Episode 2.5 – Binary Representation of Analog Values: Fitting Infinite Inside a Computer

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.

Converting unsigned integers to binary is only one of the many ways computers use ones and zeros. We have so much more to cover. In fact, it's going to be some time before we get to fractions. In Episode 2.1, we started counting in unsigned binary and found that each bit position represented increasing powers of two from right to left. We started counting at zero, where all the binary digits were cleared to zero, and then counted up to the point where all the bits were set to one. This upper limit or maximum value was one less than two raised to the power of the number of bits in our binary value. For example, if we had eight bits, then the highest value we can count to is $2^8 - 1$ or 255. It seemed odd that we needed to subtract 1, but remember that 0 is the first of the represented values and takes up one of the 256 possible patterns of ones and zeros available with eight bits. The purpose of this episode is to show how to map all those patterns of ones and zeros to different values in an analog range instead of across integers.

Let's begin our discussion with an example: temperature. What values can temperature take on? Well, we could start with absolute zero, theoretically the lowest possible temperature. In Celsius, that would be about -273 degrees. Okay, that's a reasonable number. We can handle that. What about the other end of the range? Is there an absolute hot? One proposed value for absolute hot is around 10^{32} degrees Celsius.[1] Wow! If we were to try to represent that temperature with a binary integer, it would take 107 bits.

To begin solving this problem, let's look at a real-world example: the medical thermometer. The absolutes at either end of the scale of possible temperatures useless to us. We'd only use an infinitesimal number of those values, say between a minimum of 96 degrees Fahrenheit and a maximum of 108 degrees Fahrenheit.

Don't know if you noticed, but there are those descriptors again: minimum and maximum. What if we map the minimum analog value, in this case 96 degrees, to the minimum binary value of 0, and map the maximum analog value, in this case 108 degrees, to the maximum binary value of all bits set to one? Then we could evenly space the remaining binary values across the analog range between the minimum and the maximum.

Let's start simple and say that we are going to use only two bits to represent the different analog values across our thermometer's range. The bit pattern 00 would map to 96 degrees while a binary 11 would map to 108 degrees. We only have two binary patterns left: 01 and 10. Where should they go? Well, to space them evenly across the analog range, one pattern should be mapped to a point one third of the way between 96 and 108 while the other should be mapped at the two thirds position. Since the difference between 96 and 108 is twelve, then our intervals are twelve divided by three or four. Four up

from 96 is 100 degrees. Two thirds of the way from 96 to 108 would be eight up from 96 which equals 104 degrees. This gives us our binary to analog mapping: 00 represents 96, 01 represents 100, 10 represents 104, and 11 represents 108.

But, what about the temperatures in between? Well, the electronics being used to convert the analog measurement to binary would have to round off to the nearest available binary value. That means that 01 could represent any temperature from 98 to 102. That doesn't give us much accuracy, does it? Well, there are two ways to improve accuracy: reduce the analog range or increase the number of bits. Since we should really stick with the 96 degree to 108 degree range, let's try increasing the number of bits. In fact, let's jump all the way up to eight bits.

We start out the same way we did before by mapping the all zeros binary value to 96 and the all ones value to 108. That leaves us with 254 patterns to map across the analog range. So, what is the spacing between each increment? Well, with zero anchoring the base of our range, it takes $2^n - 1$ steps to get up to the maximum, right? That means each time we increment our binary value, we move up our analog range by one over $2^n - 1$. Our analog range is 12 degrees, so each time we increment our digital value, we move up 12 divided by 255 or about 0.047 degrees away from 96 and closer to 108. This gives us accuracy better than 1/20th of a degree with only eight bits. That's a lot fewer bits than the 110 bits we were going to need if we tried to represent all possible temperatures to the tenth of a degree!

So, what if the eight-bit value returned to us from the electronic thermometer is 00110111? What is this in degrees Fahrenheit? Well, 00110111 is equal to 32 + 16 + 4 + 2 + 1 or 55 in decimal. That means the thermometer read a value that is 55 increments up from 96 degrees. Since each increment represents 12/255 degrees, then the value the thermometer read was 55 * 12 / 255 + 96 or about 98.6 degrees Fahrenheit. And now a note about rounding. The temptation is to compute the size of the increment and round it. For example, if we'd rounded our increment size to 0.05 degrees instead of using the fraction 12/255 degrees/increment, our answer would have been off by between one and two tenths of a degree. Where accuracy is important, you'll want to leave the increment size as an un-rounded fraction. This error gets worse the further you get away from the minimum value.

So let's recap. To use an n-bit binary number to represent an analog value, we need to place limits on what is being measured. Once we have the minimum and maximum analog values, we need to decide how many bits we are going to use to represent those values. Next, we map the analog range to the binary range by setting the lower limit of the analog range to the all zeros binary value and the upper limit of the analog range to the all ones binary value. Lastly, we compute the increment size by dividing the analog range by $2^n - 1$, where n equals the number of bits. If we find the accuracy isn't good enough, we either make the analog range smaller or use more bits.

In our next episode, we're going to dive a bit deeper into this topic of representing analog values digitally. The problem lies with the fact that analog values can take on an infinitesimal accuracy at limitless speeds. There are consequences when we try to handle this with a computer. Until then remember that while the scope of what makes a computer is immense, it's all just ones and zeros.

References:

1 – Tyson, P. (2008, January 8). Absolute Hot: Is there an opposite to absolute zero? Retrieved April 5, 2019, from pbs.org: https://www.pbs.org/wgbh/nova/article/absolute-hot/