Chapter 1: Start-up lecture

A. Course introduction

Course startup. Teacher presentations. Overall goals, aims, and learning outcomes. Course PM.
Design flow and EDA tools.
Course organization.

- Lectures: CMOS theory on Tuesdays, adder design on Thursdays
- Exercises on Thursdays
- Home assignments. One hand-in per week.
- Hands-on lab sessions

The aim of the first chapter is to show how CMOS logic gates can be designed for functionality based on very simple assumptions concerning the basic MOSFET design element which can be regarded as a simple ON-OFF switch.
Background knowledge required for this session: Boolean truth tables, Karnaugh maps, min terms, max terms, prime implicants, sum-of-products (SOP), product-of-sums (POS), de Morgan's theorem, and bubble shuffling. If you feel uncertain, please consult the short summary of the most basic Boolean logic presented in Appendix 1.

B. Design of CMOS logic gates

Since CMOS is a complementary technology, logic gates must be designed by using both a pull-up network of p-type switches that are ON for zero input voltages, and a complementary pull-down network of n-type switches that are ON for positive input voltages. The two networks are not allowed to be ON at the same time, but must be complementary – one being ON, the other one being OFF - thereby pulling the output node to the corresponding rail voltage (V_{DD} or V_{SS}) to which the ON network is connected. The design principle is illustrated in figure 1.1 using 2-input NAND and NOR gates as design examples.

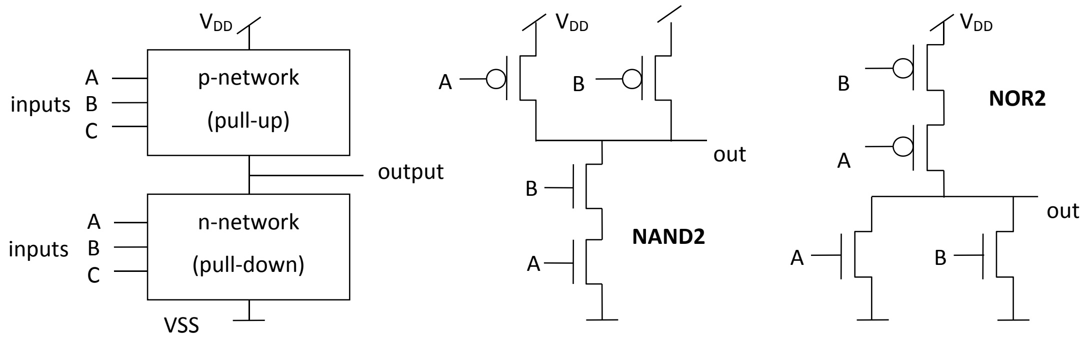


Fig. 1.1. CMOS logic gate design principle using pull-up and pull-down networks.

Some of the most common logic gates, that we will learn how to design at the MOSFET level in this session, are shown in Fig. 1.2. Both inverting and non-inverting gates are shown in this figure. One of our first observations is that basic CMOS gates are by nature always inverting. Non-inverting gates like AND, OR, and AND-OR gates always use output inverters to get the output logic correct. The AND gate, for instance, consists of a NAND gate and an output inverter. The logic symbols used in Fig. 1.2, and in American text books, are, just, American. The corresponding European AND, OR and XOR symbols are shown in Fig. 1.3.

As already stated above, the basic idea when designing static CMOS gates is to use two networks: one pull-up network of PMOS-transistors connected to the supply voltage (V_{DD}), and one pull-down network of NMOS-transistor connected to ground (V_{SS}). When the Boolean function is true, the pull-up network pulls the gate output up to V_{DD}, and when the Boolean function is false the pull-down network pulls the output down to V_{SS}. It was also mentioned in the introduction above that CMOS gates are inverting. Examples of inverting gates are the NAND, NOR, AND-OR-INVERT (AOI), and OR-AND-INVERT (OAI) gates. Non-inverting logic gates, like AND, OR, AND-OR and OR-AND gates, need an inverter at the output to get the output logic correct.

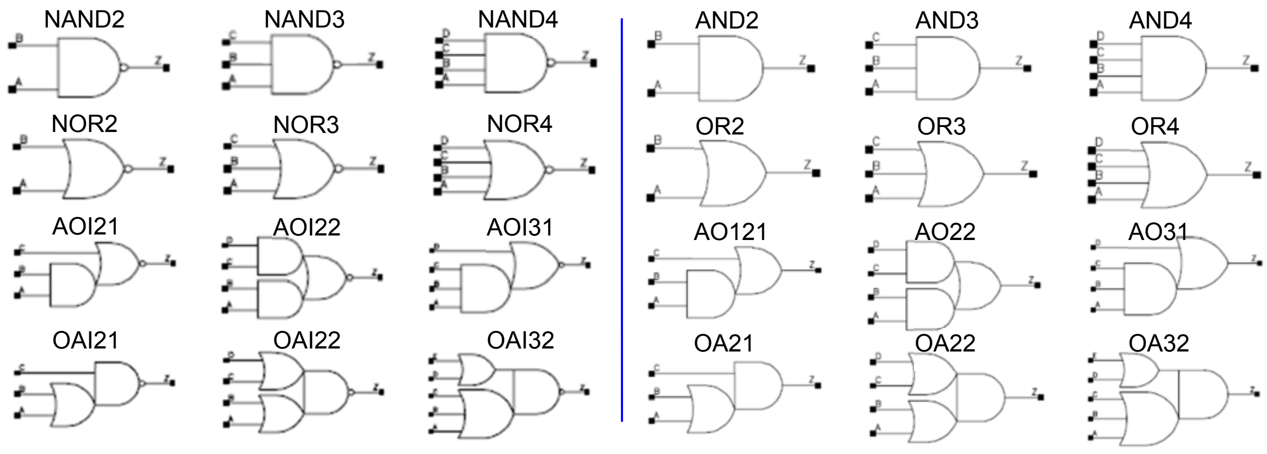


Fig. 1.2. Basic inverting and non-inverting logic gates.

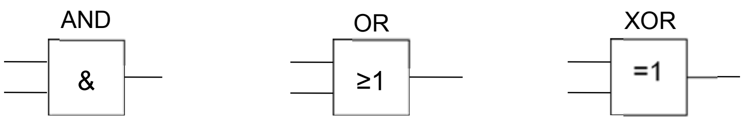


Fig. 1.3. European logic gate symbols.

The only MOSFET model we need to make a correct gate implementation of a logic function is a simple switch model. In CMOS, the n-channel MOSFET can be modeled by an n-type switch; i.e. a switch that is ON when its control input is high, and OFF when its control input is low. This simple model is sufficient to build the n-type pull-down network. Conversely, the p-channel MOSFET can be modeled by a p-type switch; i.e. a switch that is ON when its control input is low, and OFF when its control input is high. This simple model is sufficient to build the p-type pull-up network. An example where the Boolean function Z=AB+CD is implemented by p- and n-switches is shown in Fig. 1.4.

Most often Boolean functions are given as sums of products (SOP). The Boolean function $F=AB+CD$ is one such example. In my view, the simplest way of implementing any Boolean function of non-inverted inputs in CMOS, is to start with the pull-down network, and to use the Boolean expression to combine n-switches accordingly. Once the corresponding pull-up network is in place, this gate will produce the inverted output function, $\overline{F} = \overline{AB + CD}$. The output inverter will then invert this signal, and produce the correct output logical function.

Any Boolean function can be inverted by using de Morgan´s theorem. In this example, the Boolean function is given as a sum of products. The inverse of our Boolean function can then be written as a product of sums (POS), i.e. $\overline{F} = \left(\overline{A}+\overline{B}\right)\left(\overline{C}+\overline{D}\right)$. Hence, the pull-up network can be implemented by

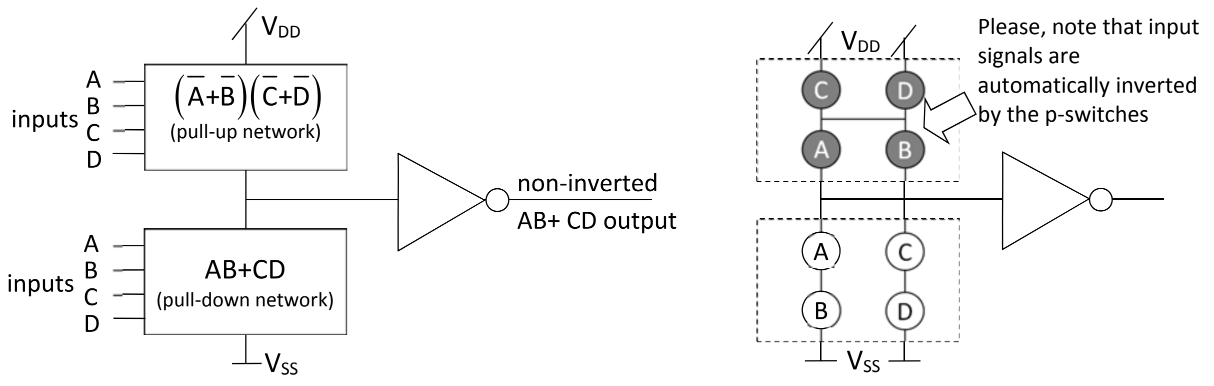


Fig. 1.4. General logic gate design using pull-up and pull-down networks.

two parallel p-switches representing $\overline{A}+\overline{B}$, in series with two parallel p-switches representing $\overline{C}+\overline{D}$. The complete solution of this 2+2 AND-OR logic gate was shown in fig. 1.4.

A little bit more complicated is the implementation of the XOR and XNOR gates since they are both functions of inverted as well as non-inverted input signals. A ten MOSFET XOR implementation is shown in Fig. 1.5.

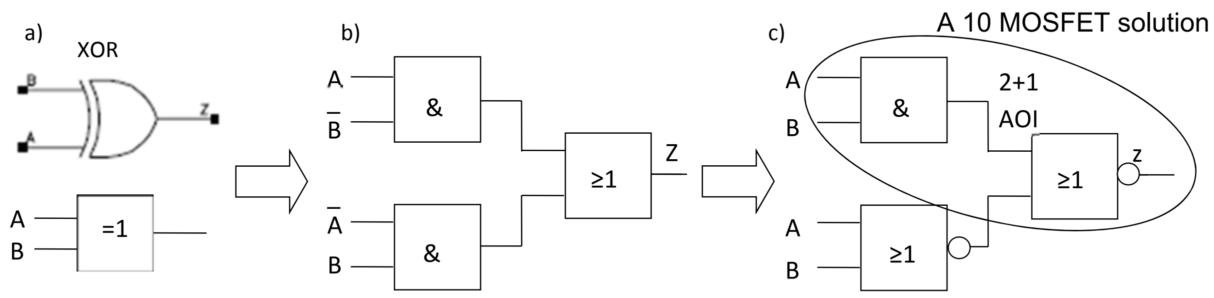


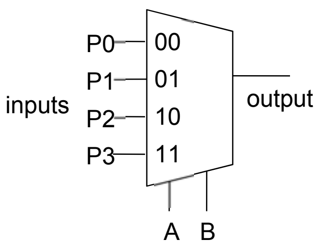
Fig. 1.5. XOR and XNOR logic gate implementations.

Exercise 1. Try to implement the design methodology outlined above for realizing the following CMOS logic gates: NAND4, NOR3, and XOR2!

Exercise 2. Try to implement the design methodology outlined above for realizing the following compound CMOS logic gates: a) AOI31, b) OAI21, c) OAI22, d) AO22, e) 211AOA or (AB+C)D, f) 212 OAO or (A+B)C+DE.

Exercise 3. Show that the XOR2 gate illustrated in Fig. 1.5 only requires 10 MOSFETs!

Exercise 4. Multiplexer logic: Determine the multiplexer inputs for realizing the following logic functions:
a) NAND2, b) NOR2, c) XOR2, d) XNOR2, e) AND2.



Exercise 5. Design an eight-bit zero-detect-circuit having a high output signal if and only if all inputs are zeroes. The circuit shall be designed as an iterative logic array!

Pre-lab assignment #1: Design at the transistor level a static compound CMOS logic cell comparing two bits *a* and *b*. The output signal Z_{out} is to be equal to a logic one if (and only if) $a>b$. The cell shall also have input state bits with the result from any previous comparison of more or less significant bits. The cell shall be designed to fit into an *N*-bit iterative logic array (ILA) so that two N-bit words *A* and *B* can be compared. Discuss whether it is best to perform the comparison starting from the most or from the least significant bit.