EECS 16B Designing Information Devices and Systems II
Fall 2018 Elad Alon and Miki Lustig

Discussion 1B

1 Digit Bases

$(N)_p$ is used to indicate that the number N is expressed in base p. For example, $(N)_2$ means that we are working in base 2 and $(N)_{10}$ means N is expressed in base 10, or decimal digits. For example,

- $(245)_{10} = 2\times10^2 + 4\times10^1 + 5\times10^0$
- $(11010)_2 = 1\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 0\times2^0$

2 Boolean Algebra

In Boolean algebra, *true* statements are denoted 1 and *false* statements are denoted 0. A Boolean function acts on a set of these Boolean values and outputs a set of Boolean values (usually just one). The most common Boolean operators used are **NOT**, **AND**, **OR**, and **XOR**.

2.1 NOT

NOT is a Boolean function that takes in one Boolean value and outputs its negation. Let x be a Boolean variable. **NOT**(x) is denoted as $\bar{x}$. The truth table of **NOT** is:

x	$\bar{x}$
0	1
1	0

2.2 AND

AND is a Boolean function that takes in two Boolean values and outputs 1 if both the values are true. Let x and y be two Boolean variables. **AND**(x,y) is denoted as $x\cdot y$. The truth table of **AND** is:

x	y	$x\cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

2.3 OR

OR is a Boolean function that takes in two Boolean values and outputs 1 if at least one of the values is true. Let x and y be two Boolean variables. **OR**(x,y) is denoted as $x+y$. The truth table of **OR** is:

x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

2.4 XOR

XOR is a Boolean function that takes in two Boolean values and outputs 1 if exactly one of the values is true. Let x and y be two Boolean variables. **XOR**(x,y) is denoted as $x \oplus y$. The truth table of **XOR** is:

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

3 De Morgan's Law

De Morgan's Law is stated as follows. Let x and y be two Boolean variables. Then,

$$\overline{x \cdot y} = \bar{x} + \bar{y}$$
$$\overline{x + y} = \bar{x} \cdot \bar{y}$$

4 Transistor Introduction

Transistors (as presented in this course) are 3 terminal, voltage controlled switches. This means that, when a transistor is "on," it connects the Source (S) and Drain (D) terminals via a low resistance path (short circuit). When a transistor is "off," the Source and Drain terminals are disconnected (open circuit).

Two common types of transistors are NMOS and PMOS transistors. Their states (shorted or open) are determined by the voltage difference across the Gate (G) and Source (S) terminals, compared to a "threshold voltage." Transistors are extremely useful in digital logic design since we can implement Boolean logic operators using switches.

Recall that in this class, Vtn denotes how much higher the gate needs to be relative to the source for the NMOS to be on, and that |Vtp| denotes how much lower the gate needs to be relative to the source for the PMOS to be on.

Figure 1: NMOS Transistor

Figure 2: PMOS Transistor

Transistors can be connected together to perform boolean algebra. For example, the following circuit is called an "inverter" and represents a NOT gate.

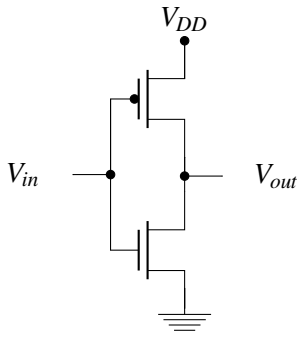


Figure 3: CMOS Inverter

When the input is high ($V_{in} \geq V_{tn}$, $V_{in} \geq V_{DD} - |V_{tp}|$), then the NMOS transistor is on, the PMOS transistor is off, and $V_{out} = 0$. When the input is low ($V_{in} \leq V_{tn}$, $V_{in} \leq V_{DD} - |V_{tp}|$), the NMOS transistor is off, the PMOS transistor is on, and $V_{out} = V_{DD}$. When working with digital circuits like the one above, we usually only consider the values of $V_{in} = 0, V_{DD}$. This yields the following truth table:

V_{in}	V_{out}	NMOS	PMOS
V_{DD}	0	on	off
0	V_{DD}	off	on

If you think of V_{DD} being a logical 1 and 0 V being a logical 0, we have just created the most elementary logical operation using transistors!

4.1 Resistor Switch Model

In the real world, transistors don't actually behave as perfect switches. Transistors have a small amount of resistance in the on state. This can be represented by a resistor in our transistor switch model.

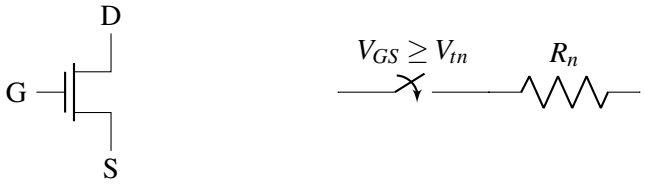


Figure 4: NMOS Transistor Resistor-switch model

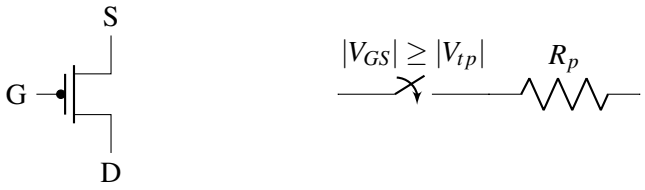


Figure 5: PMOS Transistor Resistor-switch model

1. Transistor Introduction

Assume V_{tn} and V_{tp} are the threshold voltages for the NMOS and PMOS transistors, respectively. Also assume V_{DD} is always a positive number.

(a) In the two above NMOS transistors, label which node is the source and which is the drain.

Answer: In an NMOS, the terminal at the higher potential is always the drain, and the terminal at the lower potential is always the source. Therefore, the drain is at V_{DD} on the left transistor and at the ground for the right, while the source is at the ground of the left transistor and $-V_{DD}$ on the right.

(b) In the two above PMOS transistors, label which node is the source and which is the drain.

Answer: In an PMOS, the terminal at the higher potential is always the source, and the terminal at the lower potential is always the drain. Therefore, the source is at V_{DD} on the left transistor and at the ground for the right, while the drain is at the ground of the left transistor and $-V_{DD}$ on the right.

2. Binary Spaces

(a) What is the largest integer that can be expressed with a 32-bit binary number?

Answer: If we assume unsigned integers (we are only representing nonnegative numbers with the bits), we can reach $2^{32} - 1$, which is roughly 4 gigabytes

(b) What is the largest integer that can be expressed with a 64-bit binary number?

Answer: $2^{64} - 1$, which is rougly 16 exabytes.

(c) What could be some advantages that a 64-bit os has over a 32-bit os?

Answer: A 64-bit os can support RAM sizes that are over 4 gigabytes, and can also perform much higher-precision arithmetic, both with large integers and floating-point numbers.

3. Adder Block Logic

If we convert a number into binary, we can express each digit as true (1) or false (0). Using this binary expression, we can implement arithmetic functions, such as addition. We will build a logical block which, given two binary digits and a carry input, determine the value of the current digit and whether the next digit requires a carry.

(a) Try adding the numbers 1110_2 and 1100_2. How do you go about computing a digit of the output? What factors to do have to consider? What results do you need to compute the digit, and perhaps the following digits?

Answer: To compute the right-most digit, we see that $0+0=0$, so the sum's digit is 0. The next digit is computed from $1+0=1$, so the sum's digit is 1. In the following digit, we have $1+1=10$ (remember, base 2!), so the current digit's value is 0, and we carry a one to the next digit. The last digit is $1+1$, with an additional 1 from the carry, which gives us the sum of $1+1+1=11$. Thus, the current digit's value is 1, and the carry tacks a leading one to the sum.

This example shows us that we not only have to consider the two digits that are being summed, but also the carry of the digit preceding. We also have to not only compute the value of the current digit, but whether or not the next digit has a carry.

(b) First, let's try making an adder without considering carry into the current digit. Let the two digits being summed be $x, y \in \{0,1\}$, the resulting sum digit be s, and the carry be c. Create logical expressions for what the current digit and the carry for the next digit would be.

Answer: We know that $s=1$ when either x or y, but not both, are equal to 1, so $s = x \bigoplus y$. $c=1$ only if $x=y=1$, so we can say $c = x \cdot y$.

(c) Finally, let's make a full adder block, incorporating an incoming carry variable c_{in}. Compute s, c, this time considering c_{in}.

Answer: The extra carry is effectively just another input we have to add to the current digit. For s, we can treat the current value of s as one of the digits we need to add, so $s = x \bigoplus y \bigoplus c_{in}$. c is true when at least two of x, y, c_{in} are true, which we can express as $c = (x \cdot y) + (x \cdot c_{in}) + (y \cdot c_{in})$.

4. De Morgan's Laws

(a) Write the truth table for $Out = \overline{AB}$.

Answer:

A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

(b) Notice that there are three combinations of inputs for which the output of the gate will be a 1. Provide logic expressions that correspond to each of these three input cases. If you wanted to write a single logic expression that captures all cases for which the output of the gate will be a 1, what logic operator should you use to combine all three of these expressions together?

Answer:

The first set of inputs that set the output to one is $\overline{A} * \overline{B}$.

The second set is $\overline{A}B$.

The final set is $A\overline{B}$.

Looking at these three combinations, we see that as long as at least 1 of the two inputs is 0, the output is 1. This can be expressed as $\overline{A}+\overline{B}$.

Setting this equal to our original logical expression:

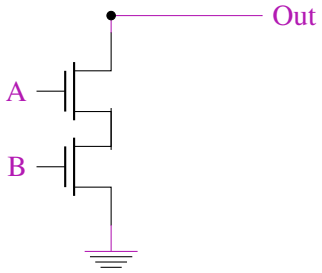
$$\overline{A\overline{B}} = \overline{A}+\overline{B}$$

Which is what DeMorgan's Law states.

(c) Create the CMOS pull down network for the NAND gate

Answer:

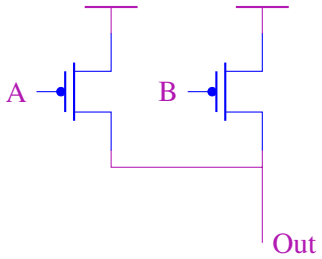
Pull down networks only use NMOS devices. To turn on the NMOS devices, $V_{gs} \geq V_{tn}$, which occurs when the gate of the NMOS is V_{DD} (or logic 1). We want the output to be connected to ground only when both A and B are 1. By putting two transistors in series, we create a circuit which corresponds to an AND operation.



(d) Next, we want to create a pull up network for the NAND gate.

Answer: Pull up networks only use PMOS devices. To turn on the PMOS devices, $|V_{gs}| \geq |V_{tp}|$, which occurs when the gate of the PMOS is 0 (or logic 0). We want the output to be connected to V_{DD} when either A or B is 0. By putting the two PMOS devices in parallel, we create a circuit which corresponds to an OR operation.

Notice how we can derive the pull up network for this circuit by taking the pull down network's logic and applying DeMorgan's law, then negating the result. The pull down logic, AB, became $\overline{A}+\overline{B}$, just like we saw in parts (a) and (b) of this question.



Contributors:

- Siddharth Iyer.
- Saavan Patel.
- Deborah Soung.
- Jaymo Kang.