PROFESSOR: Instead of building all of our logic functions directly in CMOS, it is easier for us to create a higher level of abstraction known as Boolean gates, which represent CMOS gates. Each gate is assigned a symbol, which can then be used in schematic diagrams that combine multiple logic gates together. In order to be able to understand what function any combination of logic gates will produce, we will begin by reviewing the basic gates together with the truth tables that define their logic.

We begin with an inverter. An inverter is a gate that has a single input and a single output. The output is simply the inverse of the input. When A equals 0, Y equals 1. And when A equals 1, Y equals 0. Next, we examine AND and OR gates. The AND function expects all of its inputs to be true, or 1, in order to produce a 1 as its output. So its truth table is for A, B equals 0, 0, Y equals 0. For A, B equals 0, 1, Y equals 0. For A, B equals 1, 0, y equals 0. And for A, B equal 1, 1, Y equals 1.

The OR function just expects at least one of its inputs to be true in order to produce a 1 as its output. So for A, B equals 0, 0, Y equals 0. But for the other three combinations, Y equals 1. NAND and NOR gates simply produce the inverse of AND and OR. The reason that we like to work with NAND and NOR gates is because they are inverting gates, and they can be implemented as a single CMOS gate, whereas the AND and OR gates cannot.

Finally, the last basic gate is an exclusive or. The exclusive or produces a 1 output if exactly one of its two inputs is a 1 and 0 otherwise. So for A, B equals 0, 0, Y equals 0. For A, B equals 0, 1, Y equals 1. For A, B equal to 1, 0, Y equals 1. And for A, B equals to 1, 1, y equals 0.

The output of one Boolean gate can be used as an input to another Boolean gate. So multiple gates can be used to generate more complex functions. For example, here we have a circuit that consists of two inputs and six gates, which are an inverter, an AND gate, and OR gate, two NOR gates, and 1 NAND gate. In order to figure out what this combination of gates produces as its output, we can work incrementally through the circuit.

We begin by enumerating all our choices of inputs for A and B. We know that the output of the inverter is the complement of B. The AND and OR gates use the A and B inputs directly. So since we just reviewed what AND and OR gates produce, we can fill in these columns, as well.

Next, we want to see what our first NOR gate produces. Its inputs are A and the complement of B. We walk through each of the input combinations and determine the corresponding output for that gate. For input 0, 1, the output is 0. For 0, 0, the output is 1. For inputs 1, 1, the output is 0. And for input 1, 0, the output is 0.

In the same manner, we evaluate the outputs for our second NOR gate. Its inputs are the outputs of the first NOR gate and the AND gate. Again, we simply walk through each combination of inputs and specify the outputs produced by the second NOR gate. When both inputs are 0, the output is 1. When one input is 0 and the other is a 1, the output is a 0.

Finally, we take those outputs, together with the outputs of the OR gate, and generate the final output of the circuit at the output of the NAND gate. Here, when both inputs are 1, we get a 0 output. Otherwise, the output is a 1. This NAND column of our truth table is the resulting output H.

Now that we have evaluated what our output H is equal to for each combination of A and B, we can represent the circuit as a single truth table whose inputs are A and B and whose output is H. At this point, we can express the function H as a combination of all the cases that make H equal to 1. This occurs if A and B are both equal to 0, or if A equals 0 and B equals 1, or if A and B are both equal to 1.

This can be expressed as a Boolean logic expression as follows. H equals not A not B, or not A B, or A B. This notation is called the sum of products notation. Any sum of products can be converted into a simple gate representation of the function using only inverters, AND, and OR gates by having one large OR gate that receives as input one AND gate per product term. Inverters are used as needed to complement the inputs to the AND gates.

One thing that's interesting to note about this combination of gates is that it can easily be converted into a circuit that consists purely of NAND gates. The way to do this is to realize that if you take the output of a gate and invert it twice, you've produced the original output. This means that we can add two inverters between each AND output and OR input. We can draw these inverters as bubbles, which represent inversion, where we place one bubble on the output of the AND gate and the other bubble at the input to the OR gate.

We know that an AND gate followed by an inversion is simply a NAND gate. We also know that an inverter is equivalent to a NAND gate with both of its inputs tied together. In addition, using

De Morgan's law, we know that not A or not B is equivalent to not of A and B. This means that the bubbles followed by the OR gate can also be replaced with a NAND gate, thus arriving at our equivalent circuit representation consisting purely of NAND gates.

The advantage of using NAND gates to implement the circuit is that NAND gates are inverting logic, where each gate can be implemented as a single CMOS gate. All functions can be expressed as a combination of NAND gates. So a NAND gate is considered a universal gate.

NOR gates are also universal and can be used to express any function. Furthermore, any circuit that can be used to implement any other function is also considered universal. To determine if a gate G is universal, one needs to check if one can convert G into either a NAND or NOR gate by simply using one or more copies of G together with low and high constant inputs.