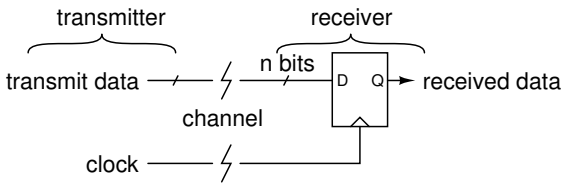
Asynchronous Serial Interfaces

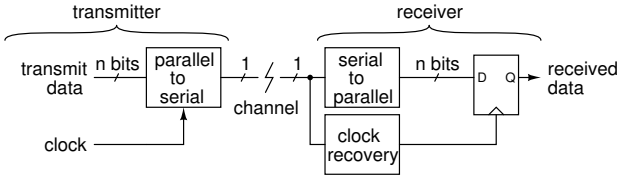
This lecture describes the RS-232 serial interface, a simple communications interface. Although no longer widely used, it demonstrates some features that are used by more advanced communication protocols: transmission of data as a serial bit stream, encoding of binary values into voltages, an error-checking technique using redundant parity bits, and a framing technique that the receiver uses to group bits.
After this lecture you should be able to: identify the purpose and signal voltages present on the data and handshaking signals; convert to/from the signal waveform and the data values transmitted; predict and explain the results of data rate and character format mismatches; explain advantages of differential "RS-422" serial interfaces; distinguish between synchronous and asynchronous interfaces.

Serial Interfaces

To transmit data over short distances we can use one connector for each bit of a word and transfer all the bits simultaneously. To synchronize the transfer we use a clock signal which causes new data to be loaded into flip-flops on the receiving side:



But over long distances it's preferable to reduce the number of cables required by transmitting the bits sequentially rather than in parallel. In addition, if we can somehow recover the clock signal at the receiver instead of transmitting it along with the data we could eliminate that signal as well:



This lecture introduces one such interface.

RS-232 Interface

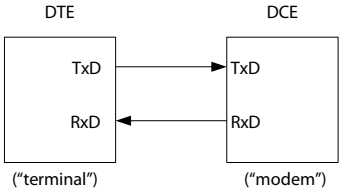
The RS-232 interface is a simple data communication interface that was widely used for low-speed (tens of kb/s) communication with devices such as modems, terminals and printers. Today its use is mainly limited to industrial automation and diagnostic interfaces.

DTE and DCE

The standard RS-232 connector was a 25-pin D-style connector often called a "DB-25".

In its simplest form the interface uses two signal pins and one ground pin. Pin 2 is called Transmit Data (TxD), pin 3 is called Receive Data (RxD) and pin 7 is signal ground. When two serial devices are connected together they are connected pin-to-pin (RxD is connected to RxD and TxD is connected to TxD). This means that RxD must be an input on one device and an output on the other device. Thus the terms RxD and TxD *do not* say whether a pin is an input or output but instead are names for pins on the connector.

The serial interface was originally designed to connect modems – "Data Communications Equipment" (DCE) to computer terminals "Data Terminal Equipment" (DTE). The terms "receive" and "transmit" are thus *from the point of view of the data terminal*. On a DTE TxD is an output and RxD is an input. Conversely, on a DCE RxD is an output and TxD is an input. Typically, DTE connectors are male and DCE connectors are female but there are exceptions.

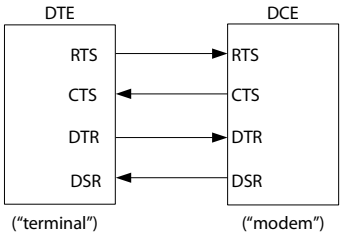


Exercise 1: Is the "Transmit Data" (TxD) signal an input or an output? How about "Receive Data" (RxD)? Is a computer a 'modem' (DCE) or a 'terminal' (DTE)?

In addition to the two data lines, most RS-232 devices implement additional "handshaking" pins that

allow the two devices to exchange status information. Of these, the most useful are called RTS (Request To Send) and CTS (Clear To Send). The RTS line is an output on a DTE and is used to tell the DCE that the DTE wants to send data (RTS was originally used to control half-duplex modems – those that can't transmit and receive simultaneously – but these are rarely seen today). The CTS pin is an output on a DCE and is used by the DCE to indicate that it can accept data on the TxD line.

Since these signals are used to control the flow of data from the DTE (and optionally from the DCE, see below) these pins are called [hardware] "flow control" signals.

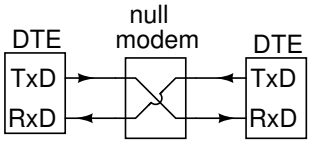


The second set of control signals are DTR (Data Terminal Ready) and DSR (Data Set Ready). These signals indicate that the DTE and DCE devices respectively are connected and operational. Typically they indicate that the power is turned on. However, by de-asserting DTR it is possible, if the modem is correctly configured, for the DTE to prevent the DCE from sending data on RxD.

A number of other handshaking signals are defined by the standard but are seldom used.

In addition to the standard DB-25 serial connector, there are a number of smaller connectors that are often used. These connectors are physically smaller and carry a subset of the RS-232 signals. The most common are the DB-9 connectors popular on some PCs and the inexpensive telephone-style "RJ-11" (6-pin) and "RJ-45" (8-pin) connectors (popular on devices with many serial interfaces or limited space for connectors).

Adapters are often used not only to convert between different styles of connectors but also to convert between male and female connectors (a "gender adapter" which allows two males or two females to be connected together) and to switch between DCE and DTE pinouts (a "null modem" which allows two DCEs or two DTEs to be connected together):



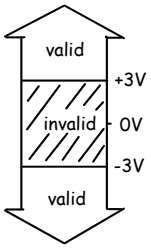
Interface Voltages

The serial interface voltage levels are bipolar (positive and negative) with respect to ground. The table below summarizes the relationship between voltage level, logical meaning on handshaking lines and the corresponding data bit value (valid on TxD and RxD only).

Signal Level	Line State	For Handshaking	For Data
negative	mark	false	1
positive	space	true	0

Note: The data lines (TxD and RxD) transmit the binary value '1' when **negative**. The control lines (e.g. CTS) are asserted (true) when **positive**. The convention for the data lines is probably the reverse of what you were expecting.

The received signal must be greater than +3 volts to be considered positive and less than -3 volts for negative. Intermediate values are considered invalid. This allows disconnected pins to be detected. A larger swing is required at the transmitter (at least ±5 V) to allow for attenuation in the cable.



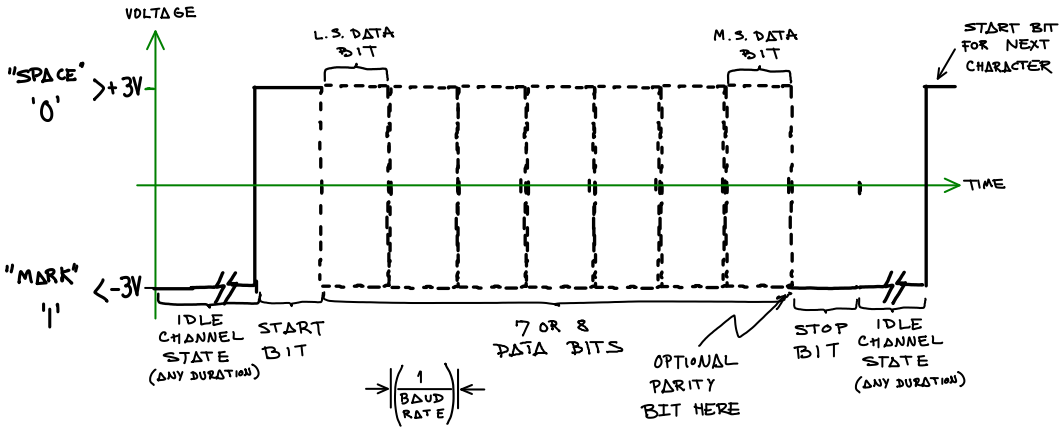


Figure 1: Asynchronous Interface Waveform.

The terms "mark" and "space" supposedly refer to the earliest telegraph machines that converted the signal current to visible marks and spaces by moving a pen up and down on a moving strip of paper. For teleprinters the term 'mark' came to mean the voltage or current when no data was being sent (see below).

Character Format

The signal format, shown in Figure 1, was originally designed to drive teleprinters – machines that replaced telegraph operators by converting electrical signals carried over telegraph wires into printed characters. "Teletype" was one manufacturer.

Data is transferred over the serial interface one bit at a time. A positive (zero) bit (the "start bit") is sent to indicate the start of the character being sent. This is followed by the bits in the character, from LS to MS bit. After sending the 7 (for ASCII) or 8 (for arbitrary bytes) bits, an optional parity bit can be sent, followed by a one "stop" bit. The parity bit is set to 1 or 0 in order to make the number of 1's in the word an even or odd number (even and odd parity respectively).

Exercise 2: Draw the waveform used to send the ASCII character 'd' (hex 64) at 4800 bps with eight data bits and even parity.

Exercise 3: Will the parity bit allow the receiver to detect all single-bit errors? All double-bit errors?

The transmitter and receiver must both be configured to the same data rate and number of data bits. The receiver waits for the rising edge of the start bit and then samples after the appropriate delays (1.5 baud periods, 1 baud period, 1 baud period,...). The use of stop and start bits ensures that there is a rising edge between each character and allows the receiver to re-synchronize itself at the start of each character. This allows for small variations between transmitter and receiver timing.

Exercise 4: What happens if the receiver's clock is running faster than the transmitter clock?

Exercise 5: What would happen if the receiver was expecting 8-bit characters and the transmitter was sending 7-bit characters? What about the reverse case?

There are a number of standard bit rates, typically powers of two times 1200 bps (1200, 2400, 4800 bps etc). The RS-232 standard specifies maximum bit rates, distances, etc but these are usually ignored in practical applications. For short distances it's possible to send at rates over 100 kbps.

The need to manually configure both interfaces, to choose between the many types of cables required by the different DTE/DCE and connector combinations and the relatively low data rates have made this interface (nearly) obsolete.

A UART (Universal Asynchronous Receiver Transmitter) is an IC that converts an asynchronous data signal to and from an interface that transfers all the bits in parallel (all at the same time). Microcontrollers often contain UARTs. Software controls a UARTs by reading and writing various data, status and control registers.

Since the microcontroller's output is at unipolar logic levels (e.g. 0 and 3.3V), an interface IC is normally required to convert to/from the bipolar RS-232 voltages. We will study one of these ICs in a lab.

Other Asynchronous Serial Interfaces

The RS-422 serial interface specification uses a similar signaling scheme but uses differential signals (opposite voltages on two signal lines) to increase immunity to noise and increase maximum transmission distance. Data rates of 1 Mbps are common. RS-422 is common in industrial applications because of its improved noise immunity.

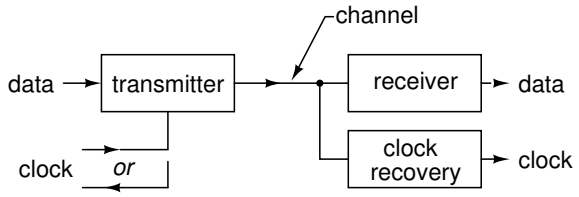
RS-485 is similar to RS-422 but the transmitter (driver) can be disabled. This allows multiple devices to be connected in parallel along one "multidrop" cable.

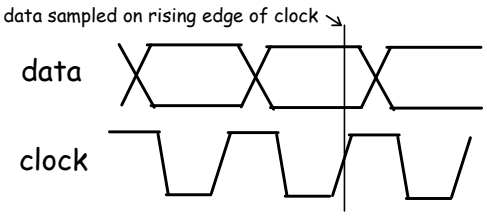
Synchronous Interfaces

Modern data communication systems (e.g. Ethernet, WiFi, etc) operate synchronously.

Synchronous communication systems are those that transmit data continuously and thus avoid the overhead of start and stop bits.

Instead of looking for a start bit at the start of each character, synchronous systems observe transitions in the data signal to generate a constant-rate clock signal. This clock is used to recover the data:





The data is often grouped into "frames" which begin with a known preamble signal that helps the receiver synchronize to the incoming data.