

Chapter 3

Basic MOSFET logic gates

3.1 Inverter

When building digital gates out of MOSFETs, we will be observing three basic rules:

1. Only use NFETs to pull the output down and PFETs to pull the output up.
2. Never allow the output to be simultaneously pulled up and down.
3. Always ensure that the output is either pulled-up or pulled-down, so that the output state is always known.

The simplest, non-trivial logic gate that satisfies rule 1 is the logic gate composed of a single NFET (to pull the output down) and a single PFET (to pull the output up). This arrangement was already described in Figure 2.6. However, the configuration of Figure 2.6(e) violates rule 2 and the configuration of Figure 2.6(d) violates rule 3. Hence only the configurations of Figures 2.6(b) and (c) satisfy all three basic rules.

We note that for the two valid configurations, both MOSFET gates must be at the same voltage. Hence we can connect these two MOSFET gates and call that the input to our logic gate. Furthermore, we note that when this input is 1 (Vdd) the output is 0 (Gnd), and when this input is 0 (Gnd) the output is 1 (Vdd). Hence the logic gate that we have constructed is an inverter. This logic gate and its operation is summarized in Figure 3.1.

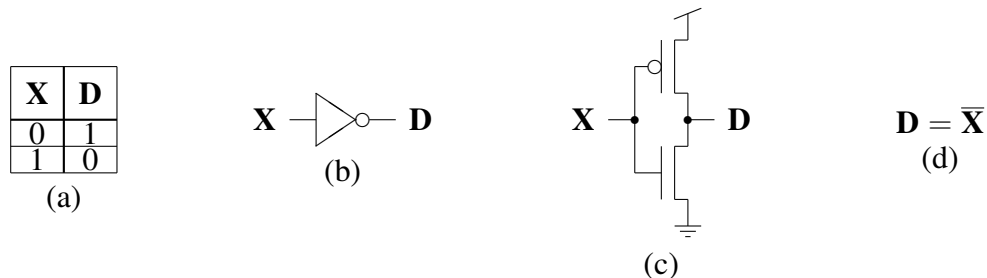


Figure 3.1: Inverter. (a) Truth table. (b) Symbol. (c) Schematic. (d) Boolean expression.

3.2 NAND gates

The next simplest gate is the two input NAND gate, shown in Figure 3.2. This gate is composed of two NFETs in series to pull the output low, and two PFETs in parallel to pull the output high, which satisfies rule 1.

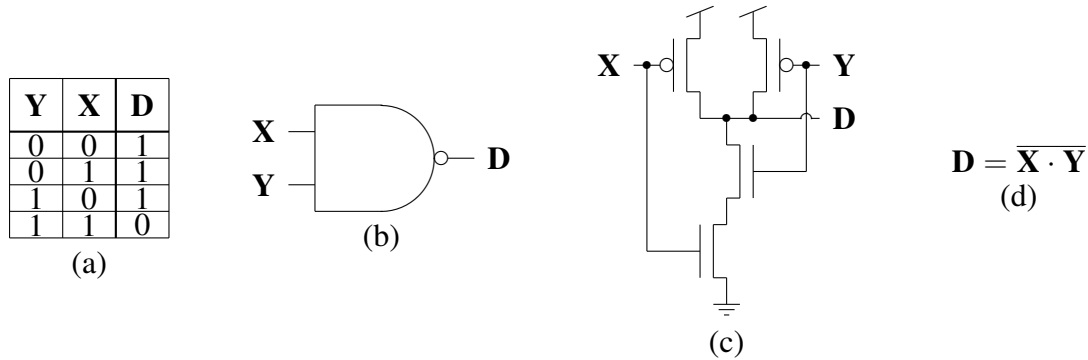


Figure 3.2: Two input NAND. (a) Truth table. (b) Symbol. (c) Schematic. (d) Boolean expression.

The operation of the two input NAND gate is as follows. When the **X** and **Y** inputs are both high, the two NFETs turn on and pull the output low. At this point, the two PFETs are both off, so neither is pulling the output high and rule 2 is satisfied.

If either **X** or **Y** is low, then at least one of the NFETs will turn off, preventing the output from being pulled low. Simultaneously, at least one of the PFETs will turn on and pull the output high, satisfying rule 3.

The two input NAND gate can be extended to three inputs by placing three NFETs in series and three PFETs in parallel as in Figure 3.3. The operation is the same as for the two input NAND gate, satisfying all three basic rules. Continuing the process, the NAND gate can be further extended to more three inputs as well.

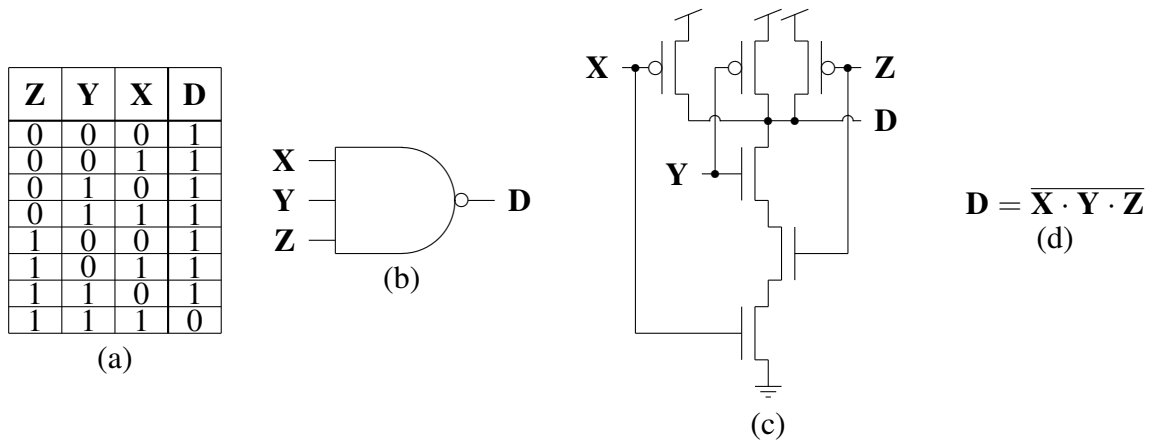


Figure 3.3: Three input NAND. (a) Truth table. (b) Symbol. (c) Schematic. (d) Boolean expression.

3.3 NOR gates

If we take the two input NAND gate and place the NFETs in parallel and the PFETs in series, we get a NOR gate as shown in Figure 3.4.

This gate still uses PFETs to pull the output high and NFETs to pull the output low, satisfying rule 1. In operation, when the **X** and **Y** inputs are low, the two PFETs turn on and pull the output high, and both NFETs are off so that rule 2 is satisfied. When either **X** or **Y** is high, then one of the PFETs will turn off, preventing the output from being pulled high, and one of the NFETs will turn on and pull the output low, so that rule 3 is satisfied.

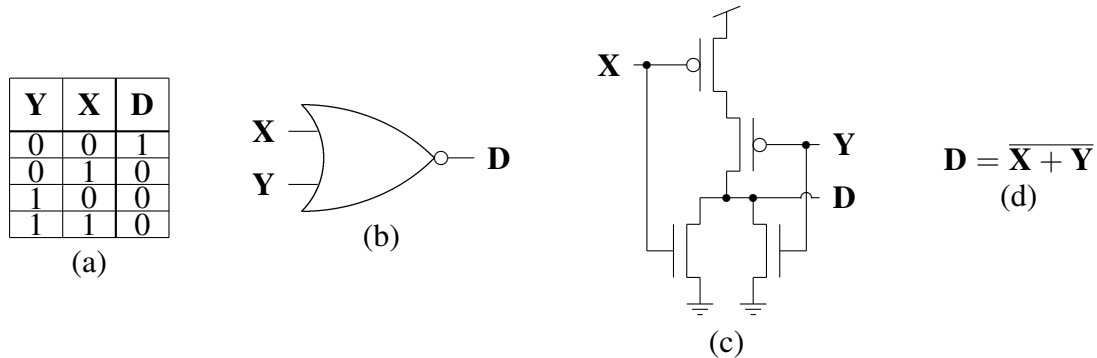


Figure 3.4: Two input NOR. (a) Truth table. (b) Symbol. (c) Schematic. (d) Boolean expression.

3.4 Multiplexor

Figure 3.5 shows the truth table, symbol, schematic and Boolean expression for a two input multiplexor.

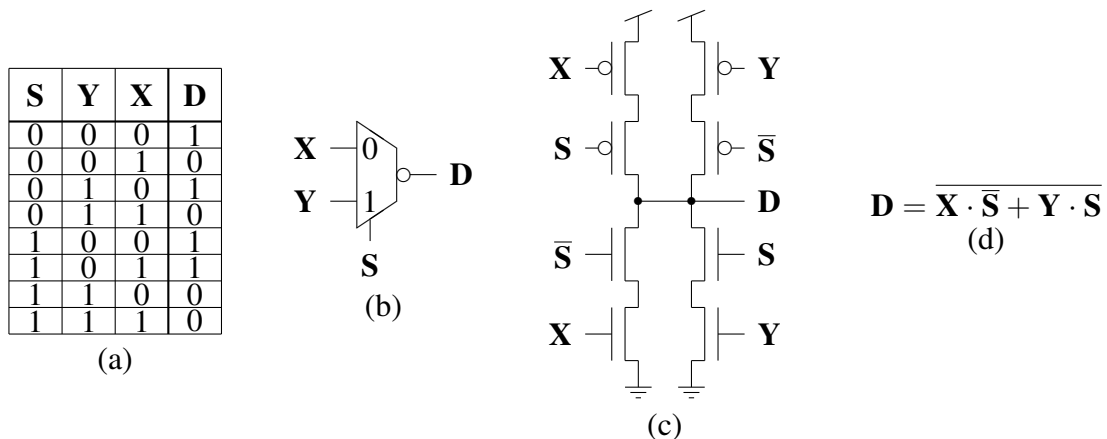


Figure 3.5: Two input multiplexor. (a) Truth table. (b) Symbol. (c) Schematic. (d) Boolean expression.

The operation of the multiplexor is as follows. When **S** is low, the PFET controlled by \bar{S} and the NFET controlled by **S** are turned off, preventing **Y** from affecting the output **D**. However, the

PFET controlled by S and the NFET controlled by \bar{S} are turned on, which allows X to control the state of the output D . Hence, when X is low, D will be pulled high, and when X is high, D will be pulled low.

Conversely, when S is high, the PFET controlled by S and the NFET controlled by \bar{S} are turned off, preventing X from affecting the output D . However, the PFET controlled by \bar{S} and the NFET controlled by S are turned on, which now allows Y to control the state of the output D . Hence, when Y is low, D will be pulled high, and when Y is high, D will be pulled low.

Chapter 4

Constructing custom logic gates

4.1 Truth table method

Although we can construct any digital system using only the two input NAND gate, this would result in a circuit that is inefficient in space, speed and power.

To construct a custom logic gate, the simplest method is to start with the truth table, and construct one branch for each line in the truth table. Given an n input truth table, each branch will be constructed of n series PFETs or n series NFETs depending on whether the output for the corresponding line in the truth table is a 1 or a 0.

For instance, suppose that we have the truth table shown in Figure 4.1(a). For line 0, we want the output to be low, so we select two series connected NFETs. In order for these two NFETs to be on simultaneously for the input state of line 0 (X and Y are both low), we need to connect the gates of the NFETs to \bar{X} and \bar{Y} . This configuration is shown in Figure 4.1(b). Similarly, for line 3, we also need NFETs, but this time the gates will be connected to X and Y . This configuration is shown in Figure 4.1(e).

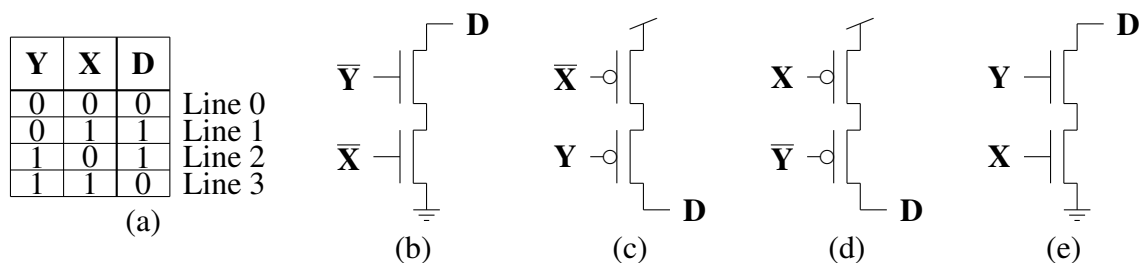


Figure 4.1: Two input XOR. (a) Truth table. (b) NFET branch for line 0. (c) PFET branch for line 1. (d) PFET branch for line 2. (e) NFET branch for line 3.

For lines 1 and 2, we want the output to be high, so we select two series connected PFETs. To turn these PFETs on for their respective input states, one pair will be connected to \bar{X} and Y (line 2) and the other will be connected to X and \bar{Y} . These two configurations are shown in Figures 4.1(c) and (d). The end result is shown in Figure 4.2.

Since a truth table with n inputs has 2^n lines, this method will result in a circuit that has 2^n branches of n MOSFETs each, or $n2^n$ total MOSFETs.

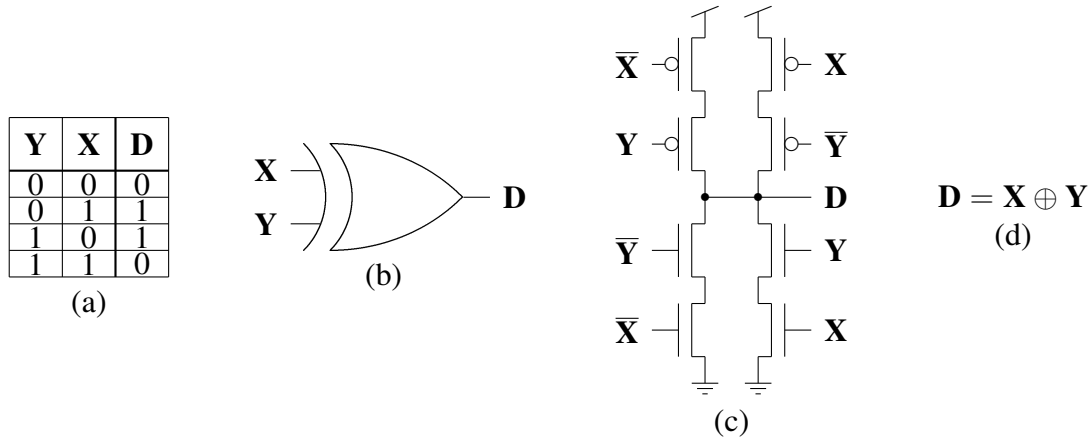


Figure 4.2: Two input XOR. (a) Truth table. (b) Symbol. (c) Schematic. (d) Boolean expression.

Logic gates can also be constructed from truth tables using parallel branches of NFETs or PFETs. To do this, we design branches to block the pull up or pull down action. For instance for the XOR used in the previous example, line 0 would convert to two parallel PFETs with gates connected to \bar{X} and \bar{Y} . This would prevent the output from being pulled high when X and Y are low.

Using branches composed of parallel MOSFETs results in gates which have the same number of transistors as using branches composed of series MOSFETs. However, experience shows that using parallel MOSFETs results in larger and slower logic gates, and for these reasons, logic gates are rarely constructed this way.

4.2 Complementary structures method

Suppose that we use the previous method to construct a NAND gate. The resulting circuit would look like the one shown in Figure 4.3. Comparing Figures 3.2(b) and 4.3(b) it is evident that the truth table method can result in an implementation with an excessive number of MOSFETs. Furthermore, implementation of Figure 4.3(b) uses the complements of the input signals \bar{X} and \bar{Y} , which themselves require inverters to generate from the original input signals. Although it is possible to reduce the circuit of Figure 4.3(b) to that of Figure 3.2(b), it is generally difficult to do so.

However, we note that the pull down branch which corresponds to line 3 of the truth table is the same in both instances. The complementary structures method takes advantage of this to form the pull up network of PFETs from the pull down network of NFETs (or *vice versa*).

With the complementary structures method, we replace NFETs with PFETs (or PFETs with NFETs), parallel branches with series branches and series branches with parallel branches. The signals going to the gates of the MOSFETs remain unchanged.

For example, the PFET branch shown in Figure 4.4(b) is the complement of the NFET branch shown in Figure 4.4(a). The PFET branch was created by converting the parallel NFETs controlled by X and Y into the series PFETs also controlled by X and Y . The NFET controlled by Z was in series with the NFETs controlled by X and Y , hence the PFET controlled by Z goes in parallel

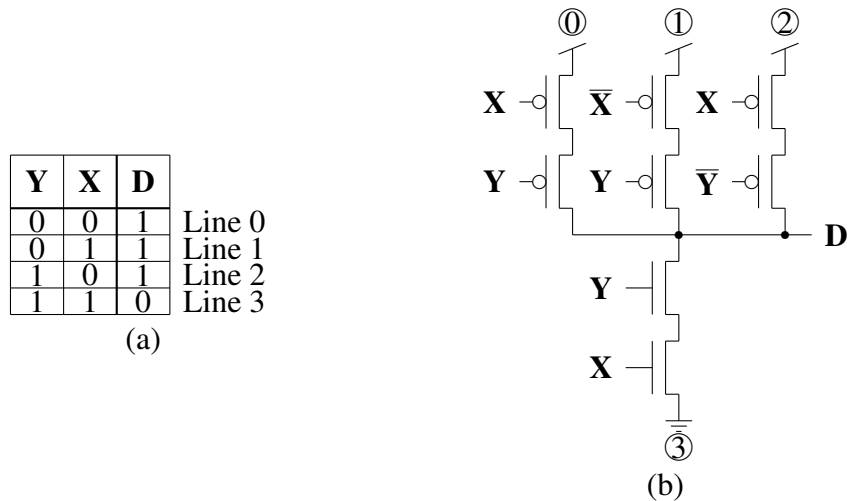


Figure 4.3: Two input NAND implemented with truth table method. (a) Truth table. (b) Schematic.

with the PFETs controlled by X and Y . Combining these two complementary structures produces the OR-NAND shown in Figure 4.5.

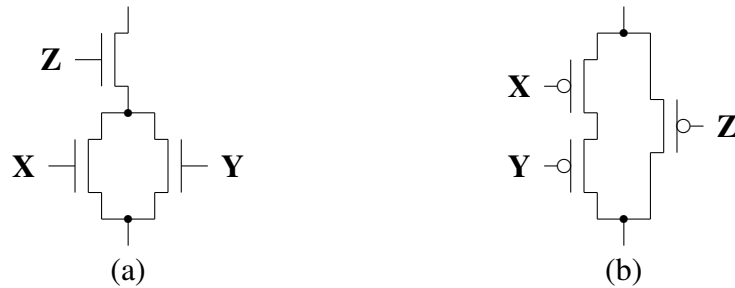


Figure 4.4: Two complementary structures. (a) NFET branch. (b) PFET branch.

4.3 Boolean equation method

The reason that the complementary structures method works is that any network of NFETs (or PFETs) can be reduced to a single NFET (or PFET) with a Boolean expression attached to the control terminal.

The basic conversion possibilities are shown in Figure 4.6. The use of this method is demonstrated in Figures 4.7 and 4.8.

Beginning with the pull down network of Figure 4.7(a), we take the two series NFETs connected to X and Y and convert them to a single NFET connected to $X + Y$ as shown in Figure 4.7(b). Now we have two NFETs in parallel, and we convert them to a single NFET connected to $(X + Y) \cdot Z$ as shown in Figure 4.7(c).

Given a single NFET pull down, we know that we need only a single PFET pull up with the same gate control signal as in an inverter. Hence the pull up PFET that we need in conjunction with the pull down NFET of Figure 4.7(c) is the PFET shown in Figure 4.8(a).

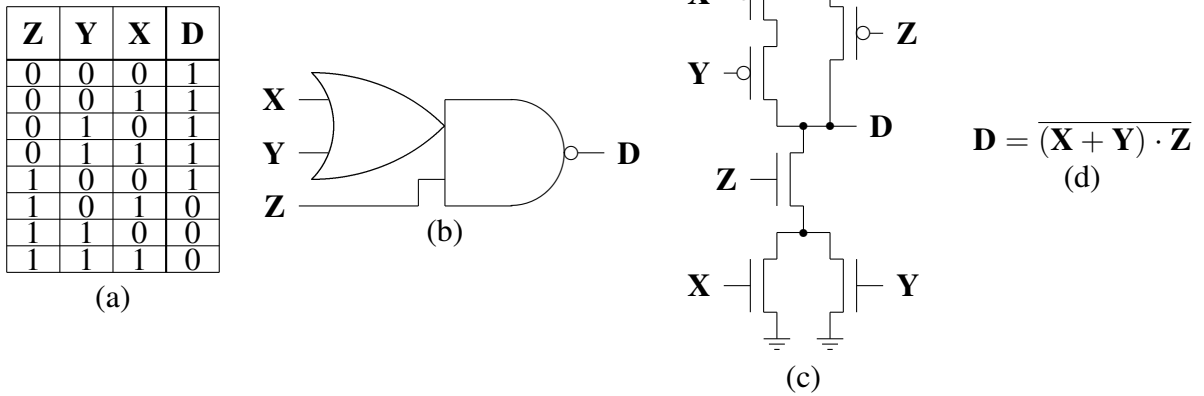


Figure 4.5: OR-NAND. (a) Truth table. (b) Symbol. (c) Schematic. (d) Boolean expression.

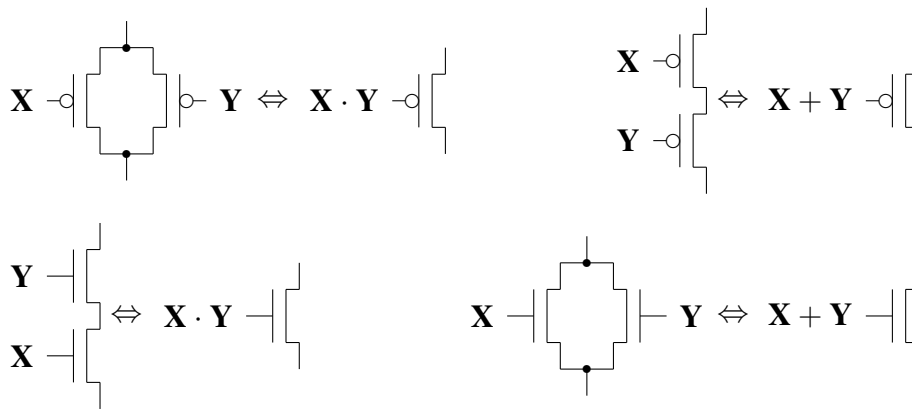


Figure 4.6: Boolean equivalent circuits.

Now we can proceed with the reverse process and reexpand the circuit, as shown in Figures 4.8(b) and (c). The complete logic gate is shown in Figure 4.9, and is indeed an AND-NOR gate as expected given the Boolean intermediate equations.

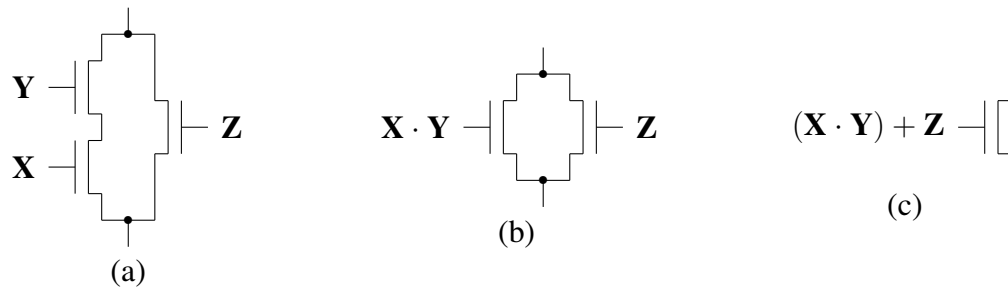


Figure 4.7: Boolean equivalent circuit reduction.

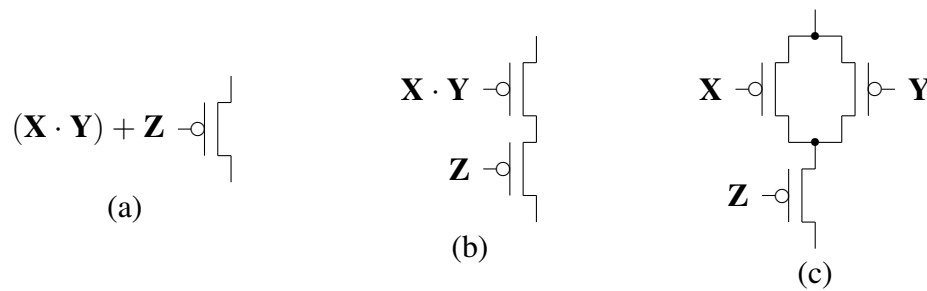


Figure 4.8: Boolean equivalent circuit expansion.

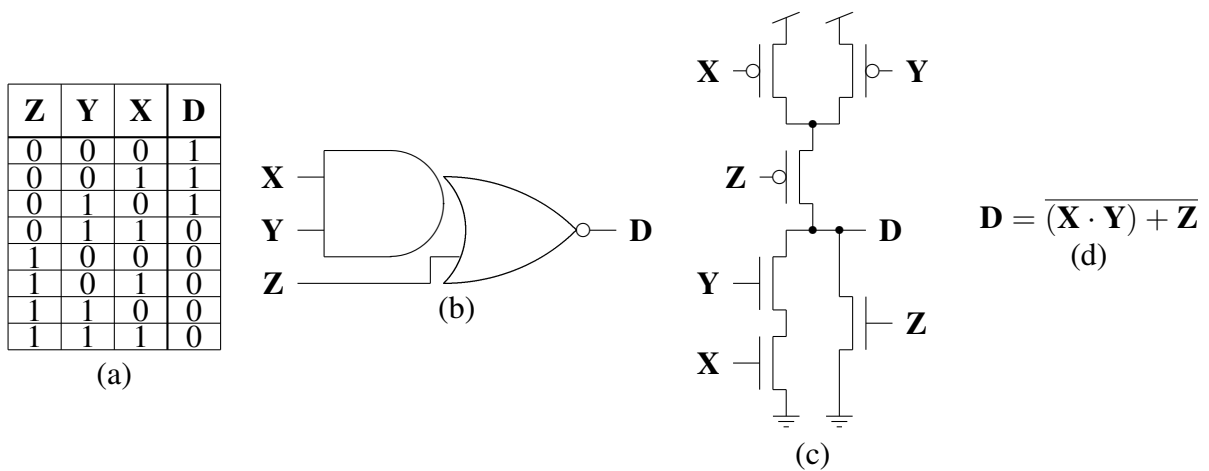


Figure 4.9: AND-NOR. (a) Truth table. (b) Symbol. (c) Schematic. (d) Boolean expression.