How to Generate Manchester-Encoded Data in Hardware and Firmware

This article discusses implementation details for a simple yet effective technique that can improve your digital communication.

In a previous article (very previous, like over a year ago), I wrote about the [“what and why” of Manchester encoding](https://www.allaboutcircuits.com/technical-articles/manchester-encoding-what-is-it-and-why-use-it/), i.e., what exactly Manchester encoding is and why you might want to incorporate it into your wired or wireless data link. In this article, after briefly reviewing this what-and-why information, we’ll discuss something rather useful, namely, how to actually generate Manchester-encoded data once you’ve decided that it could be beneficial for your system.

Manchester encoding is a form of modulation, but don’t worry, it’s much simpler than the RF techniques ([FSK](https://www.allaboutcircuits.com/textbook/radio-frequency-analysis-design/radio-frequency-modulation/digital-modulation-amplitude-and-frequency/), [PSK](https://www.allaboutcircuits.com/textbook/radio-frequency-analysis-design/radio-frequency-modulation/digital-phase-modulation-bpsk-qpsk-dqpsk/), etc.) that are more commonly associated with this word. This modulation scheme requires only digital data, the corresponding clock signal, and a bit of hardware or firmware. The differences between the modulated data and the original data are the following:

* Data is represented using logic-level transitions rather than logic levels.
* The clock signal does not need to be sent to the receiver because the clock is embedded in the modulated data stream.

The following diagram shows you how a data-plus-clock pair corresponds to a Manchester-encoded signal.

In addition to eliminating the need for a transmitted clock signal, Manchester encoding makes it possible to use AC coupling, because the modulated signal doesn’t stay at logic high or logic low for an extended period of time. An important disadvantage of Manchester encoding is the reduced data rate: because ones and zeros are represented using transitions instead of logic levels, one logic state in the original signal turns into two logic states in the Manchester signal.

### **Manchester via Hardware**

In theory, it is extremely easy to generate a Manchester-encoded data stream via hardware. In contrast to the carefully designed analog circuitry or the powerful digital signal processors that are used for advanced RF modulation techniques, Manchester encoding requires only an XOR gate.

You can easily verify this using the timing diagram shown above: when clock and data are at the same logic level, the Manchester signal is low; when they’re at different logic levels, the Manchester signal is high.

The reason that this is a more “theoretical” implementation is because the output of the XOR gate will be subject to spurious transitions, aka glitches. This is a nontrivial problem because Manchester encoding is all about the transitions. These spurious transitions occur because the XOR gate is a very straightforward device. All it does is compare two inputs and generate an output according to the XOR truth table. If the clock and data signals are not perfectly aligned (and it’s never a good idea to expect perfection), one will transition before the other, and the XOR gate is fast enough to update its output based on this transient (and erroneous) combination of the two input signals.

I don’t think that the nothing-but-an-XOR approach is useless. If you can ensure very good synchronization between the data and clock signals—such as when the two signals are generated and XORed inside an FPGA—it could probably perform quite well. Also, the effect of imperfect synchronization is less significant when the data period is very long relative to the glitch duration, because the (higher-frequency) spurious transitions can be filtered out before the Manchester data is interpreted by the receiver.

As far as I know, this glitch issue cannot be resolved simply by adding a gate here or a flip-flop there. Internet searches don’t produce much information, probably because nowadays it is much more common to generate the Manchester data in firmware (see the next section). However, it appears that Harris A. Quesnell Jr. was determined to generate glitch-free Manchester data without the aid of firmware, and you can read about his circuit in the [patent documentation](https://patents.google.com/patent/US4100541).

Like I said, not the sort of thing you can accomplish by adding a couple gates or a flip-flop. I appreciate hardware solutions, but in this case, I recommend that you either cope with the glitches or move the Manchester action into firmware.

### **Manchester via Firmware**

A firmware implementation eliminates the spurious transitions because the original data is converted into a Manchester sequence of ones and zeros before it becomes a normal electrical signal. All you have to do is drive this sequence of ones and zeros onto a GPIO pin, and you’re done. In this case, the advantages over a hardware implementation are quite significant. Most systems these days already include a processor that could handle this task.

There’s not much to say about the details required to accomplish firmware-based encoding. You take the original data and replace each one or zero with a one-zero pair (i.e., a falling-edge transition) or a zero-one pair (i.e., a rising-edge transition). You could then drive these Manchester bits onto an output pin using an interrupt service routine associated with a timer that overflows according to the desired data rate. Another option is to use one of the processor’s serial-communications peripherals.

### **A Microcontroller-Plus-Hardware Approach**

Silicon Labs makes microcontrollers that include a configurable logic unit (CLU). The CLU includes some typical digital hardware that can be customized by the user. In other words, it’s like a tiny FPGA that allows the microcontroller to perform certain tasks without any CPU intervention. Anyways, [this app note](https://www.silabs.com/documents/public/application-notes/AN921.pdf) from Silicon Labs presents some CLU implementation examples, and one of these examples ([page 10](https://www.silabs.com/documents/public/application-notes/AN921.pdf#page=10)) is a Manchester encoder/decoder. The SiLabs scheme uses the SPI peripheral in conjunction with two CLUs to generate (presumably glitch-free) Manchester data.

If you expect to have a microcontroller in your system but don’t want to dedicate significant processor resources to the Manchester conversion, I suggest that you take a close look at this implementation.

### **Conclusion**

We briefly reviewed the salient characteristics of Manchester encoding, and we discussed hardware- and firmware-based techniques for converting original logic-level data into a Manchester signal. In a future article, we will discuss another rather essential aspect of this topic, namely, how to convert Manchester data back into a standard serial data stream.