Bit and Frame Synchronization Techniques

Martin Probst , Lars Trieloff
Hasso-Plattner-Institute for Software System Engineering
martin@probst.com, lars@trieloff.net

All graphics in this paper are created by the authors unless otherwise stated.

Bit and Frame Synchronization techniques are used in order to ensure that signals transmitted from one participant of the communication can be correctly decoded by the receiver. To achieve this goal, certain timing information must be passed to the receiver such as where communication units begin and end.In this article various bit and frame synchronization techniques are presented as well as underlying encoding schemes and applications using the described techniques.

Table of Contents

1 Introduction

If one wants to transfer binary information over a physical medium in oder to establish communication between two computer systems there are several problems to be solved. Among the most basic tasks are choosing the best available transmission media, agreeing on a unambiguous encoding scheme and verifying the integrity of the received data.

Bit and frame synchronization techniques are among the most basic problems to be handled at the physical layer. In this paper we are going to describe fundamental problems and their solution in the context of binary data transmission at a very low level. In the next sections we will first dicsuss the basic prerequisites of binary data transmission and later on specific problems of encoding schemes, synchronization issues and their solutions.

Digital data will always differentiate between two distinct values 0 or 1, but every physical transmission medium will provide an analogous signal, thus a conversion scheme between digital data and their physical representation must be defined. We will take a look at the most common transmission media and signaling scheme in the introductory section of this article.

The next section will feature several basic encoding principles that are used to ensure that the transmitter can identify the borders between two adjacent bits on the transmission medium. This section will also include a description of two simple encoding schemes used for translating bits into physical signals and vice versa.

In the fourth section we are going to cover the basics of ensuring synchronization at the level of single bits, which is needed when the system clocks of sender and receiver must be in synchronism. As different systems use different quartz clocks, which tend to run ahead or late it must be possible to set the receiver's clock to match the transmitter's signal.

Using atomic clocks is no solution, because these devices are very expensive and difficult to maintain for a non-scientist. If you try to use the broadcasted clock signal of an atomic clock, you will note that this signal will propagate through space with a limited speed and will most possibly not arrive at the transmitter and at the receiver in the same instant. Additionally the clocks signal must be encoded somehow, and this was exactly the starting point of this problem.

We will show how the clock signal can be encoded in the data stream or transmitted on top of it and by which circuits the receiver system can extract the clock signal from the data signal.

The fifth section discusses the problems encountered when data is transmitted in bigger blocks (typically about one thousand bytes) called frames or packets. As a receiver needs to distinguish between the single frames a method of frame synchronization is needed. We will explain the common techniques of start & end flags (which brings up the problem of bit or byte stuffing explained thereafter), packet length indication and coding violations.

Bit and byte stuffing are techniques of in-band-signalling. With them it is possible to use reserved words or characters on the transmission medium.

In the final section, we are going to compare different real-world applications of the principles and techniques described in the sections before.

2 Fundamentals of Binary Data Transmission

In this section we will discuss the main facts one needs to know in order to understand the idea and principles of synchronization techniques at the physical layer.

The physical layer in the ISO/OSI-Model (International Organization for Standardization/ Open Systems Interconnection) of networks communication is the level that deals with network hardware to hardware communication at the electrical interface. Efficient algorithms and structures at the physical level enable protocols and applications at a higher level to be reliable and performant.

Because of the strong orientation towards physical signals, most structures and procedures described in the following sections are implemented in hardware, for exmaple in network cards or serial interface controllers.

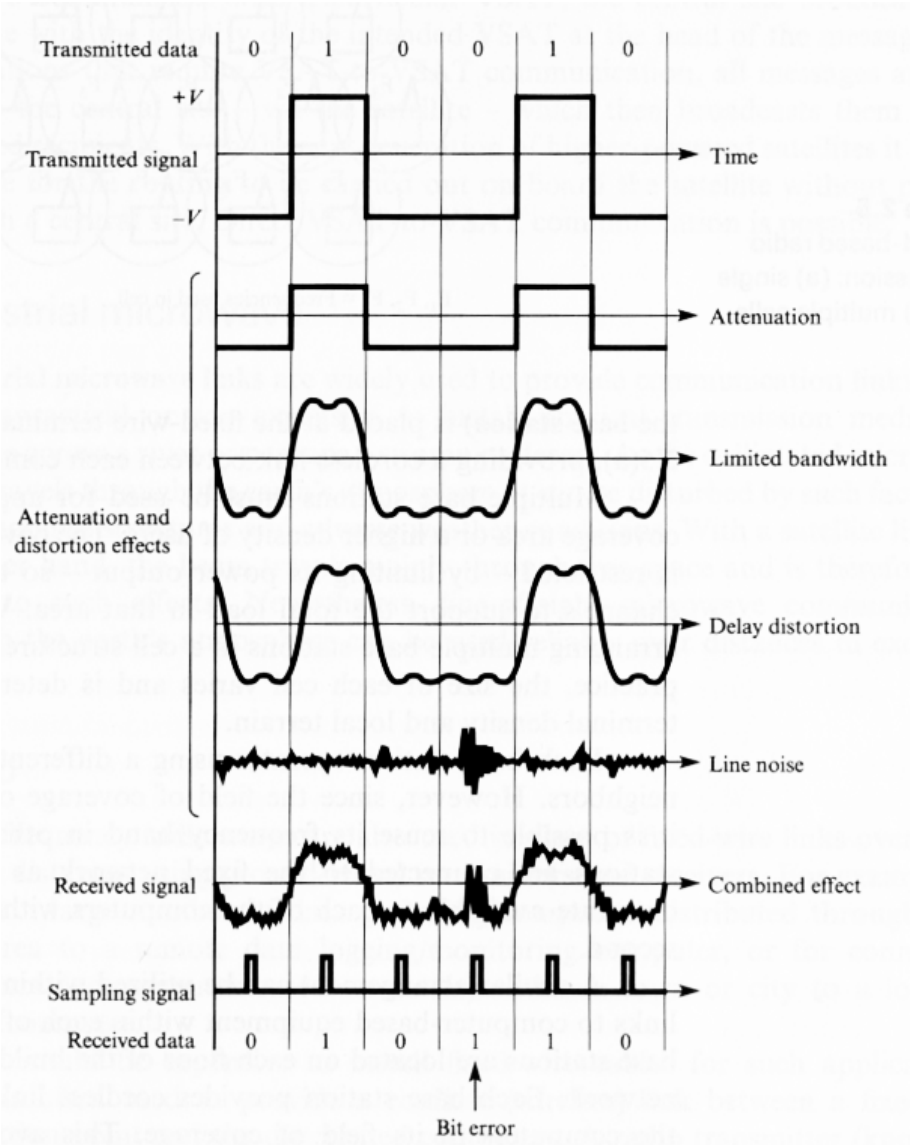
2.1 Binary Data Transfer

Todays computer systems work on binary data and for establishing a connection between two or more computer systems binary data transfer is needed. To transfer data bits between two computers they must be converted into physical signals. These signals may be signals on an electrical conductor, optical signals or signals in form of magnetic waves.

All those physical signals are vulnerable to errors due to attenuation and distortion of the transmitted signal with the effect that the received signal does not have to match the transmitted signal. The transmitted signal is a function of time, that means the strength and quality of the signal changes over time. This behavior is also known as serial data transfer because the physical representation of one bit is transferred after another.

In order to convert the received signal electrical into a logical unit of information that can be dealt with it is necessary to provide a sampling signal that will be `true` for a certain time span and a defined time interval.

Figure 1. Transfer of binary information over an imperfect medium



from [halsall95], page 32, figure 2.6

The figure above shows the limitations of every transmission medium: the signal is attenuated, causing a loss of information, a time shift is added due to the limited transmission speed and possible noise in the transmission line can cause additional errors.

The misinterpretation of a signal is called bit error.

What will be transferred? The sender of digital data will encode the information in a certain way. This means it creates code words consisting mostly of eight code characters. Each code character is either one or zero. Code words may be arranged in message blocks which are also known as frames.

As data transmission media contain in most cases only one “connection line”, but almost all networked devices operate in parallel mode (they do not operate on single bits but on bytes and words), a conversion between parallel mode and serial mode is necessary. This is usually done through shift registers with parallel load, which are also

known as SIPO registers (Serial In, Parallel Out) or PISO (Parallel In, Serial Out). Nonetheless they will work in both directions.

2.2 The Need for Synchronization

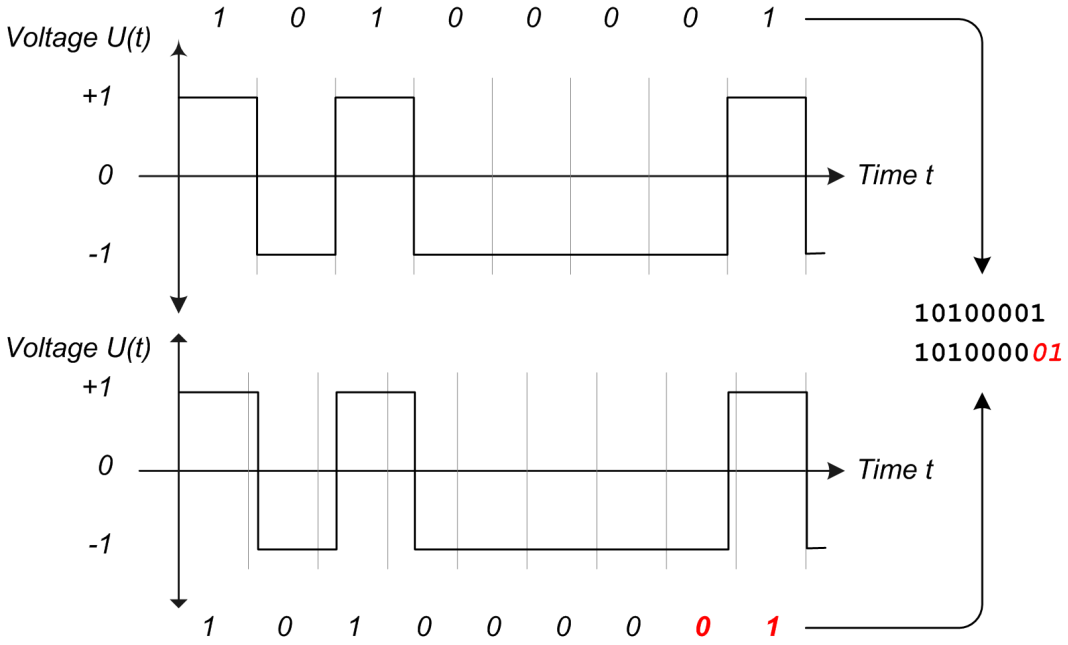
There are several points which make it clear why synchronization is essential for serial transfer of digital data. The first problem is that sender and receiver clocks may not run synchronous so that the sampling instant will shift from the beginning of the signal to the end and may overflow.

This is especially likely when there is no hint, where a data entity, may it be a bit or byte or a complete packet starts and ends and when a long line of similar signals are transferred, for example a row of logical 1 which would be encoded as a constant voltage of a certain value.

Example 1. Clocks running out of synchronization

Imagine a communication system with two Data Terminal Equipments (DTEs) that have a major difference in the speed of their system clocks. The receiver's clock is running 12,5% ahead of time than the sender's one. If the sender transmits a 8-bit word, the receiver will interpret it as a 9-bit word.

Figure 2. Synchronization problems



But as you have seen in the graphic above, the receiver will not only sample to much bits, it will also sample wrong bits. The conclusion should be that there can be no unambiguous interpretation of a common signal, if there is not a certain degree of synchronization of clocks.

Another well-known problem of time dispersion is called intersymbol interference. Signals belonging to different symbols can be observed on the medium at the same time, leading to interpretation errors at the receiver's end.

Synchronization techniques will guide the receiving system in determining where data entities start and end and at which time interval the sampling result is least error-prone.

You can see bit and frame synchronization as a very basic mechanism of error control which will reduce the need for error control at higher levels.

What is needed to decode the received signal? The receiver will have to determine where a signal cell (representing a bit) starts and ends in order to sample the signal as near at the middle of the signal as possible. It will have to know where a character or a byte starts or ends and, for packet based transmission, where each message block starts or ends.

Finding out where a signal cell starts and ends is known as bit or clock synchronization, determining the character boundaries is known as character or byte synchronization and the last task is called block or frame synchronization.

2.3 Modes of Transfer

Synchronous and Asynchronous Data Transmission

There is one common way to differentiate data transmission modes concerning the synchronization techniques utilized. The criteria of distinction is here whether the transmitter clocks are independent or synchronized.

Asynchronous communication. When there is no synchronization of transmitter and receiver clocks, it is called asynchronous transmission. This mode is used when each element of the transmission (character or byte) should be treated independently.

Synchronous communication. On the other hand, if the complete block of transferred characters is considered one entity of communication, we will speak of synchronous transmission.

For asynchronous transmission the receiver will have to find out, where each byte or character starts and where it ends. This does not imply the need for clock synchronization.

In synchronous transmission mode the clocks of sender and receiver have to run in sync for the time of the transfer of one block. For the next block a new synchronization may be possible.

2.4 Characteristics of Transmission Media

To get a basic understanding of physical network communication it is essential to know the different transmission media that are being used for communication networks as well as their strengths and weaknesses for different applications.

For an introductory explanation it should be enough to present the most important groups of transmission media.

2.4.1 Conductor Cable

Electric conductor cables are the most simple transmission media. In their simplest form they consist of two isolated wires, one for the actual signal and another for a relative grounding connection. The resulting signal is computed from the difference of voltage on the two cables.

It is not unusual to use one grounding line which is shared by multiple signal lines, thus allowing to send more signals at once. The simple variant with two lines is called two-wire open line, the multiple line variant is either a multi-core cable or a flat ribbon cable.

Common problems with two-wire open line and variants. There are two frequent problems with open-wire lines: The first is that they are vulnerable to noise signals from other electrical sources or from electromagnetic radiation. The lines act as antennas. The problem is that it is possible that the noise signal is received by only one of the two lines, causing an aberration in the computed difference. The second problem is called crosstalk. It describes the “cross-coupling of electrical signal between adjacent wires

in the same cable" [halsall95 p. 25]. The deeper cause of this problem is capacitive coupling between the two lines.

The combination of these problems sums up to a limitation of length and bit rate of the transmission media. With a maximum length of 50 meters and a bitrate of at most 19200 bit per second this technology is mostly used for connecting a computer with peripherals like modems.

Twisted-pair lines. This problems can be partially solved by twisting a pair of lines together. The resulting product is called twisted pair cable and it can reduce crosstalk and, as it is more likely that a noise signal is picked up by both lines, the caused signal difference between grounding and signal line will be reduced.

Twisted pair cables can achieve bit rates of 10 megabit per second (Mbps) when used for distances no longer than 100 meters and lower bit rates for longer distances. This makes twisted pair lines ideal for Local Area Networks (LANs).

A third important variant of electrical conductor based connection are coaxial cables. Twisted pair lines suffer from the so called skin effect, which increases the electrical resistance of the cable for high frequencies, caused by high bit rates. The effect of an increase in electrical resistance is higher attenuation. This limits the overall bit rate that can be reached with twisted pair lines.

Coaxial cables. Another common type of conductor cables are coaxial cables, which consist of an inner conductor which is in line-form, an isolating coating and an outer conductor which will coat the coating of the inner conductor. The outer conductor shields the inner conductor from interfering radiation and also nearly eliminates the skin effect.

This leads to bit rates as high as 10 Mbps over several hundred meters of line. But on the other hand, coaxial cables are - due to their construction - relatively stiff and thus hard to lay. This is good for permanent connections but limits the usefulness when literally more flexibility is needed.

2.4.2 Optical transmission

Most optical transmission is done through optical fiber cables. There are several advantages of optical fibers as transmission medium over electric conductor based media:

- light waves have a higher bandwidth and frequency than electrical waves, allowing higher bit rates

- light waves are not vulnerable to electric phenomena resulting of electromagnetical fields like crosstalk or interference

An optical fiber cable consists of lots of single fibers, one for each signal. The coating of the fiber protects it from outside light sources and reflects the light inside the fiber. A better reflection results in less attenuation and by that in a sharper received signal. This is because every reflection extends the distance a light beam has to cover which in turn reduces the intensity of the light.

“The light signal is generated by an optical transmitter, which performs the conversion from normal electrical signals as used in a DTE. An optical receiver is used to perform the reverse function at the receiving end. Typically the transmitter uses a light-emitting diode (LED) or laser diode (LD) to perform the conversion operation while the receiver uses as light-sensitive photo diode or photo transistor.” [halsall95 p. 27]

The level of dispersion of the resulting signal is determined by the construction of a fiber. If the inner part of the fiber has another dispersive index than the outer part, light beams that reach the boundary between the two part are reflected (unless their angle is too large) and propagated through the fiber.

This behavior creates many reflexions at a high angle and thus a large spreading of times for the light beams to cross the distance which results in a dispersed received signal. The dispersion can be minimized by not using a single hard boundary between the inner and outer part of the fiber but rather a variable, rising optical reflection index. With this methodology, light beams with a smaller angle will be reflected earlier, resulting in a overall shorter time to cross the cable.

The third option is to reduce the width of the fiber to the wavelength of a single beam. This is called mono mode. It allows exactly one beam to go through the fiber without being reflected a single time. Mono mode fibers are used with laser diodes and can transmit data at bit rates of more than 100 Mbps.

Important

Unlike electrical transmission lines, optical fibers do not allow bipolar encoding. An electrical voltage can be positive or negative, but light can only be either on or off.

2.4.3 Electromagnetic Wave Transmission

The third important group of transmission media does not rely on a physical connection between the two communication participants. Instead it will create electromagnetic waves that will propagate though the air or — in case of satellites — through free space.

Satellites. For satellites a collimated microwave beam is used as transmission medium. The satellite will receive the microwave beam and retransmit it to another location at the ground. The circuit that will receive and transmit signals is called transponder.

Satellites are a very powerful means of communication as a satellite channel has a high bandwidth of 500 MHz and more and is able to “provide many hundreds of high bit rate data links” [halsall95 p. 29] by dividing the enormous bandwidth of the beam into various sub channels of one frequency band each.

There are two options for retransmitting the received signal: Either the new beam is focused at a certain location, which is appropriate for two-way communication or it is unfocused, so that the signal can be received over a wide geographic area. This is mostly done for transmitting television.

Satellites are vunerable to certain weather conditions such as thunderstorms, but not as much as terrestrial microwave connections.

Terrestrial microwave. The same principle applies for terrestrial microwave. It is important that there is a line of sight between sender and receiver. Terrestrial microwave

is used in cases where it is too expensive or impossible to run a line. Those connections are vulnerable to weather conditions, but they can cross distances of over 50 kilometers.

Radio. For a lower frequency band radio transmission is common. It does not have the bandwidth of microwave beams, but radio waves do not require a line of sight between sender and receiver. It is common to use one base station to cover a certain radio cell. If there should be adjacent radio cells, the corresponding base station must use a different frequency.

The usable data rate for DTEs in a radio based network is around 100 kilo bit per second (Kbps) but it may reach higher values, if the frequency is higher and thus the cells are smaller.

All electromagnetic wave-based media have no possibility to use bipolar encoding as it is used for direct electrical connection based communication.

2.5 Modulation Techniques

A conversion scheme between digital data and analog waveforms is called modulation. There are many different ways to modulate a logical signal into a waveform at the electrical or optical interface, based on the available transmission media and its characteristics as well as on the characteristics of the data to transfer.

Why is modulation important? In reality there are no binary media. Every bit of digital data has to be passed over an analogous medium in oder to transfer data. Every representation of digital data in the physical world is analogous. There needs to be one single scheme of encoding and decoding binary data into physical signals that sender and receiver have to know of in oder to establish data communication. This essential scheme of encoding and decoding is a modulation scheme. Without modulation there would be no possibility to represent digital data by physical signals.

In the previous section we described the most common transmission media and the fundamental thing they have in common, that waves (electrical, optical or electromagnetic) are used for data transmission. This means that there are common methods of keying information in waves, because every physical wave form has three important properties: amplitude, frequency and phase shift.

Most common are amplitude shift keying, frequency shift keying and phase shift keying. A method used on non-binary transmission media is quantization of signals which allows to transfer two or more bits at once.

For the following subjections it is nessecary to understand that we use herein the model of sinusodial carrier, meaning that the base signal on the medium has the shapes of a sinus wave and is thus characterized through fundamental properties amplitude (what are the maxium values measured on the carrier), frequency (how often does the polarization of the value change) and phase (what delay time can be measured between the basic sinus curve and the signal on the carrier). By variing these properties it is possible to encode data in a physical signal.

2.5.1 Amplitude Shift Keying

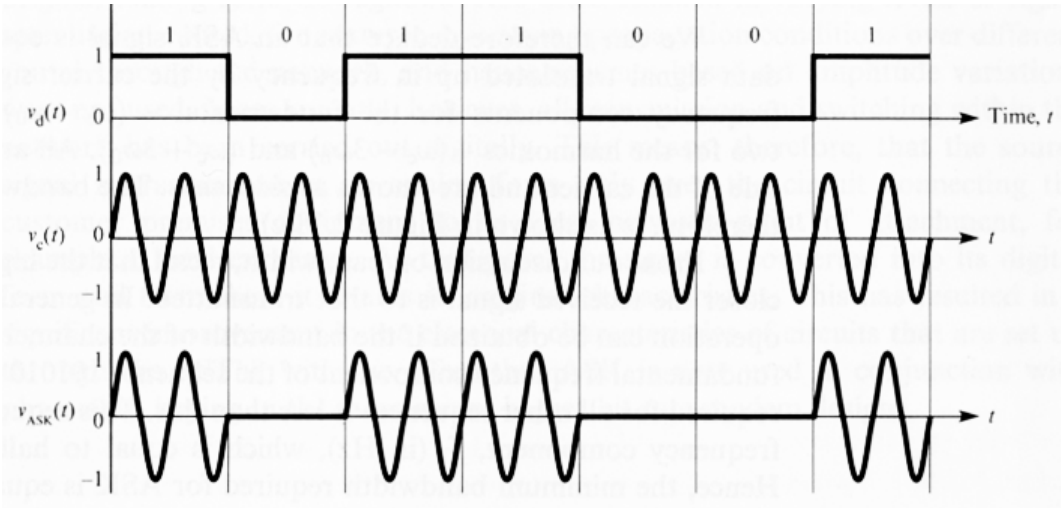
The term amplitude shift keying describes the idea that the you assign every possible bit value a corresponding amplitude of the waveform. The simplest approach would be

to use a high amplitude for a bit denoting 1 and no amplitude at all for a bit with the value 0. But there can be cases in which it is useful to use a certain amplitude for 1 and the half amplitude value for 0, because it will make it possible to distinguish between 0 and no signal at all due to a physical error in the connection.

The transmitter device will have to multiply a base amplitude with the bit value in some way to get an amplitude-shift-keyed signal. The receiver must introduce threshold value to determine which is high and which is low amplitude.

Using a fixed frequency, the transmitter device may count the reached peak values in the signal stream to count the number of bits received. If an encoding scheme is chosen which assigns no signal to zero, this procedure is impossible, because there will be no peak values for a 0-bit.

Figure 3. Amplitude Shift Keying



from [halsall95], page 59, figure 2.18

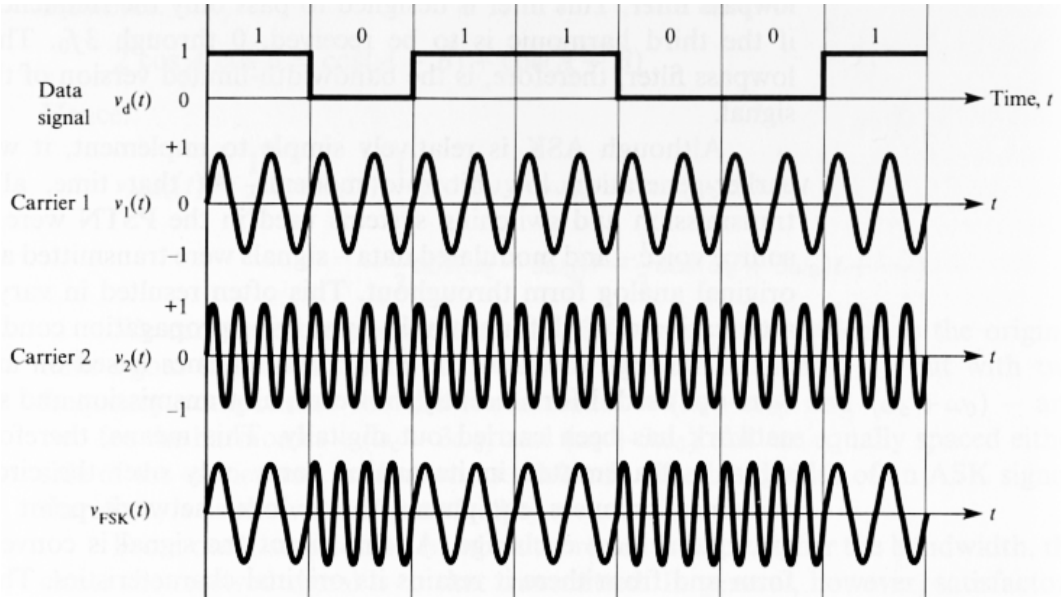
Figure Figure 3, “Amplitude Shift Keying” shows the basic operating principle. The carrier signal will be transmitted whenever the input is 1 and muted when the input is 0. Another name for this specific encoding scheme is On-Off-Keying, or OOK.

According to Fred Halsall [halsall95 pp. 58 ff.] amplitude shift keying is vulnerable to attenuation and strain which characterized the analogous telephone networks at the time when first generation modems where used. Because of this behavior the first modems used frequency shift keying instead which we will describe in the next section.

2.5.2 Frequency Shift Keying

In this keying method there are two base frequencies used to key the bit values. A high frequency is usually used to encode 1-bits and a lower frequency for 0-bits. For every base frequency there is one carrier. The switch between the two carriers is called frequency shift.

Figure 4. Frequency Shift Keying



from [halsall95], page 62, figure 2.19

The example above shows how the transmitter switches between the two carrier signals to emit the frequency-encoded signal.

To allow frequency shift keying, the bandwidth[1] of the carrier must be at least as high as the difference between the high and the low frequency. If the bandwidth is too low, it might happen that the difference of high to low frequency is too small and a high frequency cannot be distinguished from a low frequency, rendering this encoding scheme useless.

2.5.3 Phase Shift Keying

As the third fundamental variable of waves is the period shift, there is a corresponding keying practice. The basic operation principle is very similar to the frequency shift keying. The transmitter has a carrier signal which will create a reference signal with fixed frequency and amplitude.

There are two variants of phase shift keying. The most obvious is phase-coherent PSK (Phase Shift Keying). In this variant a 0-bit is encoded by sending the carrier signal and a 1-bit is encoded by sending the carrier signal with a 180° phase shift (which is simply the inverse signal). But how does the receiver know whether the received signal is the transmitters carrier signal or its inverse?

There is no trivial answer to this problem which means that the carrier signal has to be transmitted additionally to offer the receiver a reference value. This behavior costs additional transmission facilities and additional demodulation circuits.

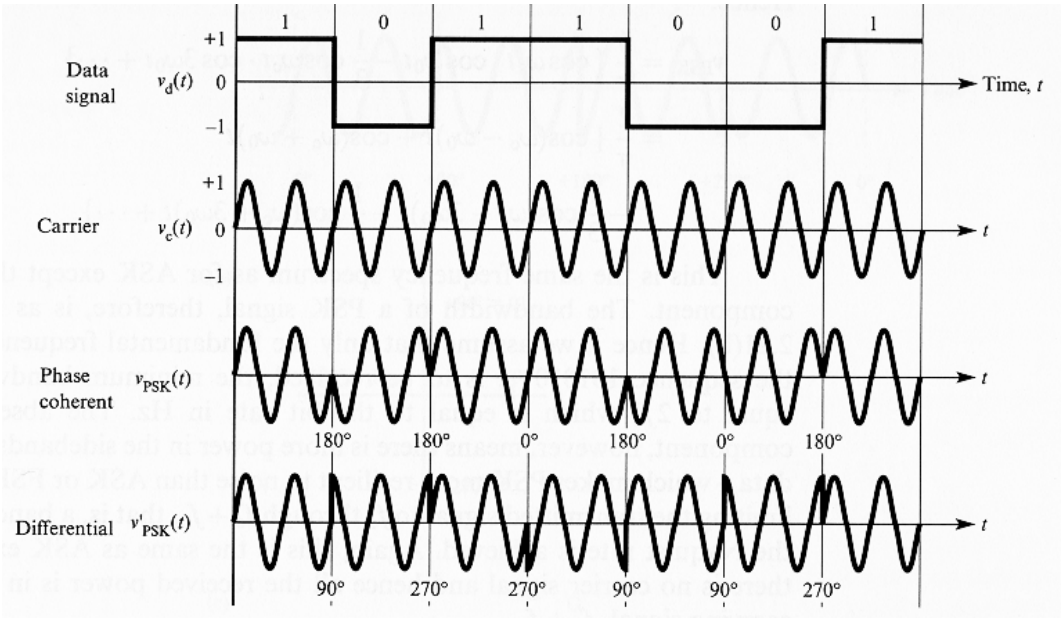
Please note the figure below, that will give an example of phase coherent phase shift keying.

[1] The bandwidth is the range of possible frequencies on the carrier medium

Differential Phase Shift Keying. The problems described above can be solved by the differential phase shift keying. For this practice there will be a phase shift at every bit transition, but the direction of the phase shift will contain information about the encoded bit. If the phase shift is positive (usually +90°), the next bit will be 1, if it is negative (-90° or +270°), the next bit is a 0.

With phase shift keying the transmitter will have to compare the phase shift of two consecutive signal entities to determine the value of one bit.

Figure 5. Phase Shift Keying schemes compared



from [halsall95], page 65, figure 2.21

In the figure above the two phase shift keying schemes are compared. The third row is an example of phase coherent phase shift keying, but it's signal would not be interpretable without knowing the carrier signal in the second row. The fourth row shows an equivalent differential phase shift encoded signal. This signal can only be interpreted by using the decoding rules described above.

There are methods to achieve even higher data rates using phase shift keying, but this requires quantization techniques which we will focus on in the next section.

2.5.4 Quantization

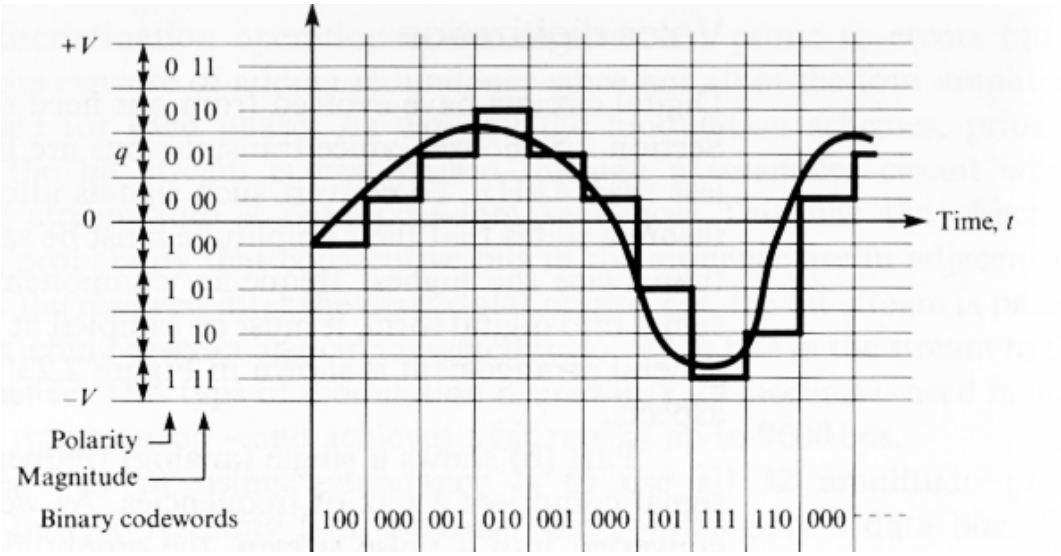
The attempt to overload the data transfer capabilities of a medium is called quantization. With quanization a stream of analog signals is discretized in oder to create digital data. The simplest quantization is intruducing a threshold value and interpreting every value above as 1 and every value below as 0. As more threshold values are introduced the data input stream is scanned at a finer level and more bit values can be tranferred in one time slot.

It can be explained easily with an example. Imagine a carrier medium that allows to send signals with a voltage between 0 and 3 volts. But both transmitter and receiver are only able to create and measure voltages in 1-volt steps.

Quantization means the utilization of the facilities described above. The sender would create 4 distinct signals, one signal for two bits. `00` would be encoded with zero voltage, `01` with on 1 volt, `10` with two volts and `11` with the maximum signal of three volts. By using this technique it is theoretically possible to double the data rate for a given bandwidth.

Theoretically, because a signal with smaller steps between two signaling units is more vulnerable to attenuation and noise.

Figure 6. Quantization of signals



from [halsall95], page 70, figure 2.24

The example above uses four distinct voltage steps as well as positive and negative polarity to encode three bits at one signal time.

Baud-rate and bit-rate

The baud rate specifies the number of signal variations on a medium in a certain span of time. If each bit is represented by one signal level, the baud rate is identical to the bit rate (the number of bits received in a certain time span), but using more sophisticated modulation techniques can lead to higher or lower bit rates at constant baud rate.

For amplitude shift keying and frequency shift keying it is clear how to quantize a bit stream to a given signal. For phase shift keying there is a more sophisticated solution available. The solution is called multilevel modulation and extends the differential phase shift keying approach.

Remember that differential shift keying used two phase shift vales (90° and 270°) to encode 0 and 1. For 4-phase shift keying the encoding scheme is following:

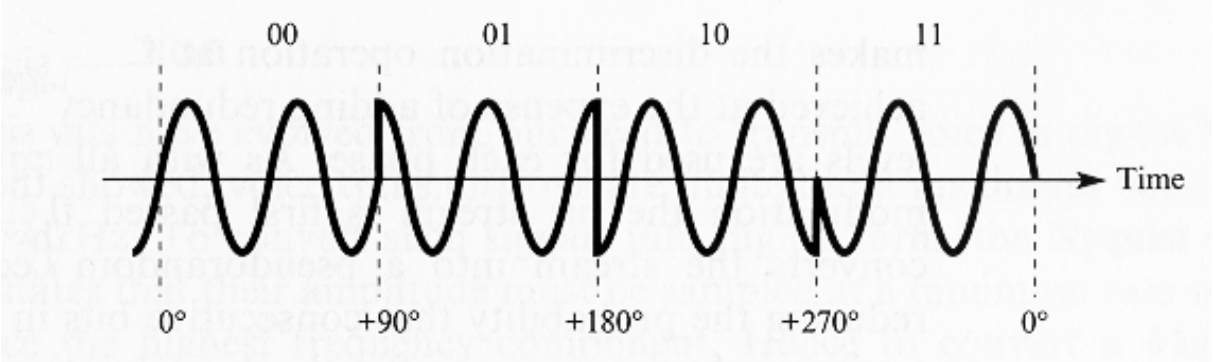
Table 1. 4-Phase Shift Encoding Scheme

Bit value	relative phase shift
00	0°
01	90°
10	180°
11	270°

As you can seen in Table 1, “4-Phase Shift Encoding Scheme” there are four different phase shift values, each a 90° step which adds up to a full 360° phase rotation. There is a very simple assignment from binary value to pahse shift: multiply the binary value with 90° and you will know the resulting phase shift. Because, unlike to the phase-coherent phase shifting, relative, not absolute phase shift values count it is possible to use 0° and 180° as values without running into ambiguities.

A resulting signal with all four possible encoded values will look as follows:

Figure 7. 4-Phase Shift Encoding



from [halsall95], page 67, figure 2.22

Quadrature amplitude modulation. This is a very powerful modulation scheme as it uses 4-PSK (Phase Shift Encoding) together with amplitude modulation and value quantization. For a given frequency amplitude and phase shift are varied, causing 16 different combinations and allowing the transmitter to transmit 4 bits at a signal time.

In the example below there are eight distinct phase shift values, and only every second possible phase-shift/amplitude-shift combination has a value assigned to avoid possible errors due to attenuation and noise.

Table 2. 16-Quadrature Amplitude Modulation

Phase Shift/Amplitude Shift	1	2	3	4
0	-	0001	-	0011
45	0000	-	0010	-
90	-	0101	-	0111
135	0100	-	0110	-
180	-	1001	-	1011
225	1000	-	1010	-
270	-	1101	-	1111
315	1100	-	1110	-

There are extensions of Quadrature amplitude modulation (QAM) that go up to 256-QAM and are often used in high-data-rate cable-modems because there is one transmission medium reserved to one pair of transmitter and receiver, causing very little interference from other communication participants.

2.6 Conclusion

The concepts introduced in this section mark the very basic of computer communications. They are essential for every communication regardless whether it is a wide or low range and whether the transmission medium is based on electrical, optical or electromanetical signals.

The fundament is to manipluate a base signal in a destinct way, that allows the receiver the identificate the received data easily and allow a high certainity when decoding the signal.

3 Asynchronous Data Transmission

Asynchronous data transmission is used whenever character or byte based data has to be transferred in irregular intervals. A common example for this are the signals from a computer keyboard. The user types at irregular intervals and the keyboard encodes each hit into a sequence of bits of a defined length.

This character is transferred to the main unit where the pulse of incoming signals is interpreted as a sequence of bytes which are afterwards interpreted as a certain typed key code or key-combination.

The Problem. How does the receiver of the signal know when a byte starts and when it ends? This can be done by certain bit patterns indicating the start and end of a byte or character.

Before we come to this kind of character-length-synchronization we will cover the different bit encoding schemes. Afterwards we will explain the nature and use of start and stop bits.

Example 2. Application of Asynchronous Data Transmission: UART

The most common application of asynchroonous data transmission are Universal Asynchronmous Receivers/Transmitters (UARTs). It is an computer component that is used in the serial communication subsystem, for example in serial ports or internal modems.

The Universal Asynchronous Transmitter will transmit bytes in a serialized way by sending bit by bit to the receiver. At the receiver side, a Universal Asybnchronous Receiver will receive the single bits and return full bytes.

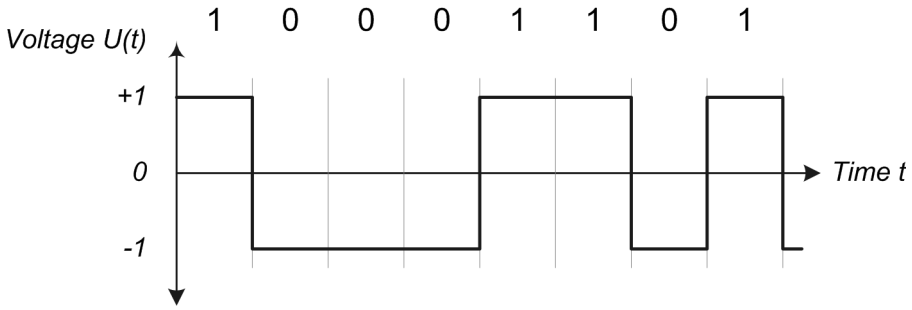
3.1 Return to Zero Signaling

The most obvious variant of bit signaling is called return to zero signaling or pulse signaling. As said before, some media on the physical layer do not only support high and low values but also positive and negative values. A signal may be represented by a positive high, negative high or low value of voltage, amplitude or phase shift. Return to zero signaling essentially means that a logic `1` is represented by a positive high value, a logical `0` is represented by a negative value. After the transmission, a low or no signal is sent, in other words the signal returns to zero.

Example 3. Return to Zero Signaling with different modulations

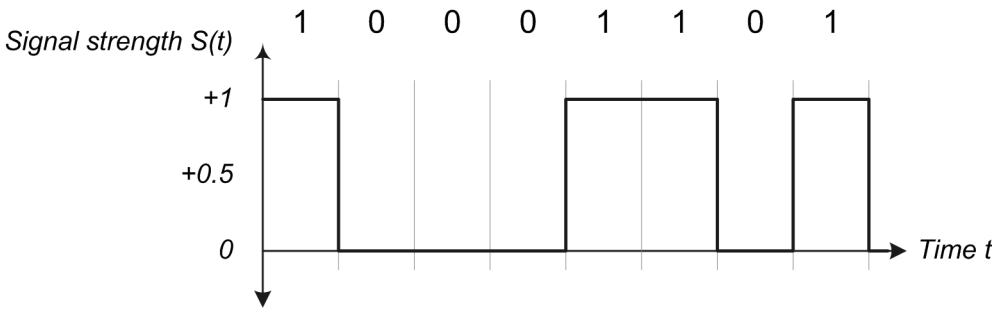
Let there be a byte with the bit sequence of `10110001`. For frequency modulation this means that there will be a frequency of 500 Hz for a high value (logical true) and a frequency of 250 Hz for a low value (logical false). Please note Figure 8, “Return to Zero Encoding (Voltage)” for an illustration of the example.

Figure 8. Return to Zero Encoding (Voltage)



For amplitude shift keying there will be a high voltage of 3 Volts for true and no voltage for false. The figure below will illustrate this example.

Figure 9. Return to Zero Encoding (Abstract)



As return to zero signaling relies on the availability of three signal states it may not be available for every medium and keying scheme.

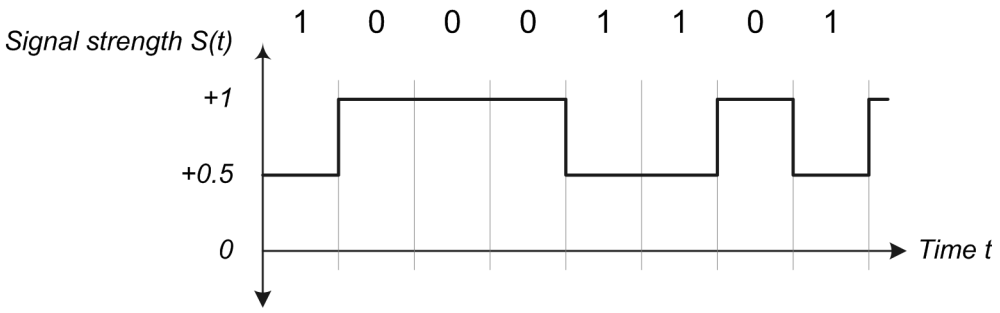
3.2 Non-Return to Zero Signaling

Non-return to zero (NRZ) code means that the signal will not be zero when data transmission is completed.

This requirement is dealt with by defining the following simple encoding scheme: a logical one is encoded as a low value, a logical zero is encoded as a high value.

Example 4. Non-Return to Zero Encoding for Amplitude Shift Modulation

Figure 10. Encoding with NRZ



As you can see in Section 3.2, “Non-Return to Zero Signaling” [20], the signal will not return to line idle, even when only `0`-bits are transmitted. A long series of logical ones will result in a long time span without any transition of amplitude or frequency.

The example above shows the main problem that Return to Zero and Non-Return to Zero encodings share. For certain protocols, where long sequences of zeros or ones are transmitted, there will be long sequences without any change in signal. As the receiver's clocks has no way to synchronize, they will run out of sync and sampling of the incoming signal will lead to wrong results because the receiver will fail to identify the bit boundaries.

Non-Return to Zero encoding and its derivatives should be used only when it can be guaranteed that the higher level protocols will not result in long sequences of identical bits. This behavior can be guaranteed for asynchronous transmission with start and stop bits, but not for synchronous transmission.

This limitations can be overcome by using the Manchester encoding, which we will cover in the next section.

3.3 Character or Byte Synchronization

The basic principle of byte or character synchronization is very simple. The receiver waits for the transmission of a start bit. After that bit has been received, it will wait for a specified number of bits (seven or eight in most cases). Every single received bit will be stored in a shift register until the specified number of bits has been received. In the next step the receiving unit will give a signal to the microprocessor that the register is filled.

The microprocessor will read the shift register in parallel mode and start computation. The emptied shift register can be filled with data from the next character, as soon as a new start bit is detected.

The register used in this process is called Serial-In-Parallel-Out register or SIPO. Its opposite is the Parallel-In-Serial-Out register (PISO) used by the sender.

This basic mode of byte synchronization is only useful if both sender and receiver know how long a character or byte is. If this is not the case, you will need start and stop bits in oder to indicate the beginning and the end of each character.

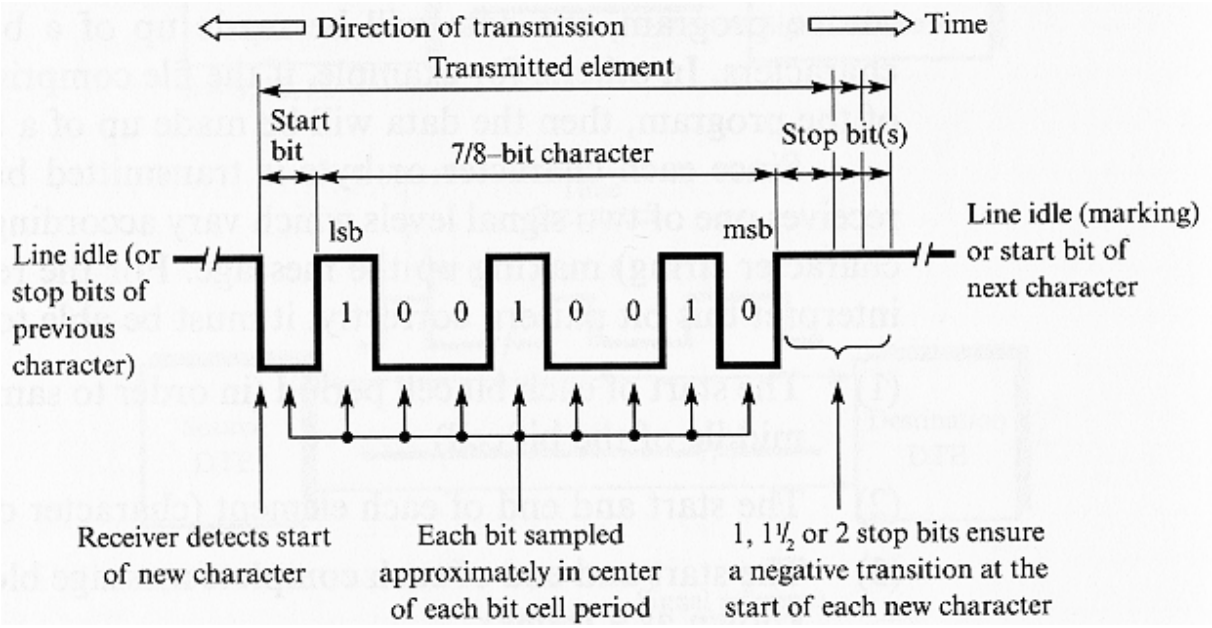
3.4 Start- and Stop Bits

To solve the problem of finding byte beginnings and endings the concept of start- and stop bits. A start bit indicates the beginning of a byte or character. There will be only one start bit for each byte. A stop bit indicates the end of a byte and there may be more than one stop bits for each character.

It is clear that the stop bit must have the opposite value of the start bit, as it will be impossible to identify a start bit after a sequence of stop bits if the start bit has the same value as the stop bit.

For NRZ encoding where line idle is encoded with 1, the start bit has the value 0 and the stop bits thus have the value 1. This variant of encoding adds at least 25% overhead to every byte transmitted, but that does not matter because asynchronous transmission is used mostly for peripheral devices such as keyboards or devices connected to the serial port of the computer.

Figure 11. Schema of start and stop bits usage



From [halsall95], page 102, figure 3.3

The figure above shows a complex start- and stop-bit example. If no data is sent, a positive signal is received. As soon as the signal value drops, a start bit is detected and the recoding of the next bits starts. Every bit is evaluated at the middle of each signal time, whose length can be approximated by measuring the length of the first start bit.

After successful transmission of seven or eight bits a sequence of stop bits will be received. If this sequence is detected, the recorded byte is complete and the SIPO re-

gister can shift the contained values and start processing the received character or byte.

The different values of start and stop bits introduce at least one transition in the bit stream which is important to avoid the problems that result of non-return to zero encoding combined with long sequences of bits of the same value that have been mentioned before.

3.5 Conclusion

Asynchronous Data transmission with start and stop bits is a very simple way to transmit data. “Because asynchronous data is ‘self synchronizing’, if there is no data to transmit, the transmission line can be idle.” [Frank Durda, Serial and UART Tutorial].

The limitation of asynchronous communication is that errors cannot be detected, e.g. when a stop bit is missed. Because of this fact, asynchronous commictaion schemata can be found mostly in a single computer or between a computer and its peripheral modem or in direct terminal connections.

4 Synchronous Data Transmission

According to [halsall95 p. 102] synchronous data transmission is characterized by the fact that the senders and the receivers clock have to be put into synchronism in order to successfully transfer and decode data.

This requirement is fulfilled by using special encoding schemes that are a bit more complex than the NRZ-encoding variants described in the previous section.

There are also ways to keep the clocks of two systems in synchronism, with- or without directly sending the clock signal.

4.1 Manchester-Encoding

The limiting factor at the physical layer is the fact that the clocks used for determining bit boundaries are never equally fast. This will lead for every encoding scheme which may create long sequences of equal signal values to an aberration of the two clock and thus to bit errors when interpreting the signal.

Principle of operation. Manchester Signaling means that every bit is represented by a change of value from high to low or low to high. The change of value happens exactly at the middle of the time span persevered for that bit. This makes determining the start and end of every bit very easy.

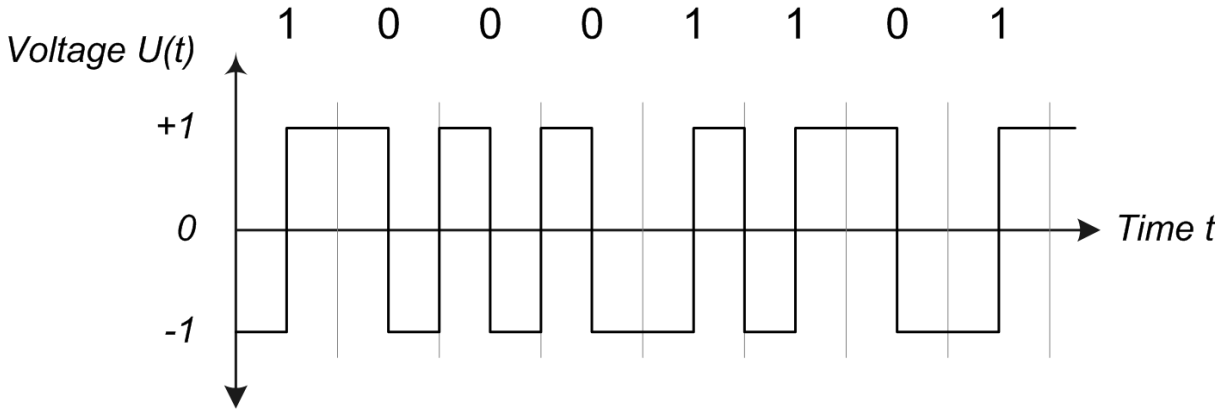
The following encoding convention has been accepted for the Manchester Encoding:

Logic Data	Physical Signal
true	transition from low to high
false	transition from high to low

Example 5. Encoding with Manchester Signaling

In this example the bit sequence 00110001 should be encoded. Figure 12, “Signal Values for Manchester Encoding” will show the signal-value-curve for this example.

Figure 12. Signal Values for Manchester Encoding



If you compare Manchester Code with Non-Return to Zero encoding or variants, you will notice that for a given signaling rate only half of the bit rate can be achieved, as one bit requires two signal elements, one high and one low.

The Manchester Encoding is used in Ethernet networks.

Differential Manchester Encoding. This variant of the Manchester Signaling scheme also uses a transition in the middle of the bit for synchronization, but the representation of each bit depends on the signaling of the previous bit. If a 0-bit should be encoded, there will be a transition at the start of the bit and in the middle of the bit. If a 1-bit is encoded there will be only a transition in the middle of the bit.

Figure 13. Differentiald Manchester Encoding Scheme

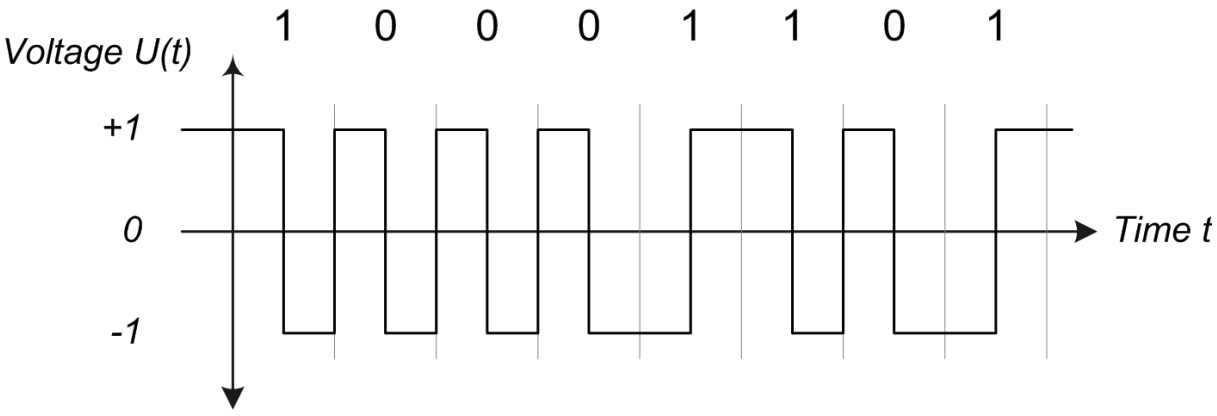


Figure 13, “Differentiald Manchester Encoding Scheme” shows the encoding of the same bits as in Example 5, “Encoding with Manchester Signaling”. The Differential Manchester Encoding variant is used for Token Ring networks.

Table 3. Differential Manchester Encoding Scheme

previous signal	logical `0`	logical `1`
high-low	high-low	low-high
low-high	low-high	high-low

4.2 Bit Synchronization

For synchronous transmission, data is not transferred byte-wise so there are no start- or stop bits indicating the beginning or end of a character. Instead, there is a continuous stream of bits which have to be split up into bytes. Therefore the receiver has to sample the received data in the right instant and the sender's and receiver's clocks have to be kept in a synchronized state.

As the main task lies in synchronizing sender's and receiver's clocks, bit synchronization is also called clock synchronization.

4.2.1 Clock encoding

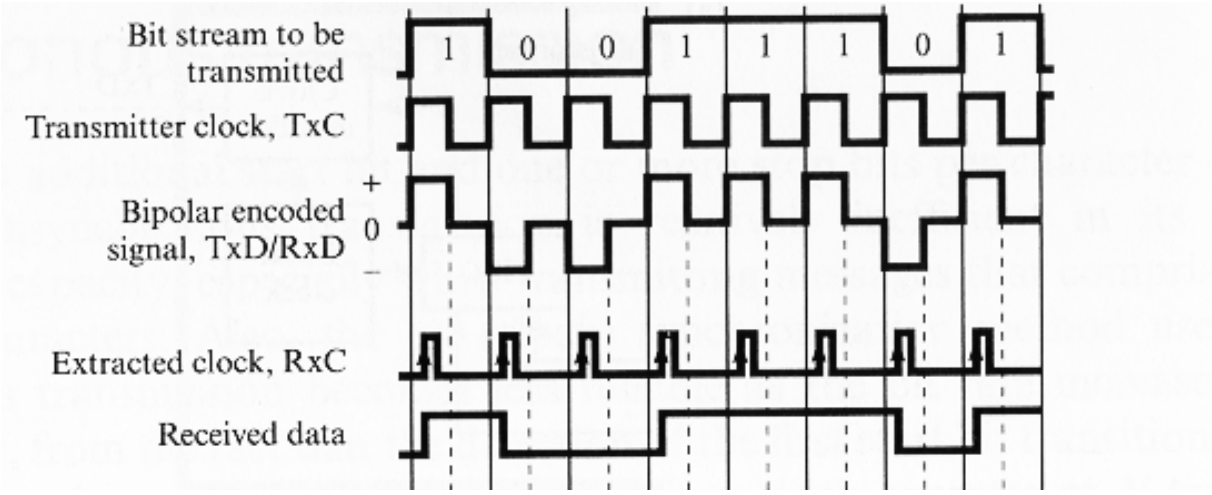
The most self-evident way to accomplish clock synchronization is to send the clock signal to the receiver. This can be done by adding the signal of the local clock to the encoded signal of the bit stream resulting in a bipolar encoded signal which the receiver will have to interpret. By using this bipolar encoding, it is not necessary to create an additional transmission line just for the clock signal.

Each bit span of the bipolar[2] signal is dived in the middle by the signal shift of the clock. There are two possible values for each bit span: high-zero and low-zero, denoting logical one and logical zero. The received signal will contain enough information for the encoder as it can determine the length of a bit by the guaranteed signal change at the end of each bit and it can determine the literal value by distinguishing between a positive or negative signal in the first half of the bit time.

In [halsall95 p. 104] there is a good example, describing bipolar clock encoding:

[2] A bipolar signal is a signal that can have positive and negative polarity. For example a signal on a coaxial cable can be bipolar (positive or negative polarity of the measured voltage), but an optical signal cannot be bipolar as light can only be either on or off.

Figure 14. Bipolar Clock Encoding



From [halsall95], page 104, figure 3.8

As can be seen in Figure 14, “Bipolar Clock Encoding” signal of the transmitter clock is added to the bit stream that should be transmitted, the resulting signal is bipolar and contains a clock signal that can be extracted in order to encoding the bit stream.

This way of clock encoding is also a return to zero encoding, as it requires a medium of capable of carrying bipolar encoded signals. This limitation can be overcome by using a Digital Phase Locked Loop (DPLL).

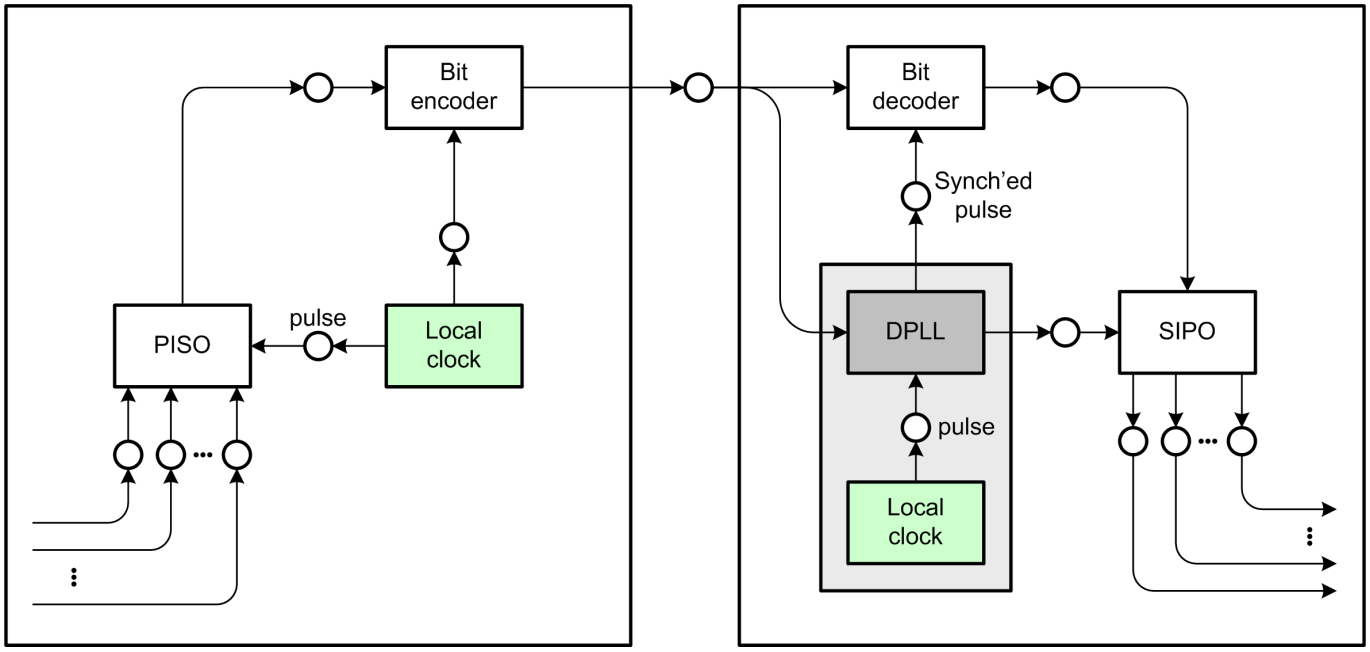
4.2.2 Digital Phase-Locked Loop

The main idea of a digital phase locked loop is that the receiver's clock is reasonably accurate, but should be resynchronized with the sender's clock whenever possible. Unlike the direct clock encoding, there is no direct transmission of the clock signal, but it is possible to extract clock information from the received data signal.

It is important that there are enough bit transitions in the received data stream which indicate bit boundaries and make it possible to deduct the duration of a bit time as well as enabling the clock controller to reset the clock to a less diverged signal. This can be ensured by using a bit scrambler which removes long sequences of zeros or one's, but a more convenient way is to use an encoding scheme which ensures a sufficient number of bit transitions like the Manchester encoding.

Assume a system structure like described in Figure 15, “Digital Phase-Locked Loop Structure (FMC)” where you have a digital clock which will have a sampling frequency that is at least 32 times as high as the bit rate of the incoming signal. This clock feeds its signal to the DPLL which also gets the received bit stream. The bit encoder encodes the bit stream and feeds it to the receiver's shift register where the serial to parallel transformation happens. The clock signal for this shift register comes from the DPLL.

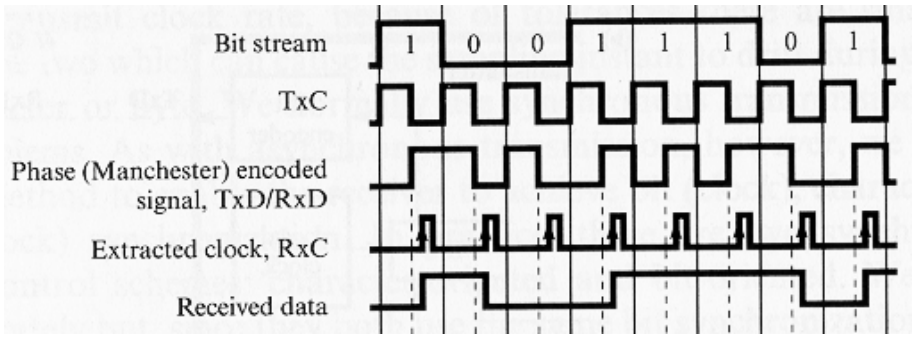
Figure 15. Digital Phase-Locked Loop Structure (FMC)



The digital phase locked loop has two inputs. The first input is the system clock which provides a signal to the DPLL in a well defined frequency. The other input is the received bit stream, where every bit has at least one bit transition from `0` to `1` or from `1` to `0`. This bit transition is either after around 16 clock signals, at the end of a bit time, in which case it will be ignored or after around 32 clock signals, in which it marks the middle of a bit time (for Manchester encoding) and will always occur.

If the clocks are in synchronism, there will be 32 clock signals between two bit middles. The DPLL will feed its next signal after exactly 32 clock signals to the shift register.

Figure 16. Manchester Encoding for encoding a Clock signal



From [halsall95], page 104, figure 3.8

The figure above uses Manchester encoding to encode the transmitter's clock signal in the bit stream without adding additional polarity to the transferred signal. The clock signal can be extracted nonetheless by using a digital phase-locked loop.

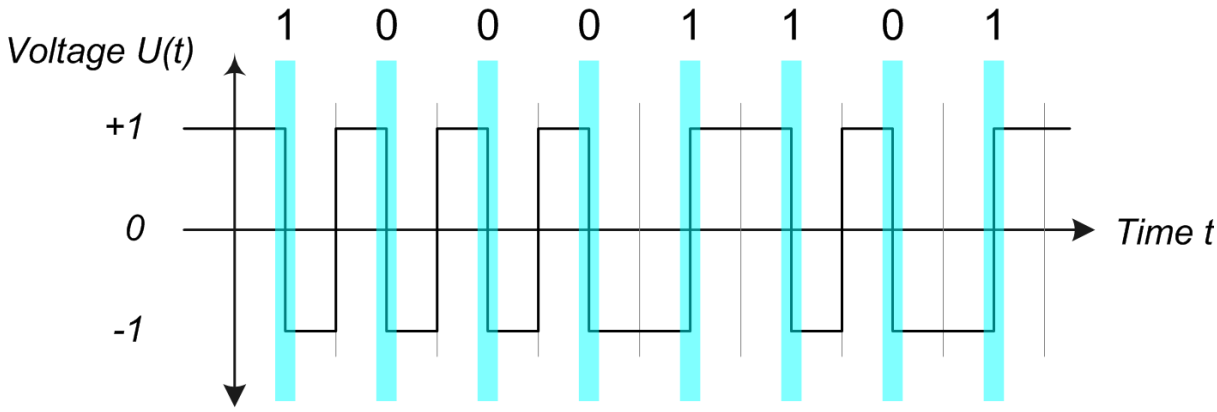
But if there are more than 32 clock signals, say 34, it means that the clock is two clock cycles ahead. In this case the digital phase-locked loop will shorten the span until sending the next signal to the shift register by two clock signals and will feed the next signal after just 30 clock signals to the shift register.

The same procedure applies if there are less than 32 clock signals time span between to bits. The signaling span will be adjusted by waiting two clock cycles longer for giving the next impulse to the shift register, adding the delay to a total of 34 clock signals.

Example 6. Digital Phase-Locked Loop with 8 clock cycles

In this example, there are only eight clock ticks for one bit time. The shaded area in the next diagram marks the time span in which a signal transition from high to low or vice versa is expected. This is the time where the DPLL adjusts the clock signal.

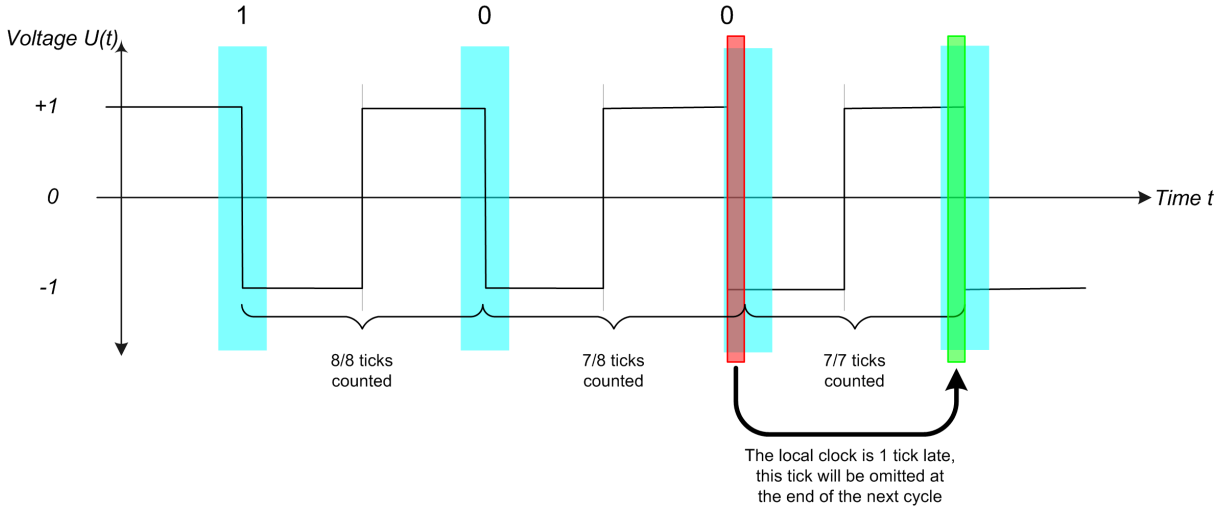
Figure 17. Manchester-encoded signal



In the next diagram we will focus on two bits of the above bit stream. For the first bit the clocks are in synchronism and no adjustment will take place, but for the second bit, the receiver's clock is one tick late, causing it to measure only seven ticks, while eight should have been there.

For the next bit the clock will count only seven ticks (the missing tick will be omitted), which will make the clock going ahead. But going ahead for one time unit and being late for another will sum up to nothing. The clocks are in synchronism again.

Figure 18. Manchester-encoded signal detailed view



A digital phase-locked loop will work with Non-return to zero encodings as well as with Non-return to zero inverted (NRZI) encodings like Manchester code or the differential

Manchester code, but NRZI-encodings require a higher signaling rate for the same bit rate and are therefore used especially in LANs.

4.2.3 Advanced DPLL use

In the prevoius section we have shown the use of the DPLL with a fixed encoding scheme that guarantees a transition for every one or two bits. But the Digital Phase Locked Loop algorithm does not need a transition every one or two bits, because in most cases the time-disperson is much lower. In this case it is enough, if every four or five bits a transition occurs, to resynchronize the clock. This can be achived by adding an encoding scheme that will not create in-bit transitions but will gurantee transitions for a fixed count of bits. One example of a encoding algorithm that will fulfil this requirement is shown in 4B5B encoder, in the final section of this article.

Another possibility is to a some scrambling sequences in oder to get enough transitions. In either case, a short learing sequence between transmitter and receiver is advisable, because otherwise no initial synchronized state could be reached.

The clock would be resynchronized by the DPLL whenever a transition occurs.

4.3 Conclusion

In this section we have shown some more sophisticated encoding schemes that have one major feature: they allow self-synchronizing communication through the use of Digital Phase Locked Loops which will detect time dispersion and correct it.

There are different other possibilities to ensure synchronization by sending a clock signal or by ensuring enough transitions between high and low value in each bit sequence.

5 Frame Synchronization

5.1 Motivation

In modern computer networks data is not transferred as a simple stream of bits or bytes but in terms of frames or packets. This enables amongst other things packet based routing, error correction and the sharing of one physical medium between multiple clients. As the medium usually is a serial link and does not have a concept of frames or separated data units the sender and receiver have to recognize frame borders in the data stream on the medium. This process is called Frame Synchronization.

5.2 Requirements

The correct detection of frames is crucial to the stability of the network. If frame borders are not recognized or the receiver erroneously detects a frame where none was sent, border data corruption is very likely to occur. Therefore frame borders must be detected unambiguously.

Because in general the transfer medium has to be considered unreliable the frame synchronization must be able to recover from error conditions, e.g. after one corrupted frame the algorithm should be able to recover and synchronize to any following frames.

Finally, frame synchronization methods can have a big impact on performance. A badly designed algorithm can significantly increase the amount of overhead per frame transmission. On the other hand complex algorithms might increase processing time of the data and by that require faster processing units or result in a slow down too.

5.3 Methods

5.3.1 Time gap synchronization

The most obvious method for frame synchronization is leaving a time gap between frames. This is of course only possible if an idle state of the transfer medium is recognizable and distinguishable from for example a long row of zeros. If the bits are encoded using return to zero signaling (see Section 3.1, “Return to Zero Signaling”) time gaps cannot be used.

While this is simple and easy to implement it has disadvantages too. To make sure each of the communication partners recognizes the time gap as such it has to be long enough compared to the length of one information cell in asynchronous data transfers. This brings a performance penalty.

Time gap synchronization is usually used in combination with another method of frame synchronization like packet length indication because it might fail if line noise is encountered and by that needs a backup technique (see Section 6.1, “CSMA/CD (Ethernet)” for an example).

5.3.2 Start & End Flags

Frame Synchronization via start and end flags is very widely used. The general idea is to separate the single frames by special data sequences, the flags. These flags are commonly referred to as "STX" which stands for "start-of-text" and "ETX" for "end-of-text". Whenever the receiver encounters a STX flag it knows it has detected the beginning of a new frame whilst ETX signals the end of the current frame. In many cases there is no need to distinguish between the start and the end of a frame. If the receiver is currently receiving a frame only an ETX character can be valid and if it is not receiving a frame only the STX character can be valid. Because of that to avoid any overhead STX and ETX are usually chosen to be the same character. Please see Figure 19, “Frame Synchronization via start & end flags” for a graphical representation of a frame embedded in STX and ETX flags.

Figure 19. Frame Synchronization via start & end flags

If the data transmitted is simple printable text, frame synchronization via Start & End flags is easy. STX and ETX are then chosen to be two unprintable characters and can

be used to unambiguously mark the beginning or the end of a frame respectively. In the ASCII code STX is the character number `0x02` and ETX is `0x03`. But because links on which only printable data may be transferred are rather ineffective as they only use a fraction of the available characters they are seldomly used.

A much more effective style of transmitting data can be achieved by not restricting the usable characters to printable ones. This makes frame synchronization a hard task as for example in a compiled binary program file every character might occur, including those chosen for STX and ETX. If one of the flags occurs within a frame the receiver will believe it had detected a frame border and expect the next data to be a new frame. This erroneous border detection will usually corrupt the data sent in both the frames and result in data loss. To avoid this ambiguity the special flag values have to be removed from the user data before it is sent on the medium. This process is called encoding or stuffing. It is a very common problem in computer science as in nearly all data encoding formats special control codes have to be removed from the user data.

Stuffing algorithms for network transmission describe a method of encoding the user data so that all occurrences of the flag are removed. This encoding has to be 0 so that the receiver can extract the user data after separating the frames. Stuffing algorithms can be classified as bit or as byte oriented algorithms. In the first case the receiver and sender examine the received data in terms of bits. The flag is a special sequence of bit values. This method is usually chosen if the algorithm has to be implemented in hardware. In the latter case both the sender and the receiver talk in terms of bytes. This is useful if the algorithm is implemented in software, because processors usually operate on byte values rather than on bits. Using bits would result in a slow down of the algorithm because the processing capabilities of the computer would not be exploited. Modern processors can process at least complete byte values at one step. Processing one bit at a time would in that case approximately increase the number of steps necessary for processing an amount of data by the factor eight.

Synchronous or Asynchronous? Frame synchronization must be implemented in synchronous and in asynchronous transfer protocols. The transfer mode only has an impact on which algorithm to choose. In asynchronous transmissions the frame synchronization is done after bit and character synchronization. As after character synchronization the data is already represented in bytes only byte oriented algorithms are used on these links.

In the following sections we will exemplarily introduce a bit stuffing algorithm and a byte stuffing algorithm. While there are slightly different algorithms the ones we show are representative and widely used. Additionally, we will introduce the Consistent Overhead Byte Stuffing (COBS) algorithm. COBS is operating on bytes like PPP (Point-to-Point Protocol) but is of special interest because it has a completely different approach to byte stuffing.

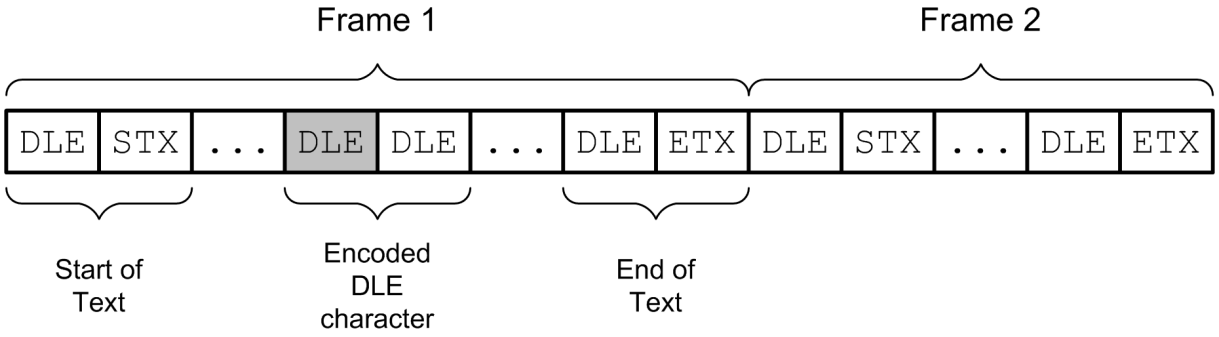
5.3.2.1 Byte stuffing

Byte stuffing is done by introducing a new signal, the Data Link Escape character DLE. To distinguish occurrences of the STX or ETX character in the user data from their special meaning as frame separators the STX and ETX characters are prepended with this character. To denote the frame start the sender inserts the sequence DLE

STX while the end is marked by DLE ETX. Within the user data no transformation is done on the STX and ETX characters, but after each DLE character another DLE is inserted.

By that the receiver can unambiguously detect frame borders. If and only if it encounters a DLE STX sequence it recognizes a frame start. Single STX occurrences are not considered frame borders. This is done equally for ETX characters. If a DLE DLE sequence is received, the second DLE is dropped from the data and the next character is not interpreted as a STX or ETX flag.

Figure 20. Byte stuffing



In Figure 20, “Byte stuffing” you see a typical sequence of frames encoded using byte stuffing. As you can see the frame start is denoted by DLE STX characters while DLE ETX denotes the end. The first frame contains a DLE DLE sequence (with the extra DLE marked gray) which represents an encoded single DLE character. By that the receiver knows that it may not interpret a character after this DLE byte as a special character like STX or ETX. If there were any STX or ETX characters somewhere within the frame it would be ignored because there would not be a DLE character preceding it.

A very common application of this algorithm can be found in the PPP protocol. PPP stands for Point to Point Protocol and is the Internet Standard for transmission of Internet Protocol (IP) packets over serial lines. PPP byte stuffing works a little bit different from the standard algorithm explained before.

As in bit stuffing (see below) PPP byte stuffing uses the bit sequence `01111110` for frame separation for both STX and ETX characters. Contiguously sent frames are separated by only one STX signal. When talking in terms of bytes this is usually written in hexadecimal format as `0x7E`. The value `0x7D` is used as the DLE character. To remove special characters as the STX character from the user data within the frame the DLE character (`0x7D`) is inserted before the character and the character itself is XOR'ed[3] with the value `0x20`. For example, STX or `0x7E` is transformed to `0x7D` 0x5E and the DLE character `0x7D` is transformed to `0x7D 0x5D`.

[3] refers to the binary operation of bitwise exclusive or.

The receiver can detect frame borders simply by listening for the 0x7E STX character. On the data within frames it performs the reverse process. Each detected 0x7D DLE character is dropped and the following character is again XOR'ed with 0x20.

Figure 21. Byte stuffing in PPP

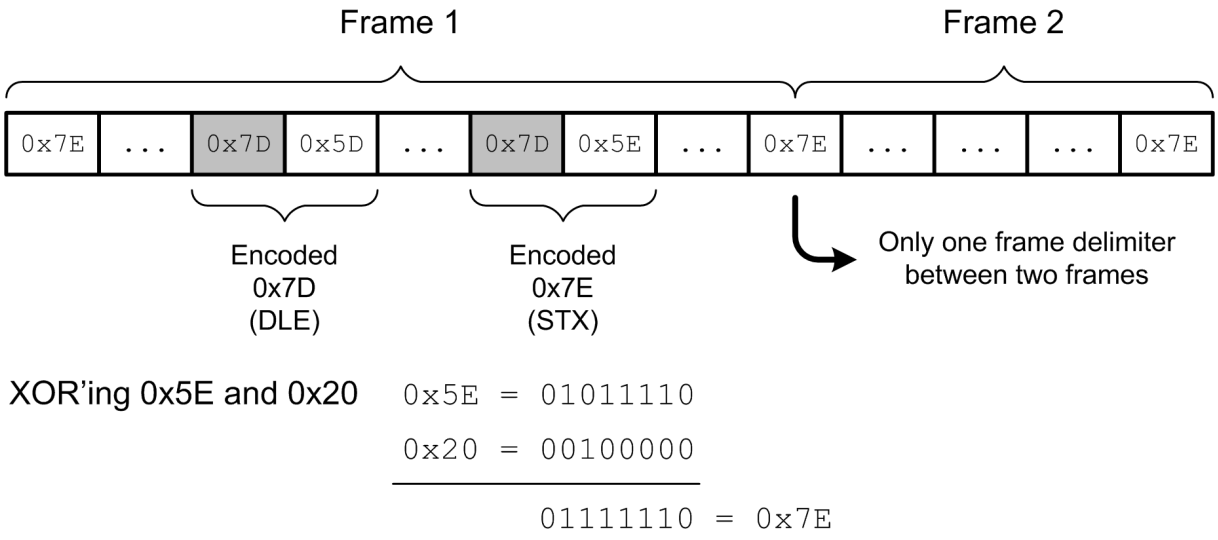


Figure 21, “Byte stuffing in PPP” shows the output of a PPP encoding procedure. As opposed to the common byte stuffing algorithm there is only one flag character used for the separation of frames. The first frame contains two encoded characters, a DLE character (0x7D) and a STX/ETX character (0x7E). The text below shows how the receiver decodes encoded characters using XOR. The character immediately following the DLE character 0x7D is XOR'ed bitwise with the value 0x20. If one writes the single bits of the characters in two rows above each other, the resulting bit value in that row is zero if both values are equal, e.g. bit-0 and bit-0 or bit-1 and bit-1, or one otherwise.

This method has the advantage of a reduced overhead if there are few special characters within the frame though the design goal was to enable the escaping of additional characters. In the PPP scheme adding additional characters is as easy as just prepending them with the DLE character and XOR'ing them with 0x20. This encoding is completely transparent and does not have to be known in advance by the receiver, as it simply XOR's all data after the DLE character with 0x20.

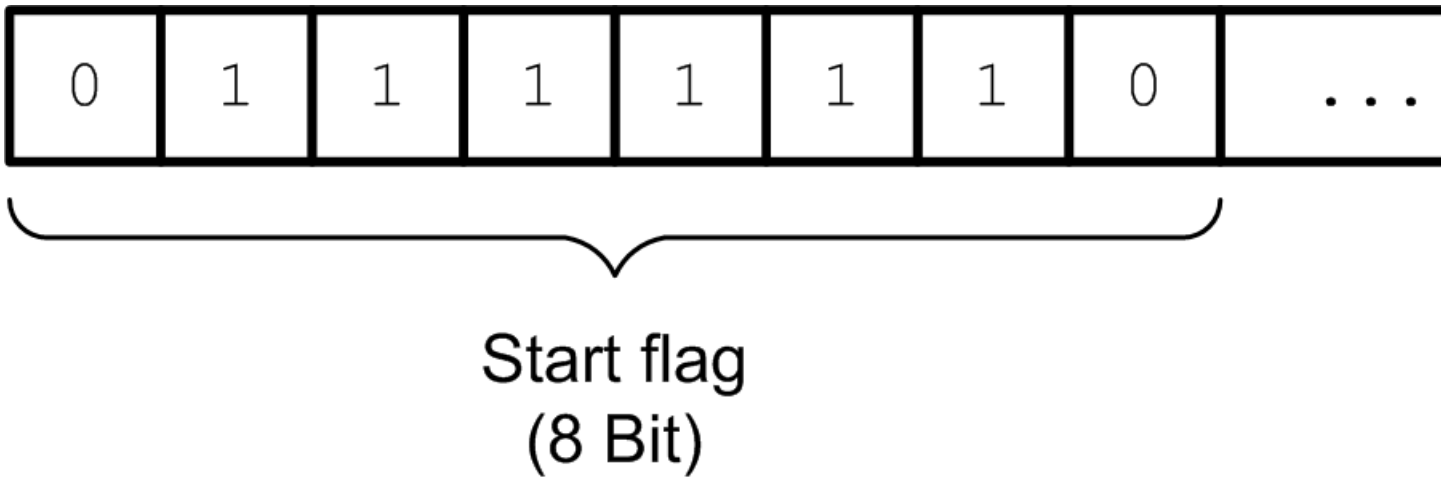
While the XOR'ing encomplicates the process of encoding its benefit is that the encoded characters are completely removed from the user data. By that the receiver can simply listen for special signals like STX/ETX (or other encoded signals) without having to remember the preceding signal as in common byte stuffing.

5.3.2.2 Bit stuffing

In bit stuffing the bit sequence 01111110 is being used as the single character for frame separation. As mentioned before, the STX signal does not have to differ from the ETX signal. The encoding algorithms used to remove the STX code from the user data is rather simple.

To remove the frame separation value the sender checks for five bit-1 values in a row and inserts a bit-0 value immediately afterwards if it encounters one as illustrated in Figure 22, “Bit stuffing”. This is done even if the next value is a bit-0 which would not result in an error as five bit-1's in a row are completely legal. This is done to simplify the process. If the value `0111110` (followed by either a bit-0 or a bit-1) would be left untouched, it would get indistinguishable from an encoded `01111110` and also had to be encoded in some way. This is because `01111110` is encoded as `011111010` which might occur in the user data if not masked. Inserting a zero after every five bit-1's prevents this as the row `011111010` is encoded as `0111110010`.

Figure 22. Bit stuffing



To decode the user data the receiver checks the data stream for five bit-1 values in a row. If the following bit is a zero, it has detected an encoded row of five bits. It then simply drops the bit-0 from the bit stream and carries on. If the following bit value is a one (`0111111_`), it has detected a special code and further processing is needed. If the next bit is a zero it's the frame separator (`01111110`).

This algorithm has the effect that not only the STX signal is transparently encoded but all signals consisting of more than or exactly six bit-1's in a row. This characteristic is used by giving all these characters which are `01111110`, `01111111` and `11111111` a special meaning.

The value `01111111` is used as the idle signal which is sent between frames to enable the receiver to keep its clock synchronized to the senders via a DPLL as described above. This is possible because this SYN (for “synchronize”) character contains a transition from bit-0 to bit-1. The value `11111111` is used as the *frame abort* signal instructing the receiver to drop the current frame.

A very common application for a bit stuffing algorithm is the High Level Data Link Control (HDLC) protocol. It is generally used for binary transfers and has such different areas of application as satellite links and ISDN (Integrated Services Digital Network) digital phone communications. Please note that again HDLC is a little different from the algorithm above - it uses the value `11111111` as the idle signal rather than `01111111`. While this means that a receiver can't synchronize it's clock while the idle signal is sent

it doesn't have any impact on HDLC. This is because in HDLC there is no idle signal sent between frames but a row of empty frames (e.g. only `01111110`'s without any content in between).

5.3.2.3 COBS - Consistent Overhead Byte Stuffing

COBS addresses a problem that occurs when data is not directly sent via a physical medium but on top of another protocol. If the lower protocol uses a concept of frames too, the frames used by the higher protocol must fit into those of the lower one. Otherwise the frames or packets would have to be fragmented and a receiver had to put them together again. This is a difficult task if the packets have to be routed to different hosts and might not be received in the correct order or at least not directly after each other. Also, if parts of packets would get corrupted their other parts would have to be dropped too.

“As well as the global overheads such as endpoint delay, fragmentation generates overheads for other components of the system - notably the intermediate gateways.”*bennet p. 21-36*

In order to make the frames fit into their lower level containers the size of the frames has to be limited. This gets critical as the resulting size of the packet might vary depending on its contents. Because reserved characters have to be escaped by stuffing algorithms they increase the resulting size of the packet. To exclude the possibility of a non fitting packet its size must be limited so it cannot burst the lower level packet.

The bit stuffing and byte stuffing algorithms described above have been designed to achieve a very small average overhead. It should be below 1% on typical data. But when it comes to worst case overhead they both have large overheads. The byte stuffing algorithm might increase the packets to 200% of their original size if all bytes have to be escaped using two bytes, bit stuffing results in a maximum overhead of 20% if the data exclusively consists of bit-1 values because for every five bit-1's a bit-0 has to be added (both without counting additional packet headers). This is very unlikely to happen, for example in a 1500 Byte packet the statistical probability of all bytes being one special character provided that the character values are distributed equally[4] is (1/256)^1500 which is pretty close to zero.

While the probability of getting packets with for example more than 50% overhead generating special characters is higher it is still very unlikely. But even if these cases are very rare they have to be addressed. If a packet supersedes the maximum size it will usually be dropped from the network. If the sender does not know that, it will try to send the packet again. This will probably result in a deadlock situation or at least a breakdown of network communication. But even if this unlikely case is considered an acceptable risk, a possible attacker could exploit this weakness and use it for a denial of service attack on the network.

[4] While real world data is not distributed equally the difference does not really matter because the protocol designers of course wanted to keep real world average overhead low and chose frame seperation characters which occur even more seldom than the rest of the data. Additionaly a lot of newer applications use compression algorithms to minimize the amount of data to be transferred which create a nearly equally distributed byte stream.

The common solution of these problems is to limit the packet size of the higher level protocol to a certain value which makes it impossible to burst the lower level packet size. In the case of byte stuffing this would mean setting it to 50% of the available size. While this is a solution, it brings a serious reduction of performance. In nearly all known protocols a frame does not only consist of the frame separators and the user data within but also has a header with meta information like address fields or checksums. These headers have a fixed size regardless of the size of the packet so if the packets sent are smaller than they could it results in more overhead. Additionally common protocols like the Transmission Control Protocol (TCP) expect a confirmation to be sent on reception of a certain number of frames. These and other influences sum up and result in reduced performance as shown in [cheshire96]. The measurements show a 33% performance loss after a reduction of the packet size by 50%.

To avoid this COBS does not escape illegal characters but in a certain way escapes the absence of illegal characters. Within COBS a special frame separation character is chosen like in the other protocols, for example the null byte `0x00`. The data to be transmitted is then broken up into smaller blocks by cutting the data at each occurrence of the special character so that the zero byte is at the end of the block. The data block *without the trailing zero* is then prepended by a byte containing the size of the block minus one. If the block is longer than or exactly 254 characters long it is prepended by the code `0xFF` and broken up after the 254th character creating a new block. This leads to the following meanings for the prepending bytes if found in the data *after* encoding:

Table 4. Meanings of code values in COBS[5]

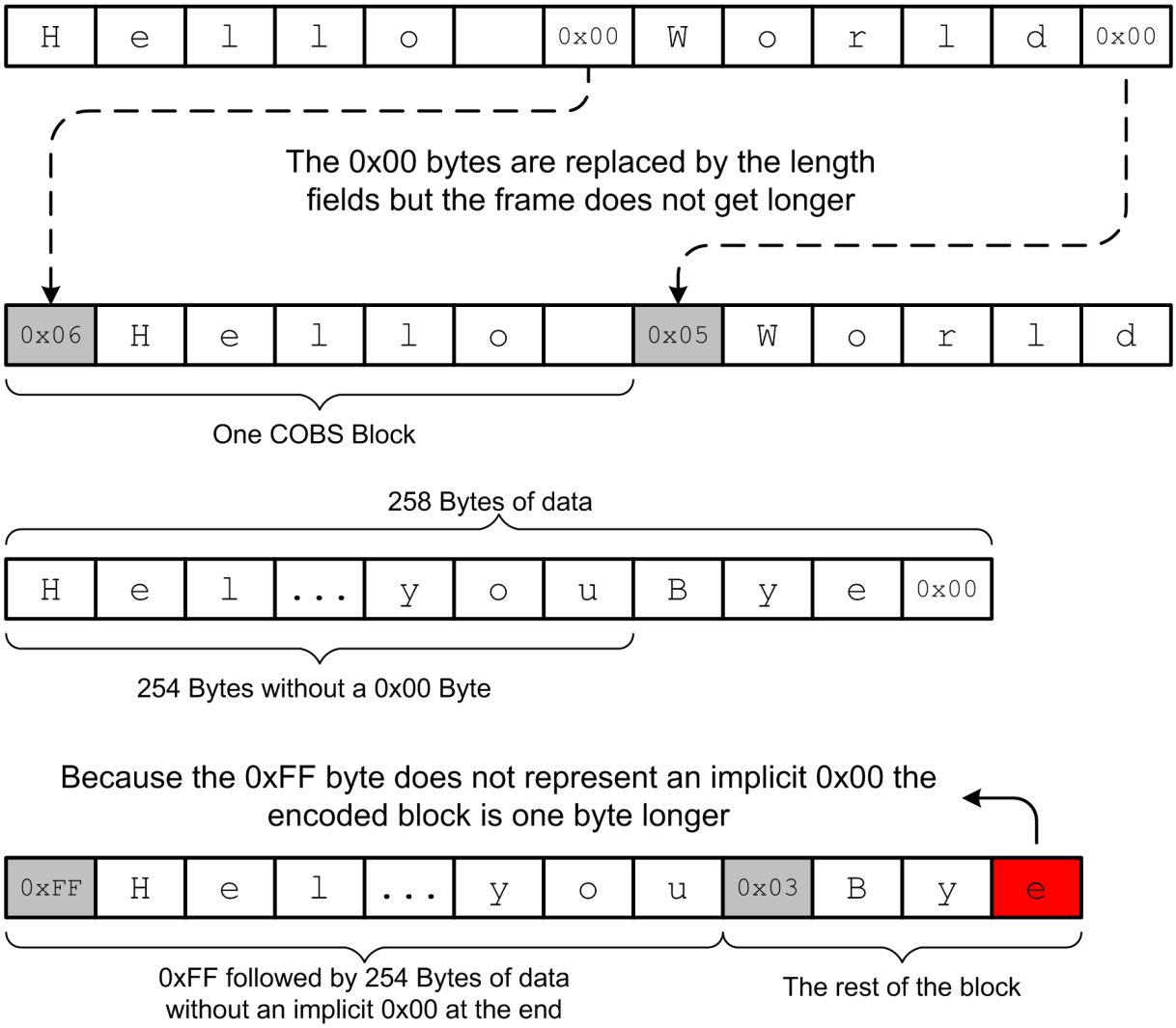
Code	Followed by	Meaning
`0x00`	(not applicable)	(not allowed)
`0x01`	no data bytes	A single zero byte
n	(n-1) data bytes	The (n-1) data bytes, followed by a single zero
`0xFF`	254 data bytes	The 254 data bytes, not followed by a zero

The receiver can then check the first byte of each block, read the next number of bytes indicated by the first byte (it's value minus 1) and append a zero after them if the byte value is less then `0xFF`. If it is `0xFF` no zero is appended.

Note that the actual implementation of a COBS algorithm would look different from the method described because first transforming the data to another representation by cutting it into blocks and afterwards putting them together again would not be very efficient. An algorithm would rather scan through the data and write the output directly. Figure 23, “COBS encoding” shouldd further illustrate this encoding scheme.

[5] from [cheshire99 p. 3]

Figure 23. COBS encoding



COBS byte stuffing incurs no overhead at all if the special character is encountered after at least every 253 Bytes. This is because every code block of less than 253 Bytes and a following byte `0x00` is encoded by a block of the number of bytes without the zero byte. This block is prepended by the length indication byte which. This way the block data stays untouched and the zero byte at the end is replaced by the length byte at the start which does not bring any overhead. If a block longer than or exactly 254 Bytes long does not contain a zero byte it has to be encoded using the `0xFF` length byte which means that the length byte is followed by 254 data bytes after which no zero follows. In this case the length byte is added to the data without removing a zero byte which leads to exactly one byte overhead for the 254 byte user data which is about 0,4 percent overhead. This is why COBS performs better if the special character is more frequent within the data.

The worst case size of a COBS encoded frame is by that exactly 100.4 percent of its original size which is hardly worse than the *average* efficiency of the byte or bit stuffing algorithms shown above. Using COBS in the example mentioned above would

enable us to set the frame size of the upper protocol to about 99.6 % which would not bring a measurable performance penalty.

As COBS is a very new technology it is currently not used in any network system. It has a big potential to be used in mobile technologies as networks like the Global System for Mobile communication (GSM) or the Universal Mobile Telecommunications System (UMTS) mobile phone systems have a basic transport system which is aware of frames and on top of which applications using other protocols, especially TCP/IP, are expected to be used.

5.3.3 Packet Length Indication

The method of packet length indication is used in Local Area Networks, for example in Ethernet. When one stations wants to send a frame, it starts with a preamble to allow the other stations within the LAN to synchronize onto its clock. Afterwards it sends a start-of-frame signal as with the start & end flag algorithms. Within the frame the first data sent is additional information. The frame starts with a header of a known fixed size containing information like the receivers address etc.; after that a fixed size field containing the length of the packet is sent before the frame data is transmitted. After the receiver has detected a start-of-frame and read the packet header it just counts the bytes from there on.

With this algorithm no stuffing is needed as the receiver knows when the packet ends and cannot be disturbed by additional start-of-frame flags within the frame. This makes the method more efficient compared to the start & end flag methods explained above as only a number of bytes relative to the maximum length of one packet needs to be used. This means because of the binary encoding of numbers that the technique scales logarithmic to the length of the frame. If the packet may be up to *n* elements (read: bytes) long only $\log_2(n)$ data units (read: bits) are needed for the length field. This means for example for Ethernet with a maximum frame length of 1500 bytes $\log_2(1500)$ [bit] = 10,550746785 [bit] which is be rounded up to 11 bits which is rounded up to 2 bytes.

5.3.4 Coding Violations

In certain encoding schemes some signals that could technically be sent are disallowed by the algorithm, e.g. with amnchester encoding (see below). Sending such an illegal signal is called a *coding violation* as it violates the rules of the encoding scheme. While this is usually an error condition it may be used as a frame delimiter signal. This is because the special signal will not occur in the user data and by that the delimiter can be detected unambiguously.

For example in Manchester encoding the sender encodes all bit values in two bits sent on the medium. This is done to enable bit-synchronization as described above. Valid bit-encodings always contain a bit-transition, either from bit-0 to bit-1 (which makes an encoded bit-1) or from bit-1 to bit-0 (which makes a 0). In that case the sender can denote a frame border by sending a special character containing coding violations like bit-0 bit-0 or bit-1 bit-1. The special codes are usually called "J" for a signal staying at the previous level or "K" for a signal changing to the opposite level. The characters

usually used with Manchester encoding (see Section 4.1, “Manchester-Encoding”) are "`JK0JK000`" for frame start and "`JK1JK111`" for the frame end.

The use of coding violations does not imply any overhead. While the manchester encoding uses half of the theoretically available bandwidth to keep the clocks of the sender and the receiver in synchronism this method of frame synchronization takes a small part of this somehow wasted bandwidth to indicate frame borders. This does not reduce the reliability of the data link because the clocks of the receiver and the sender should be accurate enough to stay synchronized for the very short period of time in which the frame delimiter is sent.

5.4 Conclusion

While frame synchronization is a rather trivial task it holds some difficulties as shown in this chapter. The algorithms described in this chapter represent all commonly used techniques for frame synchronization. While the real implementation might be slightly different most if not all networking technologies make use of one or multiple methods of the four techniques presented above. As you will see in Section 6, “Real World Applications” it is very common to use more than one method to seperate frames.

6 Real World Applications

While we have shown several techniques in the precedent sections they are all somewhat theoretical approaches to bit and frame synchronization. As always the real world implementations of these techniques are a little different. Also, as synchronization is crucial to any network most if not all of the real world applications do not only rely on one technique but rather use a combination of several approaches.

To give an example of these combinations we will explain the different approaches taken in the Institute of Electrical and Electronics Engineers (IEEE) standard 802.3 (see *IEEE Std 802.3*), better known as Ethernet or Carrier Sense Multiple Access/Collision Detection (CSMA/CD), and FDDI.

FDDI stands for Fiber Distributed Data Interface and is a set of standards describing high-speed computer network infrastructure based on optical transmission.

6.1 CSMA/CD (Ethernet)

The Carrier Sense Multiple Access/Collision Detection protocol also known as Ethernet was developed at the Xerox PARC in the first half of the 1970s. It has become the by far most wide spread technology for local area networking in the 1990s and has nearly completely replaced all other networking techniques, especially in personal computers. While the actual networking protocol is out of the scope of this document we want to show how bit and frame synchronization are implemented in Ethernet.

6.1.1 Bit synchronization

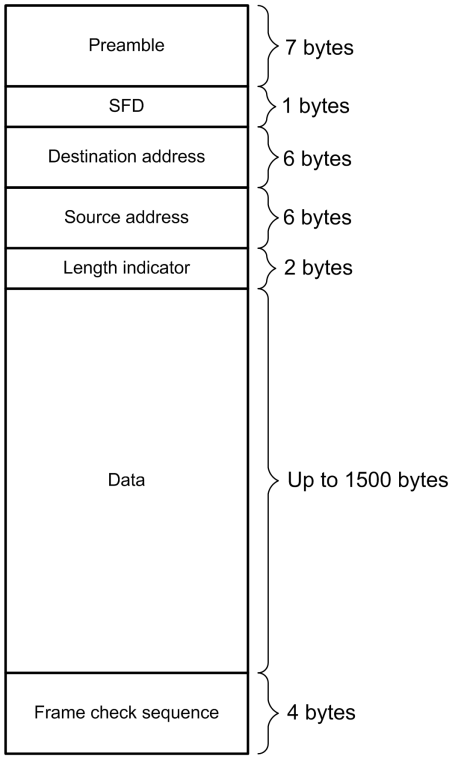
Ethernet uses synchronous transfer with Manchester encoding (see Section 4.1, “Manchester-Encoding”). Every frame sent has to start with a sequence called the *preamble*. It consists of 7 octets with the value `10101010` which allows the receiver to

initially synchronize its clock with the one of the sender. Because this initial synchronization might get lost as the frame is transmitted a DPLL (see Section 4.2.2, “Digital Phase-Locked Loop”) is used to keep the receiver and the transmitter synchronized.

6.1.2 Frame synchronization

Ethernet uses a mixture of time gap and length indication for frame synchronization. Figure 24, “Ethernet frame” shows the structure of an Ethernet frame. Each frame starts with the already mentioned preamble which is followed by the start-of-frame delimiter SFD which is binary `10101011`. It is followed by a destionation address and a source address both of which are 6 bytes long and a length field of 2 bytes indicating how many bytes will be within the following data field. After this the actual frame data is sent which may not be longer than 1500 bytes. Finally a four byte frame check sequence containing a Cycling Redundancy Check (CRC) value of the data is sent.

Figure 24. Ethernet frame



After the sender starts with the preamble all receivers[6] synchronize to its clock while scanning the data for the SFD signal. When a receiver encounters the signal it starts recording the data sent on the medium. After it has received the next 14 bytes which make the header of te frame it reads the last 2 bytes to determine the amount of data that will be sent in the data field. By that it knows that it has to expect the end of the frame after the specified number of bytes plus four for the CRC field.

[6]Ethernet is a bus networking system where usually all Data Terminal Equipments (DTEs) receive all packets sent. The exception to this is Ethernet switching.

Apart from that, Ethernet also uses a time gap to separate frames as the CSMA/CD protocol involves waiting between frames for medium access control anyway. A common value for this time gap is 9.6 [µs] in a 10 [Mbps] Ethernet. So if a DTE does not know when the next frame border is about to happen, for example because it is new to the network and did not read the last length field or somehow lost synchronization, it can just listen for data on the medium and if it encounters a pause of at least 9.6 [µs] it can be sure to have detected a frame border.

Because of this an erroneous detection of a frame border is literally impossible.

6.1.3 Error Detection

Ethernet uses a mechanism for Collision Detection which ensures that the sender recognises packets that have collided and by that can retransfer them. But even if it is ensured that collided packets are retransfered the problem that single packets might be corrupted is still present. If line noise is added to the signal transferred on the medium single bits might get corrupted and a receiver might erroneously recognize a bit-1 instead of a bit-0 or vice versa.

User data. Within the user data errors should usually be detected by the CRC checksum contained in the last four bytes of each frame. The probability that this checksum failes is only approximately 1 of 4.3 billion cases for a 32 bit checksum. If this happens the corrupted data will be given to the upper layers in the ISO/OSI model. If there is no upper layer which incorporates an error correction itself like for example TCP/IP the corrupted data will be forwarded to the application layer and might result in a major data loss or system crash. However as this is very unlikely it is considered an acceptable risk. Implementing a higher level of data security would cause a higher amount of overhead and slow down the overall performance of the network.

Address fields. However this checksum is only calculated over the user data within the frame. If one of the header fields is corrupted the receiver does not have a possibility to detect that. In case of a corrupted receiver address the packet will either not be received by anyone - which is the likely case as erroneously picking the address of an existing DTE within 6 bytes of address data is very unlikely - or be erroneously received by a DTE and probably ignored on a higher level.

Length field. If data within the length field is corrupted there are two possibilities. Either the receiver erroneously cuts off a part of the frame or it expects more data to come after the end of the frame. In each case only the user data and the checksum field will be affected which should be detected the way described above.

6.2 FDDI - Fiber Distributed Data Interface

The Fiber Distributed Data Interface is a set of standards that were originally developed by the American National Standards Institute (ANSI) and which describe an optical fiber based network architecture that operates at bit rates of 100 Mbps.

Just like Token ring architectures it uses two rings and each ring has a token. Both tokens are handed over in opposite directions. This will enhance reliability. FDDI is used to create backbone networks as up to 500 DTEs can take part in communication and the total length of the ring can be up to 100 kilometers.

Between two DTEs there is a multimode optical fiber cable, but there have been different enhancements and adaptions to FDDI, making it work with simple copper cables too. We will concetrate in this section on the original specification and intents of FDDI.

6.2.1 Bit synchronization

Unlike primitive token ring networks there is no single ring monitor in FDDI networks that provides a common encoded clock signal for all participants. Instead each ring interface has its own clock. The signal of the system clock is included in the transmitted signal, to ensure bit synchronization over the network and to guarantee that the received data has enough value transitions to ensure bit synchronization.

There cannot be a transition in the middle of each bit, because in that case a very high signaling rate would be needed. Instead, a 4 of 5 group code is used to ensure that a transition occurs at least every two bit cell periods. In fact there is a two-step transition of the bit stream. First 4 bit groups are converted into 5 bit groups, and later on the five bit groups are encoded to ensure enough signal transitions.

4B5B encoder. The 4 of 5 group code will be created by a 4B5B (4 bit to 5 bit) encoder. This encoder will take a 4 bit sequence, say `0111` and create a corresponding 5 bit sequence. There are many possible encoding schemes cogitable, but the most important fact is that all sequences of zeros longer than three should be elliminated. The following example will ilustrate such an encoding scheme.

Example 7. 4B5B encoder

4 bit to 5 bit translation rules I

4 bit data group	5 bit data group
0000	11110
0001	01001
0010	10100
0011	10101
0100	01010
0101	01011
0110	01110
0111	01111
1000	10010
1001	10011
1010	10110
1011	10111
1100	11010
1101	11011
1110	11100
1111	11101

Further encoding. In the next step every 1-bit is encoded as a signal transition and every 0-bit as a signal without transition. The 4B5B encoder created no sequence

of more than two consecutive zeros, so there cannot be more than three identical signals in a row. The following list shows a possible encoding scheme, if you assume that the first signal of every sequence is a 0 or a 1.

4 bit to 5 bit translation rules II

5 bit data group	signal	alternative signal
11110	(0)10100	(1)01011
01001	(0)01110	(1)10001
10100	(0)11011	(1)00100
10101	(0)11001	(1)00110
01010	(0)01100	(1)10011
01011	(0)01101	(1)10010
01110	(0)01011	(1)10100
01111	(0)01010	(1)10101
10010	(0)11100	(1)00011
10011	(0)11101	(1)00010
10110	(0)11011	(1)00100
10111	(0)11010	(1)00101
11010	(0)10011	(1)01100
11011	(0)10010	(1)01101
11100	(0)10111	(1)01000
11101	(0)10110	(1)01001

Please remember that the leading zero does not belong to the encoded signal, and if it would be a 1 instead of the 0, a completely different signal would be the result.

6.2.2 Frame synchronization

As you may have noted, a five-bit encoding scheme offers 32 different encodings, but only 16 have been used so far. That means there are some 5-bit words left for encoding status information and sending control symbols. These control symbols are:

IDLE	11111
J	11000
K	10001
T	01101
R	00111
S	11001
QUIET	00000
HALT	00100

There are two types of frames that are transferred in FDDI networks: Information frames and tokens. A token consists of a preamble, a two symbol long start delimiter the frame control block (2 symbols) and the end delimiter (1 or two symbols).

An information frame starts with a preamble, a start delimiter, frame control block, the destination address, the source address, the actual information and a frame check squence (8 symbols) as well as an end delimiter and three symbols of frame status.

“The preamble field is comprised of 16 or more IDLE symbols [...]. The start delimiter field consists of two control symbols (J and K) which enable the receiver to interpret the following frame contents on the correct symbol boundaries. [...] The end delimiter field contains one or two clock symbols.” [halsal95 p. 380]

6.3 Conclusion

This section has shown that the concepts introduced before find their application in real-world computer networks. Partly they can be used as described, partly - like in the FDDI standard - they have to be adapted to the real application needs.

The current technologies for bit and frame synchronization are rather exploited - they provide a reliable data transmission at a very small overhead. Used together with additional techniques as in Ethernet they almost completely exclude an undetected error.

7 Conclusion

Bit and frame synchronization have been one of the first problems that had to be solved when computer networks were first developed. The techniques we presented in this paper are all at least ten years old or are derivatives of older techniques. They are theoretically well fundamented and have been in use for decades. There are reference implementations available for each process either as circuit designs or C programs.

While one would expect that there is hardly anything unexplored in this area of communication technology as new applications are being developed new questions arise bringing the need for new answers like the one COBS gives.

Glossary

4B5B	four bit to five bit encoder.
ANSI	American National Standards Institute.
ASCII	American Standard Code for Information Interchange. The de-facto standard to encode digits, letters and other characters in an alphabet of 128 or 256 elements with seven or eight bits.
COBS	Consistent Overhead Byte Stuffing. A byte stuffing technique resulting in a defined and relatively small overhead.
CRC	Cyclic Redundancy Check
CSMA/CD	The Carrier Sense Multiple Access / Collision Detection protocol better known as Ethernet. Standardized by the IEEE in *IEEE Standard 802*.

DLE	The data link escape character. A character that is used as a meta character within a transmission. It is used to mark the following character as having a special meaning. If two DLEs follow each other one of them is ignored (dropped from the data) and the next character is not considered special.
DPLL	Digital Phase-Locked Loop.
DTE	Data Terminal Equipment.
ETX	The end-of-text flag used in frame synchronization as the symbol denoting the end of a frame. Originally ETX was only a non printable symbol within the ASCII code but today it is also used as a general term for a start of frame character.
FDDI	Fiber Distributed Data Interface.
GSM	The Global System for Mobile communication. A digital system for mobile phone communication.
HDLC	The High Level Data Link Control protocol. A data link control protocol residing on layer 2 of the ISO-OSI model, standardized in ISO 3309 and ISO 4335 as an enhanced and generalized version of IBM™s SDLC protocol.
IEEE	Institute of Electrics and Electronical Engineers.
IP	The Internet Protocol. IP is a shorthandle for TCP/IP which is part of the TCP protocol suite. The standard protocol of the internet.
ISDN	The Integrated Services Digital Network. A digital telephone communication network designed to replace analog telephony.
ISO	International Organization for Standardization.
Kbps	Kilo (thousand) bit per second.
LAN	Local Area Network.
Mbps	Mega (million) bits per seconds.
NRZ	Non-return to zero.
NRZI	Non-return to zero inverted.
OOK	On-Off-Keying. See Also RZE.
OSI	Open Systems Interconnection (network model).
packet	There are communication protocols which transfer data in form of packets to increase reliability, error control and flow control. This packet based communication is synchronous. Packet-based protocols enable multiple participants to transfer data over one single medium.
PISO	Parallel Out, Serial In shift register.
PPP	Point to Point Protocol. The internet standard for transmission of IP packets over a serial line.

PSK	Phase Shift Keying.
QAM	Quadrature Amplitude Modulation.
RZE	Return to Zero encoding. An encoding scheme in which a zero value stands for `0` and any higher value for `1`.
SIPO	Serial In, Parallel Out shift register.
SFD	Sort of frame delimiter.
STX	The start-of-text flag used in frame synchronization as the symbol denoting the start of a frame. Originally STX was only a non printable symbol within the ASCII code but today it is also used as a general term for a start of frame character.
SYN	A character sent in digital transmissions to enable receiving DTEs to synchronize their clocks to the senders clock. The bit sequence usually contains a change of value as this is needed for synchronization with a DPLL.
TCP	The Transmission Control Protocol. A suite of protocols used for data transmission in the Internet.
UART	Universal Asynchronous Receiver/Transmitter
UMTS	The Universal Mobile Telecommunications System. A digital telecommunication system for mobile phones and smart devices.
WAN	Wide Area Network
XOR	The binary boolean operation eXclusive OR. XOR returns logical true or bit-1 if and only if exactly one of the operands is logical true respectively bit-1.

Bibliography

[halsall95] Fred Halsall. *Data Communications, Computer Networks and Open Systems*. Addison-Wesley Publishing Company Inc.. 1995. ISBN: 0-20-42293-X.

[mano] M. Mano and Charles R. Kime. *Logic And Computer Design Fundamentals*. Prentice Hall Inc.. 1997. ISBN: 0-13-031486-2.

[bennet] Chris J. Bennet. *The overheads of transnetwork fragmentation*. Computer Networks [and ISDN Systems]. vol. 6February 1982.

[cheshire99] Stuart Cheshire and Mary Baker. *Consistent Overhead Byte Stuffing*. IEEE/ACM Transactions on Networking. vol. 7 pp. 159-172 April 1999.

[cheshire96] Stuart Cheshire and Mary Baker. *Experiences with a wireless network in MosquitoNet*. IEEE Micro. vol. 16 pp. 44-52February 1996.

[fairhurst] Gorry Fairhurst. *Data Communications Engineering Course Syllabus [http://www.erg.abdn.ac.uk/users/gorry/course/syllabus.html]*.

[pppfaq] Ignatios Souvatzis. *PPP FAQ [http://theory.cs.uni-bonn.de/ppp/faq.html]*.

[swiathomski] Guy Swiatlowski and Shai Homski. *Quadrature Amplitude Modulation [http://www.hait.ac.il/staff/commEng/Michael_Bank/qam/qam.html]*.

[rfc1662] W. Simpson. *PPP in HDLC-like Framing (RFC 1662) [ftp://ftp.rfc-editor.org/in-notes/rfc1662.txt]*. July 1994.

[ieee802.3] *IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements*. Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications. 06-23-1986.

[gtc96] *Federal Standard 1037C [http://www.its.bldrdoc.gov/fs-1037/fs-1037c.htm]*. Glossary of Telecommunication Terms.. 08-07-1996.

[fddi1] *ISO 9314-1:1989*. Fibre Distributed Data Interface (FDDI) -- Part 1: Token Ring Physical Layer Protocol (PHY). International Organization for Standardization. 1989.

[fddi2] *ISO 9314-2:1989*. Fibre Distributed Data Interface (FDDI) -- Part 2: Token Ring Media Access Control. International Organization for Standardization. 1989.

[fddi3] *ISO/IEC 9314-3:1990*. Fibre distributed Data Interface (FDDI) -- Part 3: Physical Layer Medium Dependent (PMD). International Organization for Standardization. 1990.

[fddi4] *ISO/IEC 9314-4:1999*. Fibre Distributed Data Interface (FDDI) -- Part 4: Single Mode Fibre Physical Layer Medium Dependent (SMF-PMD). International Organization for Standardization. 1999.

[fddi5] *ISO/IEC 9314-5:1995*. Fibre Distributed Data Interface (FDDI) -- Part 5: Hybrid Ring Control (HRC). International Organization for Standardization. 1995.

[fddi6] *ISO/IEC 9314-6:1998*. Fibre Distributed Data Interface (FDDI) -- Part 6: Station Management (SMT). International Organization for Standardization. 1998.

[fddi7] *ISO/IEC 9314-7:1998*. Fibre Distributed Data Interface (FDDI) -- Part 7: Physical layer Protocol (PHY-2). International Organization for Standardization. 1998.

[ffdi8] *ISO/IEC 9314-8:1998*. Fibre Distributed Data Interface (FDDI) -- Part 8: Media Access Control-2 (MAC-2). International Organization for Standardization. 1998.

[fddi9] *ISO/IEC 9314-9:2000*. Fibre Distributed Data Interface (FDDI) -- Part 9: Low-cost fibre physical layer medium dependent (LCF-PMD). International Organization for Standardization. 2000.

[fddi13] *ISO/IEC 9314-13:1998*. Fibre Distributed Data Interface (FDDI) -- Part 13: Conformance Test Protocol Implementation Conformance Statement (CT-PICS) Proforma. International Organization for Standardization. 1998.

[uart96] Frank Durda IV. *Serial and UART Tutorial [http://www.freebsd.org/doc/en_US.ISO8859-1/articles/serial-uart/]*. The FreeBSD Project. 01-13-1996.