CS99S
Laboratory 2 Preparation

October 1, 2001

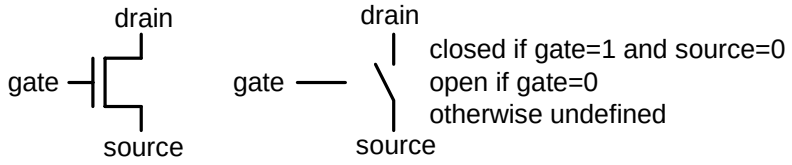
Objectives:

1. Understand the principle of static CMOS gate circuits
2. Build simple logic gates from MOS transistors
3. Evaluate these gates to observe logic function, DC characteristics, and AC characteristics

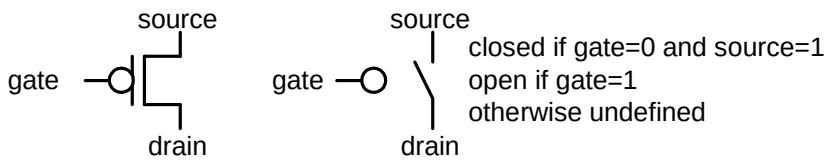
The MOS Transistor

Almost all of modern digital electronics are built from one simple device, the metal-oxide-semiconductor (MOS) field-effect transistor. MOS transistors come in two flavors, n-channel transistors – sometimes called NFETs, and p-channel transistors (PFETs).

A FET has three terminals – the *gate*, the *source*, and the *drain.* The schematic symbol for an NFET is shown below with the terminals labeled. The schematic symbol does not distinguish between the source and drain terminals. This is because they are interchangeable, for an NFET, whichever terminal is more negative is the source and the other is the drain.



For the purposes of understanding logic circuits, we can model the NFET as a switch between the source and the drain that is controlled by the gate. When the gate voltage is high (logic 1) and the source voltage is low (logic 0), the switch is closed – effectively connecting the source and drain. When the gate is low (logic 0), the switch is open – disconnecting the source and drain. When the gate is high (logic 1) and the more negative of the other two terminals is not at zero, the state of the transistor is undefined. For this reason, we can use an NFET to pass zeros, but not ones – we use a PFET to pass 1s.

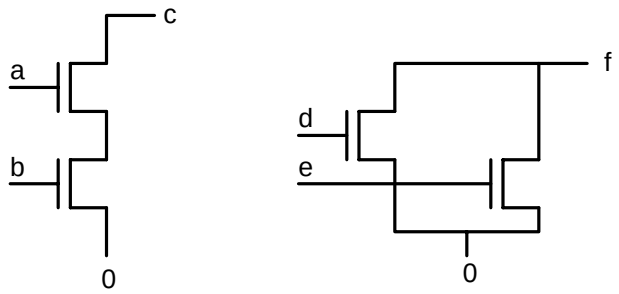


The PFET works exactly the same way an NFET does if you reverse 1 and 0. The PFET is open (or off) when the gate is high and closed (or on) when the gate is low and the source is high – hence it can pass a logic 1 or not under control of the gate. The schematic symbol for a PFET is shown above along with our switch model for the PFET. For the PFET the source is the more positive of the two interchangeable terminals, so we

tend to draw it on the top (by convention we draw schematics with voltage decreasing from top to bottom and signals propagating from left to right). The symbol for the PFET is the same as the NFET except that it includes an *inversion bubble* on the gate terminal. This bubble indicates a logical inversion – from 1 to 0 or 0 to 1 – indicating that the device is on when the gate is 0 – as opposed to the NFET which is on when the gate is 1.

Series and parallel switches

We can perform a *logical* function by connecting two switches – or two FETs together in series or parallel. For example, terminal c at the left below will be connected to 0 if a AND b are both 1. Similarly, terminal f at the right below will be connected to 0 if either d OR e (or both of them) is 1.

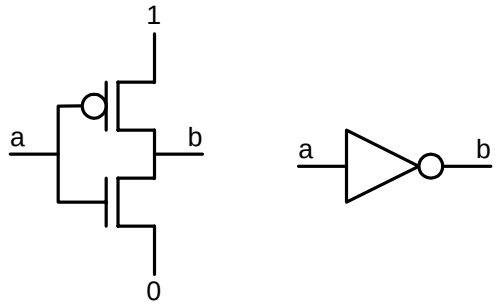


We can build similar series and parallel circuits out of PFETs. Also, while these are 2-input series and parallel networks, we can clearly extend these circuits to handle an arbitrary number of inputs.

Prep question 1: Draw an NFET network that connects the output to 0 when a AND (b OR c) is true ($a \wedge (b \vee c)$ in shorthand).

Making an Inverter

To be *composable* a circuit must generate an output that is suitable for use as an input to a similar circuit. A simple series or parallel combination of NFETs as shown above won't do this since the input needs to be either 0 or 1, but the output is either 0 or open circuited. We need to add PFETs (or a resistor) to generate a 1 on the output in this state.



The simplest composable circuit is the inverter, shown above along with its schematic symbol. When input a is a 1, the NFET is on and the PFET is off, so output b is 0. Similarly, when input a is 0, output b is connected to 1 via the PFET and the NFET is off. To save time in drawing, we use the schematic symbol on the left to indicate an inverter

and omit the power supply connections (to 1 and 0). In short, the inverter generates the logical inverse of its input. We say that b = NOT(a), or b = ~a (for shorthand).

Logic Functions and Boolean Algebra

While inverters are useful (signals never seem to be the polarity you want) we need to combine multiple logic signals to compute most functions. We can specify a logical function of several variables in several ways including an equation or a truth table. For example, the equation for the NAND (not and) function is x = NOT(a AND b), or x = ~(a ∧b) for shorthand. A truth table shows the value of the function for every possible combination of input values. For example, the truth table for the NAND function is:

a	b	x
0	0	1
0	1	1
1	0	1
1	1	0

Sometimes it is convenient to write our truth-table out in two dimensions with the values of a along one axis and the values of b along the other. This form of a truth table for the NAND is shown below and is called a Karnaugh (pronounced Car-gnaw) map.

b \ a	0	1
0	1	1
1	1	0

Just like we are used to using algebraic identities to simplify equations using + and * and real number variables, we can use the identities of Boolean Algebra to simplify equations using NOT (~) AND (∧), and OR (∨) and binary-valued variables. The basic laws of Boolean Algebra are

Zero	$a \wedge 0 = 0$	$a \vee 1 = 1$
Identity	$a \wedge 1 = a$	$a \vee 0 = a$
Negation	$\sim 0 = 1$	$\sim 1 = 0$
	$\sim(\sim a) = a$	
	$a \wedge \sim a = 0$	$a \vee \sim a = 1$
Commutativity	$a \wedge b = b \wedge a$	$a \vee b = b \vee a$
Associativity	$a \wedge (b \wedge c) = (a \wedge b) \wedge c$	$a \vee (b \vee c) = (a \vee b) \vee c$
Distributivity	$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$	$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$
Idempotence	$a \wedge a = a$	$a \vee a = a$
De Morgan's	$\sim(a \wedge b) = \sim a \vee \sim b$	$\sim(a \vee b) = \sim a \wedge \sim b$

Note that these rules are similar to, but not identical to those of algebra over + and *. For example, we can distribute OR over AND and AND over OR, but we can only distribute * over + and not the other way around. Also, + and * are certainly not idempotent. For this reason I discourage people from using + and * to represent OR and AND respectively although it is common practice.

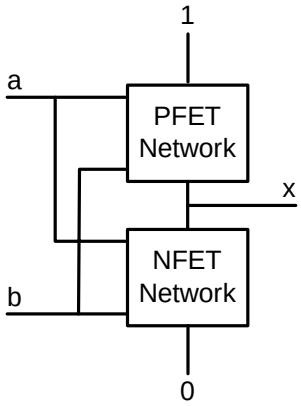
We can easily convert back and forth between truth tables and equations. To write the truth table for an equation, just substitute all possible values of the input variables into the equation and evaluate the resulting expression. Each set of values gives one row of the truth table. To convert from a truth table to an equation, we can write the logic function in *normal form* as a sum of products (an OR of ANDs). For each line of the truth table for which the output is a 1, write down the AND that corresponds to that line (e.g., ~a ∧ ~b corresponds to the line a=0, b=0) and OR the resulting ANDs together. For example, the normal form for a NAND gate is (~a ∧ ~b)∨(~a ∧ b)∨(a ∧ ~b). We can then apply the laws of Boolean algebra to simplify this expression to ~a∨~b.

Prep question 2: Write down the normal form and simplify the equation for the following truth table.

a	b	x
0	0	1
0	1	1
1	0	0
1	1	0

Logic Gates

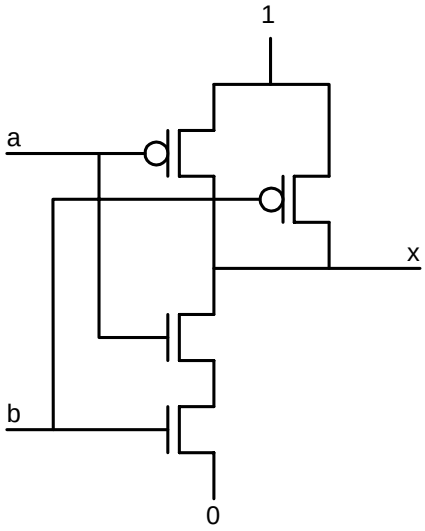
To implement a logical function of multiple signals, we build a logic gate, as shown below, by replacing the PFET of the inverter with a network (e.g., series or parallel) of PFETs that pulls up (connects the output to 1) when some logical function, f, of the inputs is true, so the output, x, is 1 when f(a, b, …) is true. To handle the case when f is false, we replace the NFET of the inverter with a network of NFETs that pulls down the output (connects it to 0) when f is false, so the output x is 0 when ~f(a,b,…) is true.

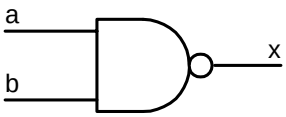


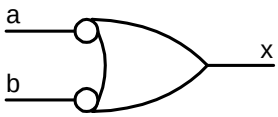
We make a NAND gate (or not-and) gate, as shown below, by using a series network for the pull-down side of the gate – so the output is 0 when a AND b are both 1 (in shorthand, when a ∧b = 1) . We use a parallel network of PFETs to pull up the output when a OR b is zero (in shorthand when ~a∨~b = 1). Thus, for this connection, f = ~(a ∧b) = ~a∨~b. and ~f = a ∧b.

The circuit diagram for the NAND and its schematic symbols are shown below. The schematic symbol on the left reflects the x = ~(a ∧b) interpretation – the squared gate

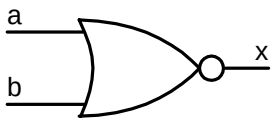
symbol means AND and the bubble (as above) means NOT, so x = NOT(a AND b) or ~(a ∧b). The symbol on the right is the x = ~a∨~b interpretation, the rounded gate symbol means OR and the inversion bubbles again mean not, so x = (NOT a) OR (NOT b) or ~a∨~b. This is the same function as ~(a ∧b) – write out the truth tables to convince yourself if you're not sure.

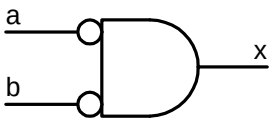






Another useful gate is the NOR gate which has the equation x = ~(a∨b). The two symbols for the NOR gate are shown below along with the truth table for a NOR

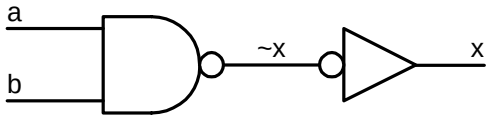


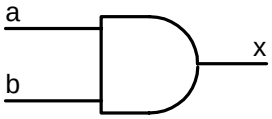


a	b	x
0	0	1
0	1	0
1	0	0
1	1	0

Prep question 3: Sketch a transistor diagram for a 2-input NOR gate.

At this point, you may have noticed that all of the gates we are making are *inverting*. This is because the NFETs take a 1 input to generate a 0 output and the PFETs take a 0 input to make a 1 output. Any static CMOS gate you make according to the rules above will be inverting. To make a non-inverting gate, like an AND we need to use two levels of gates as shown below. The figure at the left shows how we realize the AND function with a NAND gate and an inverter. Note that we can draw an inverter with the bubble on either side and here we obey the convention that we connect bubbles to bubbles to preserve the polarity of the logic. The figure at the right is the schematic symbol for an AND gate. Finally we also show the truth table for an AND.





a	b	x
0	0	0
0	1	0
1	0	0
1	1	1

As noted in the reading, an inverting gate like a NAND is complete in that we can build any function out of just NAND gates. This is not true of AND gates since there is no way to make an inverter out of an AND.

Prep question 4: Show how to create an OR gate by composing inverting gates.

Another useful logic function is exclusive-or, sometimes called XOR. The equation for XOR is $x = (a \wedge \sim b) \vee (\sim a \wedge b)$ and we sometimes abbreviate XOR by writing $x = a \oplus b$. The truth table for XOR is shown below in both forms.

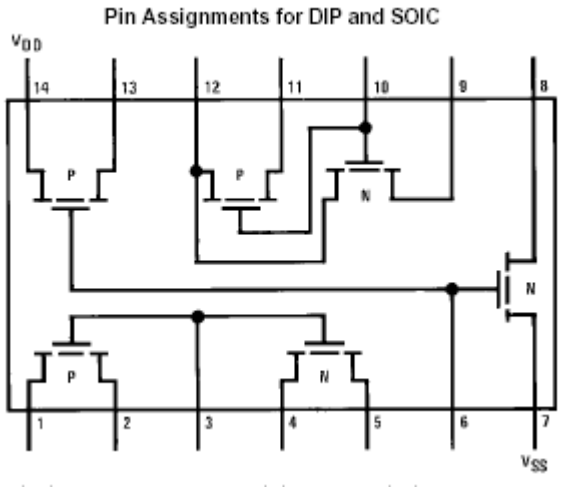
a	b	x
0	0	0
0	1	1
1	0	1
1	1	0

b \ a	0	1
0	0	1
1	1	0

Prep question 5 (optional challenge question): Sketch an implementation of an XOR gate using NFETs and PFETs. A prize goes to the solution with the minimum number of transistors.

The CD4007

To experiment with building static CMOS logic gates from MOS transistors, we will be using the Fairchild CD4007 integrated circuit. As shown in the pinout below, this device consists of three NFETs and three PFETs with some of their terminals tied together. Note that you must tie pin 14 to 1 (Vdd) and pin 7 to 0 (GND) for this part to work properly.



Using this device, we will wire up some simple logic gates and characterize their AC and DC characteristics.

The gates we will wire up will be

1. An inverter

2. A 2-input NAND gate
3. A $\sim(a \wedge (b \vee c))$ gate (see prep question 1).
4. (optional) Your XOR gate from prep question 5

In preparation for the lab, sketch how you will wire up these gates using the CD4007. Specifically, draw a schematic showing the pins used by the source, gate, and drain of each transistor.

Once you wire up these three gates, you will evaluate each of them using the following three steps.

1. Verify their logical operation using switch inputs and an LED as output. Note that due to the low drive strength of the CD4007, you may need to buffer the LED using your 74AC14.
2. Trace the DC transfer curve – input voltage vs. output voltage for at least one of your gates.
3. Observe the AC transfer curve – input and output waveforms vs time.