Using more complicated series/parallel networks of switches, we can build devices that implement more complex logic functions.

To design a more complex logic gate, first figure the series and parallel connections of PFET switches that will connect the gate's output to V_DD for the right combination of inputs.

In this example, the output F will be 1 when A is 0 OR when both B is 0 AND C is 0.

The "OR" translates into a parallel connection, and the "AND" translates into a series connection, giving the pullup circuit you see to the right.

To build the complementary pulldown circuit, systematically walk the hierarchy of pullup connections, replacing PFETs with NFETs, series subcircuits with parallel subcircuits, and parallel subcircuits with series subcircuits.

In the example shown, the pullup circuit had a switch controlled by A in parallel with a series subcircuit consisting of switches controlled by B and C.

The complementary pulldown circuit uses NFETs, with the switch controlled by A in series with a parallel subcircuit consisting of switches controlled by B and C.

Finally combine the pullup and pulldown circuits to form a fully-complementary CMOS implementation.

This probably went by a quickly, but with practice you'll get comfortable with the CMOS design process.

Mr. Blue is asking a good question: will this recipe work for any and all logic functions?

The answer is "no", let's see why.

Using CMOS, a single gate (a circuit with one pullup network and one pulldown network) can only implement the so-called inverting functions where rising inputs lead to falling outputs and vice versa.

To see why, consider what happens with one of the gate's inputs goes from 0 to 1.

Any NFET switches controlled by the rising input will go from off to on.

This may enable one or more paths between the gate's output and GROUND.

And PFET switches controlled by the rising input will from on to off.

This may disable one or more paths between the gate's output and V_DD.

So if the gate's output changes as the result of the rising input, it must be because some pulldown path was

enabled and some pullup path was disabled.

In other words, any change in the output voltage due to a rising input must be a falling transition from 1 to 0.

Similar reasoning tells us that falling inputs must lead to rising outputs.

In fact, for any non-constant CMOS gate, we know that its output must be 1 when all inputs are 0 (since all the NFETs are off and all the PFETs are on).

And vice-versa: if all the inputs are 1, the gate's output must be 0.

This means that so-called positive logic can't be implemented with a single CMOS gate.

Look at this truth table for the AND function.

It's value when both inputs are 0 or both inputs are 1 is inconsistent with our deductions about the output of a CMOS gate for these combinations of inputs.

Furthermore, we can see that when A is 1 and B rises from 0 to 1, the output rises instead of falls.

Moral of the story: when you're a CMOS designer, you'll get very good at implementing functionality with inverting logic!