Encoding

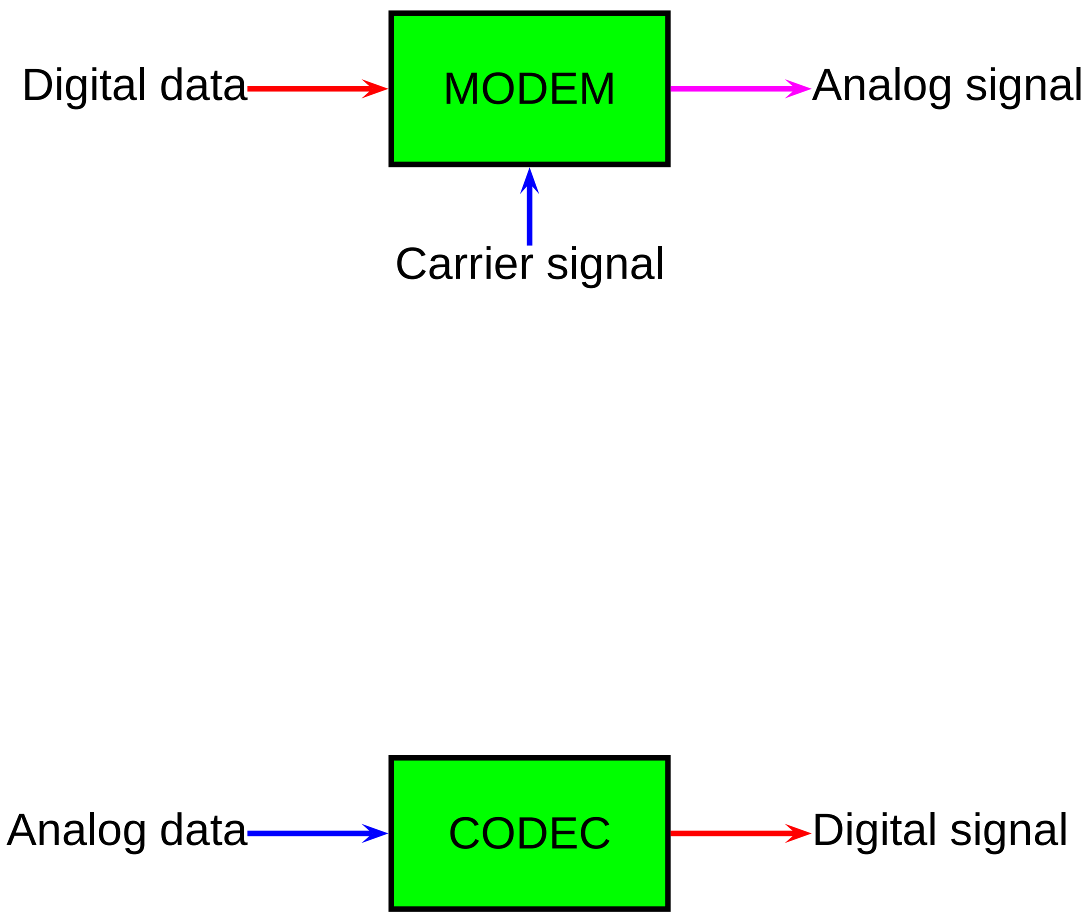
There are 4 possibilities:

1. Digital data, digital signals: the simplest form is to assign different voltage levels to 0 and 1, possibly with a third level (usually 0) assigned to silence.

2. Digital data, analog signals: a modem converts digital data to analog, altering the carrier frequency to represent 0 and 1.

3. Analog data, digital signals: periodic sampling the analog data gives a numeric value that can be transmitted.

4. Analog data, analog signals: a modem adds the signal to the carrier frequency (“modulated”) and transmits it.


Digital data
MODEM
Analog signal
Carrier signal
Analog data
CODEC
Digital signal

Basics

- The bit error ratio is proportional to the transmission rate.
- The bit error ratio is inversely proportional to the SNR.
- The bit error ratio increases with the transmission fundamental frequency.

Basic encoding formats

Return–to–Zero (bipolar or AMI).

Non–Return–to–Zero (NRZ–L).

Non–Return–to–Zero–Inverted (NRZI).

Manchester encoding and **Differential Manchester**.

mBnL *m* bits encoded using *n* voltage levels.

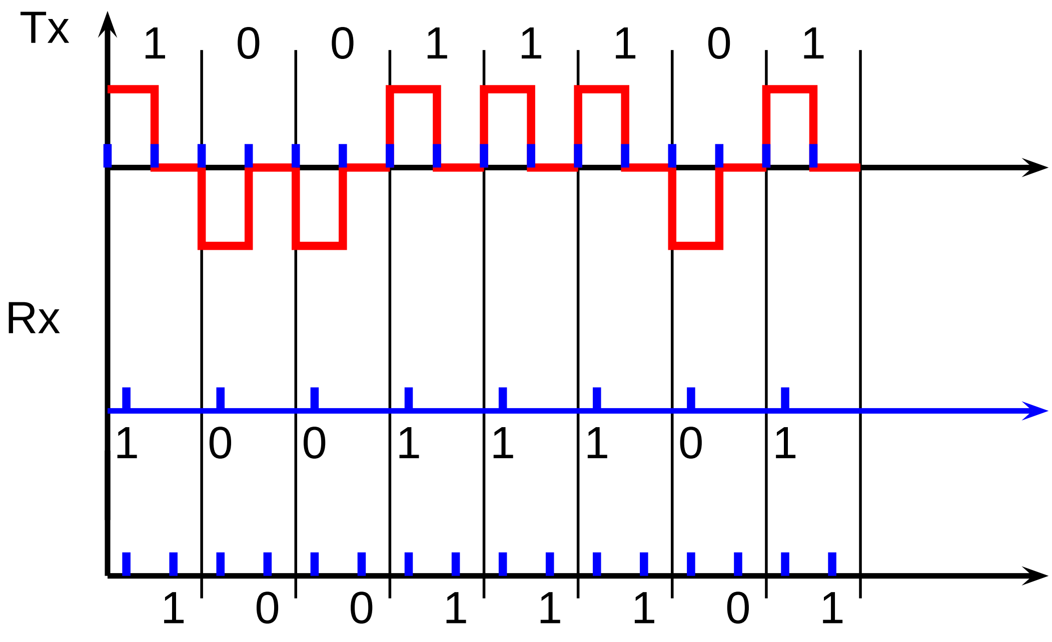
mBnB *m* bits encoded into groups of *n* bits. These bits are further encoded using one of the other encodings.

Main issues

- Clock synchronisation.
- Clock drift.
- Bit rate versus frequency (pulse rate).
- DC component.

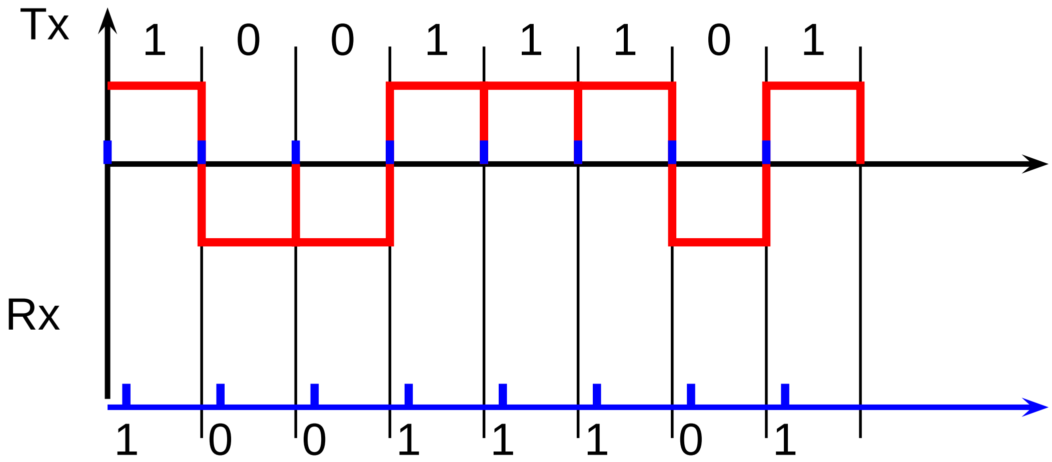
Return–to–Zero

This is a family of encodings which all require that the signal level returns to 0 (volts) after each bit. The most popular is the **RZ** encoding in which, for each bit, the first pulse gives the bit value (+V for 1 and -V for 0) and the second is 0V.

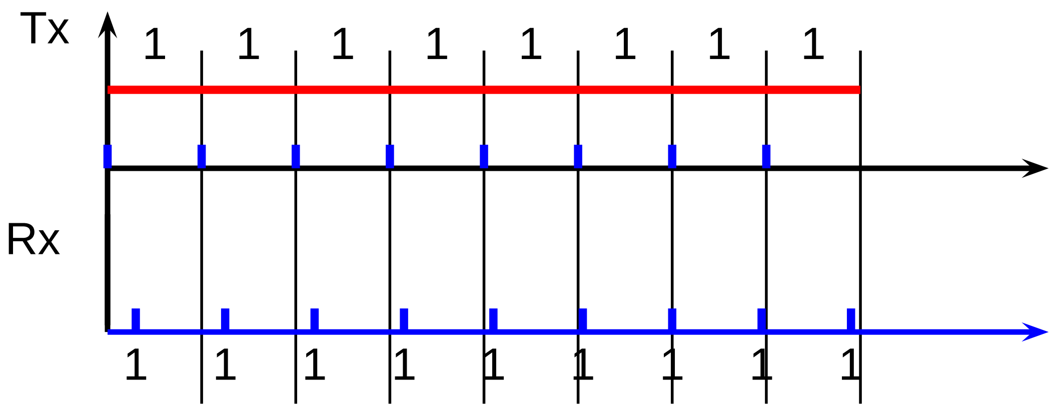


Non–Return–to–Zero

In its 2 signal level variation:



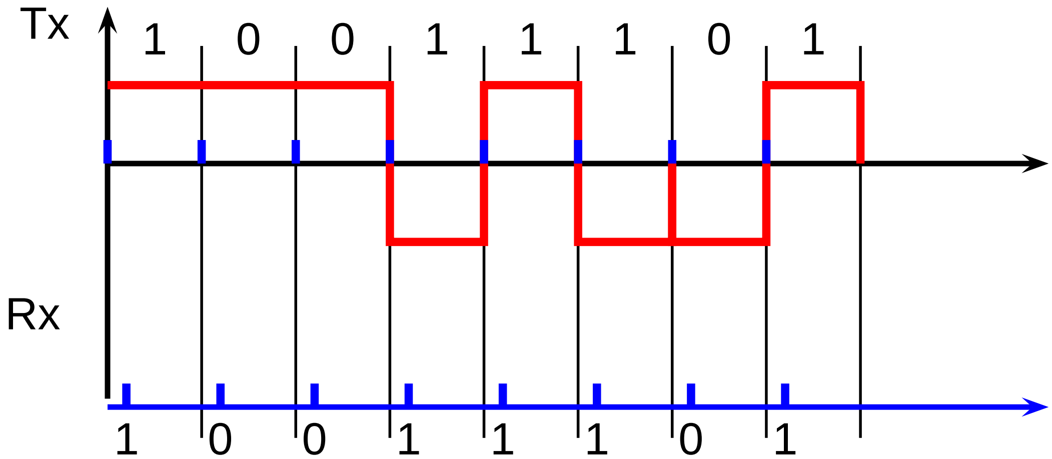
The main weakness of NRZ-L is vulnerability to clock drift:



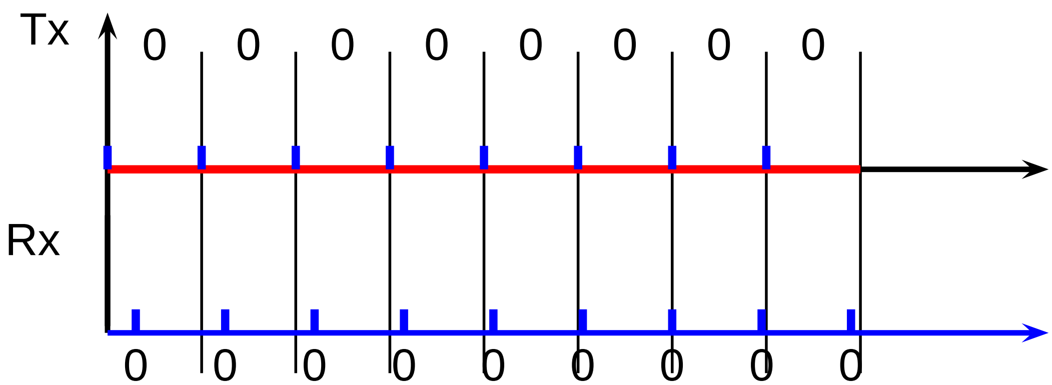
Another problem is the potential presence of a **dc** component.

Non–Return–to–Zero–Inverted

Similar to NRZ, but signal transitions occur on the beginning of a “1” bit only.



The main weakness of NRZ-I is vulnerability to clock drift during a long sequence of “0” bits:

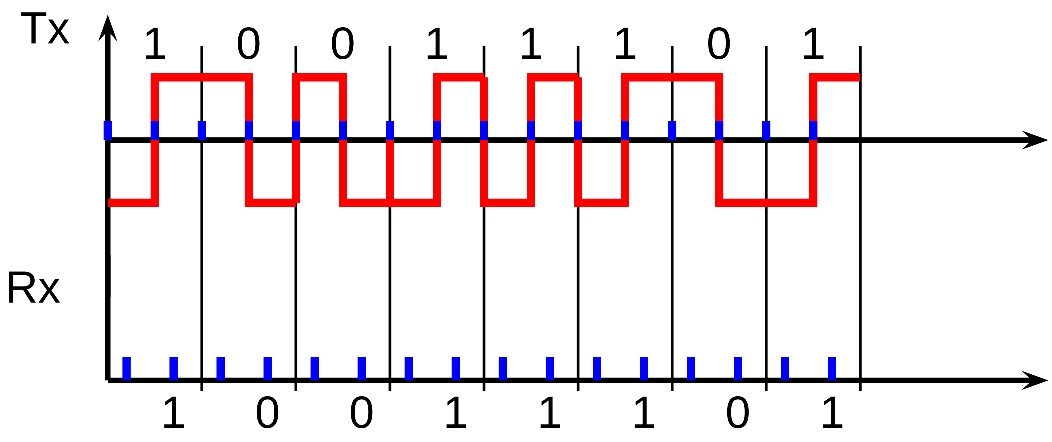


Another problem is the potential presence of a **dc** component when a long sequence of “0”” bits occurs.

Manchester encoding

There is a transition in the middle of every bit (similar to RZ). The transition serves a mechanism for clock synchronisation; it also gives the value of the bit:

- A low to high level transition means a “1” bit.
- A high to low level transition means a “0” bit.



Bandwidth vs. bit rate

NRZ and NRZI require one pulse per bit while bipolar or Manchester encodings require 2 per bit. Hence, NRZ(I) can carry the same number of bits with half the bandwidth (which implies half the frequency).

As a result, Manchester encoding is used mainly in LANs, while NRZI and similar schemes are used mainly in WANS. The problems of clock drift and dc component make NRZI unatractive over long distances, so different schemes are used over long–distance lines.

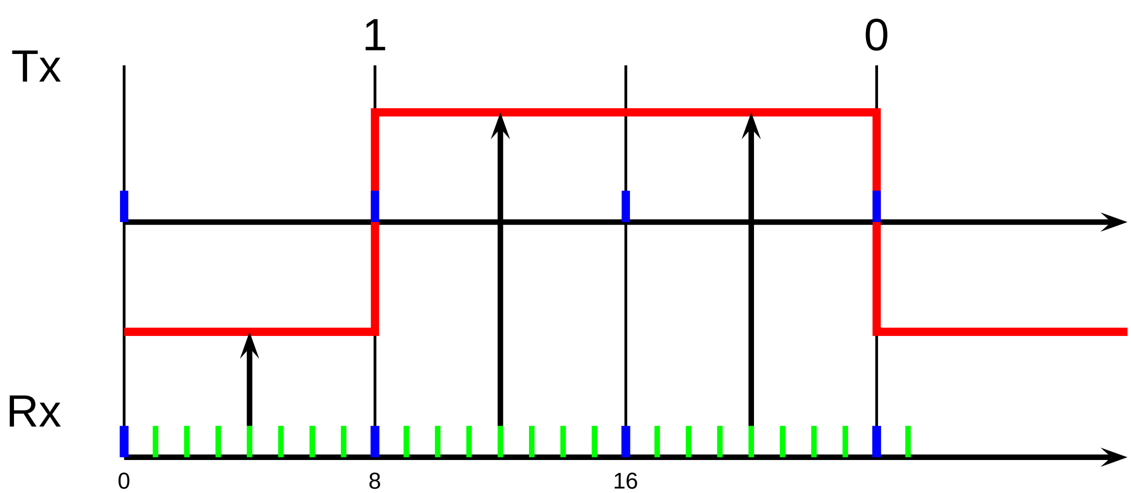
Decoding

The receiver uses its clock to sense the flow of bits. the transmission rate is known to the receiver, so its clock ticks an integer number of times per pulse (or bit). The most common ticking rate is 32 ticks per bit. The problems:

- There is an unknown propagation delay between the sender and the receiver, so it is impossible to synchronise their clocks permanently. Hence, the receiver must figure out when a bit starts.
- No two clocks have an identical ticking rate, at least because of crystal impurities. Hence, the receiver must continuously adjust the position of the tick that indicates the start of a bit.

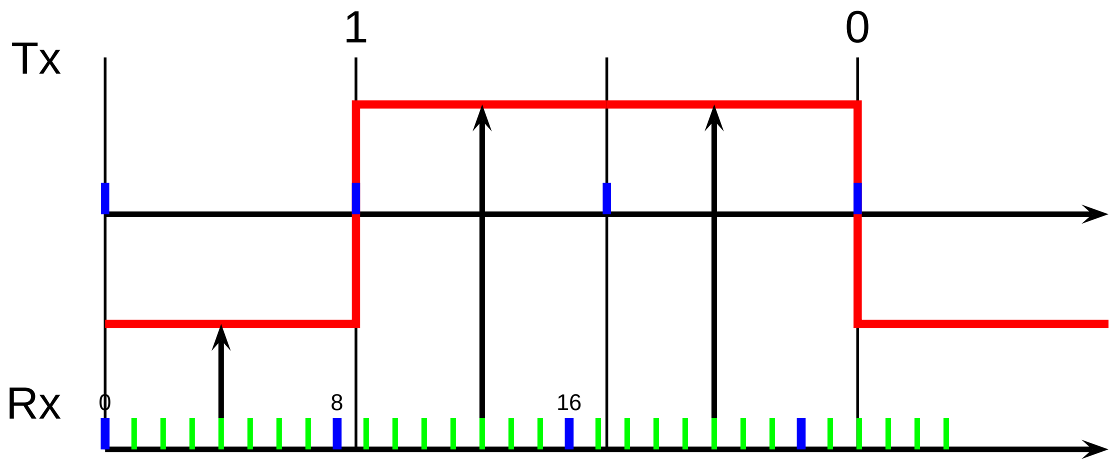
Clock synchronisation

An ideal scenario with Manchester encoding and 16 ticks per bit:

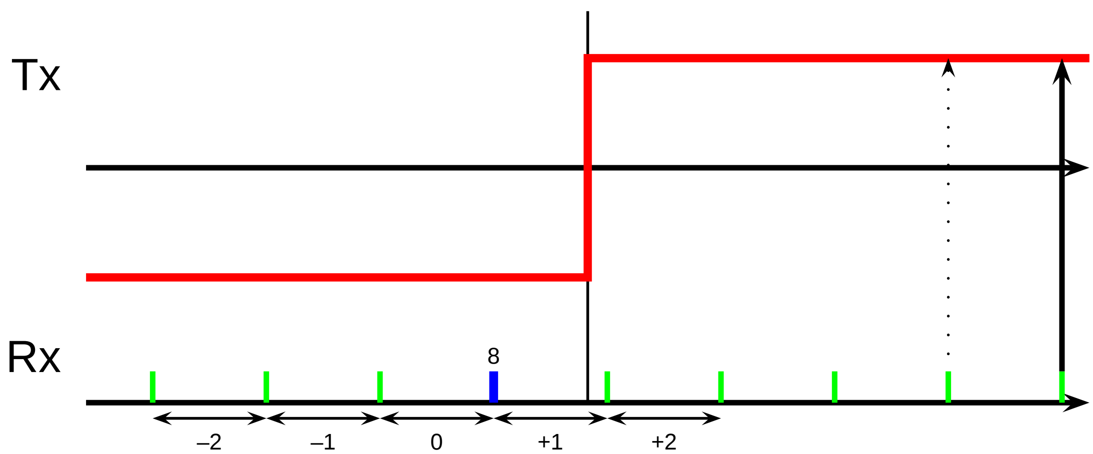


Clock synchronisation

A not so ideal scenario with Manchester encoding and 16 ticks per bit:



The transition part magnified:



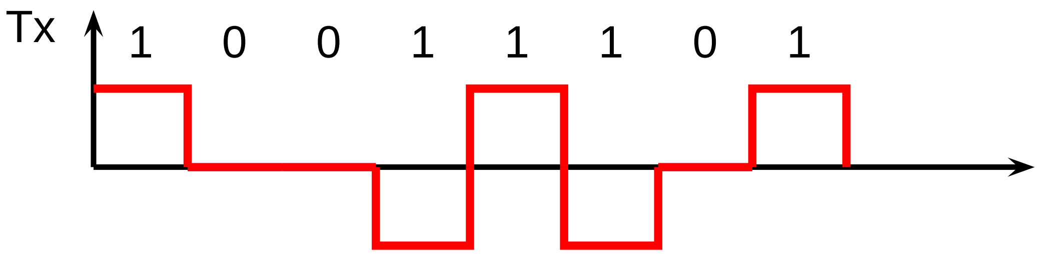
Bipolar encoding

Bipolar codes use 3 power levels (typically one of them being 0).

The simplest bipolar encoding is the **RZ** encoding.

A common variation is the **AMI** encoding.

Alternate mark inversion is a code in which 1s ("mark") bits are inverted in an alternating way, i.e. if the previous 1 was a positive voltage, the next 1 will be a negative voltage. 0s are sent as silence.



AMI was developed as an alternative to **NRZ**: it has the same signal rate (1 bit per baud) but it has no DC component.

The **clock drift** problem remains.

Multilevel codes

The simplest multilevel code, **mBnL**, sends **m** bits as an **n**–element signal of *L* levels.

Example: *2B1Q* (or **2B1L=4**) has 2 bits sent as one 4–level signal element. It is used in **DSL** (Digital Subscriber Line) telephone connections.

2B1L also uses inversion.

The table show how a pair of bits is encoded depending on whether the

The table shows how a pair of bits is encoded depending on whether the previous signal was positive or negative.

2B	Previous level	
	+	−
00	+1	-1
01	+3	-3
10	-1	+1
11	-3	+3

8B6T

This code sends 8 bits as 6 ternary (three–level) signal elements. Considering that $2^8 = 256$ and $3^6 = 729$, the **DC** component problem is solved because:

- Only elements that have a DC component of 0 or +1 are legal.
- Whenever two elements with a non–zero DC component follow each other, the second is inverted.

Example: $11111111 \longrightarrow +0-+00$ (this is the traditional notation, with $+$ standing for +1V and $-$ standing for -1V). This ternary signal element has a DC component of +1V. So, it a very long sequence of 1s is to be sent, it will be encoded as:

$$(+0-+00)(-0+-00)(+0-+00)(-0+-00)...$$

mBnB **codes**

There are two interpretations of the name of **mBnB** codes:

Multilevel codes with two signal levels (Binary). Hence **4B5B** is an **mBnL** code with **L = 2**.

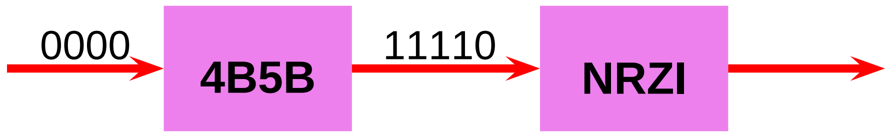
Block codes which turn **m** bit datawords into **n** bit codewords.

There is also **64B66B** which is none of the above (it has a preamble of 2 bits followed by 64 bits).

4B5B

This is the original **mBnB** code. It was used in conjunction with NRZI to get rid of the clock drift problem.

In NRZI a long sequence of 0s is sent as a signal of unchanging amplitude. If the input sequence is first converted to a **4B5B** code, this problem disappears:



Note that NRZI has no problems with a sequence of 1s.

8B10B

This code is used in many protocols (SATA, Firewire, USB 3.0, etc.). It is a combination of **5B6B** and **3B4B**: a sequence of 8 bits is divided into two groups of 5 and 3 bits each. The first group is encoded using **5B6B**, the second using **3B4B**.

To address the usual problems, the encoding guarantees no more than 5 consecutive 0s or 1s and guarantees that in each codeword the difference between the number of 0s and 1s will be no more than 2 (at most 6 zeroes or ones in a codeword).

Gadgets

The idea behind **B8ZS** (substitution of 8 zeroes) is used in **AMI** (and other encodings).

It replaces 8 consecutive zeroes with a sequence:

$$000VV^{-1}0VV^{-1}$$

where V is a violation of the inversion rule (i.e. negative when positive was required or the other way around) while V^{-1} is the inverse of V.

A similar idea is behind *HDB3* which replaces 4 zeroes with one of two sequences: $000V$ or $V^{-1}00V$ depending on the DC bias.