The RS-232 Serial Communications Protocol As Used Today

Serial Protocol Definition

Serial communication involves sending one bit at a time. This is one of the slowest means of data transmission, however, it is still widely used in the scientific community because it is also one of the simplest and most expedient. There are a few serial communications standards, this article will address mainly RS-232 (ANSI/EIA-232) and the simple three wire transmissions for illustration of the underlying principals of serial communication.

History

In 1963 the ASCII standard was published and is still used today in Serial communications. In 1969 the RS-232 protocol was published. The serial protocol is very capable of sending binary data, and was used for mostly that until the mid 1990s. However in 1995 and 1996 the FireWire and USB protocol respectively relegated RS-232 to the scientist, engineer, and the ham radio operator. FireWire and USB standards brought very high speeds to serial data transmission and provide much more current to drive low power nodes and other devices.

Benefits

As of today, simple three wire serial communication is the easiest way to connect smart devices quickly. When the devices do not need to transmit huge amounts of data, like engine sensors, gps devices, air plane instrumentation, robotic controls, cooling systems for mainframes, etc, serial is the way to go. With a couple of lines of code and few minutes with a soldering iron, chances are, you're up and running! Some high-tech watches even communicate there internal problems to a computer via serial, so the technician may better diagnose the problem! www.brainlubeonline.com/watchpage.html This is because serial UARTS and/or microcontrollers that support the serial protocol can be exceedingly small as a result of the protocols simplicity.

Limitations

Without complex phase modulation techniques, 9600 baud transmissions are limited to around 250 feet of shielded cable with RS-232. This is an extremely slow data rate by todays standards. Another draw back is the protocol doesn't leave any room for anything more than truly simple devices to be powered by the serial cable its self. The good news is, the protocol leaves ample room for the devices that can afford to power objects through the serial cable to do just that; for example, a PIC18X microcontroller can power a device (or another PIC) with 5 volts @ 1 amp for a 100% duty cycle, That's a 5 watt device!

Low Level Composition

Methods:

At the simplest level, a send, a receive, and a ground wire are the only connections needed for full

duplex communications. www.brainlubeonline.com/rs232.html . However, with a UART (Universal Asynchronous Receive/Transmit) IC, or a method of coding called Bit-Banging that calls functions to access the computers clock, much higher speeds are possible. There are two root types of serial transmissions, synchronous and asynchronous. Synchronous serial transmissions, send bits from sender to receiver and back, to synchronize both transceivers. This allows higher speed data transmissions than asynchronous communications because a start and stop bit don't need to be appended to the data packets as they would need to be with the asynchronous scheme.

Specifications:

RS-232 by the books is limited to fifty feet maximum cable length, many serial hot-rodders have surpassed this. Serial works by (as much of as possible) a square wave (voltage over time). Period and duty cycle will be decided by the agreed upon baud rate. Good house keeping denotes a positive 2.5 to 15 volts equal a logic 0 and negative 2.5 to 15 volts give you a logic 1. However, with many microcontrollers and embedded systems not equipped with UARTs, a positive 5 volts and a nice pull-to-ground is just fine for a logical 0 and 1 respectively. This is inverted as compared to most systems and so makes easy work for testing communication lines with a multimeter. Nine pins are required for the full spectrum of what RS-232 has to offer. For hardware handshaking, five pins are needed; Data Carrier Detect, Data Terminal Ready, Data Set Ready, Request To Send, and Clear To Send. Packet transmission needs two dedicated pins, Serial Data Input, and , you guessed it, Serial Data Output. The ninth pin is a ring indicator (for modem use, or to “wake” some device up).

Testing the Physical Link:

To test the serial data output line, turn off stop/start bits and parity. Then, understand your baud rate, for instance if you are running at 9600 baud that's a bit time of 1/9600 of a second, or 104ms. 9.6 kilocycles a second (kilohertz), and a duty of 104ms high, means a 100% time duty cycle. To test this,

put the line to an oscilloscope, and create a send loop transmitting an ASCII 32 (or any base two Binary number, 2, 4, etc.). These creates a logical bit level 1 every 9600 divided by n number of data bits of time, this will scope in your idea pulse rate. If the time error is greater than ~1% maybe you need to retune your equipment.

Mid Level Composition

Baud Rate:

The speed of serial transmissions is timed in a ratio denoted Baud. In its simplest form 1 baud is 1 bit per second. Therefore, 1200 baud is 1200 bits per second; since the serial protocol is one bit at a time, the baud rate also tells us the sampling rate of the transmission. This means two transceivers communicating at 33,600 baud both have there UART clocks oscillating at 33,600 cycles per second or 33.6 kilohertz. Transmission speed is limited by many variables. Distance limits clock speed mainly because of the internal resistance of the transmission wires, but also because of induction. The hall-effect has some small amount of degradation to the transmission. Over vast distances, hardware will start to see an RMS reading of two distinct pulses. In other words, it is possible to turn the transmitter into a pulse width modulator or a digital to analog converter. In the 70s, through the 90s, RS-232 was used through the phone lines for access to primitive internet (haha). Speed then was limited because of phone company band pass filters and noise laden cables. However in short distances, the author has made serial interfaces from microcontroller to microcontroller at 115.2 kilobaud!

Data Bits:

Many bits are packaged together in groups of like numbers for data transmission. This could be thought of as people (representing bits) gathering onto a train car. When one car is full the next car pulls

forward and loads the same number of people (or bits) to be carried off to some distant land. These packages of data are called packets. RS-232 will let sender and receiver agree to any number of packet size (so long as both transceivers are in accordance) however, standard packet sizes are 5, 7, and 8 bits. These are the actual raw data elements that needs to be transmitted, not including the start/stop bits, and parity, which will be discussed later. When the number of data bits is being set for packet size, one must take into account what type of data will be predominantly sent. For instants, if transmitting standard ASCII, a 7 data bit packet would be preferred (0-127), if extended ASCII were to be sent, then an 8 data bit packet would be desired (0-255). This not only makes for a more efficient transfer it also cuts down on the appending of “filler bits” in the transmission. When an 'N' amount of bits cannot be evenly divided by the packet data bit size, you will have partial end packets, these packets are filled with ASCII NULL (0). This can increase your file size and must, in most situations, be trimmed at the software level.

Stop bits:

Stop bits signal the end of a packet. Since the data is essentially represented by a square wave (voltage/time), and the sender and receiver clocks can slip out of synchronization, stop bits give the computers or routers or whatever is communicating, room for error in clock speed (measured in Parts Per Million of error counts). RS-232 suggests typical values of 1, 1.5, and 2 bits. However, any number of stop bits may be used. The greater the number of stop bits the slower the data rate, this is simply because there is now more data to convey (and more data to process if on a limited system such as a router controller or small robot). The greater the number of stop bits, the greater the modulation error that can be afforded in two separate phase locked loops. This means when using a technique such as bit-banging (software calling hardware clock functions) where clock error will be common, one needs to use a high number of stop bits.

Parity:

Parity is an error detection scheme for serial communication. Parity can be turned on or off (used or not used), and there are four methods of parity. The first two are denoted odd parity, and even parity. In this system, a logical 1 or 0 is inserted at the end of the data set(data bit portion) of the packet. For odd parity, if the data bits were 0011100, the parity bit would be set to 0 to keep the dataset odd, if the data transmission were 0111010, the parity bit would be set to a logical 1, again to keep the data set odd. This can be inversed by using even-parity. The 7-bit data set 0110000 would have an even parity bit of 0 to keep the count even. The remaining to methods are marked parity and spaced parity. These methods do not check the data bits at all, instead they set the parity bit to high for marked and low for spaced. The receiver knows to expect these parity bits and if it does not, it knows there is a synchronization problem, excessive noise or other problems in the transmission.

High Level Composition

Error Correction:

In RS-232 error correction is handled on the software level(or programmed hardware). In a Windows environment, library functions for Microsoft's C, and Visual Basic, and whatever they have decided Sun's JAVA is today, include an EV_ERR, which indicates an “error event” or mis-communication. The COMSTAT structure will tell where in time, packet, and process the even occurred. There are many complex error correction methods that sometimes do not require the resubmission of the packet in error, however for RS-232 the most effectual way of error correction is to transmit a call to resend the packet or packets.

Encryption:

Since most of the RS-232 work now is machine commands and not really “sensitive data” this is rarely

an issue now. To compound this most machines do not have a time or resources to decode encrypted material. There are however simple (easily crackable) algorithms that can jumble the data sent and be decoded with one simple transmission of a “key”. One method is, a level of permutation (given by the key number) that combinatorially sorts the data bits into a mess only to be resorted at the receiving computer or node. The level of sort and the amount of data is directly proportional to the amount of time it takes to code and decode with this method. Lowering the resolution of the permutations in possible if binaries and not words are being sent, by sorting packets instead of individual bits. An example of a suitable permutator code using recursion can be found here (this version allows a user to input numbers to be sorted to illustrate the device). www.brainlubeonline.com/cpa.html Anything more advanced than this will be a severe strain of devices with limited resources, and there are other protocols that are better suited for this type of work.