Chapter 2

Spring 2010 Edition

Gates: Basic Building Blocks of Digital Circuits

A SUMMARY of what you learned in Chapter 1:

1- What is a digital circuit?

2- Why may we prefer digital circuits to analog circuits?

3- Binary number system.

4- Truth tables: how to convert a problem description in natural language into a generic one called a truth table.

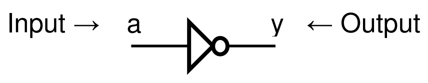
5- The physical meanings of logic values 0 and 1 in digital circuits.

In Chapter 1 we examined how to develop truth tables, but we do not know yet how to ***realize*** a truth table and come up with ***a real digital circuit*** that performs the intended function specified by the truth table. The next step to reach this goal is taken in this chapter. But to complete the design cycle we have to wait until Chapter 4.

In this chapter some electronic devices called *logic gates* are introduced. Logic gates are the *basic building blocks* of digital circuits. As you will see in the coming chapters, a digital circuit (such as an adder) is constructed as a physical network of properly interconnected logic gates; so that the terminal behavior of the resulting circuit will be exactly what is prescribed by the corresponding truth table.

NOT gate or inverter

The logic symbol, truth table (or definition) and two different algebraic notations for an inverter (with input a and output y) are shown in Figure 1*a*, Figure 1*b* and Figure 1*c*, respectively. In this book we use the second algebraic notation (y = a') as it is easier to type. (But the top notation is much more readable, especially in complex expressions.) Here the prime (') is an algebraic operator, a is its operand and a' is an expression. The algebraic notation of a gate is in fact a mathematical model for that gate. We will employ these models to mathematically manipulate logical expressions after *switching algebra* is introduced in Chapter 3. Remember from Chapter 1 that a and y are normally called *signals* in Figure 1*a*, but *variables* in Figure 1*b* or Figure 1*c*.



(a)

a	y
0	1
1	0

(b)

$y = \overline{a}$

$y = a'$

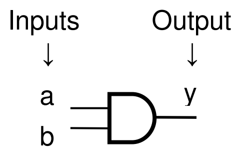
(c)

Figure 1. NOT gate: (a) logic symbol, (b) truth table, (c) algebraic notations

As illustrated by the truth table in Figure 1*b*, the output of an inverter is the opposite (complement) of its input. So, if we apply a 1 to an inverter, we will receive a 0 output. And if we apply a 0 to this gate, a 1 will be produced.

AND gate

The logic symbol, truth table and two different algebraic notations for an AND gate (with inputs a and b, and output y) are shown in Figure 2*a*, Figure 2*b* and Figure 2*c*, respectively. Use the second notation with care, and only when no confusion is caused, as the reader may not know whether ab is one variable, or by ab you mean a . b! Here the dot (.) is an algebraic operator, a and b are its operands, and a . b is an expression. The AND function is also called *logical multiplication*, although it performs the 1-bit-by-1-bit *traditional* multiplication as shown in Figure 2*b*.



a b	y
0 0	0
0 1	0
1 0	0
1 1	1

$y = a \cdot b$

$y = ab$

(a) (b) (c)

Figure 2. AND gate: (a) logic symbol, (b) truth table, (c) algebraic notations

As illustrated by the truth table, the output of an AND gate is 1 only if both inputs are 1; even one 0 input *pulls the output down* (i.e., makes the output 0).

Question: The output of an AND gate is 1. What can you tell about the two inputs of this gate?

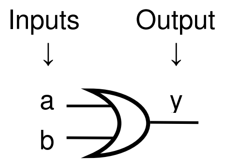
Answer: Both inputs are necessarily 1.

Question: The output of an AND gate is 0. What can you tell about the two inputs of this gate?

Answer: At least one of the inputs is 0.

OR gate

The logic symbol, truth table and algebraic notation for an OR gate (with inputs a and b, and output y) are shown in Figure 3*a*, Figure 3*b* and Figure 3*c*, respectively. The OR function is also called *logical addition*, which is the same as the traditional addition except for 1 + 1. As shown in the last row of the truth table, 1 + 1 equals 1, where + is logical addition (OR).



a b	y
0 0	0
0 1	1
1 0	1
1 1	1

$y = a + b$

(a) (b) (c)

Figure 3. OR gate: (a) logic symbol, (b) truth table, (c) algebraic notation

As illustrated by the truth table, the output of an OR gate is 0 only if both inputs are 0; even one asserted input *pulls the output up* (i.e., makes the output 1).

Question: The output of an OR gate is 0. What can you tell about the two inputs of this gate?

Answer: Both inputs are necessarily 0.

Question: The output of an OR gate is 1. What can you tell about the two inputs of this gate?

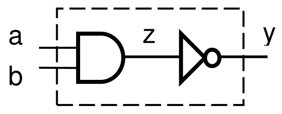
Answer: At least one of the inputs is 1.

Notice: You see an obvious *similarity* between an AND gate and an OR gate. This similarity will be elaborated on in Chapter 3.

NAND gate

What happens if we connect the output of an AND gate (z) to the input of an inverter, as shown in Figure 4*a*? Now, we have the composition of two functions, NOT of AND, resulting in a third function called NAND (which stands for **N**ot **AND**), with two inputs, a and b, arriving at the AND gate and one output, y, coming from the inverter. This output (y) is low whenever the output of the AND gate (z) is high.

Similarly, y is high whenever z is low. In other words, if the output column of AND's truth table is bit-wise complemented (i.e., 0 and 1 are swapped) as shown in Figure 4*b*, the resulting truth table will belong to a NAND gate.

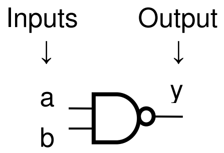


a b	z	y
0 0	0	1
0 1	0	1
1 0	0	1
1 1	1	0

(a) (b)

Figure 4. AND followed by NOT: (a) logic circuit, (b) truth table

The logic symbol (one AND followed by an *inversion bubble*), truth table and two different algebraic notations for a NAND gate (with inputs a and b, and output y) are shown in Figure 5*a*, Figure 5*b* and Figure 5*c*, respectively.



a b	y
0 0	1
0 1	1
1 0	1
1 1	0

$y = (a . b)'$

$y = \overline{a . b}$

(a) (b) (c)

Figure 5. NAND gate: (a) logic symbol, (b) truth table, (c) algebraic notations

As illustrated by the truth table, the output of a NAND gate is 0 only if both inputs are 1; even one 0 input pulls the output up.

Question: The output of a NAND gate is 0. What can you tell about the two inputs of this gate?

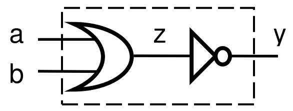
Answer: Both inputs are necessarily 1.

Question: The output of a NAND gate is 1. What can you tell about the two inputs of this gate?

Answer: At least one of the inputs is 0.

NOR gate

Let's connect the output (z) of an OR gate to the input of an inverter, as shown in Figure 6*a*. Again, we have the composition of two functions but this time NOT of OR, resulting in a third function called NOR (which stands for **N**ot **OR**) with two inputs, a and b, arriving at the OR gate and one output, y, coming from the inverter. This output is low whenever the output of the OR gate (z) is high. Similarly, y is high whenever z is low. In other words, if the output column of OR's truth table is bit-wise complemented (i.e., 0 and 1 are swapped) as shown in Figure 6*b*, the resulting truth table will belong to a NOR gate.

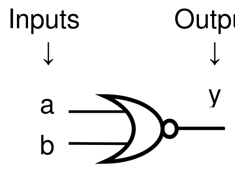


a b	z	y
0 0	0	1
0 1	1	0
1 0	1	0
1 1	1	0

(a) (b)

Figure 6. OR followed by NOT: (a) logic circuit, (b) truth table

The logic symbol (one OR followed by an inversion bubble), truth table and two different algebraic notations for a NOR gate (with inputs a and b, and output y) are shown in Figure 7*a*, Figure 7*b* and Figure 7*c*, respectively.



(a)

a b	y
0 0	1
0 1	0
1 0	0
1 1	0

(b)

$y = (a + b)'$

$y = \overline{a + b}$

(c)

Figure 7. NOR gate: (a) logic symbol, (b) truth table, (c) algebraic notations

As illustrated by the truth table, the output of a NOR gate is 1 only if both inputs are 0; even one 1 input pulls the output down.

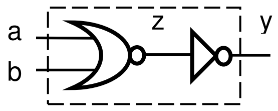
Question: The output of a NOR gate is 1. What can you tell about the two inputs of this gate?

Answer: Both inputs are necessarily 0.

Question: The output of a NOR gate is 0. What can you tell about the two inputs of this gate?

Answer: At least one of the inputs is 1.

Let's see what happens if a NOR gate is followed by an inverter, as shown in Figure 8*a*.



(a)

a b	z	y
0 0	1	0
0 1	0	1
1 0	0	1
1 1	0	1

(b)

Figure 8. NOR followed by NOT: (a) logic circuit, (b) truth table

Now, we have NOT of NOR. Using the procedure illustrated in Figure 6*b*, we will come up with the truth table shown in Figure 8*b*. But this is the truth table of an OR gate (ignore z); therefore, NOT of NOR becomes OR. This conversion is graphically illustrated in Figure 9, where a NOR gate is first replaced with an OR gate followed by an inverter, and then the two inverters are removed. As shown in this composition, *two consecutive inverters cancel out each other*.

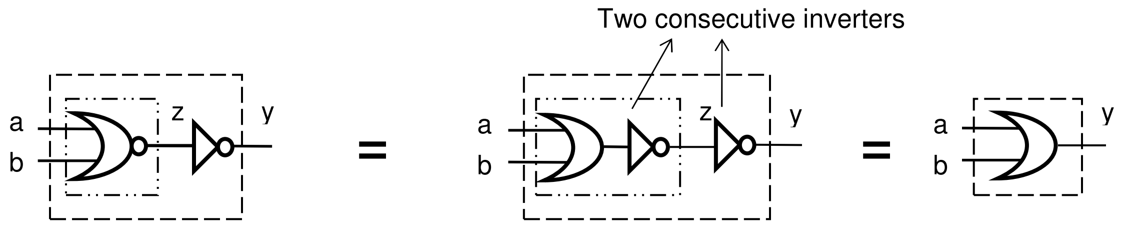


Figure 9. NOR followed by NOT becomes OR

In a similar way, it can be shown that a NAND gate followed by an inverter is equivalent to an AND gate as shown in Figure 10.

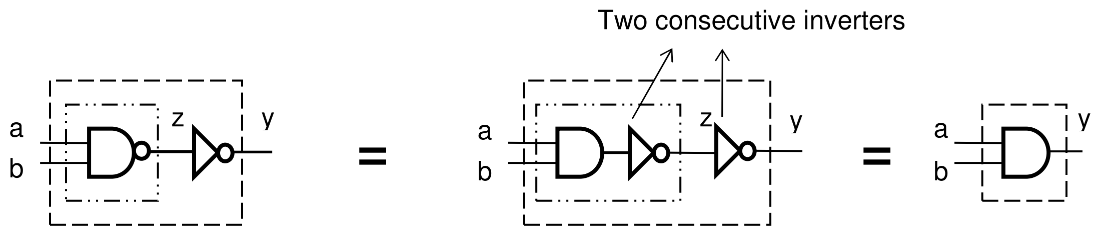


Figure 10. NAND followed by NOT becomes AND

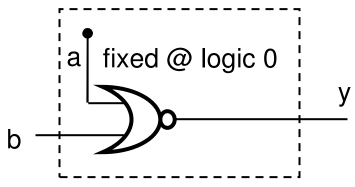
Question: Which gate, NOR or OR, is *undecomposable* into simpler gates? In other words, which one is built first, NOR or OR?

Answer: What normally happens is illustrated in Figure 9, i.e., a NOR gate is first realized (say in CMOS[1] technology) and then it is converted to an OR gate by adding an inverter. Similarly, a NAND gate is built first and then converted to an AND gate by inverting the output. We will see simplified electronic models for CMOS NAND and CMOS NOR gates in the last section of this chapter.

Example 1. Convert a NOR gate into an inverter.

There are two different ways for this conversion:

Method 1. Let's connect one of the inputs of a NOR gate, say a, to logic 0 as shown in Figure 11*a*. This means that the last two rows of NOR's truth table in which a = 1 are not relevant anymore and can be deleted, as shown in Figure 11*b*. The remaining part of the truth table is shown in Figure 11*c*. Since a does not change in this table, we can remove a to reach the truth table shown in Figure 11*d*. But this is the truth table of an inverter with input b and output y. In other words, the circuit shown in Figure 11*b* is an inverter.



(a)

a b	y
0 0	1
0 1	0
~~1 0~~	~~0~~
~~1 1~~	~~0~~

(b)

a b	y
0 0	1
0 1	0

(c)

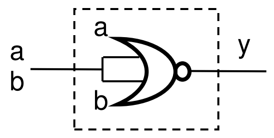
b	y
0	1
1	0

(d)

Figure 11. NOR gate is converted to inverter, method 1

Method 2. Now let's tie the two inputs, a and b, of a NOR gate to each other as shown in Figure 12*a*. Since a = b, the second and third rows of NOR's truth table in which a ≠ b are not relevant anymore and can be crossed out, as shown in Figure 12*b*. The remaining part of the truth table is shown in Figure 12*c*. Since a = b we can remove one of the input columns to reach the truth table shown in Figure 12*d*. But this is again the truth table of an inverter, with a or b (two different names for the same signal) as the input, and y as the output. In other words, the circuit shown in Figure 12*b* is an inverter.

[1] CMOS stands for **c**omplementary **m**etal-**o**xide **s**emiconductor, and is the dominant technology used in today's revolutionary and gigantic chips with tens of millions of transistors commercially available in the market. Transistors are electronic building blocks of logic gates. It is interesting to notice that a transistor could be as small as 65 nm in length in today's CMOS technology, where one nm is 10^{-9} meter!



(a)

a b	y
0 0	1
~~0 1~~	~~0~~
~~1 0~~	~~0~~
1 1	0

(b)

a b	y
0 0	1
1 1	0

(c)

a = b	y
0	1
1	0

(d)

Figure 12. NOR gate is converted to inverter, method 2

Through a similar approach we can prove that each of the circuits shown in Figure 13*a* and Figure 13*b* is functionally equivalent to an inverter.

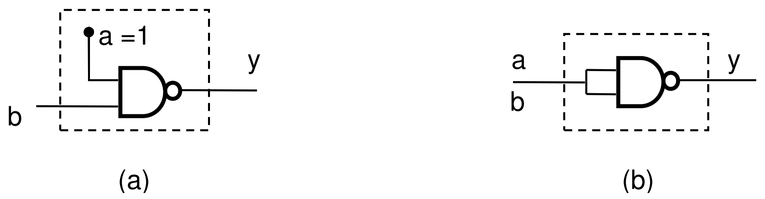
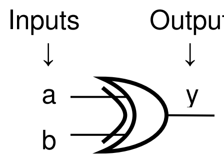


Figure 13. NAND gate is converted to inverter: (a) method 1, (b) method 2

Exclusive OR (XOR) gate

The symbol, truth table and algebraic notation for an XOR gate (with inputs a and b, and output y) are shown in Figure 14*a*, Figure 14*b* and Figure 14*c*, respectively.



(a)

a b	y
0 0	0
0 1	1
1 0	1
1 1	0

(b)

$y = a \oplus b$

(c)

Figure 14. XOR gate: (a) logic symbol, (b) truth table, (c) algebraic notation

The name is self-descriptive: in XOR the output goes up only if exactly one input is at logic 1 (compare it with OR). Take a closer look at the truth table. You see that this gate in fact *compares* the two input bits, and determines whether or not they are equal. Equal inputs produce a 0 output. An asserted output signifies different inputs.

Question: The output of an XOR gate is 0. What can you tell about the two inputs of this gate?

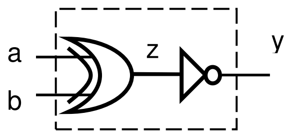
Answer: The inputs have the same value.

Question: The output of an XOR gate is 1. What can you tell about the two inputs of this gate?

Answer: The inputs have different values.

Exclusive NOR (XNOR) gate

If an XOR gate is followed by an inverter (i.e., negated), the resulting gate is called Exclusive NOR or XNOR for short. Therefore, if the output column of XOR's truth table is bit-wise complemented (0 and 1 are swapped) as shown in Figure 15*b*, the resulting truth table will belong to XNOR. This means that XNOR is a comparator as well; the output is pulled *up* if the inputs are *equal*, otherwise it is pulled down (compare this gate with XOR). In other words, the output of XNOR is pulled down only if *exactly one* input is high (compare it with NOR).



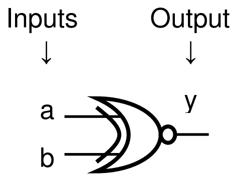
(a)

a b	z	y
0 0	0	1
0 1	1	0
1 0	1	0
1 1	0	1

(b)

Figure 15. XOR followed by NOT: (a) logic circuit, (b) truth table

The logic symbol (one XOR followed by an inversion bubble), truth table and two different algebraic notations for an XNOR gate (with inputs a and b, and output y) are shown in Figure 16*a*, Figure 16*b* and Figure 16*c*, respectively.



(a)

a b	y
0 0	1
0 1	0
1 0	0
1 1	1

(b)

$y = \overline{a \oplus b}$

$y = a \odot b$

(c)

Figure 16. XNOR gate: (a) logic symbol, (b) truth table, (c) algebraic notations

Question: The output of an XNOR gate is 1. What can you tell about the two inputs of this gate?

Answer: The inputs have the same value.

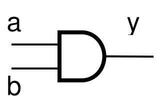
Question: The output of an XNOR gate is 0. What can you tell about the two inputs of this gate?

Answer: The inputs have different values. ◊

All the binary functions discussed above are *commutative*; for example *a* **.** *b* = *b* **.** *a*. By a *binary* function we mean a function with *two* inputs. Therefore, the NOT function is not binary; it is called *unary*.

A Different Look at Gates

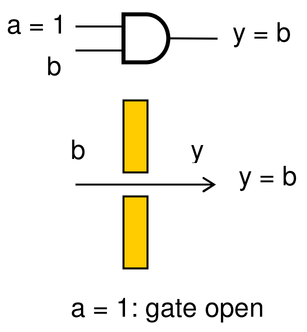
The logic symbol and truth table of an AND gate (with inputs a and b, and output y) are again shown in Figure 17*a* and Figure 17*b*, respectively, for ease of reference. Let's take a different look at this gate. As the first two rows of the truth table read, if a = 0 then y = 0, no matter what b is. However, if a = 1, then y = b, i.e., b is copied onto y if a = 1, as shown by the last two rows of the truth table. According to this interpretation, the device is similar to a mechanical *gate* (with one input, b, and one output, y) which can be either open (on) if a = 1 or closed (off) if a = 0. Input data at b passes through an open gate and appears at the output as shown in Figure 17*c*. However, in a closed gate, no matter what b is, the output is stuck at 0 (see Figure 17*d*), i.e., the signal at b cannot pass through the closed gate. This is called signal *gating*: preventing a signal from further propagation.



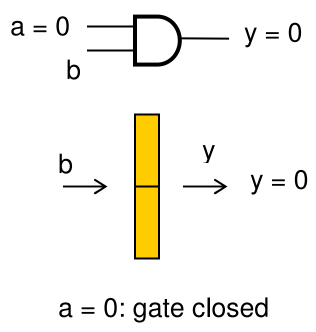
(a)

a b	y
0 0	0
0 1	0
1 0	0
1 1	1

(b)



(c)



(d)

Figure 17. AND gate: (a) logic symbol, (b) truth table, (c) open gate, (d) closed gate

This interpretation of an AND gate has been tabulated in a *compressed* truth table in Figure 18*a*.

a b	y
0 x	0
1 x	b

(a)

a b	y
x 0	0
x 1	a

(b)

Figure 18. AND gate's compressed truth table: (a) input a is controller, (b) input b is controller

A new notation, x, (read it *don't care*) has been introduced in this table, which has fewer rows than a traditional truth table has. (There are only two rows in Figure 18*a*, while considering the number of variables (2), a traditional truth table needs four rows.) That is why we call the table in Figure 18*a* (and similar tables) a compressed truth table. Here in Figure 18*a* the first row (which represents the first *two* rows of the original uncompressed truth table in Figure 17*b*) reads: if a = 0, then y = 0, no matter what b is. The second row in Figure 18*a* (which represents the second *two* rows of the original uncompressed truth table in Figure 17*b*) reads: if a = 1, then y = b (i.e., b is copied to y). So, if in one row (of a compressed truth table) a don't care is assigned to an input variable, it means that both logic values (0 and 1) are valid for that particular variable in that specific row. In other words, if a row in a (compressed) truth table has one don't care entry, that row is an *abbreviation* for two rows in the corresponding uncompressed truth table. Therefore, we may consider the compressed truth table in Figure 18*a* as a shorthand notation for the original uncompressed truth table in Figure 17*b*. We can of course swap a and b to come up with the compressed table shown in Figure 18*b*.

Example 2. Read the compressed truth table shown in Figure 19, and then convert it into a traditional truth table.

The first row reads: if a = 0, then b is copied onto y, i.e., y = b.
The second row reads: if a = 1, then no matter what b is, y will be pulled up.

a b	y
0 x	b
1 x	1

Figure 19. Compressed truth table in Example 2

In order to expand a compressed truth table and reach a traditional one, each row with a x entry is duplicated and then one of the two x entries is replaced with a 0 and the other one is replaced with a 1. Take a close look at Figure 20. The compressed table of Figure 19 is first split into two rows and then, following the procedure just mentioned, each row is expanded into two new rows or partial truth tables as shown in Figure 20*a* and Figure 20*b*. These two tables are eventually combined to reach the truth table of Figure 20*c*, which describes an OR gate. Therefore, the compressed truth table in Figure 19 belongs to OR.

(a)

a b	y
0 x	b

a b	y
0 0	0
0 1	1

(b)

a b	y
1 x	1

a b	y
1 0	1
1 1	1

(c)

a b	y
0 0	0
0 1	1
1 0	1
1 1	1

Figure 20. Table expansion: (a) first row, (b) second row, (c) traditional truth table

The concept of open/closed AND gates can be generalized to NAND gates. A NAND gate is closed if one of its inputs is tied to logic 0. The output of a closed NAND is stuck at 1. A 1 applied to an input will open the NAND gate. But as the input signal passes through a NAND, the signal is inverted before it is placed at the output.

Similar to AND gates, an OR gate can also be imagined as a mechanical gate, provided that appropriate duality is taken into consideration: Now the gate is open when a = 0 as shown in Figure 21*a*, and it is closed when a =1 as illustrated in Figure 21*b*. Notice that the output of a closed OR gate is always 1.

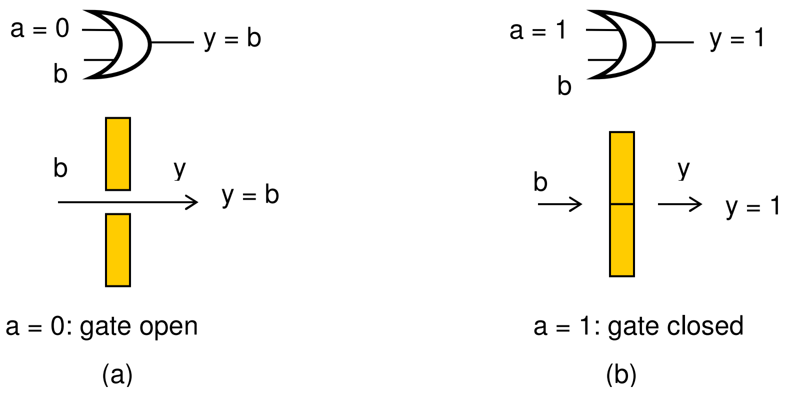


Figure 21. OR gate: (a) open gate, (b) closed gate

The concept of open/closed OR gates can be generalized to NOR gates. A NOR gate is closed if one of its inputs is tied to logic 1. The output of a closed NOR is stuck at 0. A 0 applied to an input will open the NOR gate. But as the input signal passes through a NOR, the signal is inverted before it is placed at the output.

Let's take a second look at XOR's truth table that is shown again in Figure 22*a* for ease of reference. The first two rows read: if a = 0, then y = b, i.e., b is copied to y as illustrated in Figure 22*b*. The second two rows read: if a = 1 then y = b', i.e., the gate becomes an inverter, as shown in Figure 22*c*. A compressed truth table describing an XOR gate is shown in Figure 22*d*.

(a)

a b	y
0 0	0
0 1	1
1 0	1
1 1	0

(b)

a b	y
0 0	0 = b
0 1	1 = b

y = b if a = 0

(c)

a b	y
1 0	1 = b'
1 1	0 = b'

y = b' if a = 1

(d)

a b	y
0 x	b
1 x	b'

y = b if a = 0
y = b' if a = 1

Figure 22. XOR: (a) truth table, (b) first 2 rows, (c) last 2 rows, (d) compressed truth table

Logic symbols have been used to illustrate the above concepts in Figure 23. Figure 23*a* shows an XOR gate with one arbitrary input, say a, called control. If control is 1, the gate will functionally be equivalent to an inverter as shown in Figure 23*b*. Otherwise (if control = 0), the XOR gate will functionally be equivalent to a piece of *one-directional* wire that *digitally* connects input b to output y, as shown in Figure 23*c*. Therefore, we may interpret an XOR gate as a *programmable inverter*. A one-directional wire (as opposed to two-directional or traditional wire) has one input and one output, such as b and y, respectively, in Figure 23*c*. Signals on such a wire can only flow from the input to the output; signal flow in the opposite direction is impossible. A digital wire (as opposed to an analog or traditional wire) can propagate only logic values (0 and 1); the voltage values at the two ends of such a wire are not necessarily identical. Back to Figure 23*c* and in TTL technology, if the voltage on b is 4.3 volts (logic 1), the voltage on y could be, say, 4.2 volts, although it is still interpreted as logic 1.

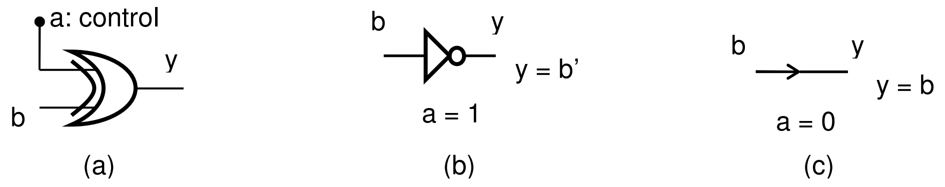


Figure 23. Different look at XOR gate: (a) control and data inputs, (b) XOR behavior when a = 1, (c) XOR behavior when a = 0.

Timing Diagrams

Remember that in digital circuits (comprised of gates) the values of input signals vary with time[2], resulting in time-varying output signals for different gates. For example, if two time-varying signals, a and b, shown in Figure 24, are applied to an AND gate, then a time-varying output, z, will be produced as shown in this figure. In this example input b goes up at time = t_1 while a is low, and then a goes up at time = t_2 while b is high, and so on. At any given instant of time, such as t_5, the logic values at a, b and z (1, 0 and 0, respectively) have to be consistent with the truth table of this gate (@t_5: 1 . 0 = 0). Pay attention to those input signal transitions (causes) that cause output signal transitions (effects). For example, the signal transition at t_2 at input a is the cause for the signal transition (effect) at t_2 at output z. In this ideal model the effect *immediately* follows the cause, as seen in Figure 24.

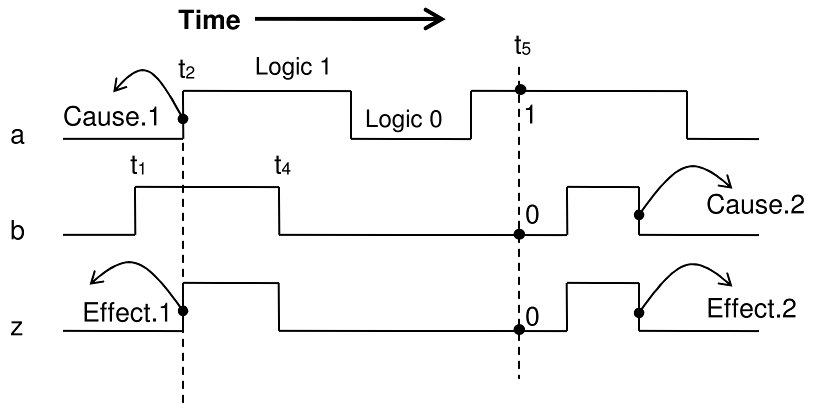


Figure 24. Timing diagram for ideal AND gate

[2] The rate of this change can easily reach hundreds of millions of transitions per second in today's technologies.

The real behavior of physical gates, however, is different from the ideal behavior illustrated in Figure 24[3]. In our AND example (and also other physical systems) there is always some delay between a cause (such as the signal transition at time = t_2 at a) and its effect (the signal transition at time = t_3 at z) as shown in Figure 25. Now, z (effect) is pulled up at $t_3 = t_2 + \Delta t$, i.e., Δt after a has gone up, where Δt is called the *propagation delay time* of this gate. During this time period (Δt), the circuit is in a *transient state*. When a transient state ends, the circuit reaches a steady (stable) state. In other words, it takes some (short) amount of time for a circuit to leave its transient state and stabilize. During this short period of time the truth table may be violated. For example, between two time instants t_2 and t_3, the timing diagram reads 1 . 1 = 0, which is not what we expect from an AND gate. Therefore, truth tables may not be valid during *transient states* of digital circuits.

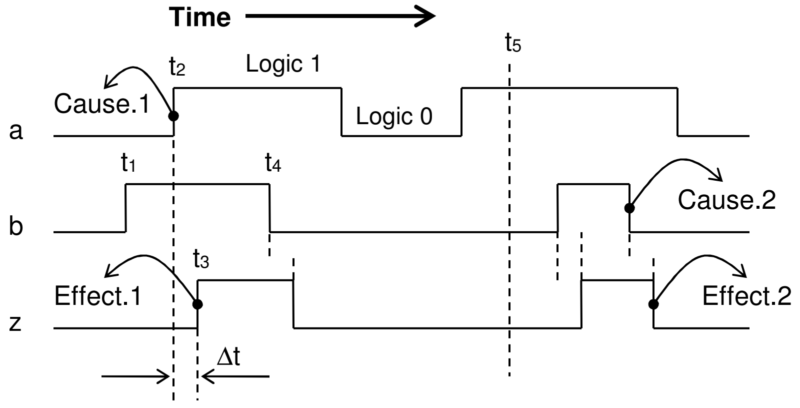


Figure 25. Timing diagram for real AND gate

Three- or More-input Gates and Analysis of Simple Digital Circuits

All of the physical devices that we have studied so far (except for inverters) have two inputs, but the concepts can be generalized to gates with three or more inputs. The functions corresponding to these gates are obviously not binary anymore, as they need more than two operands.

An n-input AND gate has an asserted output only if all inputs are asserted; otherwise the output is pulled down.

An n-input OR gate has a deasserted output only if all inputs are deasserted; otherwise the output is pulled up.

An n-input NAND gate is functionally equivalent to an n-input AND gate followed by an inverter. Therefore, an n-input NAND gate has a deasserted output only if all inputs are asserted; otherwise the output is pulled up.

An n-input NOR gate is functionally equivalent to an n-input OR gate followed by an inverter. Therefore, an n-input NOR gate has an asserted output only if all inputs are deasserted; otherwise the output is pulled down.

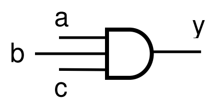
An n-input XOR gate has an asserted output only if it has an *odd* number of 1 inputs; otherwise the output is pulled down.

An n-input XNOR gate has an asserted output only if it has an *even* number of 1 inputs; otherwise the output is pulled down.

[3] Another ideal behavior illustrated in this timing diagram is the *instantaneous* signal transitions. In real circuitry signals always rise and fall with a finite slope. This, however, does not concern us in this introductory book.

Example 3. 3-input AND gates

The symbol, truth table (definition) and two different algebraic notations for this gate are shown in Figure 26*a*, Figure 26*b* and Figure 26*c*, respectively. As mentioned above, in this table there is exactly one row (a b c = 1 1 1) with an asserted output (y = 1). Therefore, even a single zero at one of the inputs will pull the output down.



(a)

Row	a b c	y
0	0 0 0	0
1	0 0 1	0
2	0 1 0	0
3	0 1 1	0
4	1 0 0	0
5	1 0 1	0
6	1 1 0	0
7	1 1 1	1

(b)

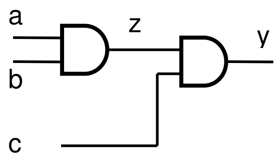
$y = a \, . \, b \, . \, c$

$y = abc$

(c)

Figure 26. Three-input AND gate: (a) logic symbol, (b) truth table, (c) algebraic notations

Now consider 2 two-input AND gates cascaded as shown in Figure 27*a*. Soon, we will prove that this circuit is functionally equivalent to a three-input AND gate (shown in Figure 26*a*). In Figure 27*a* the *output* of one gate becomes an *input* for the following gate. This is similar to what we examined before, such as the circuit shown in Figure 8. We always need this type of interconnection to build larger digital circuits from smaller ones.



(a)

Row	a b c	z	y
0	0 0 0	0	
1	0 0 1	0	
2	0 1 0	0	
3	0 1 1	0	
4	1 0 0	0	
5	1 0 1	0	
6	1 1 0	1	
7	1 1 1	1	

(b)

Row	a b c	z
0	0 0 x	0
1	0 1 x	0
2	1 0 x	0
3	1 1 x	1

(c)

Figure 27. Two cascaded AND gates: (a) logic circuit, (b) truth table for z, (c) compressed truth table for z

Question: When are two digital circuits functionally equivalent?

Answer: When they have identical truth tables.

Therefore, what we need here is to obtain the truth table of the circuit in Figure 27*a* and then check whether or not this table is identical to the table shown in Figure 26*b*.

Let's obtain the truth table of the circuit shown in Figure 27*a*. What we know about this table is that it has $2^3 = 8$ rows because there are three input signals (a, b and c) involved in this circuit. On the other hand, the final output, y, in Figure 27*a* is clearly a function of z and input c. Therefore, we draw an 8-row table with two output columns, z and y, as shown in Figure 27*b*. Since we don't know yet how y behaves, let's fill out column y later. However, we already know how z behaves: z is the output of an AND gate with inputs a and b, i.e., z = a . b. So, z is asserted only if both a and b are high (rows 6 and 7 in Figure 27*b*). Obviously, z is not a function of c; that is why every two adjacent rows (starting with row 0) in Figure

27*b* have identical z entries. This fact (that z is a function of only a and b, and not a function of c) has been highlighted in Figure 27*c*, which shows a compressed version of the original truth table. As a reminder, row 2 in this table reads: if a = 1 and b = 0, then z becomes 0, no matter what c is. We call a signal, such as z, an *intermediate* signal if it is neither the final output of the circuit, y, nor an input coming from the outside world, such as a in Figure 27*a*.

Our final goal is to determine y, which is the output of an AND gate with inputs c and z: y = c . z (see Figure 27*a*); therefore column y of the truth table in Figure 27*b* should be filled out accordingly. This table has been redrawn and column y has been filled out in Figure 28*a* for ease of reference. Remember that y is asserted only if both z and c are asserted. And this happens in row 7 only, as you see in Figure 28*a*. Now we need to compare this table with the truth table of a three-input AND gate, which is shown again in Figure 28*b* for ease of reference. As you see the two tables happen to have identical entries for y; hence the logic circuit in Figure 27*a* is functionally equivalent to a three-input AND gate (shown in Figure 26*a*).

Row	a b **c**	**z**	y
0	0 0 **0**	**0**	0
1	0 0 **1**	**0**	0
2	0 1 **0**	**0**	0
3	0 1 **1**	**0**	0
4	1 0 **0**	**0**	0
5	1 0 **1**	**0**	0
6	1 1 **0**	**1**	0
7	1 1 **1**	**1**	1

(a)

Row	**a b c**	**y**
0	0 0 0	0
1	0 0 1	0
2	0 1 0	0
3	0 1 1	0
4	1 0 0	0
5	1 0 1	0
6	1 1 0	0
7	1 1 1	1

(b)

Figure 28. (a) Truth table for two cascaded AND gates, (b) truth table for 3-input AND gate

Question: In Figure 27*a* can we algebraically express y in terms of inputs a, b and c?

Answer: Yes, we can. This is in fact the simplest problem in this category. However, the procedure is the same no matter how complicated the circuit is. Starting with the gate that generates y, write an algebraic equation for y:

y = z . c (1)

Now substitute for z with its algebraic expression, z = a . b.

y = (a . b) . c (2)

(In general, we repeat this procedure until all the variables on the right of equation belong to the set of input variables, i.e., until all intermediate variables, such as z in (1), are eliminated.)

Since all permutations of input variables (a, b and c) in Figure 27*a* result in the same truth table, the AND function is *associative*, i.e., (a . b) . c = a . (b . c). The logic circuit realizing the righthand side of this equation, a . (b . c), is illustrated in Figure 29.

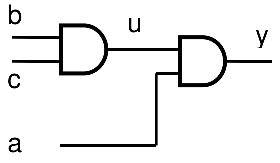


Figure 29. Another input permutation for cascaded AND gates

Similar conclusions are valid for also OR and XOR gates as summarized in Figure 30 and Figure 31, respectively.

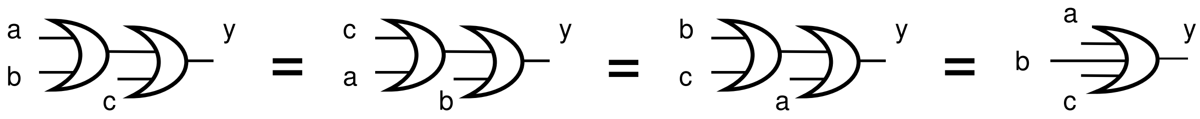


Figure 30. Two cascaded two-input OR gates are functionally equivalent to a 3-input OR gate

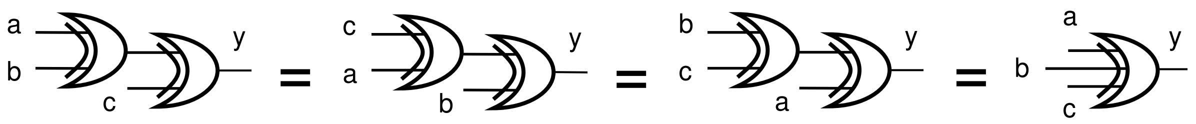


Figure 31. Two cascaded two-input XOR gates are functionally equivalent to a 3-input XOR gate

In a similar way, we may use smaller gates to realize AND, OR and XOR functions with four or more inputs. Three logic circuits equivalent to a five-input NAND, a five-input OR and a five-input XOR are shown in Figure 32*a*, Figure 32*b* and Figure 32*c*, respectively. These three circuits have different configurations to highlight the fact that to reach a large gate, smaller gates with different sizes can be cascaded in any order. To make an optimal choice one important criterion is the propagation delay time. Remember that the more gates are placed in series, the longer the propagation delay time is.

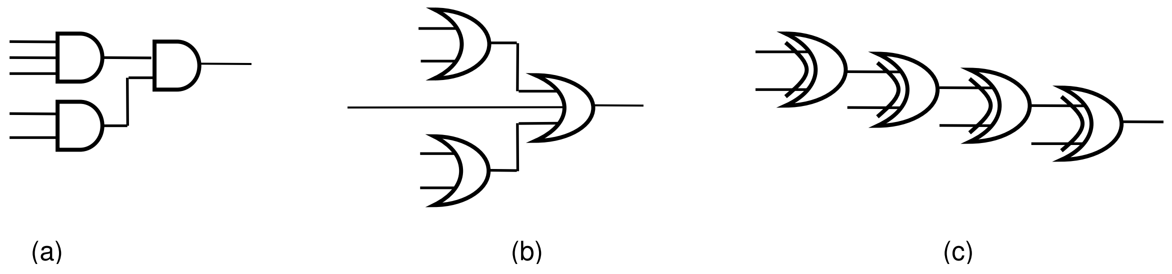


Figure 32. (a) five-input AND, (b) five-input OR, (c) five-input XOR

However, the above conclusions are not true for NOR, NAND or XNOR gates. For example, the logic circuits shown in Figure 33*a* and Figure 33*b* are not functionally equivalent. Furthermore, none of them are functionally equivalent to a three-input NAND gate. Why? The answer is "because they do not have identical truth tables". Remember that checking the truth tables is one straightforward way to verify the equivalence of two logic circuits.

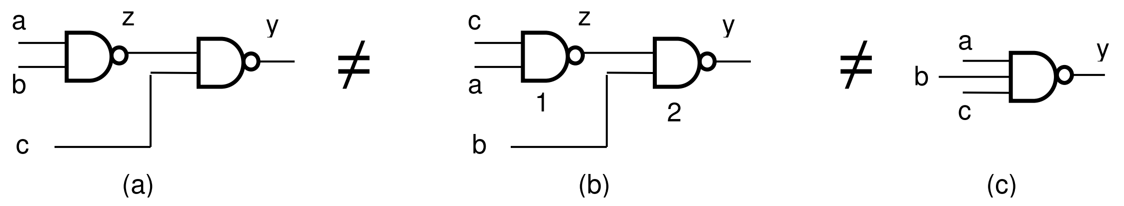


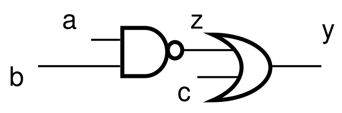
Figure 33. Three NON-equivalent logic circuits

To prove nonequivalence, we do not have to obtain the whole truth tables; if we find an input combination that generates two *different* outputs in two different circuits, then we have proved that the two circuits do not have the same truth tables, hence are not equivalent. For example, input combination abc = 110, asserts y in Figure 33*a* and Figure 33*c*, while the same combination pulls y down in Figure 33*b*. The conclusion is that both circuits in Figure 33*a* and Figure 33*c* are functionally different from the circuit shown in Figure 33*b*. Now consider input combination abc = 111, which asserts y in Figure 33*a*, while the output is pulled down in Figure 33*c*, and this means that the circuit in Figure 33*a* is not functionally equivalent to the circuit in Figure 33*c* either. And the final conclusion is what we see in Figure 33: the three logic circuits are functionally different from each other.

Let's take a closer look at how input combination abc = 110 (for example) pulls y down in Figure 33*b*. Since ac = 10 is applied to NAND gate 1, the output, z, of this gate becomes high. (See NAND's truth table in Figure 5*b*.) Now consider NAND 2 with two asserted inputs, z and b, and remember that the output of a NAND gate with two asserted inputs is deasserted as shown in Figure 5*b*. Therefore, in Figure 33*b* input combination abc = 110 deasserts the output.

Example 4. Obtain the logic expression and truth table for the logic circuit shown in Figure 34*a*.

Remember we have already solved a similar problem for the circuit shown in Figure 27*a*. The same procedure is valid for any logic circuit.



(a)

Row	a b c	z = (a . b)'	y = z + c = (a . b)' + c
0	0 0 0	1	1
1	0 0 1	1	1
2	0 1 0	1	1
3	0 1 1	1	1
4	1 0 0	1	1
5	1 0 1	1	1
6	1 1 0	0	0
7	1 1 1	0	1

(b)

Figure 34. Example 4: (a) two-gate logic circuit, (b) truth table

Start with

y = z + c (3)

Eliminate z from (3). Notice that z is the output of a NAND gate.

z = (a . b)' (4)

Substitute for z from (4) into (3)

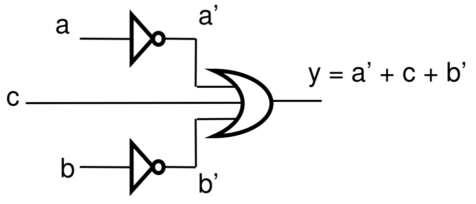
y = (a . b)' + c (5)

Since y is a function of three variables (a, b and c), the truth table of this circuit has 2^3 = 8 rows, as shown in Figure 34*b*. Again, in this truth table there is a column for the intermediate signal z, which is the output of a NAND gate, z = (a . b)'; so column z is filled out accordingly. And finally column y is filled out based on equation (3).

Example 5. Prove that the logic circuits shown in Figure 34*a* and Figure 35*a* are functionally equivalent; in other words prove that

(a . b)' + c = a' + b' + c

where the left-side expression corresponds to Figure 34*a* as obtained in (5), and the right-side expression corresponds to Figure 35*a*, as shown in this figure.



(a)

Row	a b c	a'	b'	a' + b' + c
0	0 0 0	1	1	1
1	0 0 1	1	1	1
2	0 1 0	1	0	1
3	0 1 1	1	0	1
4	1 0 0	0	1	1
5	1 0 1	0	1	1
6	1 1 0	0	0	0
7	1 1 1	0	0	1

(b)

Figure 35. Example 5: (a) logic circuit, (b) truth table

Remember that two functionally equivalent logic circuits have identical truth tables. So, we first need to obtain the truth table of the logic circuit shown in Figure 35*a*. Following the procedure carried out for the previous example, the truth table is obtained in Figure 35*b*. Since this table is identical to the table of Figure 34*b*, it is proved that the two logic circuits are functionally equivalent.

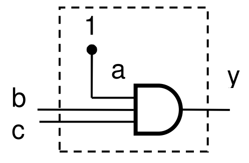
Unused inputs

We have already leaned how to use gates to obtain larger gates (with more inputs). Now we do the opposite and see how we may use, say, a 3-input AND gate as a 2-input AND gate.

The truth table of a three-input AND gate is illustrated again in Figure 36*a* with four highlighted rows, 4, 5, 6, and 7 corresponding to a = 1. For the remaining rows of this table a = 0. But the highlighted region of this table describes a 2-input AND gate with inputs b and c, i.e., y = b . c provided that the third input, a, is tied to logic 1. Therefore, by fixing the unused input, say a, at logic 1, a 3-input AND gate is converted to a 2-input AND gate as shown in Figure 36*b*. This is in fact a generalization of the concept that we examined in Figure 17 to keep a two-input AND gate open. The same conclusion is valid for NAND gates as well. In a similar way it can be shown that unused inputs of OR, NOR, XOR and XNOR gates can be tied to logic 0 in order to obtain a fewer-input version of the original gate. For XOR and XNOR gates we can further generalize this rule as follows: the unused inputs of an XOR or XNOR gate must be tied to an even number of 1s. (Note that "no 1 at all" means an "even number of 1s" as well.)

Row	a b c	y
0	0 0 0	0
1	0 0 1	0
2	0 1 0	0
3	0 1 1	0
4	**1 0 0**	**0**
5	**1 0 1**	**0**
6	**1 1 0**	**0**
7	**1 1 1**	**1**

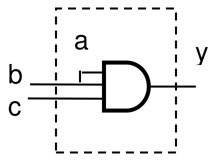
(a)



(b)

Figure 36. Unused inputs: (a) truth table of 3-input AND with highlighted rows for a =1, (b) 3-input AND gate converted to 2-input AND gate by fixing input a at logic 1

Another method to get rid of unused input(s) of an AND gate is to tie the unused input(s), say a, to used one(s), say b, as shown in Figure 37*a*. Since a = b, the truth-table rows in which a ≠ b (i.e., rows 2, 3, 4 and 5 in the truth table of a 3-input AND gate) are not relevant anymore, hence can be removed as shown in Figure 37*b*. The resulting truth table which describes a two-input AND gate is shown in Figure 37*c*. In a similar way it can be shown that the same method is valid for NAND, OR and NOR gates as well.



(a)

Row	a b c	y
0	0 0 0	0
1	0 0 1	0
~~2~~	~~0 1 0~~	~~0~~
~~3~~	~~0 1 1~~	~~0~~
~~4~~	~~1 0 0~~	~~0~~
~~5~~	~~1 0 1~~	~~0~~
6	1 1 0	0
7	1 1 1	1

(b)

Row	a=b	c	y
0	0	0	0
1	0	1	0
2	1	0	0
3	1	1	1

(c)

Figure 37. Unused input in AND gate: (a) unused input connected to a used one, (b) relevant rows (0, 1, 6 and 7) highlighted in truth table of 3-input AND, (c) final truth table

Transistor-level or Internal Architecture of CMOS Gates

Logic gates themselves are built up of electronic devices called *transistors*. Rigorous transistor-level analysis of logic gates is beyond the scope of this book. In this section, only *simplified* views of *standard* CMOS gates, namely NOT, NAND and NOR, are presented to give you some ideas about the operation principles of this logic family. You need more prerequisites in electronics to take a similar look at TTL gates. In CMOS technology (see the footnote on page 5) two *complementary* transistor types, namely NMOS and PMOS are used[4] each with three terminals called G, D and S. We may model each transistor with an on/off switch (with switch terminals called D and S) as shown in Figure 38, where each switch is controlled by a third terminal called G. A PMOS switch is closed (or on) if the voltage on G (V_G) is low (logic 0), and is open (or off) if V_G is high[5] (logic 1). (See Figure 38-top.) An NMOS switch has the opposite behavior; it is closed (or on) if V_G is high (logic 1) and is open (or off) if V_G is low (logic 0), as shown in Figure 38-bottom.

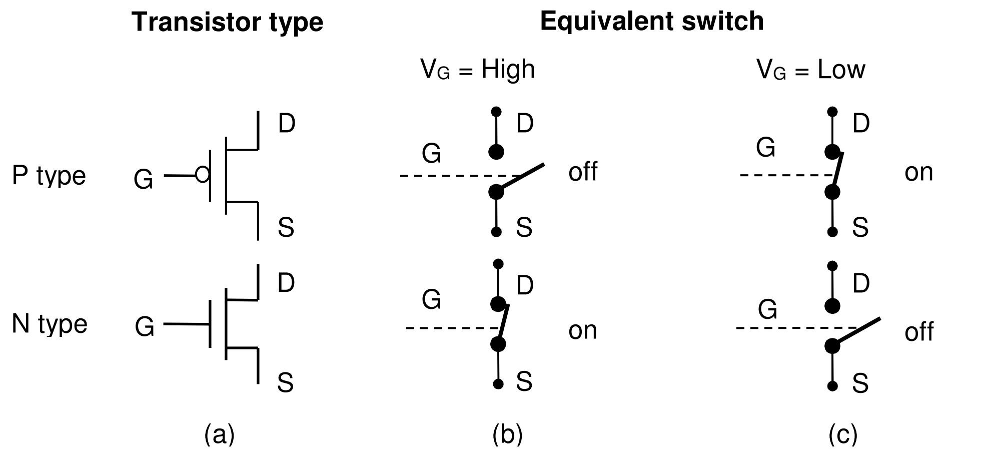


Figure 38. Complementary MOS transistors:
Top. PMOS: (a) symbol, (b) switch model when V_G = high, (c) switch model when V_G = low
Bottom. NMOS: (a) symbol, (b) switch model when V_G = high, (c) switch model when V_G = low

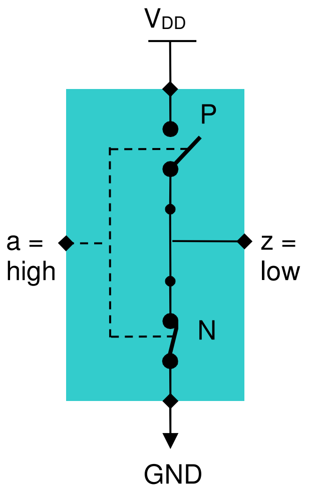
[4] A transistor is an atomic electronic device, i.e., it is not comprised of any smaller electronic devices. The transistors used in CMOS technology are called MOSFET standing for **m**etal-**o**xide **s**emiconductor **f**ield **e**ffect **t**ransistor or just MOS for short. The “C” in CMOS stands for *complementary*, signifying the presence of two different N and P type MOSFETs (called NMOS and PMOS transistors, respectively) in this technology.

[5] Different CMOS processes use different high and low voltage levels. For example in the 54/74HC family the levels are *approximately* as follows: 3.5 volts $< V_H <$ 5 volts, and 0 volt $< V_L <$ 1.50 volts. However, remember that in today’s gigantic CMOS chips these voltage levels are much lower to reduce power consumption significantly.

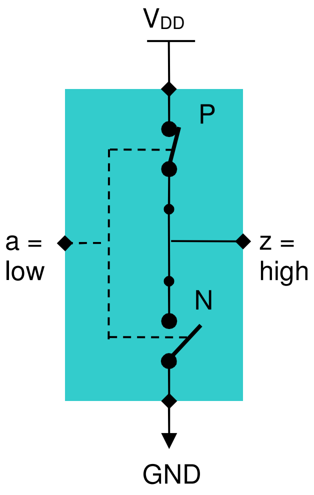
Considering these (simplified) transistor models, a CMOS inverter is constructed using two complementary switches put in series between V_{DD} (the positive terminal of power supply) and GND (the negative terminal of power supply) as shown in Figure 39*a*. The input signal, a, (which is high in the figure) goes to the control inputs of both switches. Pay special attention to the output pin, z, which has a path to V_{DD} through a P-type switch from the top, and one path to ground through an N-type switch from the bottom. Since the control inputs of these two complementary switches are driven by the same (input) signal, *exactly one switch* is on (closed) at a time, and the other one is necessarily off (open). And this means that at any given instant of time z is either high or low, as elaborated on below.

In Figure 39*a* the input voltage is high (logic 1), resulting in an open P-switch (top) and a closed N-switch (bottom). In other words, in this figure z is disconnected from V_{DD} and connected to ground, and this means that z is low (logic 0), which is consistent with the truth table of an inverter, input = 1 => output = 0.

When the input changes to a low voltage (logic 0), the P-type switch turns on and the N-type switch turns off, resulting in a path from z to V_{DD}, which pulls z up (input = 0 => output = 1) as shown in Figure 39*b*. An operation summary (function table) of a CMOS inverter is shown in Figure 39*c*.



(a)



(b)

Input	P-type	N-type	Output
low	on	off	high
high	off	on	low

(c)

Figure 39. CMOS inverter: (a) switch diagram when input = high, (b) switch diagram when input = low, (c) function table

Figure 40*a* shows a switch diagram for a two-input CMOS NAND gate with one low and one high input. In this gate two P-type switches are put in parallel between the output, z, and V_{DD}. Even one closed (on) P-type switch is able to connect z to V_{DD}; to isolate z from V_{DD} both P-type switches have to be off. On the other hand, z has a path to ground through two N-type switches in series; so that in order to connect z to ground both NMOSs have to be on; even one open N-type switch is able to isolate z from ground.

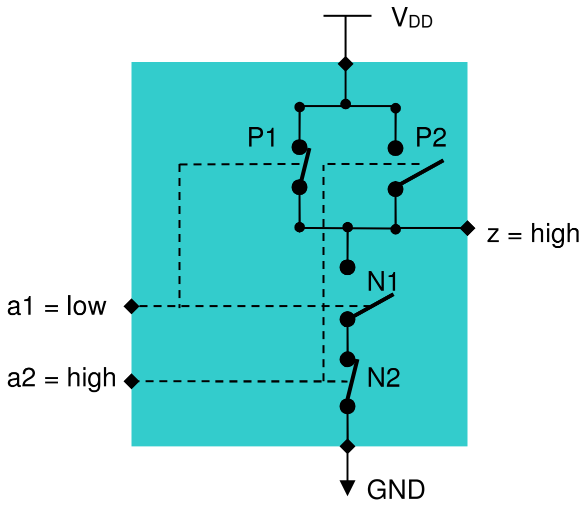
You should note that as a general rule in CMOS NOT, NAND and NOR gates there is a one-to-one correspondence between P-type switches and N-type switches. More specifically, each N-type switch is paired up with one P-type switch and vice versa, so that each pair of P-type/N-type switches is driven by the same input. In other words, in CMOS gates each input drives one P-type and one N-type switch. Therefore, at any given instant of time and in each pair exactly one switch is on and the other one is necessarily off. This is in fact a simple generalization of what we saw in the CMOS inverter in Figure 39. Back to Figure 40*a*, switch pair P_1/N_1 is driven by input a_1, and switch pair P_2/N_2 is driven by input a_2.

With this background refer to the function table shown in Figure 40*b*, in which all four possible combinations of input levels have been listed. As an example consider the second row in which a_1 = low and a_2 = high, which corresponds to the situation illustrated in Figure 40*a*. Since a_1 = low, P_1 turns on (closed) and N_1 turns off (open). Similarly, since a_2 = high, P_2 turns off and N_1 turns on. So, we have

P_1: on, and P_2: off => there is a path between z and V_{DD}

N_1: off, and N_2: on => there is no path between z and ground

The net effect is z is pulled up (logic 1). And this is consistent with the truth table of a NAND gate.



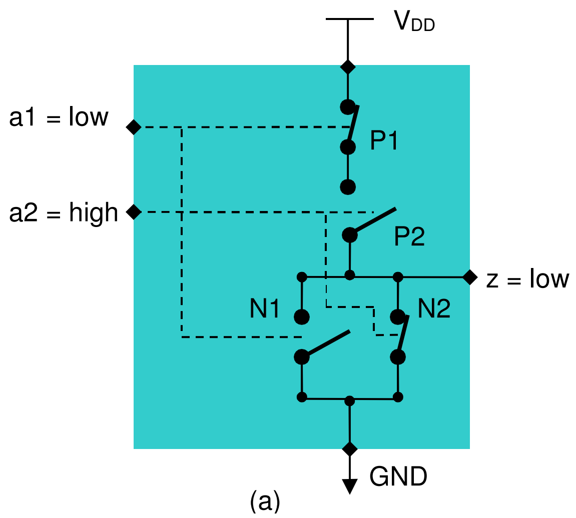
(a)

a1	a2	P1	P2	N1	N2	z
low	low	on	on	off	off	high
low	high	on	off	off	on	high
high	low	off	on	on	off	high
high	high	off	off	on	on	low

(b)

Figure 40. CMOS NAND: (a) switch diagram when inputs = low high, (b) function table

Figure 41*a* shows a switch diagram for a two-input CMOS NOR gate with one low and one high input. The function table of this gate is shown in Figure 41*b*. What has changed here is that now the P-type switches are placed in series while the N-types are in parallel. But what never changes in the design of CMOS gates is that P-type transistors are always placed on the top between VDD and the output, and N-type transistors are placed on the bottom between the output and ground. Let's take a look at the second row of the function table, as an example, where a_1 = low and a_2 = high. This situation is shown in Figure 41*a*. (Remember that in CMOS each input drives exactly two switches, one P-type and one N-type, resulting in one closed and one open switch in each pair.) In this switch diagram a_1 (= low) drives P_1 and N_1; considering the behavior of P- and N-type switches summarized in Figure 38, P_1 would be closed (on) and N_1 would be open (off), as shown in Figure 41*a*. Input a_2 (= high), on the other hand, controls P_2 and N_2, resulting in open P_2 and closed N_2. The net effect of these four switches in this specific situation is that z will be disconnected from V_{DD} by P_2 and connected to ground through N_2, and this results in z = low, which is consistent with the truth table of a two-input NOR gate: input 01 produces a 0.



(a)

a1	a2	P1	P2	N1	N2	z
low	low	on	on	off	off	high
low	high	on	off	off	on	low
high	low	off	on	on	off	low
high	high	off	off	on	on	low

(b)

Figure 41. CMOS NOR: (a) switch diagram when inputs = low high, (b) function table

The above concept can be generalized to three- (or more-) input CMOS NAND or NOR gates. An n-input NOR gate has n P-type series transistors (between V_{DD} and the output) and n N-type parallel transistors (between GND and the output). Similarly, an n-input NAND gate has n P-type parallel and n N-type series transistors (between V_{DD} and the output and between GND and the output, respectively). However, as the number of series transistors increases, the electrical characteristic of the gate degrades, and this limits the number of inputs (*fan-in*) of CMOS NAND and NOR gates to around 6 and 4, respectively. Another restriction that is also due to electrical characteristics of CMOS gates and should be taken into consideration in the design of digital circuits is the maximum capacitive loading or the maximum number of gates (AC *fanout*) that a gate is able to drive before performance is degraded beyond a threshold. Similar restrictions apply to other logic families as well. Electrical characteristics of digital gates are beyond the scope of this book. Interested students may refer to textbooks on digital electronics for more information.