Episode 4.04 – NAND, NOR, and Exclusive-NOR Logic

Welcome to the Geek Author series on Computer Organization and Design Fundamentals. I'm David Tarnoff, and in this series we are working our way through the topics of Computer Organization, Computer Architecture, Digital Design, and Embedded System Design. If you're interested in the inner workings of a computer, then you're in the right place. The only background you'll need for this series is an understanding of integer math, and if possible, a little experience with a programming language such as Java. And one more thing. Our topics involve a bit of figuring, so it might help to keep a pencil and paper handy.

Did Benjamin Franklin mess things up? There are some who say he did. Back in Ben's day, people believed that the phenomenon of electricity was due to the interaction of two fluids flowing towards one another in an effort to cancel each other out. Ben proposed a single fluid theory suggesting that the flow was from an excess of fluid to an absence of fluid. We won't get into how it was fashionable at the time to prove these theories by sending electrical charges through human chains and rate each individual's level of pain. Let's just say that there was no dual fluid canceling happening.[1] In his single fluid theory, Benjamin Franklin used the term "positive" to describe an excess of fluid and "negative" to describe an absence of fluid.[2] As any present-day physics student can tell you, while current flows "down" from positive to negative, what's actually doing the flowing are the negatively charged electrons, not some invisible fluid.

Skipping about 200 years of history, we come to the transistor, the device used by most digital machinery to process binary. Because electrons have faster mobility than an absence of electrons does, circuits designed around the negative charge have faster switching speeds.[3] The types of transistors that use electrons as charge carriers are referred to as NPN transistors. These transistors go "active" by making a connection to ground or zero volts. In digital circuitry, this signal level has traditionally been associated with a logic zero. That means that the active state of a logic gate works quicker as a logic zero rather than a logic one.

The term "active low" is used to describe the situation where a logic zero is considered the "on" or active condition. In other words, we describe a signal as active low when a logic zero is associated with the on or true value. A logic one level indicates an idle or off value. This alternate way of looking at a binary signal forces us to re-examine how we define the behavior of our basic logic gates. Instead of saying that a gate outputs a logic one based on certain input conditions, we now say that a gate goes active based on those input conditions.

If we apply this to the AND gate keeping in mind that the AND gate goes "active" when all of its inputs are one, we see that we can make a quicker logic gate by having it output a zero when all the inputs are one and a one when any input is a zero. In other words, the circuitry used to realize an AND gate with an inverted output is faster than a classic AND gate circuit. This alternate form of AND gate with an inverted output, in other words, a NOT-AND, is so common that it has a name all its own: the NAND gate.

Later in this series, we will show how the NAND gate has an additional benefit. It turns out that by using the proper design techniques, every possible digital circuit can be realized with combinational logic made entirely of NAND gates. This condition, where any complete digital circuit can be realized using only one type of gate, is referred to as functional completeness.[4]

As for drawing a NAND gate, it's not that much different than the AND gate. Remember in Episode 4.01 – Intro to Logic Gates, we described how an inverter is drawn by placing a small circle (sometimes referred to as an "inversion bubble") at the output tip of a triangle. We also mentioned how it is the small circle that represents the inversion operation, not the triangle. The schematic symbol for a NAND gate works the same way. By placing an inversion bubble against the curved end of the AND gate and drawing the output coming from the bubble, we've drawn the NAND gate.

The OR gate also has an active low output version called the NOR gate. The NOR gate outputs a logic one only if all of the inputs are zero. It outputs a logic zero if any of its inputs are one. The schematic symbol for the NOR gate places an inversion bubble at the output tip of the OR gate and has the output leaving the gate from that bubble. And like the NAND gate, the NOR gate is functionally complete.
Lastly, the Exclusive-NOR gate outputs a one as an indication that an even number of ones is being input to the gate and outputs a logic zero otherwise. Its schematic symbol places an inversion bubble at the tip where its output is connected.

Before we conclude this episode, let's take a moment to discuss one of the more important uses of the NAND gate. In digital circuitry, we often want to indicate when a specific condition has occurred. For example, let's say we want to create a signal that goes active when a 4-bit value equals 0-1-1-0. We would also like that signal to be active low in order to take advantage of the performance of the NPN transistor.

To begin with, let's label each bit of the 4-bit number. We do this using integer subscripts that match the power to which two is raised for that bit position. In other words, D_0 represents the 2^0 position, which is the least significant bit position, D_1 represents the 2^1 position, D_2 represents the 2^2 position, and D_3 represents the 2^3 position, which is the most significant bit position.

The circuit that detects when our number equals 0-1-1-0 can be defined with the sentence, "Our output needs to be active when D_3 is NOT one, when D_2 is one, when D_1 is one, and when D_0 is NOT one." There are two things to note here. First, the contraction used is "and", hence we will be using the AND operation. Since we want the output to be active low, we will be using the NAND gate. Second, notice that both D_3 and D_0 are supposed to be NOT one for a successful compare. That means that both the D_3 and D_0 signals should pass through inverters before entering the NAND gate inputs. We represent this by placing inversion bubbles right at the point where D_0 and D_3 enter the back of the NAND gate. That way, an input pattern of 0-1-1-0 will actually be a pattern of 1-1-1-1 immediately at the inputs of the NAND gate, thereby making its output go active, which will be a logic zero.

This type of circuit is referred to as a "binary decoder", and its purpose in life is to go active for exactly one pattern of ones and zeros at its inputs.[5] For a set number of inputs, n, there are 2^n possible decoders. For example, if we have four bits, there are 2^4 or 16 different ways we can set up inverters at the inputs to the NAND gate in order to detect one of the 16 possible patterns of ones and zeros available for the four bits.

In our next episode, we present a mathematical-like method for representing logic circuits along with some techniques to manipulate them for faster performance or a lower chip count. These tools can then be used to effectively design the components of a computer system.

For transcripts, links, or other podcast notes, please visit us at intermation.com where you will also find links to our Instagram, Twitter, Facebook, and Pinterest pages. Until the next episode remember that while the scope of what makes a computer is immense, it’s all just ones and zeros.

References:

1 – https://www.revolutionary-war-and-beyond.com/benjamin-franklin-and-electricity.html
2 – https://whyy.org/articles/does-our-confusing-electrical-nomenclature-start-with-ben-franklins-theory/
3 – https://www.elprocus.com/difference-between-npn-and-pnp-transistor/
4 – https://www.geeksforgeeks.org/digital-logic-functionality-completeness/
5 – https://www.geeksforgeeks.org/digital-logic-binary-decoder/
3 – https://www.elprocus.com/difference-between-npn-and-pnp-transistor/
4 – https://www.geeksforgeeks.org/digital-logic-functionality-completeness/
5 – https://www.geeksforgeeks.org/digital-logic-binary-decoder/