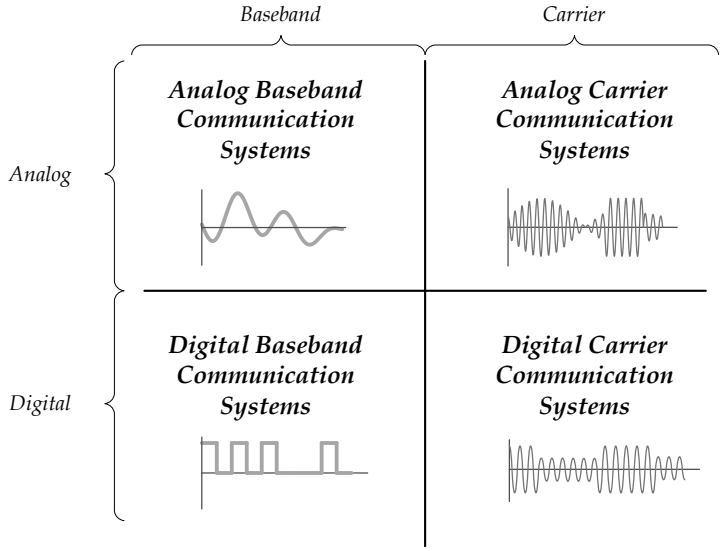
EE 421: Communications I
Dr. Mohammed Hawa

Introduction to Digital Baseband Communication Systems

For more information: read Chapters 1, 6 and 7 in your textbook or visit http://wikipedia.org/.

Remember that communication systems in general can be classified into four categories: Analog *Baseband* Systems, Analog *Carrier* Systems (*using* analog modulation), Digital *Baseband* Systems and Digital *Carrier* Systems (*using* digital modulation).



Digital baseband and *digital* carrier transmission systems have many advantages over their analog counterparts. Some of these advantages are:

1. Digital transmission systems are more *immune to noise* due to threshold detection at the receiver; and the availability of regenerative repeaters, which can be used instead of analog amplifiers at intermediate points throughout the transmission channel.
2. Digital transmission systems allow *multiplexing* at both the baseband level (e.g., TDM) and carrier level (e.g., ~~FDM~~, CDMA and OFDMA), which means we can easily carry multiple conversations (signals) on a single physical medium (channel).
3. The ability to use *spread spectrum techniques* in digital systems help overcome jamming and interference and allows us to hide the transmitted signal within noise if necessary. In addition, the use of *orthogonality* is easier and allows increasing the transmission rate by overcoming impairments such as fading.
4. The possibility of using *channel coding techniques* (i.e., error correcting codes) in digital communications reduces bit errors at the receiver (i.e., it effectively improves the signal-to-noise ratio (SNR)).
5. The possibility of using *source coding techniques* (i.e., compression) in digital communications reduces the amount of bits being transmitted, and hence, allows for more bandwidth efficiency. *Encrypting* the bits can also lead to privacy.

6. Digital communication is inherently more efficient than analog systems in *exchanging* SNR for bandwidth, and allows such exchange at both the baseband and carrier levels.
7. Digital hardware implementation is *flexible* and permits the use of microprocessors. Using microprocessors to perform *digital signal processing* (DSP) eliminates the need to build expensive and bulky discrete-component devices. In addition, the price of microprocessors continues to drop every day.
8. Digital signal storage is relatively easy and inexpensive. Also the reproduction of digital messages can be extremely reliable without deterioration, unlike analog signals.

Actually, due to those important advantages most of communication systems nowadays are digital, with analog communication playing a minor role (we still, *for example*, listen to analog AM and FM radio).

This article provides a very quick overview of some of the main concepts that are relevant to digital *baseband* transmission. You will study more about this topic in the EE422 "Communications II" course. The main concepts to be emphasized here will be the analog-to-digital conversion process and the line coding concept.

Digitization (and Analog-to-Digital (A/D) Conversion):
Signals that result from physical phenomena (such as voice or video) are almost always analog baseband signals. Converting such analog *baseband* signals into digital *baseband* signals is not as simple as one might expect, especially when considering modern communication systems such as Digital TV, SDH, Ethernet, ADSL, etc. To summarize, we say digitization involves the following steps:
1. **Sampling** (in which the signal becomes a *sampled* analog signal, also called a *discrete* analog signal).
2. **Quantization** (the signal becomes a *quantized discrete* signal, but not a digital baseband signal yet).
3. **Mapping** (the signal becomes a stream of **1**'s and **0**'s). Mapping is sometimes *confusingly* called encoding.
4. **Encoding** and **Pulse Shaping** (after which the signal becomes a digital baseband signal).

These four steps are shown in Figure 2 below and are explained in the following subsections.

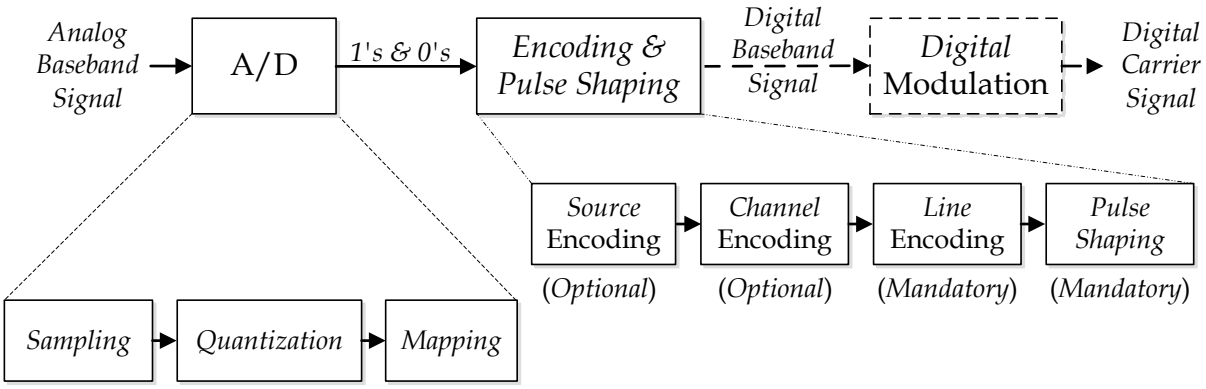


Figure 2. The Many Steps of Digitization.

I. Sampling:

Sampling is the first step in converting an analog baseband signal into a digital baseband signal. Sampling is defined as the process in which only a relatively-small set of values, called discrete samples $\{m_n\}$, are used to represent the signal $m(t)$ instead of the (time-continuous) infinite set of values included in the original analog signal (see Figure 3 below, which shows the process of *ideal sampling*).

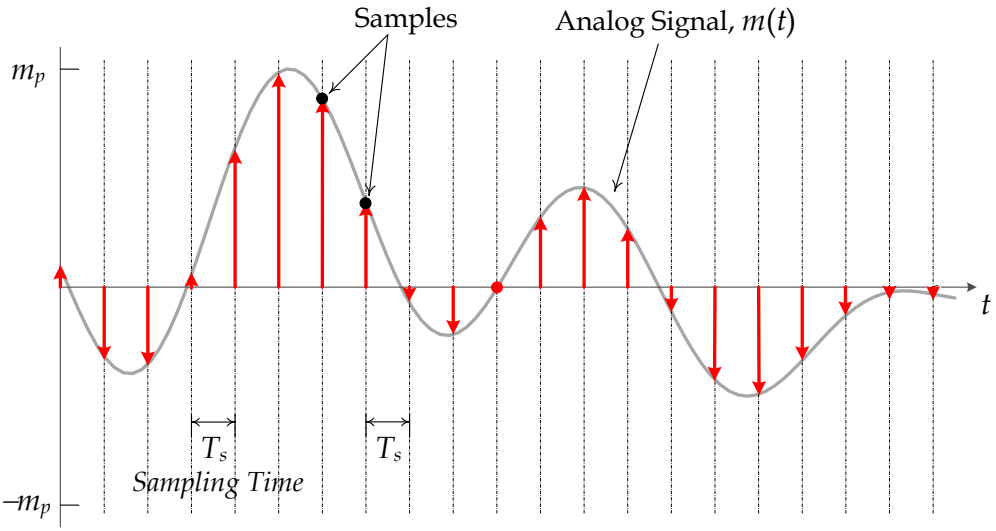


Figure 3. Ideal Sampling.

In *uniform* sampling, the time interval between successive samples is set to a constant value equal to T_s, called the *sampling time*. In this case, the *sampling frequency* is $f_s = 1/T_s$.

Nyquist–Shannon sampling *theorem* states that for the samples $\{m_n\}$ to *truly* represent the original signal $m(t)$, we need the sampling frequency f_s to be at least *twice* as high as the bandwidth B of the band-limited analog signal $m(t)$ (i.e., $f_s \geq 2B$). Such a condition will prevent *aliasing*. Aliasing should be avoided[1] at all costs since it means that the signal $m(t)$ cannot be recovered from the discrete samples $\{m_n\}$ by simple low-pass filtering (LPF) at the receiver.

As a specific example, telephone conversations are sampled at 8 kHz (twice the 4 kHz bandwidth of the human voice signal[2]), while compact disc (CD) audio is sampled at 44.1 kHz (more than twice the 20 kHz bandwidth of music signals[3]).

It is interesting to notice that practical A/D integrated circuits (such as ADC0801, ADC0820, ADC0832, TDA9910, etc) cannot generate impulses as shown in Figure 3 above since impulses require infinite energy (impulses are theoretical signals by definition). Hence, such A/D ICs generate instead *practically-sampled* signals in which rectangular pulses (of width T_s) are used to represent the samples instead of

[1] If the Nyquist criterion cannot be satisfied, an anti-aliasing filter with a maximum bandwidth of f_s / 2 should be used to prevent aliasing. This will result in distortion in the sampled signal, which is undesired, but is better than aliasing distortion.

[2] Subjective tests show that signal articulation (intelligibility) is not effected if all components above 3400 Hz are suppressed. Hence, strictly speaking, voice bandwidth is 3.4 kHz not 4 kHz.

[3] Some music instruments generate signals with bandwidths exceeding 20 kHz but a human ear cannot hear sounds above the 20 kHz mark.

impulses, as shown in Figure 4 below[4]. We will explain in class the difference between *practically-sampled* signals, *naturally-sampled* signals, and *ideally-sampled* signals.

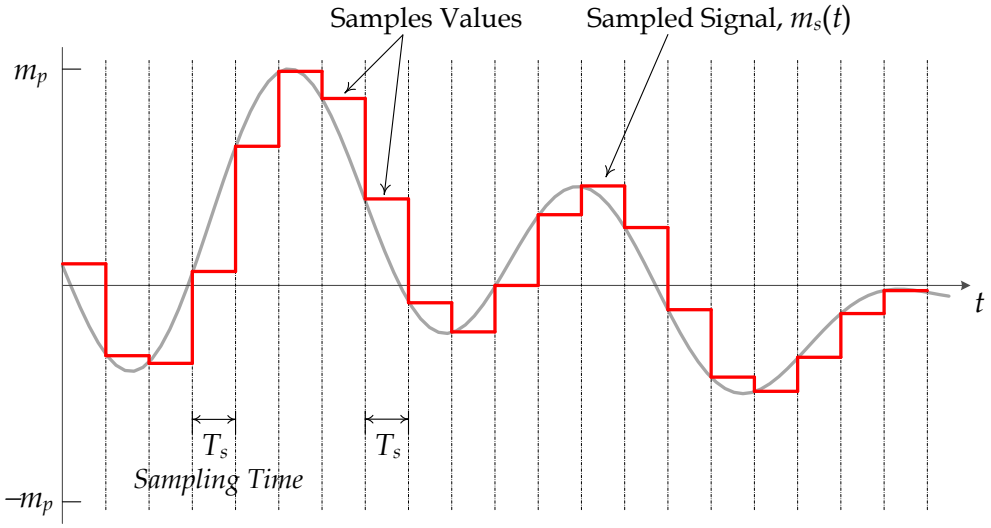


Figure 4. Practical Sampling.

II. Quantization:

The next step in analog-to-digital conversion is Quantization, which limits the digital data to be sent. Quantization is the process in which each sample value is *approximated* or "*limited to*" a relatively-small set of discrete *quantization levels*. For example, in uniform quantization if the amplitude of the signal $m(t)$ lies in the range $(-m_p, m_p)$, we can partition this continuous range into L discrete intervals, each of length $\Delta v = 2\ m_p\ / L$, and the value of each sample is then approximated to only one of these L levels.

Notice that quantization can be done in several different ways. In one method the value of each sample can be "truncated" to the quantization level just below it. This is shown in Figure 5 below for $L = 5$ levels.

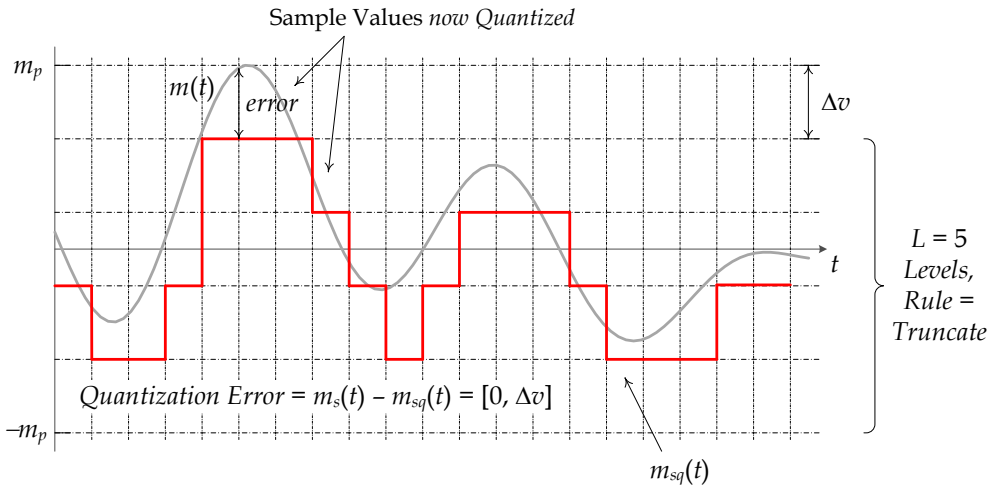


Figure 5.

[4] Typical IC-based ADC chips perform sampling, quantization and mapping all on the same chip.

Notice that the *quantization error* (i.e., the difference between the original sample value and the quantized sample value) is limited to the range [0, Δv]. This quantization error is a *deliberate* error introduced by the transmitter to control the transmitted bit rate. Notice, however, that this error can be controlled by reducing the value of Δv, which can be achieved by increasing the number of quantization levels L as shown in Figure 6 below where L = 10 levels now.

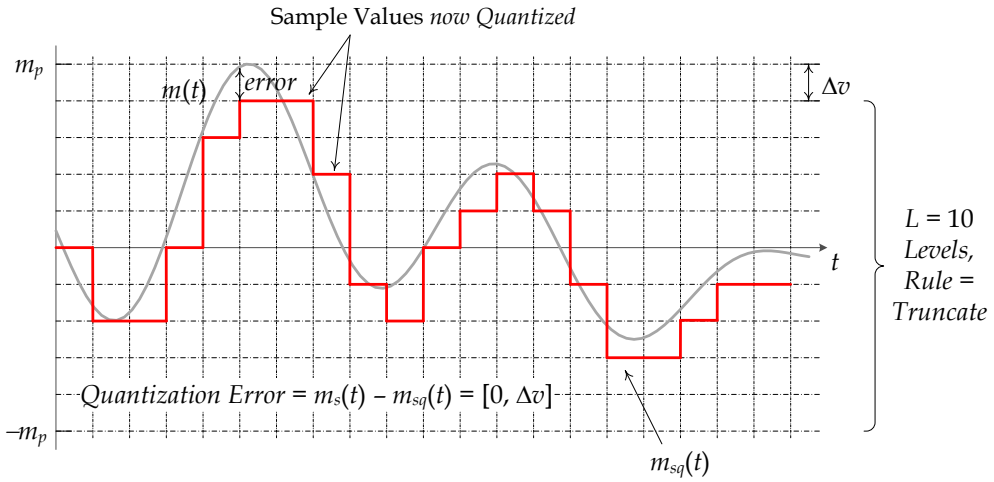


Figure 6.

Another valid method of quantization is where samples are "rounded off" to the nearest quantization level either below it or above it. This is shown in Figure 7 below. Notice that in this method the quantization error is now limited to [$-\Delta v/2$, $\Delta v/2$].

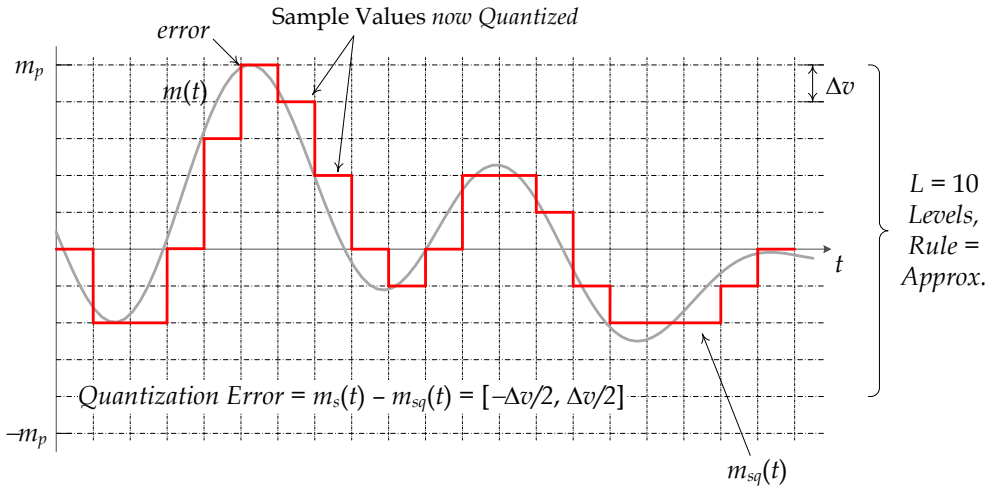


Figure 7.

The number of quantization levels L is an important parameter in digital systems because it decides (see next section) how many bits will be used to represent the value of each sample. For example, if L = 256, the value of each sample can be in one of 256 possibilities, which means that each sample must be mapped (encoded) into 8 bits. This is because 8 binary bits can be in 2^8 = 256 possible states (00000000, 00000001, 00000010, 00000011, . . ., 11111111). For L = 65,536, we need 16 bits to encode each sample value.

In voice telephony, for example, the number of quantization levels is chosen to be $L = 2^8 = 256$ since *intelligibility* (rather than high fidelity) is required, while for compact disc (CD) audio, the number of quantization levels is $L = 2^{16} = 65{,}536$ possible values per sample. Of course, the bigger value of L means a smaller quantization error range Δv, and thus better quality.

We will discuss in class how quantization can be visualized as a noise signal. We will also explain how adaptive quantization can improve the signal-to-quantization-noise ratio (SQNR).

III. Mapping:
After the analog signal is discretized in time (i.e., sampled) and value (i.e., quantized), it is converted into a binary *bit stream* using a process called mapping, in which each of the quantized sample values (e.g., 6V, 10V, *etc*) is mapped to a corresponding binary code (e.g., 0110, 1010, *etc*). The result is a stream of **1**'s and **0**'s.

Notice that each sample is represented by $n = \log_2(L)$ bits, and hence the bit time T_0 is given by $T_0 = T_s / \log_2(L)$, where T_s is the sample period. For example, if $L = 256$, we have $T_0 = T_s/8$ (see Figure 8 below).

The total number of bits generated in one second is called the **data bit rate** $f_0 = 1 / T_0$ measured in bits/s (*or* bps) and is given by:

$$f_0 \text{ [bps]} = f_s \text{ [samples/s]} \times \log_2(L) \text{ [bits/sample]}$$

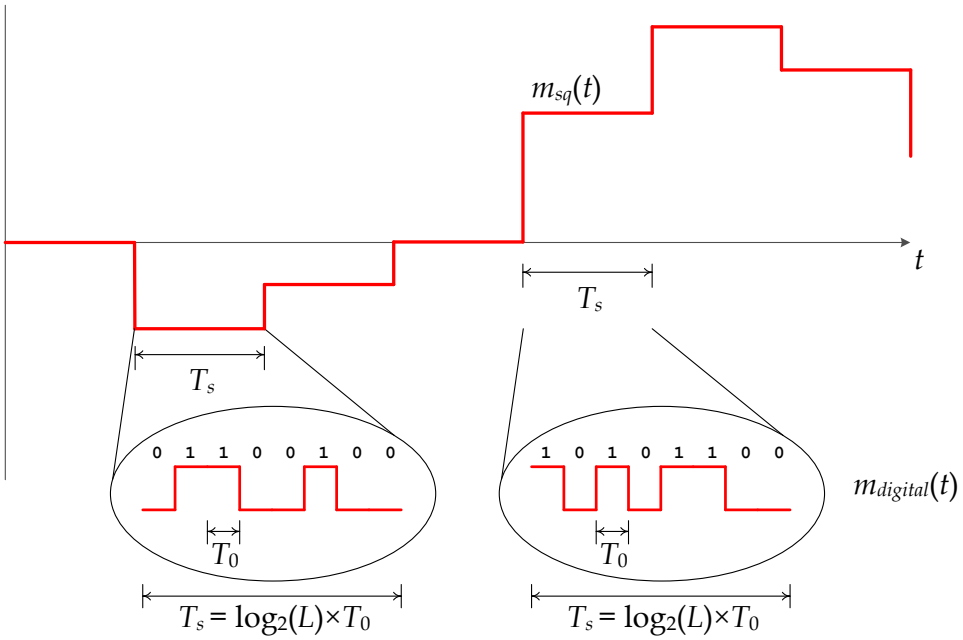


Figure 8. The Sampled Signal being Mapped into 1's and 0's.

IV. Encoding:
Now that we have a bit stream of **1**'s and **0**'s, those bits are first manipulated into a new (and *better*) sequence of 1's and 0's, and then converted into voltages appropriate for transmission on a physical channel. This manipulation and conversion is collectively called encoding. Encoding schemes are usually divided into: source encoding, channel encoding and line encoding (see Figure 2).

A. Source coding: Source coding refers to the process of **compressing data**. This is typically done by replacing long binary *codewords* that occur frequently by shorter ones, and those that occur less frequently by longer codes. For example, a 4-bit sequence "0110" occurring frequently can be mapped into the shorter 2-bit sequence "01", while another 4-bit sequence "1011" occurring less frequently can be mapped to the longer 6-bit sequence "111001". This makes sure that sequences that occur more often in the bit stream are the shorter ones.

In information theory, **Shannon's noiseless coding theorem** places an upper and lower bounds on the expected compression ratio. Examples of source codes currently in use are: Shannon codes, Huffman codes, run-length encoding (RLE), arithmetic coding, Lempel-Ziv coding, MPEG-2 and MPEG-4 video coding[5], Linear Prediction Coding (LPC) and Code-Excited Linear Prediction (CELP) coding, etc.

B. Channel coding: Channel coding refers to **error correcting codes**. The most obvious example of such codes is the simple *parity* bit system. Such codes are used to protect data sent over the channel from corruption even in the presence of noise. In other words, channel codes can improve the signal-to-noise ratio (SNR) of the received signal.

The theory behind designing and analyzing channel codes is called **Shannon's noisy channel coding theorem**. It puts an upper limit on the amount of information you can send in a noisy channel using a *perfect* channel code. This is given by the following equation:

$$C = B_{ch} \times \log_2(1 + SNR)$$

where C is the upper bound on the *capacity* of the channel (bit/s), B_{ch} is the bandwidth of the channel (Hz) and SNR is the Signal-to-Noise ratio on the channel (unitless). Examples of channel codes currently in-use include: Hamming codes, Reed-Solomon codes, convolutional codes (usually decoded by an iterative Viterbi decoder), Turbo codes, etc.

C. **Line coding:** Line coding refers to the process of representing the final bit stream (**1**'s and **0**'s) in the form of voltage or current variations optimally tuned for the specific properties of the physical channel being used.

The selection of a proper line code can help in so many ways: (*a*) One possibility is to aid in *clock recovery* at the receiver. A clock signal is recovered by observing transitions in the received bit sequence, and if enough transitions exist, a good recovery of the clock is guaranteed, and the line code is said to be **self-clocking**. Self-

[5] MPEG is short for Moving Picture Experts Group.

clocking is important in digital systems as all digital receivers require the existence of the clock to function properly (this is similar to the synchronous detection process in DSB-SC demodulators).

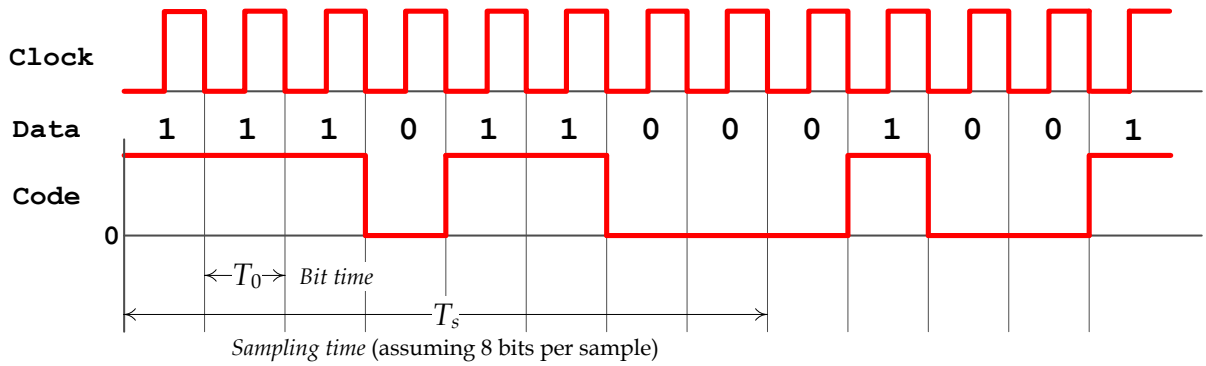
(*b*) Another advantage is to the ability to control the DC component in the resulting line code. This is important because most long-distance communication channels cannot transport a DC component[6], and hence, most line codes try to eliminate the DC component before being transmitted on the channel. Such codes are called *zero-DC* or *DC equalized*.

Other advantages of proper line coding include the (*c*) possibility of transmitting at a higher data bit rate while requiring smaller *bandwidth* for the resulting baseband signal, (*d*) the possibility of increasing the average power of the baseband signal compared to noise (i.e., improving SNR), and (*e*) reducing the amount of power at low-frequency components of the spectrum, which is important in telephone line applications, where the channel exhibits heavy attenuation below 300 Hz.

Some types of line encoding in common-use nowadays are unipolar, polar, bipolar, Manchester, MLT-3 and Duobinary encoding. These codes are explained here:

1. Unipolar (Unipolar **NRZ** and Unipolar **RZ**):
Unipolar is the simplest line coding scheme possible, but the least used in practice due to its many disadvantages. Unipolar coding uses a positive rectangular pulse $p(t)$ to represent binary **1**, and the absence of a pulse (i.e., zero voltage) to represent a binary **0**.

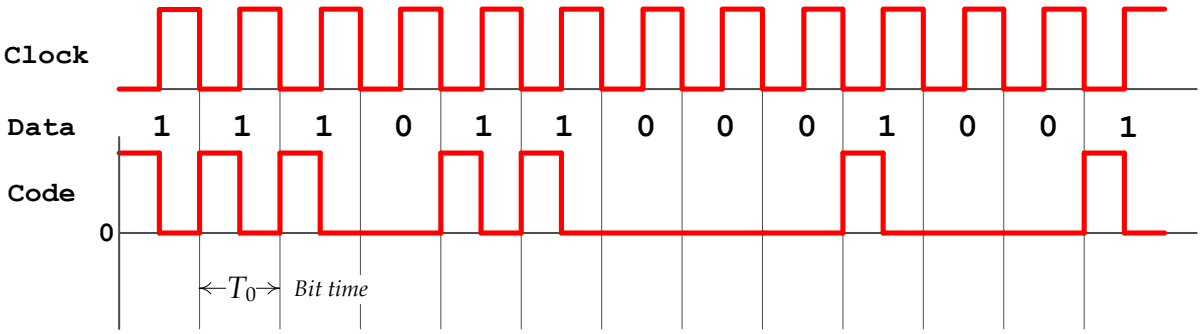
Two possibilities for the pulse $p(t)$ exist[7]: Non-Return-to-Zero (NRZ) rectangular pulse and Return-to-Zero (RZ) rectangular pulse. The difference between Unipolar NRZ and Unipolar RZ codes is that the rectangular pulse in NRZ stays at a positive value (e.g., +5V) for the full duration of the logic **1** bit, while the pule in RZ drops from +5V to 0V in the middle of the bit time. The figure below shows the difference between Unipolar NRZ and Unipolar RZ for the example bit stream 1110110001001.



Unipolar NRZ Code

[6] DC-values create excessive heat generation in the channel, they cause baseline drift and also do not fit systems that carry an additional small direct current to power intermediate amplifiers (an example is telephone networks).

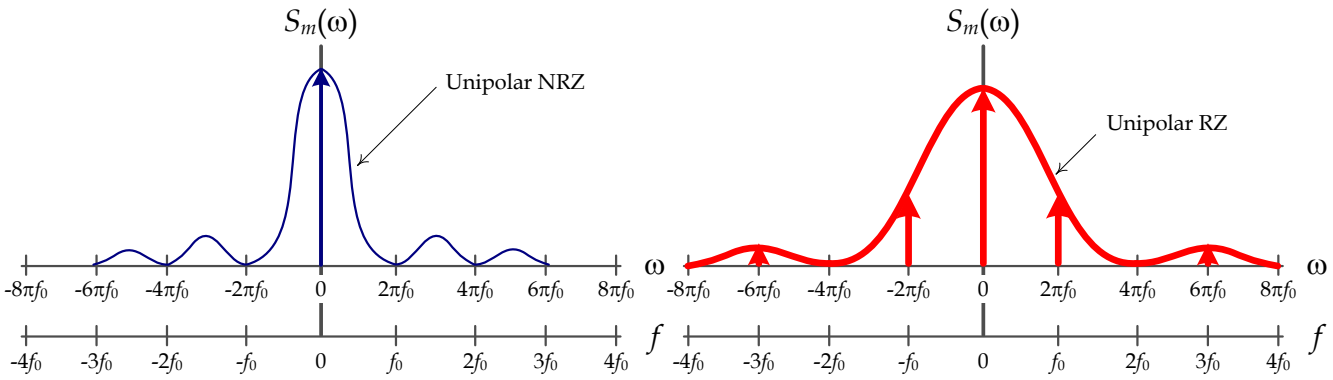
[7] Actually there are so many possibilities for the pulse shape $p(t)$; not just a rectangular NRZ or rectangular RZ pulses. Changing $p(t)$ waveform is called **Pulse Shaping** and affects the characteristics of the line code as will be explained later.



Unipolar RZ Code

An advantage of unipolar NRZ is that it is compatible with TTL logic. However, a drawback of unipolar (RZ and NRZ) is that its average value (i.e., DC value) is not zero (see the impulse at zero frequency in the corresponding power spectral density (PSD) of this line code shown in the diagram below). As we explained earlier, a DC-value is not desired in long-distance communication systems. Another disadvantage of such unipolar (RZ and NRZ) signaling is that it does not include clock information especially when the bit stream consists of a long sequence of **0**'s.

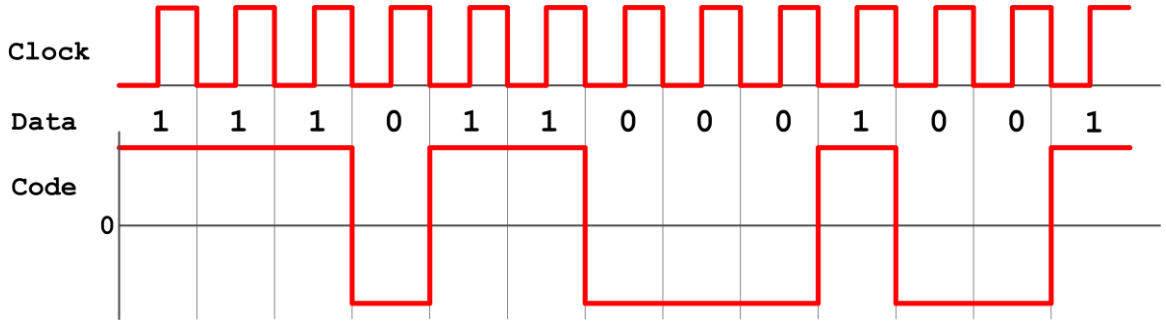
The disadvantage of unipolar RZ compared to unipolar NRZ is that each rectangular pulse in RZ is only half the length of the NRZ pulse. This means that unipolar RZ requires twice the bandwidth of the NRZ code. This can be seen from the PSD of both signals shown below[8].



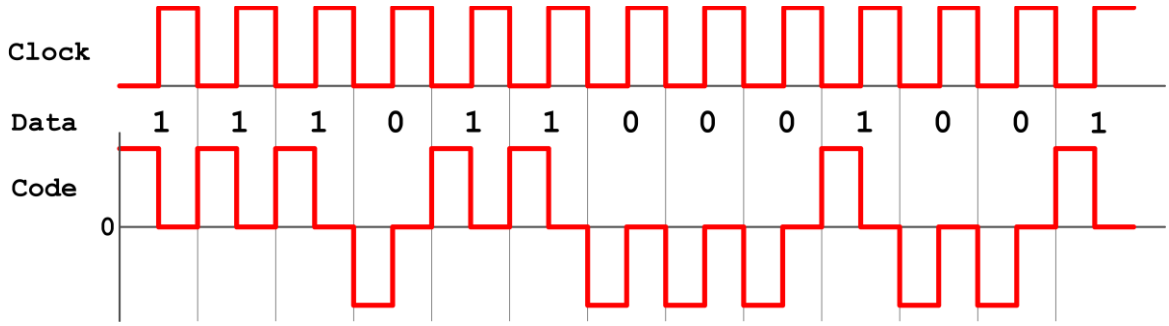
2. Polar (Polar **NRZ** and Polar **RZ**):
In Polar NRZ line coding a binary **1** is represented by a pulse $p(t)$ (e.g., +5V) and a binary **0** is represented by the negative of this pulse $-p(t)$ (e.g., –5V). Polar (NRZ and RZ) signals are shown in the diagram below.

Using the assumption that in a regular bit stream a logic **0** is just as likely as a logic **1**, polar line codes (whether RZ or NRZ) have the advantage that the resulting DC-component is very close to zero.

[8] The above spectra were calculated based on the assumption that logic **1**'s and logic **0**'s are equally likely in the transmitted bit sequence. This is a simplifying assumption that we use throughout this article.



Polar NRZ Code

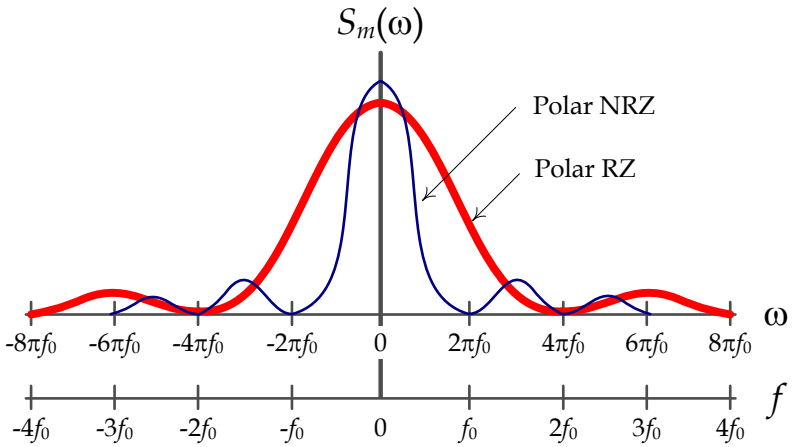


Polar RZ Code

In addition, the rms value of polar signals is bigger than unipolar signals, which means that polar signals have more power than unipolar signals[9], and hence have better SNR at the receiver. Actually, polar NRZ signals have more power compared to polar RZ signals.

The drawback of polar NRZ, however, is that it lacks clock information especially when a long sequence of **0**'s or **1**'s is transmitted. This problem does not exist in polar RZ signals, since the signal drops to zero in the middle of each pulse period.

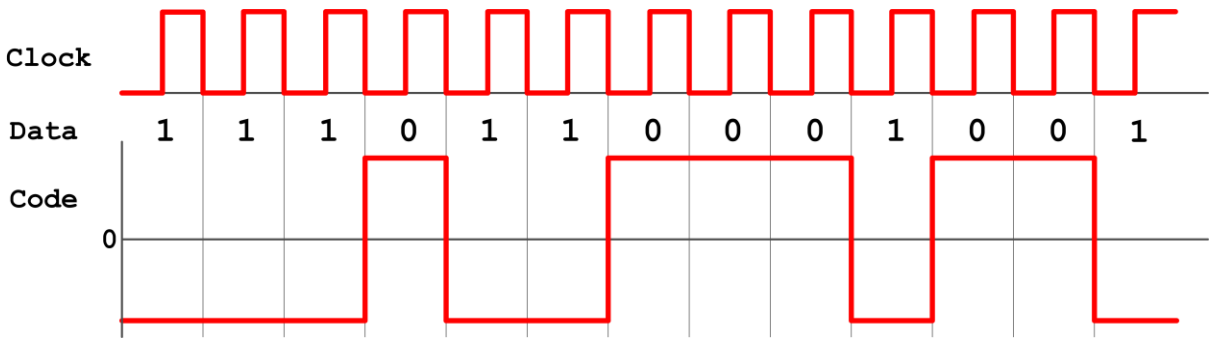
The power spectral densities (PSD) of both polar NRZ and polar RZ are shown below.



Signals transmitted on a computer motherboard often use Polar NRZ code. Another useful application of this encoding is in Fiber-based Gigabit Ethernet (1000BASE-SX and 1000BASE-LX).

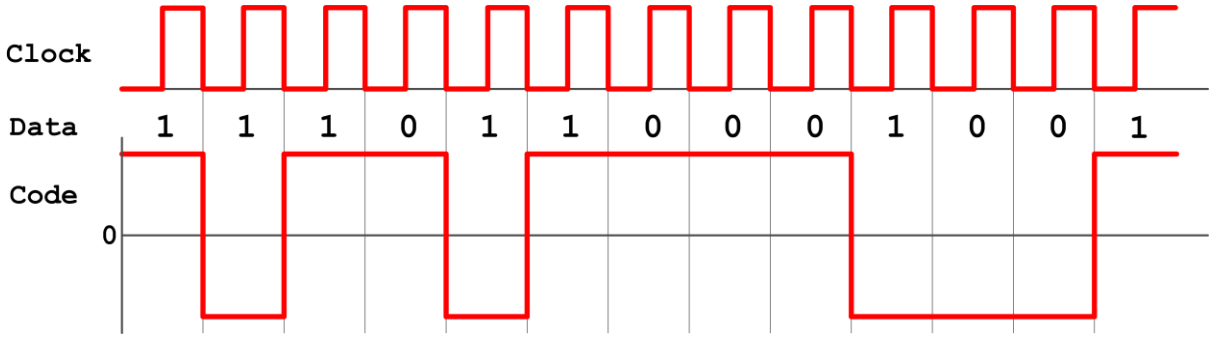
[9] Remember that the average power in a signal is the square of its rms value ($P_x = x_{rms}^2$).

A variant of Polar NRZ is Non-Return-to-Zero-Level (**NRZ-L**) in which the **1**'s and **0**'s are represented by $-p(t)$ and $p(t)$, respectively. This is Polar NRZ using negative logic. An example where NRZ-L is used is the legacy **RS-232 serial** port communication.



NRZ-L Code

3. Non-Return-to-Zero, Inverted (NRZI):
NRZI is a variant of Polar NRZ. In NRZI there are two possible pulses, $p(t)$ and $-p(t)$. A transition from one pulse to the other happens if the bit being transmitted is a logic **1**, and no transition happens if the bit being transmitted is a logic **0**.



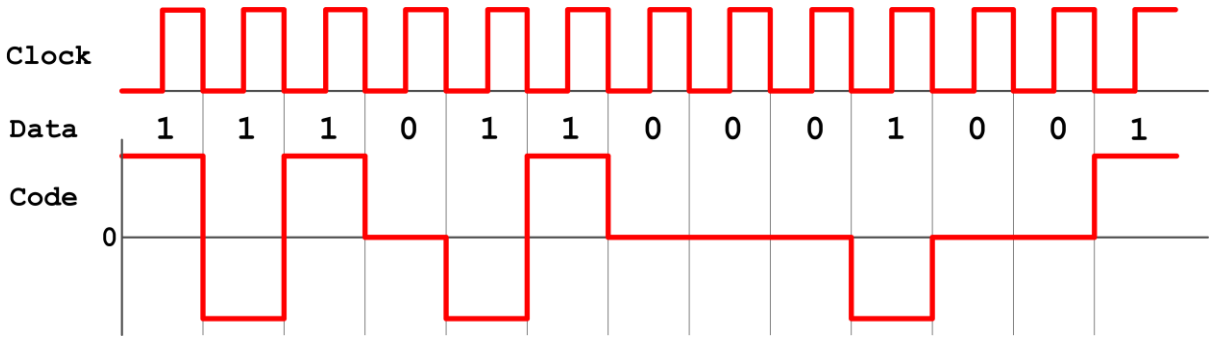
NRZI Code[10]

This is the code used on compact discs (CD), USB ports, and on fiber-based Fast Ethernet at 100-Mbit/s (100Base-FX).

NRZI can achieve synchronization between the transmitter and receiver, if we make sure that there are enough umber of **1**'s in the transmitted bit stream.

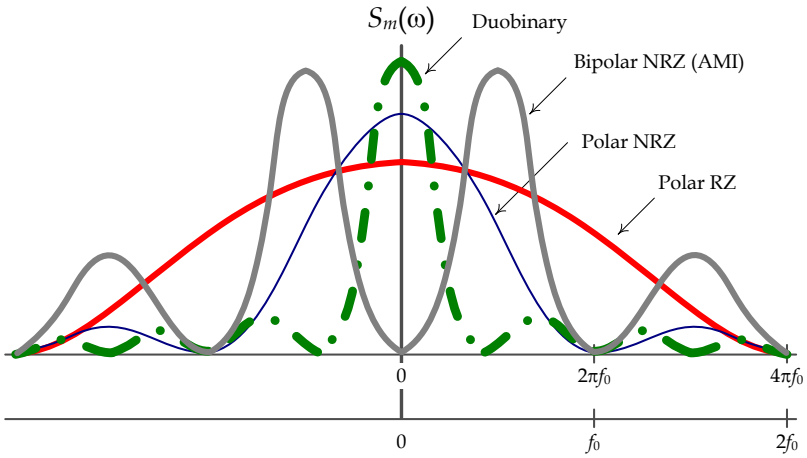
4. Bipolar encoding (*also called* **Alternate Mark Inversion (AMI)**):
Bipolar (or AMI) is a three-level system that uses $p(t)$, $-p(t)$, and the absence of pulses (e.g. +5V, -5V, 0V) to represent logical values. A logic **0** is represented with an absent pulse, and a logic **1** by either a positive or negative pulse. The direction of the pulse is opposite of the pulse sent for the previous logic **1** (mark) (see the Figure below).

[10] NRZI is always polar *not* unipolar.



Bipolar (AMI) Code

The alternating code in bipolar encoding prevents the build-up of a DC voltage in the cable. You can also observe the absence of low frequencies (including the DC component) from the PSD for AMI shown below.



Line Code	Bandwidth
Unipolar NRZ	f_0
Unipolar RZ	$2\,f_0$
Polar NRZ	f_0
Polar RZ	$2\,f_0$
Bipolar NRZ	f_0
Duobinary	f_0 / 2

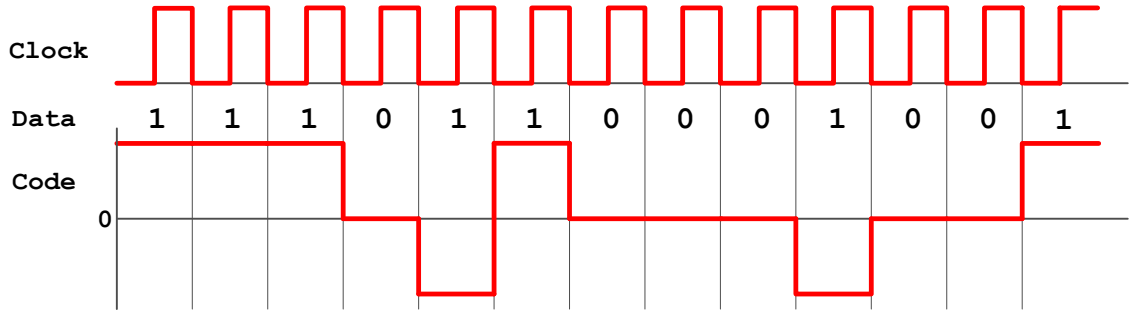
AMI coding was used extensively in first-generation digital telephony PCM networks. AMI suffers the drawback that a long run of **0**'s produces no transitions in the data stream, and a loss of synchronization is possible. This was solved in telephony by adopting improved variants of AMI encoding to ensure regular transitions in the baseband signal even for long runs of **0**'s. The **Binary-with-8-Zero-Substitution (B8ZS)** is the line coding scheme that was adopted for North America T1 system, while **High-Density Bipolar 3-Levels (HDB3)** was the line coding scheme used in the European E1 system.

This is not part of the exam material …

Note: A very similar encoding scheme to AMI, with the logical positions reversed, is also used and is often referred to as **pseudoternary encoding**. This encoding is essentially identical to AMI, with marks (**1**'s) being zero voltage and spaces (**0**'s) alternating between positive and negative pulses.

Note: Coded Mark Inversion (CMI) is another variation of AMI, where **0** bits are represented by a transition in the middle of the bit time instead of zero voltage.

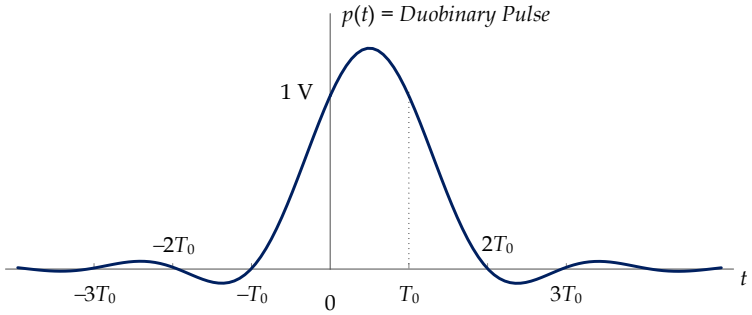
5. Duobinary:
In a duobinary line code a **0** bit is represented by a zero-level electric voltage; a **1** bit is represented by a $p(t)$ if the quantity of **0** bits since the last **1** bit is *even*, and by $-p(t)$ if the quantity of **0** bits since the last **1** bit is *odd*. An illustration of the duobinary line code is shown below.



Duobinary Code

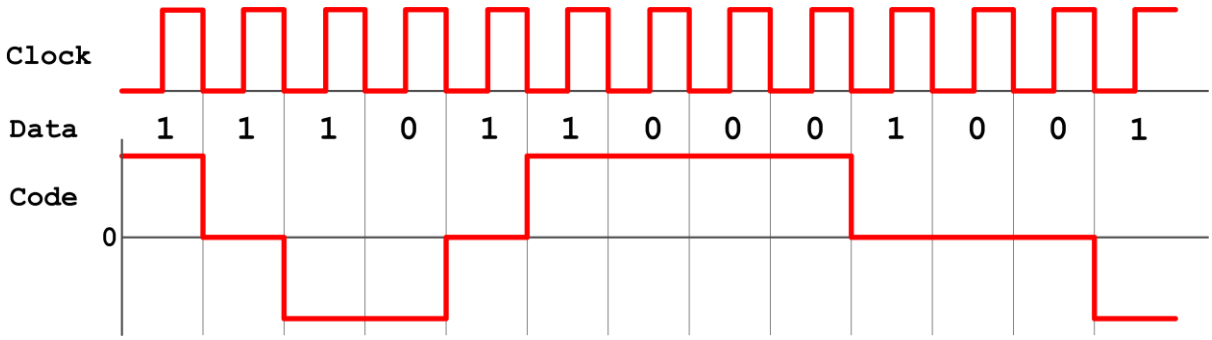
For a bit rate of f_0, duobinary line code requires $f_0/2$ bandwidth, which is the minimum possible (theoretical) bandwidth for any digital baseband signal (called *Nyquist bandwidth*). In addition, the duobinary line code permits the detection of some transmission errors without the addition of error-checking bits. However, duobinary line codes have significant low frequency components as seen by the PSD shown earlier. The differential version of the duobinary line code is common in the 20 Gbit/s and 40 Gbit/s uncompensated optical fiber transmission systems.

It is important, however, that you do not confuse a duobinary *line code* (explained *above*) with something completely different called a duobinary *pulse* (shown *below*). This pulse (which you are going to study in the "Communications II" course) is commonly used in controlled inter-symbol interference (ISI) scenarios. Confusingly, duobinary signaling refers to using the duobinary pulse with a *polar line coding* rule (*not* a duobinary line coding rule).



6. Multi-Level Transmission 3-Levels (MLT-3):
MLT-3 encoding is used mainly in 100BASE-TX Fast Ethernet, which is the most common type of Ethernet nowadays. MLT-3 cycles through the states $-p(t)$, 0, $p(t)$, 0, $-p(t)$, 0, $p(t)$, 0, ... etc. It moves to the next state to transmit a **1** bit, and stays in the same state to transmit a **0** bit.

MLT-3 has many advantages including emitting less electromagnetic interference, requiring less bandwidth than unipolar, polar, and bipolar (AMI) signals operating at the same data bit rate. The PSD of MLT-3 code is shown in Figure 22.



MLT-3 Code

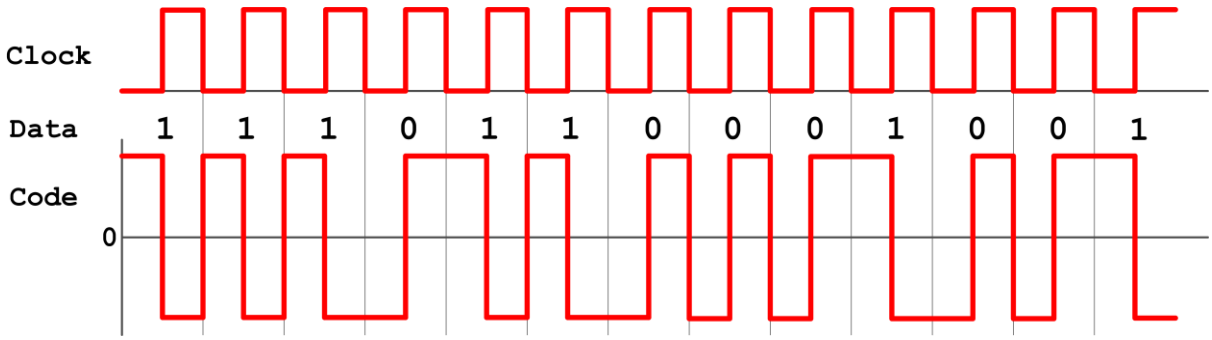
7. Manchester:

There are two opposing conventions for the representation of Manchester codes:

The **first convention** of these was first published by G. E. Thomas in 1949 and is followed by numerous authors (e.g., Andrew S. Tanenbaum). It specifies that for a **0** bit the signal levels will be **Low-High** with a low level in the first half of the bit period, and a high level in the second half (see figure below). For a **1** bit the signal levels will be **High-Low**.

The second convention is also followed by numerous authors (e.g., Stallings) as well as by **IEEE 802.4 and IEEE 802.3 (Ethernet** 10 Mbps 10Base-T**) standards**. It states that a logic **0** is represented by a **High-Low signal** sequence and a logic **1** is represented by a **Low-High signal** sequence.

If a Manchester encoded signal gets inverted somewhere along the communication path, it transforms from one variant to another. In this article, we will adopt the first convention (see figure below).

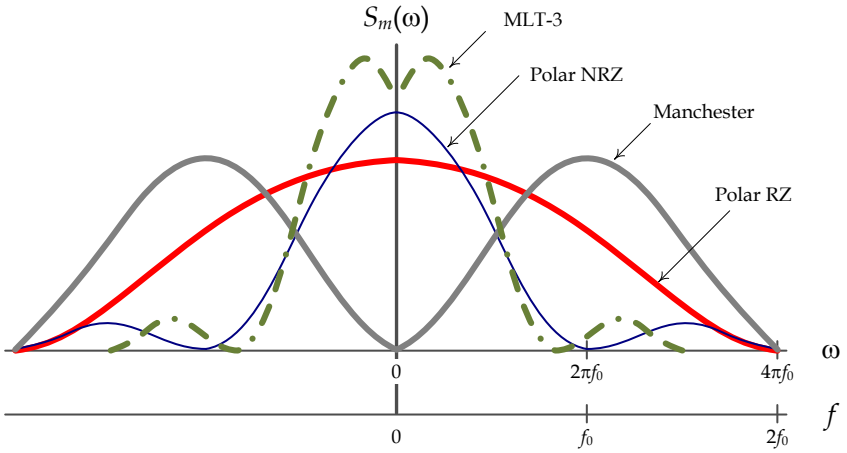


Manchester Code[11]

In a Manchester code each bit of data is signified by at least one transition. Manchester encoding is therefore considered to be self-clocking, allowing an accurate clock recovery from the data stream. In addition, the DC component of the encoded signal is zero.

[11] Manchester is always polar *not* unipolar.

Although transitions allow the signal to be self-clocking, it carries significant overhead as there is a need for essentially twice the bandwidth of a simple Polar NRZ or NRZI encoding (see the PSD below). This is the main disadvantage of the Manchester code.



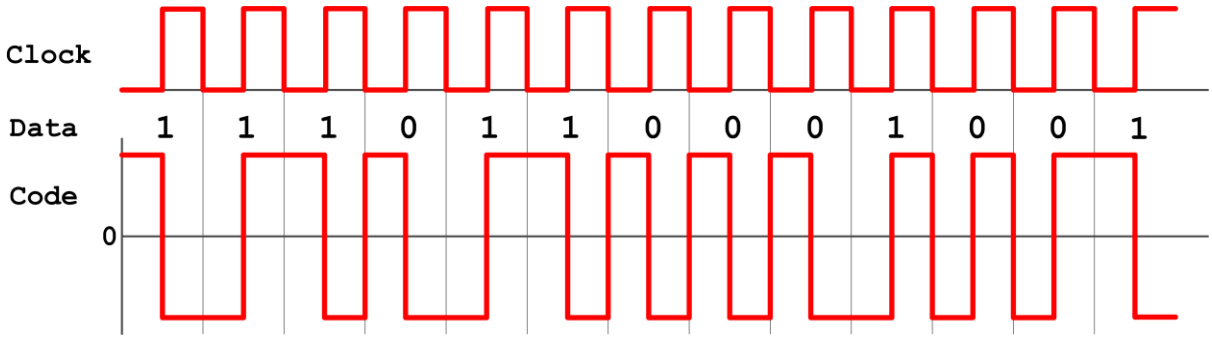
Line Code	Bandwidth
Polar NRZ	f_0
Polar RZ	$2\,f_0$
MLT-3	$0.9\,f_0$
Manchester	$2\,f_0$

Figure 22.

This is not part of the exam material …

Differential Manchester encoding

In Differential Manchester encoding a **1** bit is indicated by making the first half of the signal equal to the last half of the previous bit's signal i.e. no transition at the start of the bit-time. A **0** bit is indicated by making the first half of the signal opposite to the last half of the previous bit's signal i.e. a zero bit is indicated by a transition at the beginning of the bit-time.



Differential Manchester Code

Because only the presence of a transition is important, differential schemes will work exactly the same if the signal is inverted (wires swapped).

In the middle of the bit-time there is always a transition, whether from high to low, or low to high, which provides a clock signal to the receiver.

Differential Manchester is specified in the IEEE 802.5 standard for IBM Token Ring LANs, and is used for many other applications, including magnetic and optical storage.

8. *M*-ary Coding

All the line coding schemes discussed above are called binary codes, since the number of bits per second is identical to the number of symbols per second (called *baud rate*). We say that for binary signaling:

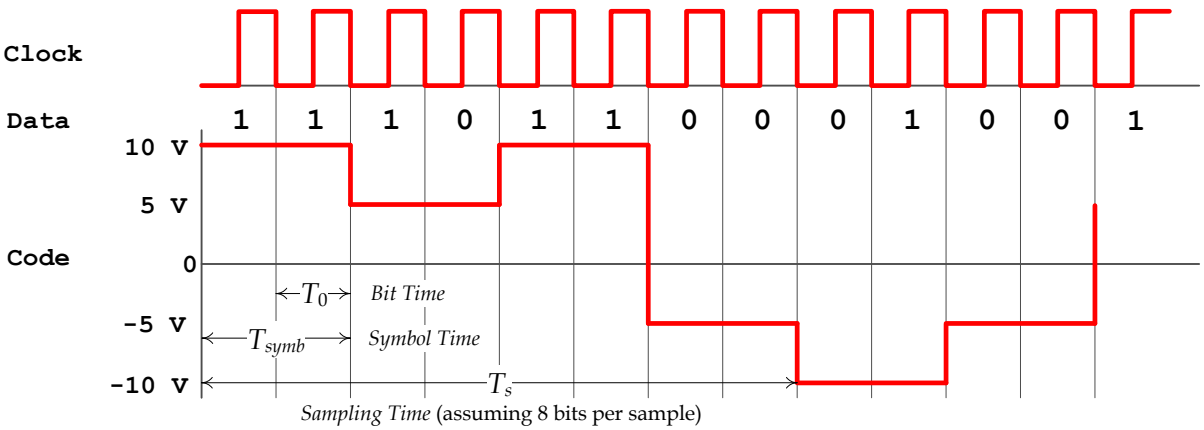
$$f_0, \text{ data bit rate [}\textit{in units of}\text{ bit/s]} = f_{symb}, \text{ symbol rate [}\textit{in units of}\text{ baud]}$$

Notice that a symbol is defined as a waveform pattern that the line code has for a certain period of time before switching to another waveform pattern (i.e., another symbol).

In *M*-ary signaling, on the other hand, a cluster of bits is grouped to represent one symbol. For example, in the 4-ary (*also called* Quaternary) case, two bits are grouped into one symbol. The two bits can be in one of 4 possible states, which means that the symbol can take M=4 different symbols. The following table shows a possible mapping of two bit values to four symbols of a Quaternary signal. The corresponding line code for this Quaternary signal representing the bit stream 1110110001001 is shown next.

Bits	Symbol
00	–5 V
01	–10 V
10	5 V
11	10 V

M=4 levels



Quaternary Code

Notice that a symbol time T_{symb} is now twice the bit time T_0. This means that there are half as much symbol transitions as there are bit transitions. We can say that:

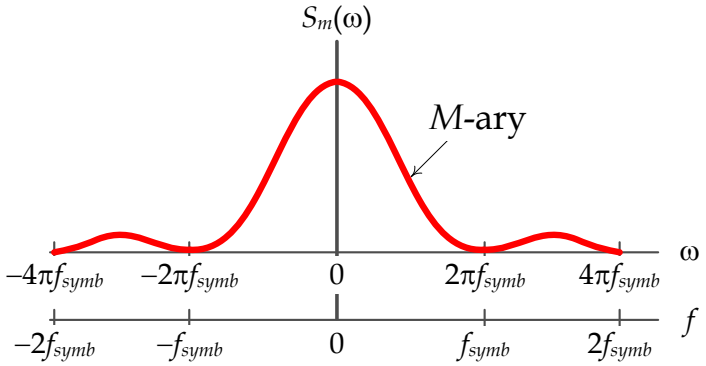
$$\text{symbol rate [}\textit{in units of}\text{ baud]} = (\tfrac{1}{2}) \times \text{data bit rate [}\textit{in units of}\text{ bit/s]}$$

For a general *M*-ary line coding scheme, we have:

$$\text{symbol rate [}\textit{in units of}\text{ baud]} = (1/\log_2(M)) \times \text{data bit rate [}\textit{in units of}\text{ bit/s]}$$

where M is the number of levels (possibilities) for a symbol. Such a drop in transition rate in the resulting signal will reduce the bandwidth of the signal by a factor of $\log_2(M)$, because the bandwidth of digital signal is actually dependent on its baud rate f_{symb} not its bit rate f_0.

Remember: An M-ary signal is a baseband signal that has a bandwidth equal to its baud rate (f_{symb}). The PSD of the M-ary signal is shown below.



V. Pulse Shaping:

In explaining the above line codes, we limited ourselves to pulses $p(t)$ shaped as either *rectangular NRZ* or *rectangular RZ* pulses. It is essential that you understand that these two are not the only choices you have; instead a variety of pulse shapes $p(t)$ can be used without compromising the information transfer process. In this section, we will discuss two possibilities: the *triangular* pulse and the *raised cosine* pulse. The Figure below shows the bit stream 1110110001001 represented using a *polar* line code combined with a *triangular* pulse shape.

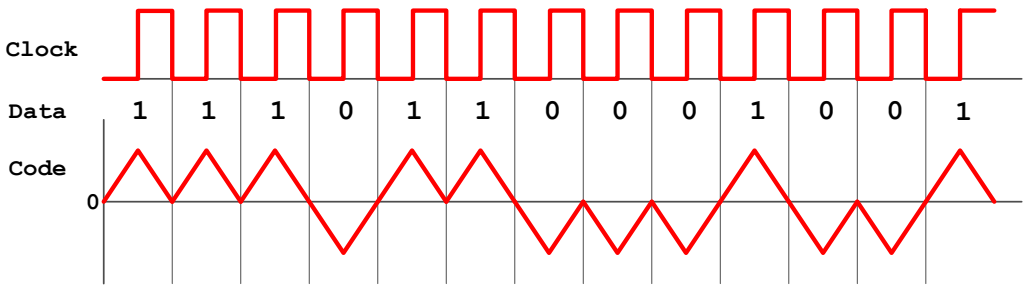


Figure 25.

You might be wondering at this point about the advantages of choosing a pulse shape different than the simple and familiar rectangular pulse? Well, there is a number of advantages for doing so, the main of which is being able to control the shape of the power spectral density PSD (and hence the bandwidth) of the resulting digital baseband signal.

To understand this, recall that the line code you chose earlier (unipolar, polar, bipolar, etc) has affected the bandwidth of the corresponding code. However, the PSD shape always looked like a $\text{sinc}^2()$ function. The reason this was the case is that the Fourier transform of a rectangular pulse has the sinc() shape and because the PSD is the square of the Fourier transform, the PSD looked like a $\text{sinc}^2()$ function (see Figure 27).

You also recall that using a RZ rectangular pulse instead of a NRZ rectangular pulse resulted in expanding the bandwidth by a factor of 2. This is an immediate result of the "Time compression, Frequency expansion" property of the Fourier transform, and is also illustrated in Figure 27.

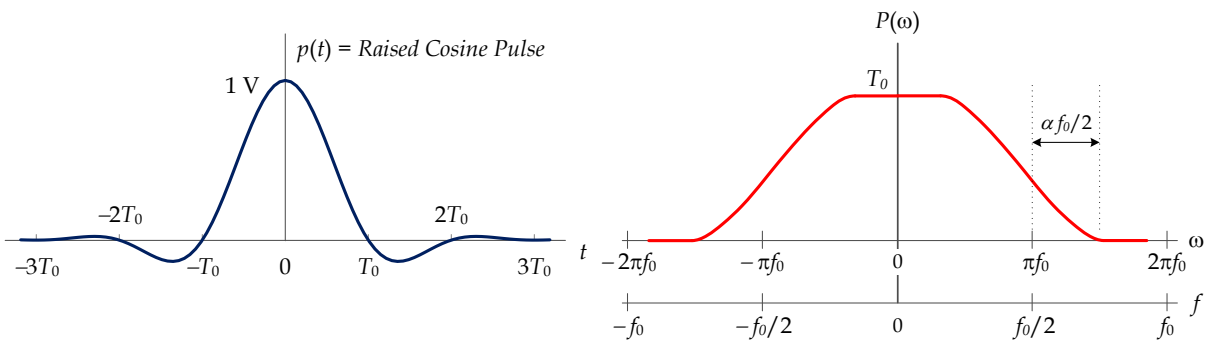
The question that arises now is this: If we pick a data stream encoded using a particular line code (say *polar encoding*), can we control its bandwidth by changing the pulse shape $p(t)$ while still keeping the polar line code rule? The answer to this question is YES; picking the right pulse shape can result in smaller bandwidth compared to the bandwidth of the rectangular pulse which is, strictly-speaking, infinity (because the sinc() function extends from minus infinity to positive infinity).

If we choose a *smoother* pulse (compared to the rectangular pulse), the high frequencies in the resulting PSD of the data stream are eliminated. For example, if we pick a *triangular* pulse instead of a *rectangular* pulse (see Figure 27) the high frequency components are heavily reduced. (This is more apparent if you look at the spectrum using a log scale). Remember that the Fourier transform of a triangular pulse is $\text{sinc}^2()$ and, hence, the PSD has the shape of $\text{sinc}^4()$ function.

Reducing high frequency components in the PSD is important so that the signal can pass through band-limited channels without too much linear distortion. Such distortion usually creates what is called **inter-symbol interference (ISI)** in digital systems, where one bit value overlaps with (and corrupts) the adjacent bits.

Using smoother pulses (and hence reducing their high frequency content) can be taken to the extreme if we pick a pulse that is not limited in time. In other words, the pulse spells outside the bit time (T_0). This will limit its Fourier transform (because an expansion in the time-domain results in compression in the frequency-domain).

One popular example used in many practical systems is the *raised-cosine pulse*. This is a pulse that looks *similar* to (*but decays much quicker than*) a sinc(t) function and has a Fourier transform similar to a raised cosine shape in the frequency domain. This pulse and its corresponding Fourier transform are shown below. The raised-cosine pulse shape changes with a parameter called the *roll-off factor* α. The Figure below shows a $\alpha = 0.5$ raised-cosine pulse with the corresponding Fourier transform.



This type of pulse satisfies what is called the **first Nyquist ISI criterion**, which states that it is OK for the pulse shape not to be limited to the bit time (T_0), so long as it does not introduce inter-symbol interference (ISI) in the transmitted pulse stream.

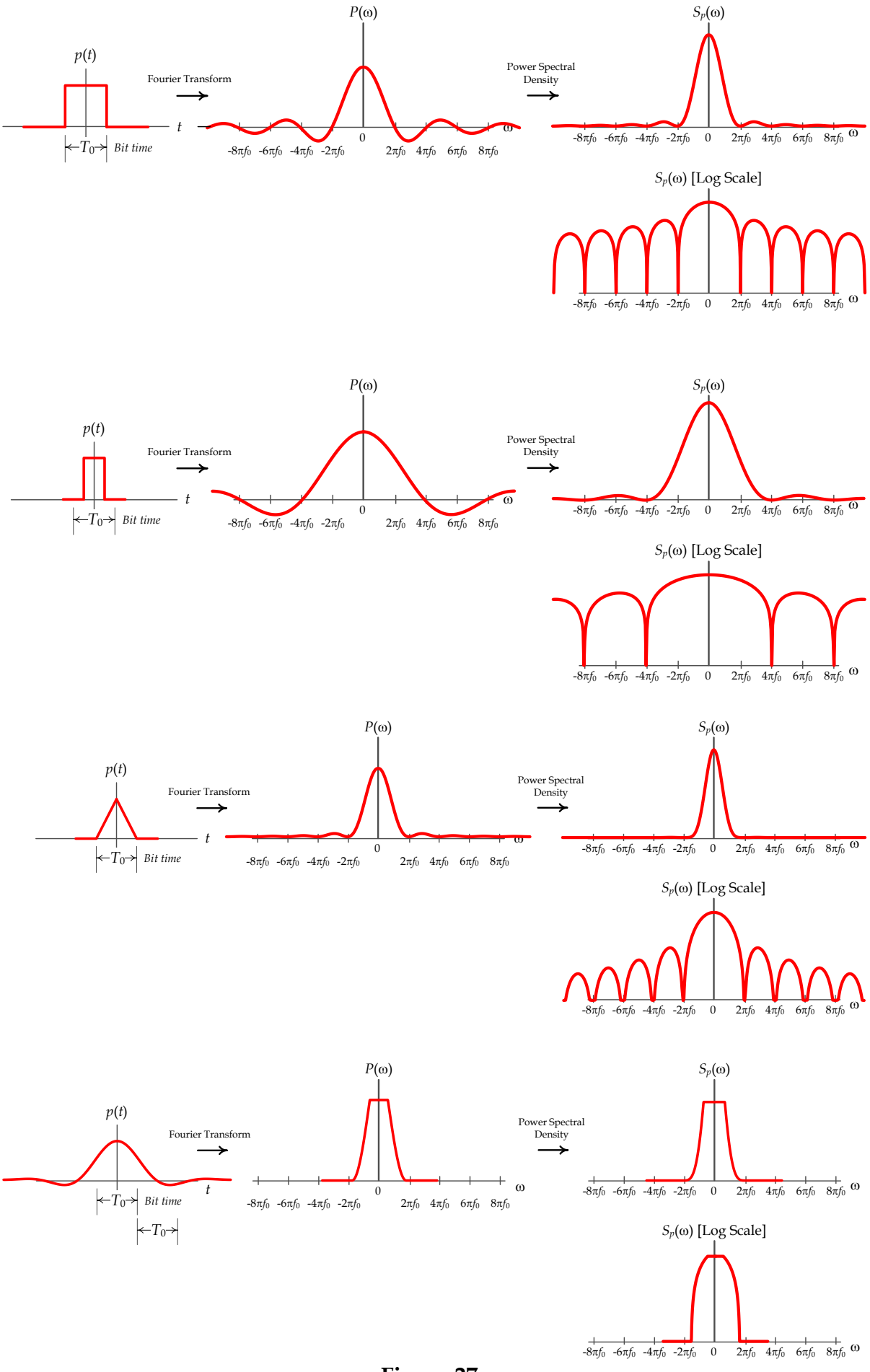

p(t)
Fourier Transform
P(ω)
Power Spectral Density
$S_p(\omega)$
t
T_0
Bit time
ω
$S_p(\omega)$ [Log Scale]
$-8\pi f_0$ $-6\pi f_0$ $-4\pi f_0$ $-2\pi f_0$ 0 $2\pi f_0$ $4\pi f_0$ $6\pi f_0$ $8\pi f_0$

Figure 27.

An example of using the raised-cosine pulse combined with polar line code is shown in Figure 28, where the binary sequence '1011011100' is transmitted. Notice that we use $p(t)$ for logic **1** and $-p(t)$ for logic **0** (i.e., the *polar* line code). The dotted lines represent the raised-cosine pulses $p(t)$ and $-p(t)$ corresponding to individual bits, while the solid line represents the result of adding these pulses (i.e., the transmitted signal on the channel). At the receiver side, sampling the received signal at *exactly* the middle of the bit time will retrieve the original bit sequence as shown in the Figure.

Our example used raised-cosine pulses with parameter $\alpha = 0.5$. In such case, the bandwidth of the resulting stream is given by the formula $(1+\alpha)\, f_0 / 2 = (1+0.5)\, f_0 / 2 = 0.75\, f_0$, where f_0 is the data rate. Hence, if $f_0 = 50$ kbit/s, the resulting bandwidth = 37.5 kHz. What would the bandwidth have been if you used a rectangular NRZ pulse? How about a rectangular RZ pulse?

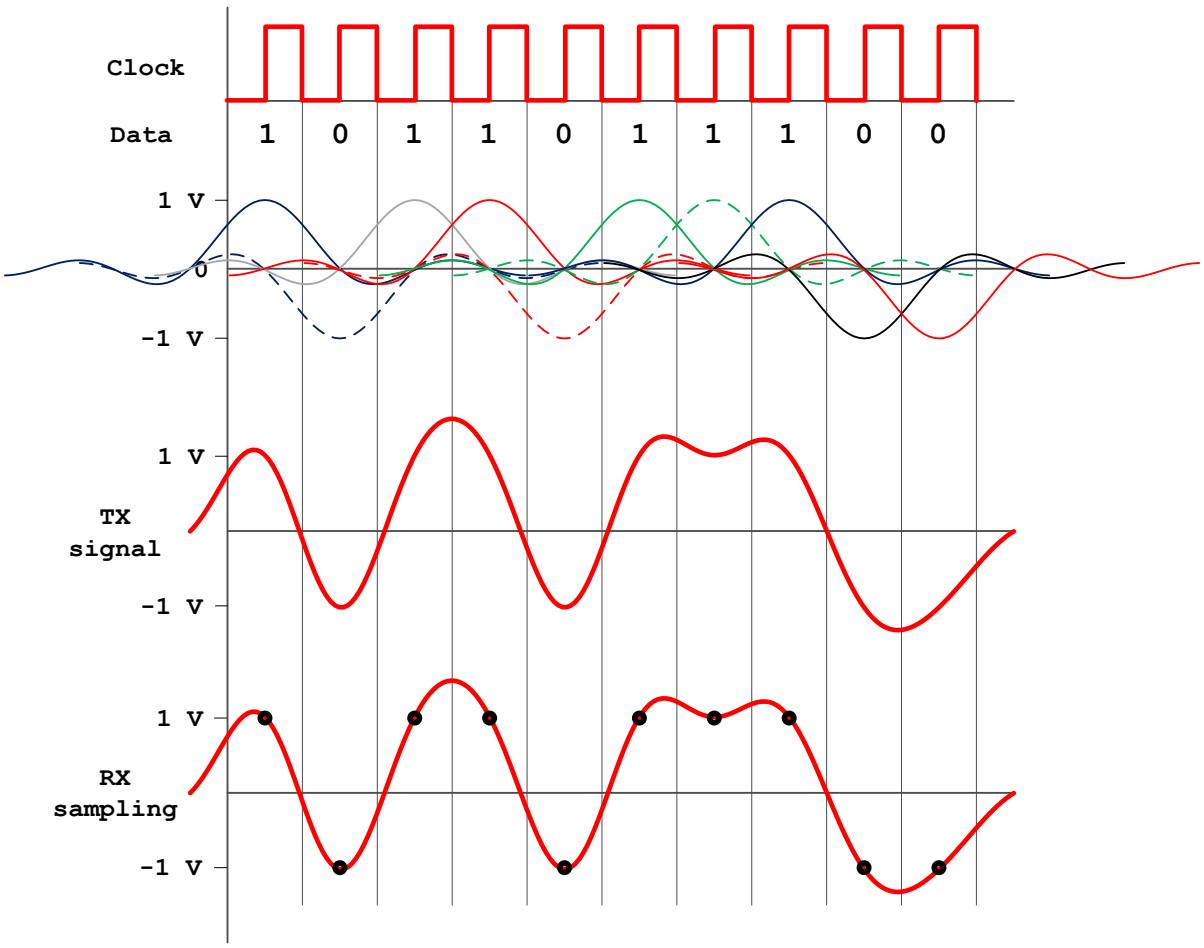


Figure 28.