TCOM 370 NOTES 99-7

DIGITAL DATA TRANSMISSION

Synchronization and Related Topics

DIGITAL DATA TRANSMISSION (BASEBAND)

- "Data" in general is a **sequence of binary digits**. It may represent a sequence of alphanumeric characters or text, it may be the contents of a binary file, or it could be digitized voice or video.

We have discussed how pulses representing binary digits (M-level pulses) can **modulate** a carrier frequency for efficient transmission through a **limited-bandwidth** link such as the analog telephone line.We will now discuss some details of binary data transmission over links using **baseband techniques** (no carrier-frequency modulation, electrical pulses transmitted directly, usually as two-level pulses carrying one bit per pulse).

These are useful in many applications involving the use of **high-bandwidth** cables such as in ISDN connections over twisted pair, PCM voice or data at 1.544 Mbps and higher rates on special cables, and local area networks (e.g. twisted pair or coaxial cable in ethernet).

The key issues that we will consider here are synchronization and line coding.

- **Synchronization** is concerned with techniques to allow receivers to work in *timing synchronism* with transmitters.

- **Line coding** is used to generate special types of electrical pulse sequences to help in synchronization as well as to match the pulse sequence signal to the characteristics of the transmission line.

(In a subsequent set of notes we will discuss the important topics of *error control* and *compression* of digital data streams, which will take us into data link layer and higher layers of the communication protocol architecture.)

ASCII – American Standard Code for Information Interchange

To transmit characters for standard **text** and associated **control** characters (e.g. "return", "backspace", "delete", "start-of-text", etc.), ASCII defines a 7-bit code word for 2^7=128 characters in the set. A subset of these are the standard printable characters (upper/lower case alphabet, numerals, punctuation and special symbols), the rest are control characters for formatting or used to control transmission.

(We may add an 8-th redundant bit to provide a degree of **error control.** The 8-th bit may be defined to give the resulting 8-bit **byte** "even" **parity,** say, so that the number of binary 1's in the word is an even number. "Odd" parity may also be used, or a dummy 8-th bit may be inserted.)

If the data is already in binary format (e.g. computer object code in a file) we may think of it as occurring as a sequence of 8-bit **octets** or bytes, although this is not always necessary. Note that for a binary data file, combinations of bits may occur which may not occur in an ASCII representation of printable characters. This must be taken into account in a transmission scheme, as we will see.

SYNCHRONIZATION

A sequence of binary digits transmitted at some regular rate over a link in the form of an electrical signal (pulse sequence) has to be recovered at the receiver from the electrical signal.

- The important consideration here is that of **clocking or timing** of the instants at which the receiver will be looking for individual bits. If the receiver clock is running at a rate different from that of the transmitter, or if there is an offset between the two clocks even when they are running at the same rate, we will not be able to recover the bits properly.

How does the receiver obtain the necessary timing information? Sending a separate clock signal is possible but not preferred because it takes away transmission resources.

There are two approaches.

- In **asynchronous** transmission the receiver has a **free-running clock** that is running nominally at a fixed multiple of the transmitter clock rate (e.g. transmitter running at 9600 clock cycles/sec, receiver clock running at 8 times this rate, nominally 76,800 cycles/sec). Short bit sequences are emitted with framing bits that allow the receiver to know when to start looking for bits. For each short sequence, provided the receiver starts at approximately the correct time, it can maintain approximately correct timing for the duration.

- In **synchronous** transmission, transmit **clock information** has to be **embedded** within the bit stream, so that the receiver can either **extract** a clock signal for its use or can use the embedded clock information to **fine-tune its own local clock** to keep it in synchronism with the transmitter clock.

ASYNCHRONOUS TRANSMISSION

Bits are represented by one of two electrical states on the line, say two different voltage levels.

Suppose the line is in "idle" condition (in logic "high" or "1" state, say. This means that the electrical state of the line is that corresponding to bit "1" on the line. An idle line has to be in some electrical state.) Then one 7 or 8 bit data group is to be sent. These bits are **preceded** by a "start bit' (logic "low" or 0) and **followed** by one to two "stop" bits (logic "high" or 1).

The receiver operates by looking for the **first transition from "high" to "low"** after its previous idle or "stop" bits; it continuously samples at rate nominally N times the transmit clock rate. Upon finding the first transition from high to low, it sets the next sampling instant to be at N/2 receive clock samples (nominally half of transmit clock period) after the first transition; this should be near the center of the "start" bit. It then samples at multiples of N receiver clock pulses after this for the required number of data bits. The ending stop bits return the line to logic high (idle state value), so that the next character or byte will again produce a high to low transition with its start bit.

The receiver works better (samples more nearly at the center of the bit intervals) if N is larger; receiver clocks with N=8 or 16 are typical. This is because the faster the receiver clock is, the sooner after the actual transition from "high" to "low" it will discover it. The circuits that provide this type of transmitter and receiver function are called UARTS (Universal Asynchronous Receive/ Transmit).

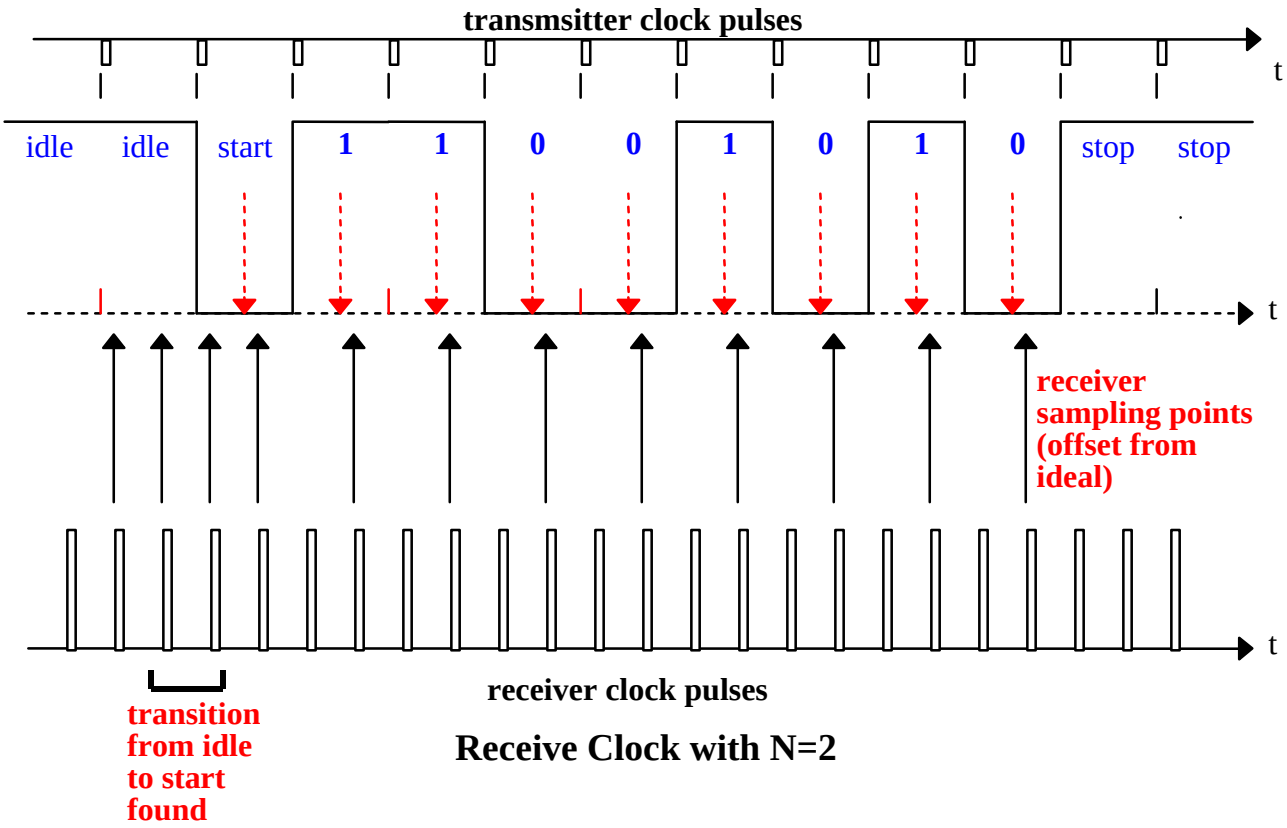
The main point here is that the receiver clock does not actually extract a clock signal from the incoming signal, nor does it try to synchronize its own clock with the transmitter clock. It assumes that its clock is reasonably accurate to maintain approximately correct timing over short spans of a few bits. This scheme is useful up to about several tens of Kbps

Frame Synchronization in Asynchronous Transmission:

If a block of characters is to be sent and received as one frame of data, we need a means to signal to the receiver the beginning and end of a frame.

This can be accomplished by sending special characters "STX" (start of text) and "ETX" (end of text) when transmitting printable ASCII characters (because these special characters do not occur in the middle of such data).

To transmit arbitrary binary digits, this needs to be modified, because any of the special control characters may occur within the data itself. For this we can use **character stuffing.** The special DLE control character (data link escape) is used to precede the STX and ETX characters marking frame start and stop. The receiver looks for these two-character combinations to determine frame boundaries. During the transmission, the transmitter inspects the outgoing bytes for the DLE character. Whenever this is found, a second DLE is stuffed in before the next byte. At the receiver, any pair of DLE's after frame start is decoded as a single DLE. The DLE-ETX pair of course denotes end of frame.



Receive Clock with N=2

ASYNCHRONOUS TRANSMISSION (START/STOP)

Notes:

"Direction of transmission" labels in figures like the above in the book always point in the opposite direction of "time". Earlier events are seen at the receiver before later events; thus the data bits above arrive at the receiver as 11001010. We could have shown a "direction of transmission" as an arrow pointing to the left. Sometimes (even more confusing) time is shown as increasing to the left and direction of arrival points to the right.

The transmitter and receiver clock waveforms generated by the timing circuits are usually a periodic square-wave at the clock frequency. The actual sampling instants are derived from the clock waveform by looking only at the rising edges or only at the falling edges (transitions) of the clock waveform, which occur once each clock cycle, of course. In the figure above we displayed only the narrow timing pulses that mark these events.

SYNCHRONOUS TRANSMISSION

Asynchronous transmission is inefficient (extra overhead bits needed; start bit, stop bits) and cannot be used at high transmission rates because of the increasing sensitivity to differences between transmit and receive clocks as the rate increases.

- In synchronous transmission, we tailor the way in which binary data is encoded onto the electrical signal to endow the signal with properties that the receiver can exploit to obtain receiver clock information.

The representation of a binary data stream as a baseband electrical signaling waveform (some type of pulse sequence) that travels on the link is called **line coding.**

Clock Extraction by Special Line Coding

In **polar Return to Zero** (RZ) line coding there is always a "transition" in the signal level in the middle of every bit interval *(this is also called "**bipolar**" in the book, but avoid using this terminology for this line coding scheme).* These transitions are from positive or negative level to zero. This can be exploited by the receiver to generate its own clock signal. (The receiver looks for transitions to and from zero level; pulses generated at these transition points constitute the clock signal. These pulses are shifted back by 1/4 clock period to get the sampling points in the centers of the first halves of the bit intervals. A circuit to implement this can be built easily).

The disadvantage of this method is (i) ***extra bandwidth,*** the actual pulse being half the size of the bit interval; (ii) ***wasted power*** in sending a three-level signal (iii) possible presence of a ***dc level*** which needs to be received (a continuous stream of all 1's or all 0's may be present in the data).

Manchester coding also always produces transitions at the bit centers, but they are between two levels only (high-low or low-high). This is also a **polar** code (two polarities are used). Each "1" bit has a transition from low electrical level to high electrical level, and each "0" bit the other type. The presence of transitions at periodic locations allows a clock extraction circuit to obtain timing information.

This code has ***no dc component,*** an advantage. It is a ***two-level code.*** However, it also ***needs extra bandwidth*** compared to straight two-level coding with no transitions in the centers of bit intervals.

[In straight two-level coding (also called **NRZ,** non-return to zero) the pulse width is the same as the bit interval, not one-half of the bit interval. Please ***do not use*** *NRZ to refer to Manchester coding,* as the book does.]

Differential Manchester Code. Here the encoding of each bit is basically like the Manchester code, except that the **type** of the transition (high to low or low to high) for a bit is determined not only by its logic state (1 or 0) but also by the preceding bit. Thus, the **type of the transition** for the bit being transmitted ***stays the same*** as that of the preceding one if the current bit is a 0, and ***switches*** if the current bit is a 1. The advantage of differential coding is that once the starting bit is known, all the bits following can be obtained *even if the entire waveform is* ***inverted*** (for example by reversal of the leads in a connector).

Note that with differential Manchester coding, there is always a transition at the beginning of a bit interval for bit "0", and no such transition for bit "1". This forms the basis of the decoder, once timing has been extracted from the transitions which are always present in the centers of the bit intervals.

- Manchester and Differential Manchester coding is used in local area networks, specifically in ethernet LANs and token-ring LANs, respectively.

Receiver Clock Synchronization

The above line coding schemes require higher bandwidths, as explained for the polar RZ case. We can get better bandwidth efficiency by using other line coding techniques. The clock recovery process is then usually based on the use of some ***local clock*** at the receiver that is ***pulled into synchronism*** by exploiting some basic clocking information in the signal, as opposed to being generated by the received signal.

The polar **NRZ** (non-Return to Zero) code is a very simple two-level code, however it does not have good properties. It can have a dc component, and is poor from the point of view of timing recovery.

The **NRZ-I** (the book's version of this is actually NRZ-S for NRZ-Space) has inversions in the following way: each 0 is signaled as a change in level from the previous one. So it is a differential code. Since 0's will cause transitions, **if we have a scheme that ensures that 0's are present at some minimum frequency** in the data stream, clock synchronization at the receiver can be accomplished. Note that here we are talking about synchronizing an existing fairly stable clock that requires small adjustments to keep it locked to the transmit clock, so that we have some more slack in the line code's clock-carrying information. The actual synchronizing is done by a digital phase locked-loop (**DPLL**).

The **AMI** code (alternate mark inversion, also known as "bipolar") encodes 1's with alternating polarity pulses and sends 0's with a zero level. (The opposite version of this sends 0's with alternate polarities; it is called ***"pseudo-ternary".*** It is essentially the same.). Thus there are transitions in AMI if the number of consecutive 0's is somehow limited.

The **B8ZS** code (bipolar 8 zero substitution) code is one modification of the AMI code achieving this. For this code, every sequence of 8 consecutive 0's is transmitted as 000VB0VB, where V is a bipolar violation pulse amplitude and B is a conforming pulse amplitude. (A violation means that two consecutive non-zero pulses of the **same** polarity occur in a signal with zero amplitude pulses in between). This scheme retains the zero dc balance and ensures a minimum rate of transitions in the signal.

Other interesting codes are the HDB3 and 4B3T ***(Read p. 119-120 in Halsall).***

Hybrid schemes employ both Manchester-type codes and DPLL to get better performance.

Frame Synchronization in Synchronous Transmission

We have discussed the bit-synchronization techniques for synchronous transmission above. We now consider briefly the situation with respect to frame synchronization, which allows the receiver to know where the frame boundaries are.

Character-Oriented:

Frame starts with a number of SYN characters and a STX character.

SYN characters allow the receiver to gain bit synch, and then find the SYN characters, hence establishing character boundaries. The STX character then signifies start of frame.

Data transparency for binary data is obtained with DLE-STX, DLE-ETX and DLE-DLE characters.

Bit-Oriented:

Start and End of frame: 8-bit "flag" 01111110.

Idle bytes: 01111111 repeatedly sent when idling to allow bit synch to be maintained.

Flag pattern should not occur in data; use 0 bit stuffing following a sequence of five data 1's.

Other schemes:

Preamble 101010---10 ("10" pairs)
followed by
Start-of Frame byte;
Fixed header (address); two length bytes (number of bytes in frame contents)

Frame tail or end of frame contents may contain error detection and correction bits.