Encodings

6 min · Justin Rajewski | Suggest Changes

▶ Table of Contents

Up until now we have only talked about discreet 0s and 1s. For many cases a single bit is all you need, however, many times you need to be able to encode numbers. The far most common way to do this is to use binary encoding. This encoding is the most obvious way to do things, and it is the closest to what you are already familiar with.

Binary

Binary numbers are not very different from the decimal numbers you know. The right most digit, or the LSB (**L**east **S**ignificant **B**it), is the bit of lowest value, while the left most digit, or the MSB (**M**ost **S**ignificant **B**it), is the bit of highest value.

For decimal systems the value of each digit is d * 10n where d is the number and n is it's position from the right starting with 0. For example, the number 15 has the value of 1 * 101 + 5 * 100 = 15. It is useful to look at numbers this way when we talk about binary and other number systems. Since we use powers of 10 for our normal decimal system, it is known as **base-10**.

Binary is a **base-2** system, since each digit only has 2 possible values. That means to compute the value of a binary number you use d * 2n where d and n are the same as before. This means that the first digit has the value of 1, the next 2, then 4, 8, 16, 32,

64, and so on. Let's look at the binary number 1010. There is a 1 in the 2's and in the 8's place. That means it has a value of 2 + 8, or 10.

Hexadecimal

Hexadecimal, or commonly called hex, is a compact way to represent binary number and it not really another encoding type. Hexadecimal is a **base-16** system. Since we don't have a digit to represent values higher than 9, hex uses the letters A-F to represent 10-15 respectively. Hex is commonly written with *0x* preceding the number to show it is a hex number and not a decimal number. For example, 0xA1 has the value of 10 * 16 + 1, or 161.

The reason hex is so common is that most systems are based on an 8-bit byte. Since the value of one hex digit can range from 0-15 it effectively represents 4 bits. That means that you only need two digits to represent a byte. So a number like 0x45 is a single byte. This also makes it easy to convert back to binary because you can convert each digit separately. For example, with 0x45 first you know that 4 is 0100 and 5 is 0101, so 0x45 is 01000101.

Gray

Gray encoding, also known as Gray code, is an alternative to binary encoding. It's main benefit is that when you change between any consecutive numbers (in other words add or subtract 1 from any number) only one bit will change. This is normally not much of a benefit and the simpler binary encoding is used instead. The place this shows up the most is the format of the K-maps where each cell can only change by one bit.

Decimal	Hex	Binary	Gray
0	0	0000	0000
1	1	0001	0001
2	2	0010	0011

Decimal	Hex	Binary	Gray
3	3	0011	0010
4	4	0100	0110
5	5	0101	0111
6	6	0110	0101
7	7	0111	0100
8	8	1000	1100
9	9	1001	1101
10	A	1010	1111
11	B	1011	1110
12	C	1100	1010
13	D	1101	1011
14	E	1110	1001
15	F	1111	1000

One-Hot

This is an encoding that is pretty common but usually does not represent numbers. It is instead used for things like state machines, which we will cover later. A one-hot encoding is generally used instead of binary because it usually makes for a simpler circuit. In many designs binary is first converted into one-hot when it is used to index something.

One-hot encodings are those where there is always a single one is a number. For example, the first state could be 0001, while the next is 0010, then 0100, and 1000. The down side to one-hot is you need as many bits as you want states, however, in the next section we will show how this encoding is commonly useful.

2's Complement

So you may have noticed with all of these encodings there is no way to represent a negative number. Since binary is only 0's and 1's you can't simply put a minus sign in front! Although some encodings of negative numbers do basically just that with what is called a **sign bit**. The naive way is to just add a 0 to the beginning of positive numbers and a 1 to the beginning of negative numbers. While that would work if you designed your circuit correctly, it is not ideal.

There are many other ways to encode negative and positive numbers but the most common, and generally most useful, is 2's complement. With 2's complement the MSB is also a sign bit where 1 means the number is negative. However, when a number is negative it's value is not simply the value of the rest of the bits in binary.

A negative 2's complement number can be turned into it's positive representation by inverting all the bit's then adding 1. For example, take 1111, which is -1. When you invert all the bits you get 0000. Then add 1 and you get 0001, or positive 1. You should note that the same exact thing works for making a positive number negative.

Why would anyone ever go though all this when just using a single bit to show if a number is negative seems so much easier? This is because when numbers are represented as 2's complement doing math with them is much similar. When you add two numbers, you don't have to have special cases for when the numbers are negative or not. You can just add any two numbers and get the correct result.

Another very nice property of 2's complement is that there is only one zero. If you just use a sign bit, you end up with a positive and a negative zero, which can cause complications.

The most useful property, and one that allows negative and positive values to be treated equally, is that the number system warps around at 0. For example, if you have a 4 bit value for 1, 0001, and you subtract 2 from it. You end up with 1111, which we have already shown is -1. If you then take -1 and add 1 to it you get 0000, or 0, because of the way the binary numbers overflow.

« PREV
How does an FPGA work?

NEXT »
Multiplexers