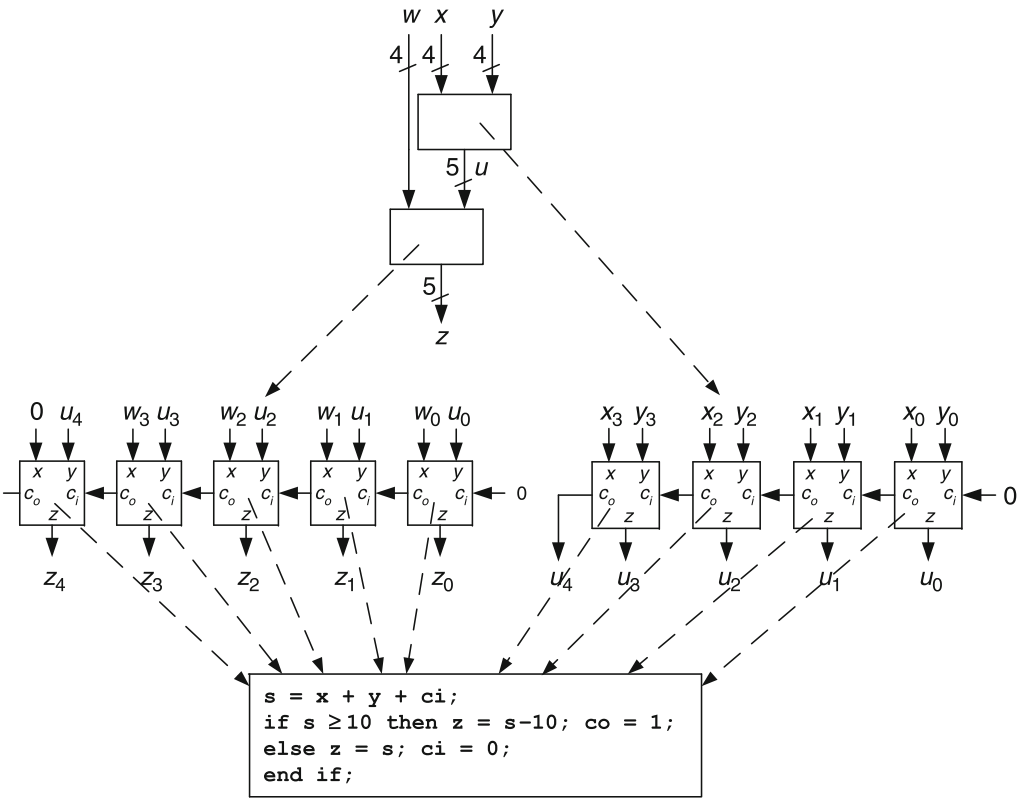


Fig. 1.10 Hierarchical description

Comments 1.3
Generally, the initial specification of a digital system is functional (a description of what the system does). In the case of very simple systems it could be a table that defines the output signal values in function of the input signal values. However, for more complex systems other specification methods should be used. A natural language description (e.g., in English) is a frequent option. Nevertheless, an algorithmic description (programing language, hardware description language, pseudo-code) could be a better choice: those languages have a more precise and unambiguous semantics than natural languages. Furthermore, programing language and hardware description language specifications can be compiled and executed, so that the initial specification can be tested. The use of algorithms to define the function of digital systems is one of the key aspects of this course.

In other cases, the initial specification already gives some information about the way the system must be implemented (see Examples 1.5 and 1.6).

In fact, the digital system designer work is the generation of a circuit made up of available components and whose behavior corresponds to the initial specification. Many times this work consists of successive refinements of an initial description: starting from an initial specification a (top level) block diagram is generated; then, every block is treated as a subsystem to which a more detailed block diagram is associated, and so on. The design work ends when all block diagrams are made up of interconnected components defined by their function and belonging to some available library of physical components (Chap. 7).

1.3 Digital Electronic Systems

The definition of digital system of Sect. 1.1 is a very general one and refers to any type of physical system whose input and output values belong to a finite set. In what follows, this course will focus on electronic systems.

1.3.1 Real System Structure

Most real digital systems include (Fig. 1.11)

- Input devices such as sensors, keyboards, microphones, and communication receivers.
- Output devices such as displays, motors, communication transmitters, and loudspeakers.

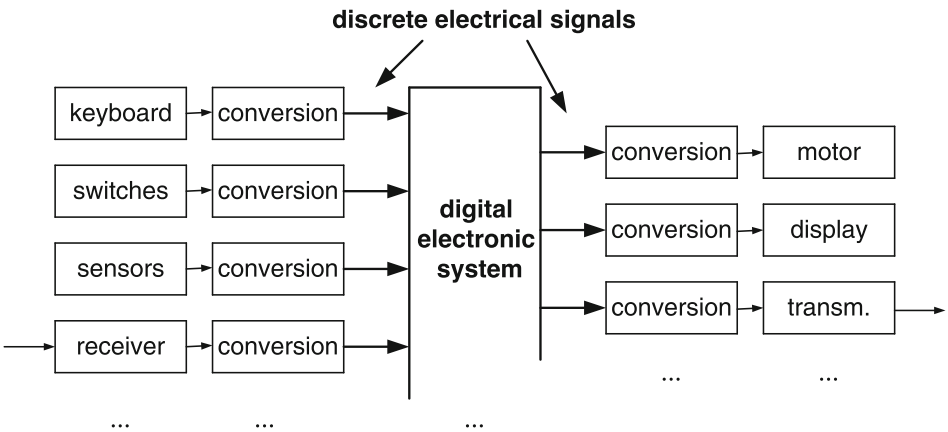


Fig. 1.11 Structure of a real digital system

- Input converters that translate the information generated by the input devices to discrete electrical signals.
- Output converters that translate discrete electrical signals into signals able to control the output devices.
- A digital electronic circuit—the brain of the system—that generates output electrical data in function of the input electrical data.

In Example 1.2, the input devices are three switches (push buttons) and a crystal oscillator, and the output device is a 7-digit display. The time computation block is an electronic circuit that constitutes the brain of the complete system.

Thus, real systems consist of a set of input and output interfaces that connect the input and output devices to the kernel of the system. The kernel of the system is a digital electronic system whose input and output signals are discrete electrical signals.

In most cases those input and output signals are binary encoded data. As an example, numbers can be encoded according to the binary numeration system and characters such as letters, digits, or some symbols can be encoded according to the standard ASCII codes (American Standard Code for Information Interchange).

1.3.2 Electronic Components

To build digital electronic systems, electronic components are used. In this section some basic information about digital electronic components is given. Much more complete and detailed information about digital electronics can be found in books such as Weste and Harris (2010) or Rabaey et al. (2003).

1.3.2.1 Binary Codification

A first question: It has been mentioned above that, in most cases, the input and output signals are binary encoded data; but how are the binary digits (bits) 0 and 1 physically (electrically) represented? The usual solution consists in defining a low voltage V_L, and a high voltage V_H, and conventionally associating V_L to bit 0 and V_H to bit 1. The value of V_L and V_H depends on the implementation technology. In this section it is assumed that $V_L = 0$ V and $V_H = 1$ V.

1.3.2.2 MOS Transistors

Nowadays, most digital circuits are made up of interconnected MOS transistors. They are very small devices and large integrated circuits contain millions of transistors.

MOS transistors (Fig. 1.12a, b) have three terminals called S (source), D (drain), and G (gate). There are two types of transistors: n-type (Fig. 1.12a) and p-type (Fig. 1.12b) where n and p refer to the type of majority electrical charges (carriers) that can flow from terminal S (source) to terminal D (drain) under the control of the gate voltage: in an nMOS transistor the majority carriers are

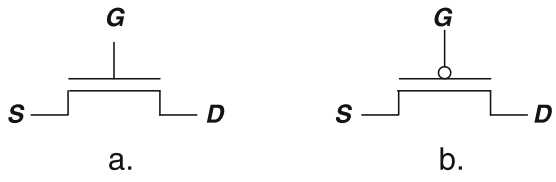


Fig. 1.12 MOS transistors

electrons (negative charges) so that the current flows from D to S; in a pMOS transistor the majority carriers are holes (positive charges) so that the current flows from S to D.

A very simplified model (Fig. 1.13) is now used to describe the working of an nMOS transistor: it works like a switch controlled by the transistor gate voltage.

If the gate voltage V_G is low (0 V) then the switch is open (Fig. 1.14a, b) and no current could flow. If the gate voltage V_G is high (1 V) then the switch is closed (Fig. 1.14c, d) and V_{OUT} tends to be equal to V_{IN}. However, if V_{IN} is high (1 V) then V_{OUT} is not equal to 1 V (Fig. 1.14b). The maximum value of V_{OUT} is $V_G - V_T$ where the threshold voltage V_T is a characteristic of the implementation technology. It could be said that an nMOS transistor is a good switch for transmitting V_L (Fig. 1.14c), but not a good switch for transmitting V_H (Fig. 1.14d).

A similar model can be used to describe the working of a pMOS transistor. If the gate voltage V_G is high (1 V) then the switch is open (Fig. 1.15a, b) and no current could flow. If the gate voltage V_G is low (0 V) then the switch is closed (Fig. 1.15c, d) and V_{OUT} tends to be equal to V_{IN}. However, if V_{IN} is low (0 V) then V_{OUT} is not equal to 0 V (Fig. 1.15b). Actually the minimum value of V_{OUT} is $V_G + |V_T|$ where the threshold voltage V_T is a characteristic of the implementation technology. It could be said that a pMOS transistor is a good switch for transmitting V_H (Fig. 1.15c), but not a good switch for transmitting V_L (Fig. 1.15d).

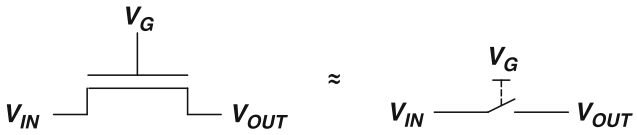


Fig. 1.13 Equivalent model

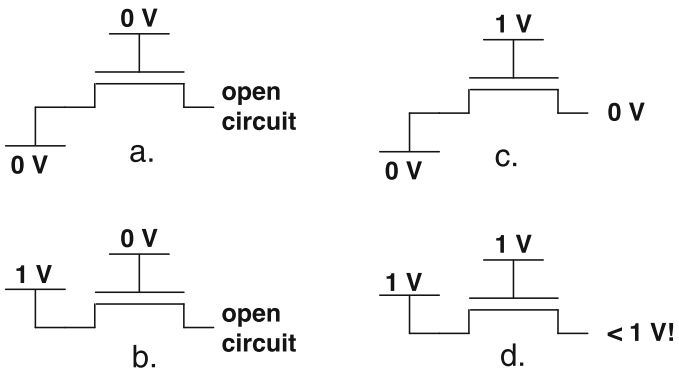


Fig. 1.14 nMOS switches

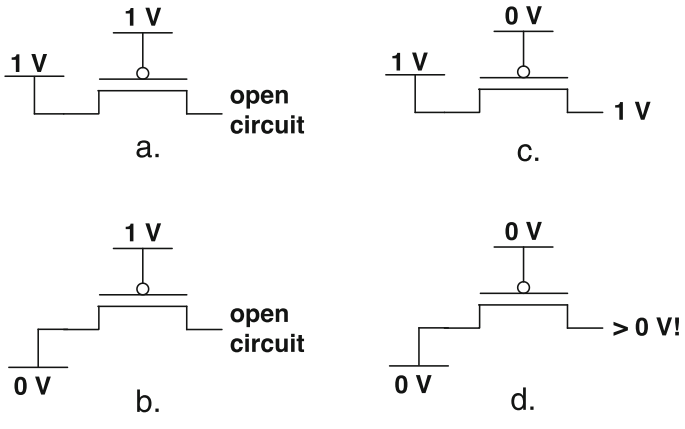


Fig. 1.15 pMOS switches

1.3.2.3 CMOS Inverter

By interconnecting several transistors, small components called logic gates can be implemented. The simplest one (Fig. 1.16) is the CMOS inverter, also called NOT gate.

A CMOS inverter consists of two transistors:

- A *p*MOS transistor whose source is connected to the high voltage V_H (1 V), whose gate is connected to the circuit input and whose drain is connected to the circuit output.
- An *n*MOS transistor whose source is connected to the low voltage V_L (0 V), whose gate is connected to the circuit input and whose drain is connected to the circuit output.

To analyze the working of this circuit in the case of binary signals, consider the two following input values:

- If $V_{IN} = 0$ V then (Fig. 1.17a) according to the simplified model of Sect. 1.3.2.2, the *n*MOS transistor is equivalent to an open switch and the *p*MOS transistor is equivalent to a closed switch (a good switch for transmitting V_H) so that $V_{OUT} = 1$ V.
- If $V_{IN} = 1$ V then (Fig. 1.17b) the *p*MOS transistor is equivalent to an open switch and the *n*MOS transistor is equivalent to a closed switch (a good switch for transmitting V_L) so that $V_{OUT} = 0$ V.

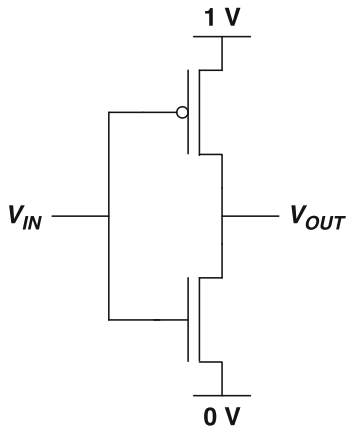


Fig. 1.16 CMOS inverter

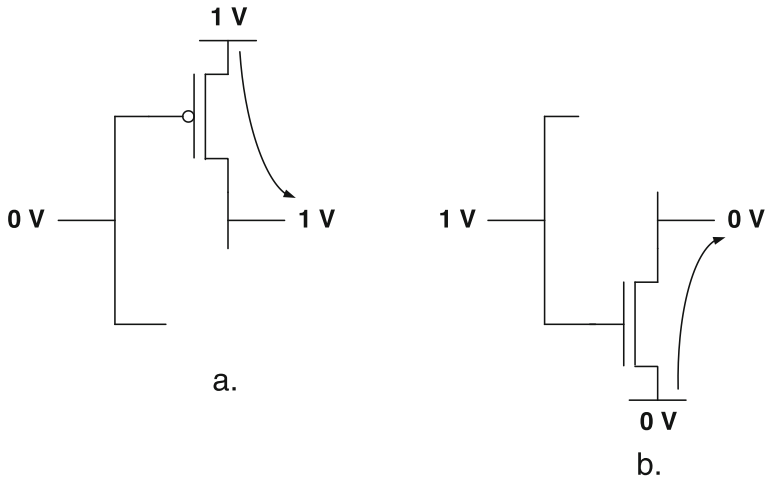


Fig. 1.17 Working of a CMOS inverter

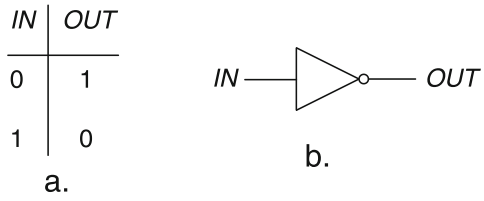


Fig. 1.18 Inverter: behavior and logic symbol

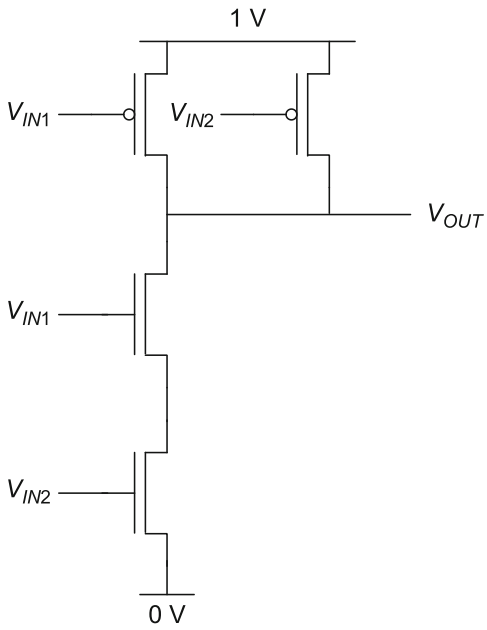


Fig. 1.19 2-Input NAND gate (NAND2 gate)

The conclusion of this analysis is that, as long as only binary signals are considered, the circuit of Fig. 1.16 inverts the input signal: it transforms V_L (0 V) into V_H (1 V) and V_H (1 V) into V_L (0 V). In terms of bits, it transforms 0 into 1 and 1 into 0 (Fig. 1.18a). As long as only the logic behavior is considered (the relation between input bits and output bits), the standard inverter symbol of Fig. 1.18b is used.

1.3.2.4 Other Components

With four transistors (Fig. 1.19) a 2-input circuit called NAND gate can be implemented. It works as follows:

- If $V_{IN1} = V_{IN2} = 1$ V then both *p*MOS switches are open and both *n*MOS switches are closed so that they transmit $V_L = 0$ V to the gate output (Fig. 1.20a).
- If $V_{IN2} = 0$ V, whatever the value of V_{IN1}, then at least one of the *n*MOS switches (connected in series) is open and at least one of the *p*MOS switches (connected in parallel) is closed, so that $V_H = 1$ V is transmitted to the gate output (Fig. 1.20b).
- If $V_{IN1} = 0$ V, whatever the value of V_{IN2}, the conclusion is the same.

Thus, the logic behavior of a 2-input NAND gate is given in Fig. 1.21a and the corresponding symbol is shown in Fig. 1.21b. The output of a 2-input NAND gate (NAND2) is equal to 0 if, and only if, both inputs are equal to 1. In all other cases the output is equal to 1.

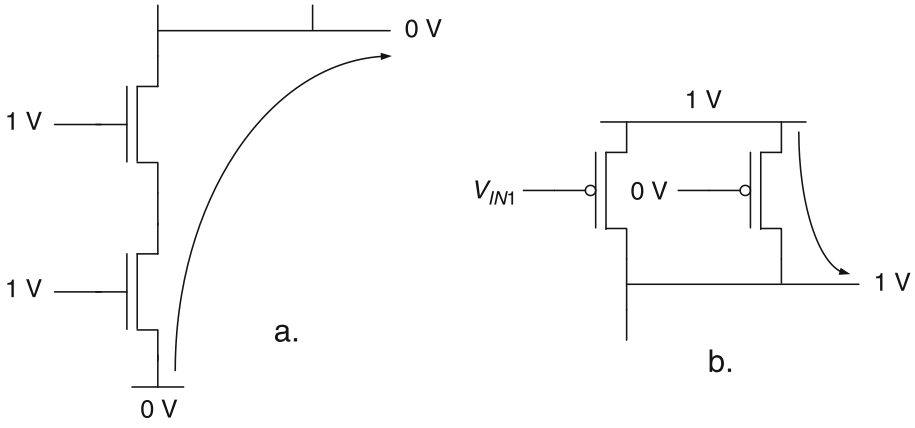


Fig. 1.20 NAND gate working

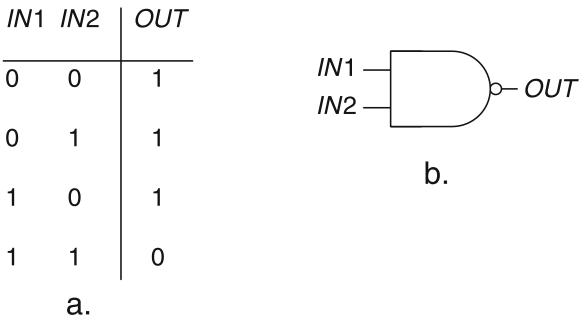


Fig. 1.21 2-Input NAND gate: behavior and symbol

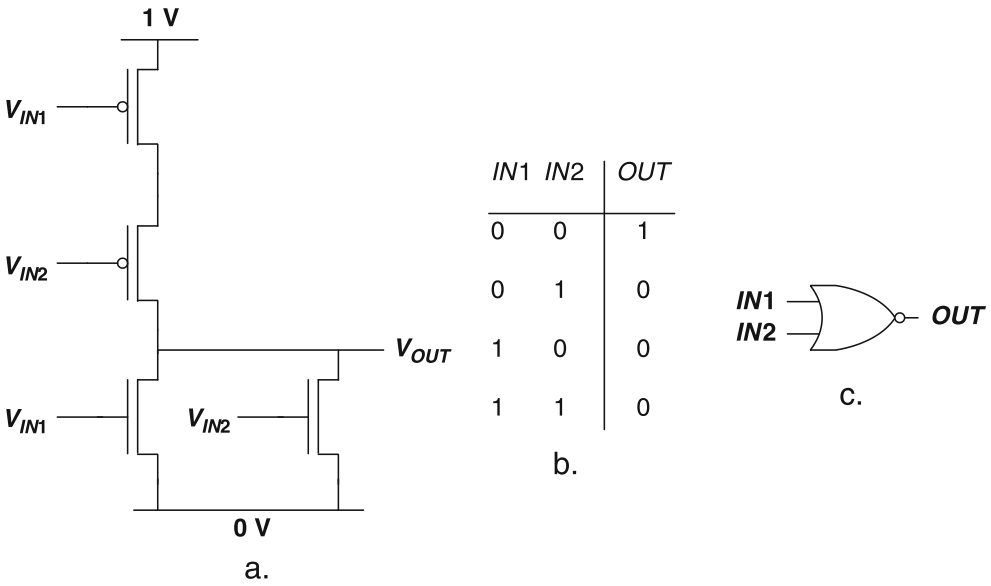


Fig. 1.22 NOR2 gate

Other logic gates can be defined and used as basic components of digital circuits. Some of them will now be mentioned. Much more complete information about logic gates can be found in classical books such as Floyd (2014) or Mano and Ciletti (2012).

The circuit of Fig. 1.22a is a 2-input NOR gate (NOR2 gate). If $V_{IN1} = V_{IN2} = 0$ V, then both p-type switches are closed and both n-type switches are open, so that $V_H = 1$ V is transmitted to the gate output. In all other cases at least one of the p-type switches is open and at least one of the n-type

switches is closed, so that $V_L = 0$ V is transmitted to the gate output. The logic behavior and the symbol of a NOR2 gate are shown in Fig. 1.22b, c.

NAND and NOR gates with more than two inputs can be defined. The output of a k-input NAND gate is equal to 0 if, and only if, the k inputs are equal to 1. The corresponding circuit (similar to Fig. 1.19) has k p-type transistors in parallel and k n-type transistors in series. The output of a k-input NOR gate is equal to 1 if, and only if, the k inputs are equal to 0. The corresponding circuit (similar to Fig. 1.22) has k n-type transistors in parallel and k p-type transistors in series. The symbol of a 3-input NAND gate (NAND3 gate) is shown in Fig. 1.23a and the symbol of a 3-input NOR gate (NOR3 gate) is shown in Fig. 1.23b.

The logic circuit of Fig. 1.24a consists of a NAND2 gate and an inverter. The output is equal to 1 if, and only if, both inputs are equal to 1 (Fig. 1.24b). It is a 2-input AND gate (AND2 gate) whose symbol is shown in Fig. 1.24c.

The logic circuit of Fig. 1.25a consists of a NOR2 gate and an inverter. The output is equal to 0 if, and only if, both inputs are equal to 0 (Fig. 1.25b). It is a 2-input OR gate (OR2 gate) whose symbol is shown in Fig. 1.25c.

AND and OR gates with more than two inputs can be defined. The output of a k-input AND gate is equal to 1 if, and only if, the k inputs are equal to 1, and the output of a k-input OR gate is equal to 0 if, and only if, the k inputs are equal to 0. For example, an AND3 gate can be implemented with a NAND3 gate and an inverter (Fig. 1.26a). Its symbol is shown in Fig. 1.26b. An OR3 gate can be implemented with a NOR3 gate and an inverter (Fig. 1.26c). Its symbol is shown in Fig. 1.26d.

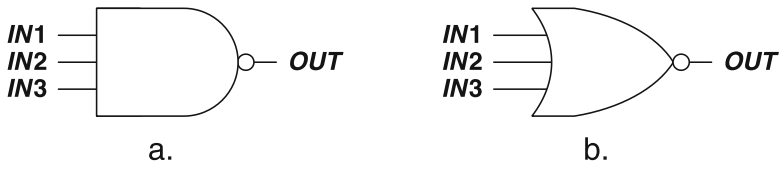


Fig. 1.23 NAND3 and NOR3

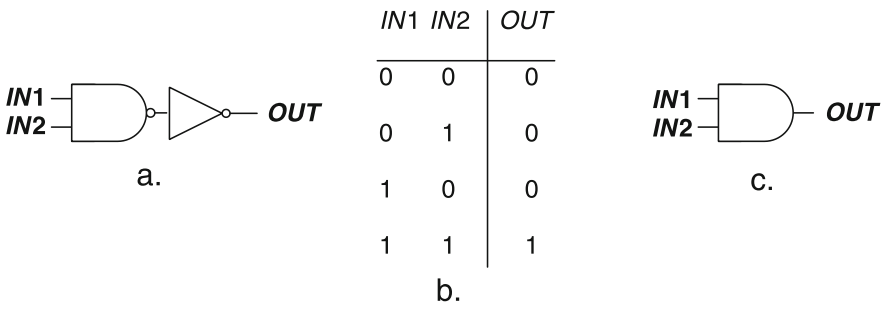


Fig. 1.24 AND2 gate

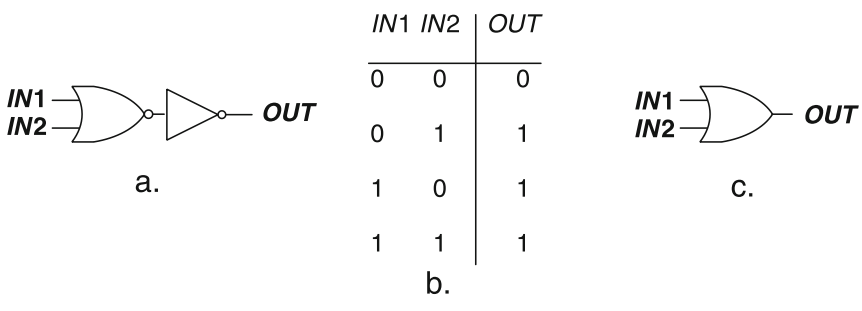


Fig. 1.25 OR2 gate

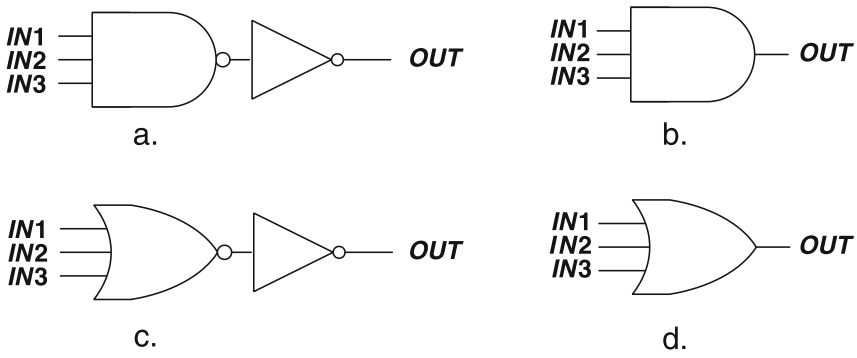


Fig. 1.26 AND3 and OR3 gates

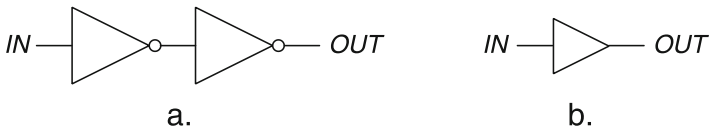


Fig. 1.27 Buffer

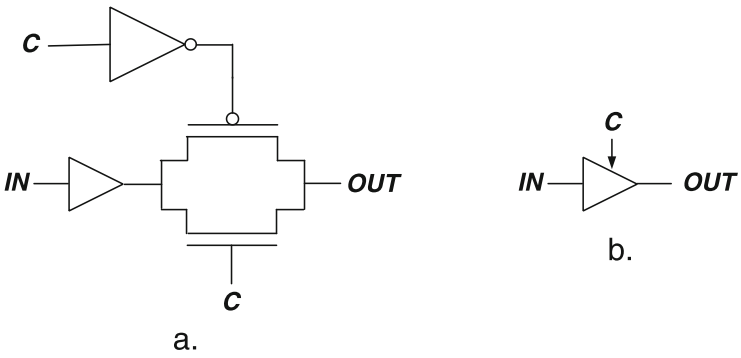


Fig. 1.28 3-State buffer

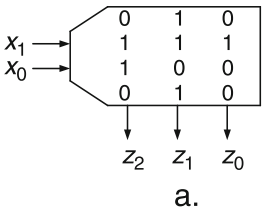
Buffers are another type of basic digital components. The circuit of Fig. 1.27a, made up of two inverters, generates an output signal equal to the input signal. Thus, it has no logic function; it is a power amplifier. Its symbol is shown in Fig. 1.27b.

The circuit of Fig. 1.28a is a 3-state buffer. It consists of a buffer, an inverter, a *p*MOS transistor, and an *n*MOS transistor. It has two inputs *IN* and *C* (control) and an output *OUT*. If $C = 0$, then both switches (*n*-type and *p*-type) are open, so that the output *OUT* is disconnected from the input *IN* (floating state or high impedance state). If $C = 1$, then both switches are closed, so that the output *OUT* is connected to the input *IN* through a good (*p*-type) switch if $IN = 1$ and through a good (*n*-type) switch if $IN = 0$. The 3-state buffer symbol is shown in Fig. 1.28b.

Other small-size components such as multiplexers, encoders, decoders, latches, flip flops, and others will be defined in the next chapters.

To conclude this section about digital components, an example of larger size component is given. Figure 1.29a is the symbol of a read-only memory (ROM) that stores four 3-bit words. Its behavior is specified in Fig. 1.29b: with two address bits x_1 and x_0 one of the four stored words is selected and can be read from outputs z_2, z_1, and z_0.

More generally, a ROM with N address bits and M output bits stores 2^N M-bit words (in total $M \cdot 2^N$ bits).



x_1	x_0	z_2	z_1	z_0
0	0	0	1	0
0	1	1	1	1
1	0	1	0	0
1	1	0	1	0

b.

Fig. 1.29 12-bit read-only memory ($3 \cdot 2^2$-bit ROM)

1.3.3 Synthesis of Digital Electronic Systems

The central topic of this course is the synthesis of digital electronic systems. The problem can be stated in the following way.

- On the one hand, the system designer has the specification of a system to be developed. Several specification methods have been proposed in Sect. 1.2.
- On the other hand, the system designer has a catalog of available electronic components such as logic gates, memories, and others, and might have access to previously developed and reusable subsystems. Some of the more common electronic components have been described in Sect. 1.3.2.

The designer work is the definition of a digital system that fulfils the initial specification and uses building blocks that belong to the catalog of available components or are previously designed subsystems. In a more formal way it could be said that the designer work is the generation of a hierarchical description whose final blocks are electronic components or reusable electronic subsystems.

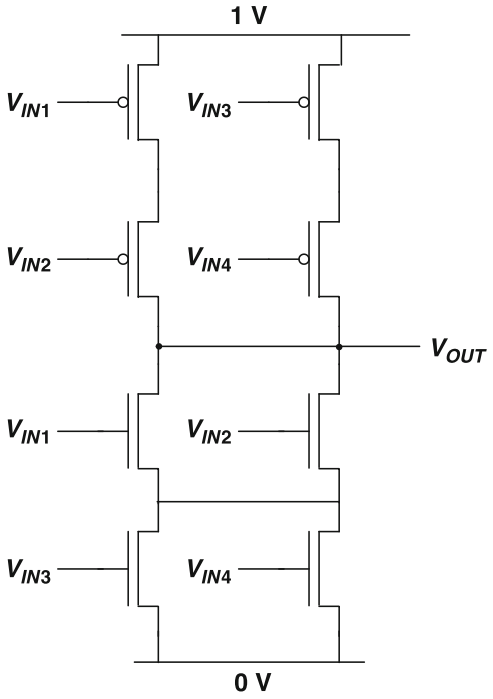
1.4 Exercises

1. The working of the chronometer of Example 1.2 can be specified by the following program in which the condition *ref_positive_edge* is assumed to be true on every positive edge of signal *ref*.

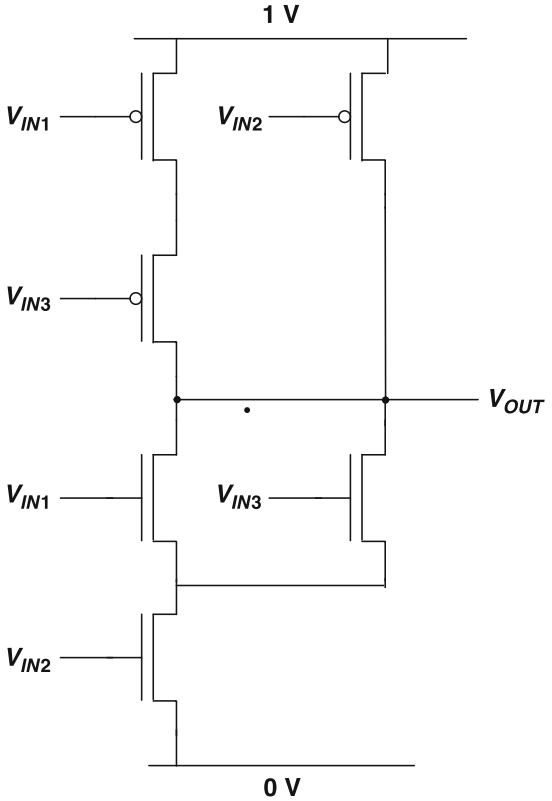
```
loop
  if reset = ON then h = 0; m = 0; s = 0; t = 0;
  elsif start = ON then
    while stop = OFF loop
      if ref_positive_edge = TRUE then
        update(h, m, s, t);
      end if;
    end loop;
  end if;
end loop;
```

The *update* procedure updates the values of h, m, s, and t every time that there is a positive edge on *ref*, that is to say every tenth of second. Generate a pseudo-code program that defines the *update* procedure.

2. Given two numbers X and $Y = y_3 \cdot 10^3 + y_2 \cdot 10^2 + y_1 \cdot 10 + y_0$, the product $P = X \cdot Y$ can be expressed as $P = y_0 \cdot X + y_1 \cdot X \cdot 10 + y_2 \cdot X \cdot 10^2 + y_3 \cdot X \cdot 10^3$. Generate a pseudo-code program based on the preceding relation to compute P.
3. Given two numbers X and $Y = y_3 \cdot 10^3 + y_2 \cdot 10^2 + y_1 \cdot 10 + y_0$, the product $P = X \cdot Y$ can be expressed as $P = (((y_3 \cdot X) \cdot 10 + y_2 \cdot X) \cdot 10 + y_1 \cdot X) \cdot 10 + y_0 \cdot X$. Generate a pseudo-code program based on the preceding relation to compute P.
4. Analyze the working of the following circuit and generate a 16-row table that defines V_{OUT} in function of V_{IN1}, V_{IN2}, V_{IN3}, and V_{IN4}.



5. Analyze the working of the following circuit and generate an 8-row table that defines V_{OUT} in function of V_{IN1}, V_{IN2}, and V_{IN3}.



References

Floyd TL (2014) Digital fundamentals. Prentice Hall, Upper Saddle River
Mano MMR, Ciletti MD (2012) Digital design. Prentice Hall, Boston
Rabaey JM, Chandrakasan A, Nikolic B (2003) Digital integrated circuits: a design perspective. Prentice Hall, Upper Saddle River
Weste NHE, Harris DM (2010) CMOS VLSI design: a circuit and systems perspective. Pearson, Boston