Episode 3.12 – Run Length Limited Coding

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.

In our last episode, we discussed differential coding as it applies to transmitting digital data across a wire. It turns out that a form of differential coding is used with magnetic storage media as well. Magnetic media? Isn't that technology going away? Well, not yet. Yes, IBM developed magnetic hard disk drive or HDD technology in the 1950s, and little has changed since then, but as of this writing, HDDs still have a place in the hierarchy of storage media. The main competition for the HDD is the solid-state drive, or SSD. While SSDs outperform HDDs in most categories including speed, lower power consumption, and reliability, the cost per unit of storage of a typical SSD is around three to five times that of an HDD.[1]

As our needs for digital storage increase each year with our growing reliance on big data, we continue to depend upon another bygone technology: magnetic tape storage.[2] Even though magnetic tape is slow, its ability to act as reliable long-term storage for large amounts of static data is attractive. The typical magnetic tape cartridge has a capacity of over 6 terabytes – over 700 times the capacity of a dual-layered, single-sided DVD.

The operation of an HDD centers on the use of a read/write head positioned at the end of an arm that moves back and forth across the radius of a spinning platter. The platters are coated with a magnetizable material. To write data to the platter, current is passed through the read/write head. This current generates a magnetic field close to the platter's surface, which magnetizes a small patch in the coating. Reversing the direction of the current magnetizes the patch in the opposite direction. The size of these magnetized patches depends on the size of the read/write head and its proximity to the platter's surface. These physical constraints limit how small the patches can be, and thus, how much data we can store to a single platter.

To read or write logic ones and zeros to the magnetized coating, the platters spin beneath the read/write heads at a constant rate, typically 7200 RPM. The bits are evenly spaced across the platter surface, which means they pass beneath the read/write head at a consistent rate.

In Episode 3.10 – Signaling and Unipolar Line Coding Schemes, we discussed how a clock signal is required in any communication scheme to identify the timing of the bits. If the signal representing the data does not carry some sort of synchronizing information, then a second connection is needed to carry a synchronizing clock signal. This is true for storing data using magnetic polarity too. Unlike electrical communication where a voltage level can be measured, the devices used to read magnetic media cannot determine a constant direction of polarity. They can only detect a change in polarity from one direction to the other. Because of this, the bits stored to magnetic media are identified by the timing of the transitions as opposed to the direction of the polarity. This is similar to differential code described in Episode 3.11.

Run Length Limited, or RLL, is a line coding scheme that places a limit on the number of bits that can pass without having a transition. If too many bits times have passed without a transition in the magnetic media, the reading device may lose its synchronization to the bit periods and corrupt the data. On the other hand, if an excess of transitions are required to distinguish the ones and zeros on an HDD platter, then data density will suffer due to the physical constraints imposed by the read/write head's limit on how closely it can reliably place the polarity transitions.

In Episode 3.11, we presented a line code called differential Manchester code. In this line code, there is always a transition in the middle of a bit time. Logic ones and zeros are distinguished from one another by the absence or presence of a transition at the beginning of a bit period. The original magnetic coding scheme was a lot like this in that a magnetic field change occurs for every bit time, the difference being that this transition occurs at the beginning of every bit period instead of the middle. A transition in the middle is used to identify a logic one while the absence of a transition identifies a logic zero. This method is referred to as Frequency Modulation or FM. To store a logic one using FM encoding, the polarization of the magnetic field must change twice within the space of a bit. This means that in order to store a single bit, FM encoding takes twice the physical length of the smallest magnetic field that can be written to the substrate by the read/write head.

If we could reduce the maximum number of polarity changes per bit, more data could be stored to the same disk. Another form of RLL, Modified Frequency Modulation or MFM, does this by changing the way in which the magnetic polarizations represent logic ones and zeros. MFM is essentially like FM except that it removes the requirement to have transitions between every bit. Changes in magnetic polarity still happen in the middle of a bit time for logic ones, so as long as ones appear regularly in the stream, synchronization is maintained. In order to maintain synchronization in the absence of ones, transitions are also placed between neighboring logic zeros.

For MFM encoding, the longest period between polarity changes occurs for the bit sequence 1-0-1. In this case, there is a polarity transition in the middle of the one in the first bit period and in the middle of the one in the third bit period. This gives us a maximum of two bit periods between polarity changes. The minimum period between polarity changes occurs for a sequence of logic ones or a sequence of logic zeros. In both cases, the polarity changes are spaced by a single bit period.

Since the minimum spacing of polarity changes for FM is half a bit period while the minimum spacing of polarity changes for MFM is one bit period, the data density of MFM is twice that of FM when using the same magnetic surface and head configuration. The hard drive controller, however, must be able to handle the increased data rate.

As a side note, different forms of RLL are characterized using two parameters, d and k. These parameters represent the minimum (d) and maximum (k) run-length for which the signal remains unchanged. For example, MFM has a minimum run-length of two-halves of a bit time before a change occurs, in other words, there is one half where no change occurs. MFM has a maximum run-length of four-halves of a bit time before a change occurs, in other words, there are three halves where no change occurs. This gives us the alternate name for MFM of RLL 1,3, where 1 represents the d parameter and 3 represents the k parameter.

We run into a problem at this point. By requiring unique transition patterns to distinguish logic ones from logic zeros while also having enough transitions to maintain synchronization across long sequences of the same bit value, we must have at least one polarity transition per bit. If instead of encoding at the bit level, we encode sequences of bits, we can further increase the density of bits stored to a platter. We simply need to ensure that we've create enough different bit sequences so that we can represent any sequence of ones and zeros that may occur in our data.

One of the first RLL encoding schemes to encode groups of data bits was RLL 2,7. It is based on seven different sequences of bits. The first two sequences are the only two that are two bits long and the only two that start with a one. The sequence 1-0 is represented with a single transition in the middle of the first bit period while the sequence 1-1 is represented with a single transition at the beginning of the first bit period. Note that neither of these sequences has a transition for the last one and a half bit periods of the sequence.

The next sequence represents the three bit pattern 0-0-0. This sequence has a transition in the middle of the second bit period, once again leaving no transition for the last one and a half bit periods of the sequence. The fourth sequence, 0-1-0 also has a transition in the middle of the second bit period, but distinguishes itself from 0-0-0 by also having a transition at the very beginning of the sequence. These two transitions are also separated by one and a half bit periods. The fifth and last of the three-bit sequences is 0-1-1. It has a transition between the first and second bit periods leaving two bit periods without a transition until the end of the sequence.

The last two sequences represent four data bits: 0-0-1-0 and 0-0-1-1. The first sequence has a transition between the first and second bits, then has a second transition one and a half bit periods later in the middle of the third bit. This last transition occurs one and a half bit periods before the end of the sequence. The bit sequence 0-0-1-1 has a single transition in the middle of the sequence leaving two bit periods before and two bit periods after it.

The thing to note about all of these groups is that the transitions within the sequence are all spaced at least one and a half bit periods apart and that all the sequences have at least one and a half bit periods between the last transition and the end of the sequence. That means that the minimum possible period between polarity changes is one and a half bit times. This increases our bit density by 50% over MFM, which has a minimum space between transitions of a single bit period. A maximum period between transitions happens when the bit sequence 0-0-1-1 occurs twice in a row. In this case, a space of four bit periods exists between polarity changes.

Note that since successive bytes are stored as long sequences of ones and zeros, it is easy to represent the sequences with complete RLL 2,7 bit groups. Only at the end of the sequence is it a problem trying to come up with a matching group of bits from the RLL 2,7 patterns. In this case, additional "throwaway" bits can be appended to the end of the sequence in order to complete a group.

Now let's talk about the d and the k parameters. RLL 2,7 has a minimum run-length of three-halves of a bit time before a change occurs, in other words, there are two halves where no change occurs, hence the two. This encoding scheme has a maximum run-length of eight-halves of a bit time before a change occurs, in other words, there are seven halves where no change occurs, hence the seven.
In our next episode, we are moving away from data representation and into the hardware that processes it. We will begin this journey by discussing logic gates. 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.

References:

1 – https://www.backblaze.com/blog/ssd-vs-hdd-future-of-storage/
2 – https://spectrum.ieee.org/computing/hardware/why-the-future-of-data-storage-is-still-magnetic-tape