Episode 3.10 – Signaling and Unipolar Line Coding Schemes

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.

Today, we're going to connect two devices. Why? Well, one device has digital data that the other device needs. The most basic way to do this is to send electrical signals down a conductor. Before a single binary digit can be shared, though, there are a few issues that must be resolved. In this episode, we're going to discuss the most basic of issues: line code. In communication systems, line code is the pattern or shape of the voltage or light used to represent the logic ones and zeros of our data.

The simplest digital connection is a conductor capable of carrying two voltage levels: a positive voltage and zero volts. This type of connection is referred to as "unipolar". Communication via light can also be considered unipolar with its levels of light and no light. In the case of electrical signals, unipolar signals are used primarily for chip-to-chip board-level communication. This is because longer range conductors have a property called capacitance which causes the voltage levels to drift toward the positive voltage thereby making it difficult to identify when the transmitted signal is supposed to be zero volts.

To solve the drift problem, some physical lines use "polar" signaling. In this case, the zero volt level is replaced with a voltage that is equal in magnitude and opposite in polarity to the positive voltage. In polar signaling, the signal should never land on zero volts; at any one time, either the positive voltage or the negative voltage is present on the line. By using opposite polarities, the voltage level drift described earlier should not happen if there is a fair balance between the time that the signal is positive and the time that it is negative.

The third type of signaling is called "bipolar" signaling. For this method, the zero volt level is brought back so that there are three electrical levels: the positive or high level, zero, and the negative or low level.

Now that we know what kinds of signal levels are available, let's look at how to create patterns out of these voltages to represent the digital zeros and ones. Before we begin, we need to lay down a few ground rules. First, the shape of the electrical signal should contain synchronization information. This is so that we can extract a timing signal, sometimes referred to as a clock signal, to identify when the next bit is coming along. If our signal does not carry this information, a second connection is needed to carry a synchronizing clock signal.

Second, the pattern representing the bits should offer good bit density. Conductors are limited in how fast they can accurately deliver signal changes. The more transitions or edges required to represent a bit, the lower the rate at which we can transfer those bits. There are other issues to address, but these two will do for now.

So, let's get to the patterns or shapes of those electrical signals beginning with two of the unipolar schemes. Since unipolar is binary in nature, we could represent a logic one with the high electrical level and a logic zero with the low electrical level. In order to keep the transmitting and receiving devices in synchronization, both the transmitting and receiving devices must agree on the time period occupied by a single bit and be synchronized as to when each bit period starts. This period must remain consistent across all the bits of the data.

This line coding scheme is referred to as Non-Return-to-Zero or NRZ. This may seem like an odd name for a signal that uses zero volts to represent one of the binary values. What we will see later is that the name distinguishes this scheme from other schemes where the electrical signal for both ones and zeros includes a transition to zero volts.

The NRZ scheme does have a problem. After a long string of successive zeros or successive ones, the receiving device may drift and lose track of where one bit ends and the next bit starts. By sending the ones and zeros as levels only, there's nothing in the signal to synchronize the timing of the bits, and therefore, a synchronizing clock should be sent in a second conductor. If we could pass along synchronizing information to the receiving device within the signal carrying the data, we wouldn't require a second conductor.

One way to do this in NRZ is to add a single bit at the beginning of a sequence of seven or eight bits. This extra bit, sometimes referred to as a start bit, lets the receiving device line up its internal clock signals so that they can read the next group of bits at the correct time intervals. The receiving device should be able to keep synchronization within the seven or eight bits of data before requiring another start bit. In order to correctly identify the moment a start bit begins, however, there also needs to be a bit of opposite level immediately preceding the start bit. This bit, referred to as a stop bit, follows every sequence of seven or eight bits and is repeated until the next start bit is sent. A sequence of consecutive stop bits indicates that the line has gone idle.

So, what about the efficiency of NRZ? Because the electrical signal of NRZ changes no faster than the bit rate, this method offers good bit density, even with the addition of the start and stop bits. Implementation is also quite simple.

Our second unipolar scheme is referred to as Return-to-Zero or RZ signaling. RZ partially solves the synchronization problem encountered with NRZ by adding a transition from one to zero in the middle of bit periods representing a logic one. In other words, a logic one is represented with a voltage pattern that is the high level for the first half of the bit period followed by the zero volt level for the last half of the period. A zero is still sent as a full bit period of zero volts. Another way that we could view this signaling is to see the first half of the bit time as equivalent to the bit value while the last half is always equal to zero.

RZ would represent a long sequence of logic ones as pulses that are half the width of the bit period. The receiving device can synchronize its internal clock to the falling edge of the signal which only occurs in the middle of a logic one. Although RZ could lose synchronization with a long series of zeros, any received logic one will resynchronize the signal. RZ does have a disadvantage with respect to NRZ, however. A logic one now requires two signal level changes per bit, which means that the signal is changing twice as fast as NRZ. This means that over the same conductor, we can now only transmit half the bits over the same period. Another way of saying this is that RZ has half the data density of NRZ.

In our next episode, we will look at five polar and bipolar schemes: NRZ-L, NRZ-I, RZ-AMI, Manchester, and differential Manchester. This may seem like a lot to cover in one episode, but if you've understood the basics of line code presented in this episode, the bipolar schemes will be a cinch. For transcripts, links, or other podcast notes, please check us out 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.