

# Manchester encoding using RS-232

Author: Adrian Mills  
Date: 30-03-2009  
www.summitelectronics.co.uk  
Rev 2.1

## INTRODUCTION

The following takes a look at transporting Manchester encoded data [1] using RS-232, over a radio frequency (RF) link.

Those familiar with RS-232 / RS-485 etc will be aware that when data is sent through a cable to remote equipment it gets there with good reliability. If error detection is required, this can take the form of a checksum as part of a packet-based protocol. However, should you want to send the same data over an RF link a number of things have to be taken into consideration. Most notably is the need to maintain a zero DC component in the serial bit stream.

RS-232 / RS-485 is generally encoded and decoded using a universal asynchronous receiver and transmitter (UART). It is the UART we shall be using to transport Manchester encoded data.

## MANCHESTER ENCODING

Manchester encoding (also known as Biphase Code) is a synchronous clock encoding technique used to encode the clock and data of a synchronous bit stream. In this technique, the actual binary data to be transmitted over the cable or RF link are not sent as a sequence of logic 1's and 0's as in RS-232 (known technically as Non Return to Zero (NRZ)). Instead, the bits are translated into a slightly different format that has a number of advantages over using straight binary encoding (ie NRZ).

The main advantages of using Manchester encoding are:

1. Serial bit stream has a DC component of zero
2. Error detection is simple to implement

In general, when transmitting serial data to a radio receiver, a DC component of zero must be maintained (over a finite time). This is so the demodulator in the receiver can properly interpret (discriminate) the received data as 1's and 0's. Manchester encoding allows us to do this.

Manchester encoding follows the rules:

1. If the original data is a Logic 0, the Manchester code is: 0 to 1 (upward transition at bit centre)
2. If the original data is a Logic 1, the Manchester code is: 1 to 0 (downward transition at bit centre)

It can be seen that there are two bits of Manchester encoded data for each bit of original data. The penalty for doing this is Manchester encoded data requires more bandwidth than NRZ encoding.

### Example of Manchester Encoding:

The pattern of 8-bits: "0 1 1 1 1 0 0 1" encodes to "01 10 10 10 10 01 01 10". The least significant bit (LSB) is shown left most. Figure 1 shows a drawing of the encoded pattern as a waveform in the time domain.

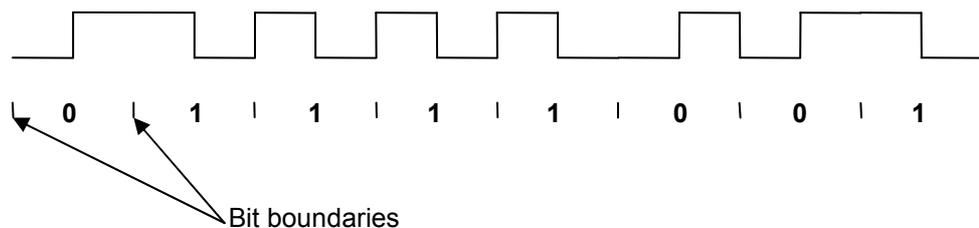
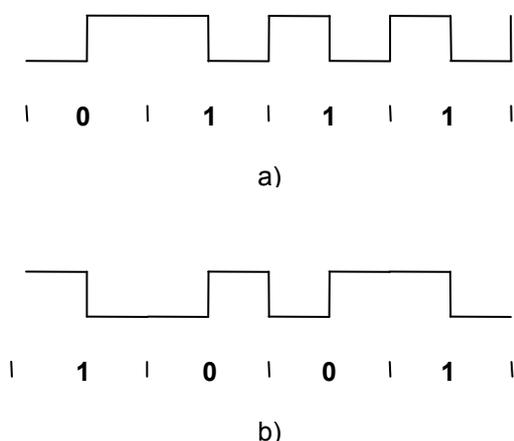


Figure 1 Waveform for the Manchester encoded bit stream carrying the sequence 01111001

Alternatively the above can be represented as two 4-bit nibbles

1. The pattern of bits " 0 1 1 1 " encodes to " 01 10 10 10 " (see Figure 2a)
2. The pattern of bits " 1 0 0 1 " encodes to " 10 01 01 10 " (see Figure 2b)



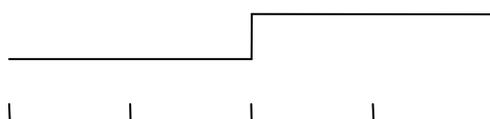
**Figure 2 Waveforms for the Manchester encoded bit stream carrying the sequence 0111 & 1001**

This step leads us to sending 8-bits of original data as two bytes of Manchester code.

### Illegal codes

Given that a 0 encodes to 01 and 1 encodes to 10, it follows that Manchester codes 00 and 11 are illegal sequences. These illegal codes are used to error check the data.

It is also possible to have the 4-bit illegal code " 00 00 11 11 ", which would be an unlikely occurrence. This 'illegal code' has the property of having a DC component of zero and has no 1 to 0 transitions. We can use this 'illegal code' (" 00 00 11 11") as a unique start/stop pattern to identify the boundaries of our Manchester encoded bit stream or data frame (see Figure 3). It will also serve to initialise the radio receivers' demodulator and synchronise the receiving UART.

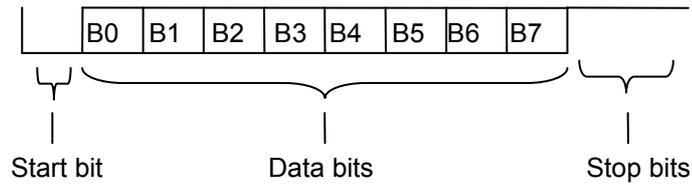


**Figure 3 Unique start/stop pattern**

### RS-232 Wave Form

In RS-232, user data is sent as a number of frames each a synchronous time-series of bits, typically eight bits in length. The diagram in Figure 4, where B0 is data LSB, shows the expected waveform from the UART when using eight data bit, no parity and two stop bit format, .

We can use RS-232 as a carrier for Manchester encoding provided that RS-232 characters (bytes) are transmitted end-to-end with no inter character delay. RS-232 has a start bit, logical 0, and a stop bit, logical 1 which have a DC component of zero. In our real life application 2-stop bits may be all that's available to us. What 2-stop bits means is that the DC component is no longer zero, so here we make a compromise and say the DC component is near zero (i.e. the extra stop bit has a 1/10 effect on the DC component). However this can be compensated for when the transmitter supports 9-bit data mode by setting the ninth bit to zero and hence cancelling out the second stop bit.

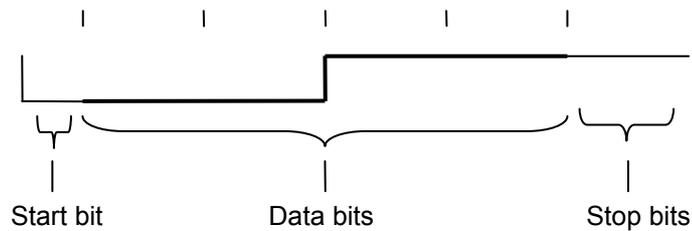


**Figure 4 RS-232 Logic wave form**

**RS-232 carries Manchester encoding**

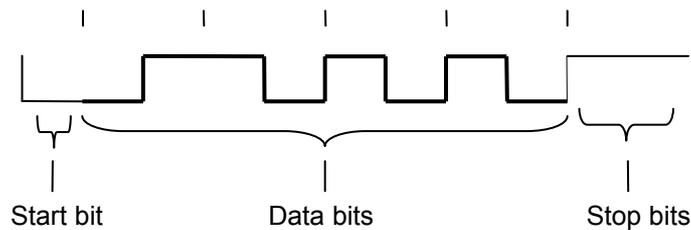
All that's required is to insert Manchester encoded data into the RS-232 data bits. There are 4 original data bits in each RS-232 byte.

Figure 5 shows the start/stop pattern inserted into an RS-232 frame. This pattern is chosen for its property of not having any 1 to 0 transitions. This means there can be only one real start bit for the UART to synchronise to.

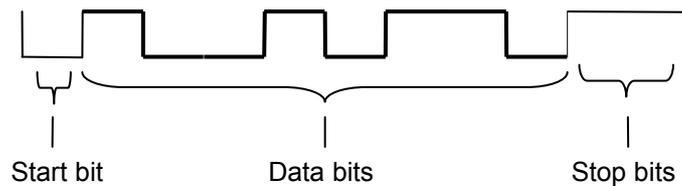


**Figure 5 Start/Stop pattern**

Figure 6 a) and b) show the sequences from Figure 2 inserted into two RS-232 frames.



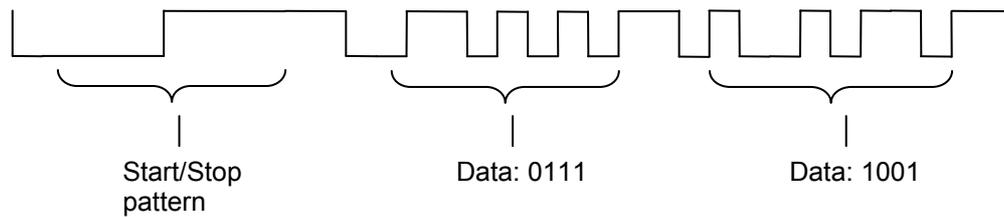
a)



b)

**Figure 6 RS-232 frames for the Manchester encoded sequence a) 0111 & b) 1001**

Finally Figure 7 shows the waveform of the combined frames from Figure 5 & Figure 6.



**Figure 7 Combined waveforms**

### APPLICATION

In this application it is assumed the 'C' functions `putc()` and `getc()` directly output or input TTL levels from a radio transmitter or receiver respectively to a UART.

#### Start data frame

The data frame is started by transmitting a preamble of start/stop patterns (SS) to the radio receiver. The number of SS patterns depends on the characteristics of the radio receiver.

In general, enough SS patterns are needed to:

1. Initialise the receiver demodulator.
2. Allow the RS-232 decoder to synchronise to the start bits of each RS-232 byte.\*
3. Indicate to the software communications handler that an SS pattern has been received and data is expected to follow.

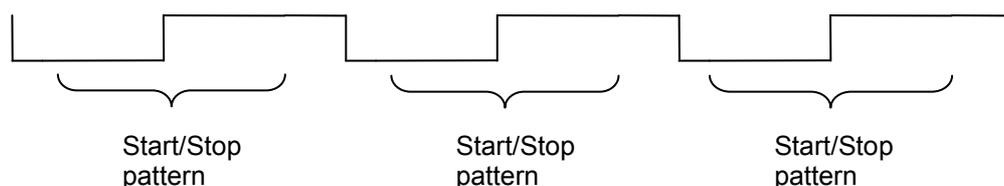
\* An additional stop bit is inserted at the end of each SS pattern to allow for software implemented RS-232 decoders (UART) to 'catch up' with the start bit. This may be omitted for hardware UARTs.

The unique SS pattern " 00001111 " (LSB left most) is equivalent to the hex code 0xF0 (LSB right most). The following 'C' function `SendSS()` will send a preamble of four SS patterns.

```
SendSS( void ) {
{
    putc(0xF0);
    delay_us(500); // *
    putc(0xF0);
    delay_us(500); // *
    putc(0xF0);
    delay_us(500); // *
    putc(0xF0);
    delay_us(500); // *
};
```

\* Additional 500us stop bit based on a bit rate of 2000 bits per second (bps).

The received stream will look similar to the waveform shown in Figure 8 below.



**Figure 8 Wave form of start/stop patterns**

Note: In a real-life radio receiver, the absence of a signal from the transmitter will generally result in a received bit stream of noise (i.e. random 1's and 0's in no recognisable pattern). It's not until the transmitter is sending a signal that the receiver can 'discriminate' the bit stream. This doesn't really matter to us, as our Manchester decoder will reject any corrupted data.

### Send data

Let's for example send the data byte 0x41. This is encoded into two Manchester encoded RS-232 bytes.

The least significant nibble is transmitted first, working out long hand:

1. 0x41 represented as an LSB first bit stream is: " 1000 0010 "
2. Lower nibble " 1000 " Manchester encodes to: " 10 01 01 01 " (i.e. 0xA9)
3. Upper nibble " 0010 " Manchester encodes to: " 01 01 10 01 " (i.e. 0x9A)

```
putc(0xA9); // lower nibble
putc(0x9A); // upper nibble
```

In real life this would be carried out by a routine similar to the one below:

```
void SendData(BYTE txbyte)
{
    int i,j,b,me;

    b = txbyte;

    for (i=0; i<2; i++) {
        me = 0; // manchester encoded txbyte
        for (j=0 ; j<4; j++) {
            me >>=2;
            if (bit_test(b,0) )
                me |= 0b01000000; // 1->0
            else
                me |= 0b10000000; // 0->1
            b >>=1;
        }
        putc(me);
    }
}
```

More data can be sent if required.

### End data frame

To end a data frame, a series of 0xF0's are transmitted using the function SendSS():

```
SendSS();
```

It could be argued that this is not necessary since the preamble in the next data frame will reset the communications handler. However the handler can use it as an indicator that the data frame is complete.

### Complete frame

The complete frame would be sent with the following 'C' functions:

```
SendSS();
SendData(0x41);
SendSS();
```

Note that the SS patterns and data must be sent end-to-end with no gaps other than the inherent start and stop bits. Failing to do this will almost certainly cause the discriminator in the receiver to produce erroneous data.

## Decoding data

Converting Manchester encoded data back to original data is straight forward and error checking may be carried out at the same time. Since data arrives 4-bits at a time in Manchester code, a simple shift and test algorithm can be implemented to retrieve the original data.

The routine below can be used to decode four bit-pairs to one 4-bits of original data:

```
BYTE DecodeData(BYTE encoded)
{
    BYTE i,dec,enc,pattern;

    enc = encoded;

    if (enc == 0xf0)    // start/end condition encountered
        return 0xf0;

    dec = 0;
    for (i=0; i<4; i++) {
        dec >>=1;
        pattern = enc & 0b11;
        if (pattern == 0b01)    // 1
            bit_set(dec,3);
        else if (pattern == 0b10)
            bit_clear(dec,3);    // 0
        else
            return 0xff;        // illegal code
        enc >>=2;
    }
    return dec;
}
```

Note that this routine only returns 4-bits of data and therefore has to be called twice by some additional code, first to retrieve the lower nibble and then again to retrieve the upper nibble. The lower and upper nibble are then assembled (using a 4-bit shift) to form one byte of original data.

## OTHER CONSIDERATIONS

When software generated RS-232 is used for receiving serial data, it is important that the latency between the start bit and the execution of `getc()` is kept to a minimum. This is not an issue if `getc()` is waiting for serial data, but becomes important when `kbhit()` is used to determine the presence of a start bit.

Typically `kbhit()` is used in a tight loop (see below). The latency is the time round the loop.

```
while(1) {
    if ( kbhit() )
        data = getc();
    ...
    some other code executed here
    ...
}
```

For example if the bit rate is 2000bps, one bit time is therefore 500us. A latency of no more than 50us is suggested. Higher bit rates require less latency.

Using a PC to send data may have drawbacks due to the non-deterministic nature of the operating system. i.e. there may be inter character delays in the transmitted data which would no longer satisfy our requirement that data is sent end-to-end. This technique is therefore most suited to a microcontroller where there is more control over sending of data.

## CONCLUSION

Manchester encoding is suitable for bit streams in radio communications and can be transported using a

standard UART. It is simple to encode, decode and has good error detection and can negate the need for 'check-summed' data. This technique is most suited for microcontrollers.

#### REFERENCES

1. Gorry Fairhurst, "Manchester Encoding"; [www.erg.abdn.ac.uk/users/gorry/course/phy-pages/man.html](http://www.erg.abdn.ac.uk/users/gorry/course/phy-pages/man.html)

#### BIBLIOGRAPHY

1. RF Solutions Ltd ([www.rfsolutions.co.uk](http://www.rfsolutions.co.uk)), "AM Super Regenerative Receivers", Data sheet DS016.
2. RF Solutions Ltd ([www.rfsolutions.co.uk](http://www.rfsolutions.co.uk)), "AM Hybrid Transmitter", Data sheet DS013.
3. Custom Computer Services Inc ([www.ccsinfo.com](http://www.ccsinfo.com)), "C Compiler Reference Manual", July 2003.
4. QuickBuilder ([www.quickbuilder.co.uk](http://www.quickbuilder.co.uk)), "AM RF Transponder"; [www.quickbuilder.co.uk/qb/libs/comms.htm](http://www.quickbuilder.co.uk/qb/libs/comms.htm).

#### Disclaimer:

Although every care has been taken to ensure the accuracy of the information in this article, such information is provided "as-is" and without any warranty or implied fitness for any purpose. In acting on any information provided by this article you accept that this is entirely at your own risk and indemnify the author against any loss or damages, actual or consequential that may occur.