Serial Port Basics

What is a serial port?

The UART (universal asynchronous receiver transmitter) serial port or just "serial port" for short, is an I/O (Input/Output) device.

Two-way communication, asynchronously (no clock).

Find serial ports on computers.

An I/O device is just a way to get data into and out of a computer.

There are many types of I/O devices, such as: serial ports, parallel ports, disk drive controllers, Ethernet boards, universal serial buses, etc.

Most PCs have one or two serial ports. Each has a 9-pin connector (sometimes 25-pin, especially on older computers) on the back of the computer.

Computer programs can send data to the transmit pin (output) and receive bytes from the receive pin (input). The other pins are for control purposes and ground.

UART is an example of LSI (large scale integration).

The UART converts the data from parallel to serial and changes the electrical representation of the data. Inside the computer, data bits flow in parallel (short wireruns, allows for higher rate of speed than serial).

Serial flow is a stream of bits over a single wire. For the serial port to create such a flow, it must convert data from parallel (inside the computer) to serial on the transmit pin (and conversely). Most of the electronics of the serial port is found in a computer chip known as a UART.

The conventional serial port (not USB or Firewire) is a very old I/O port. Most desktop PCs have them. Not likely to be present on newer laptops. Macs after mid-1998 have only the USB port. It's possible to put a conventional serial port device on the USB bus which is on all modern PCs (including laptops and Macs).

The following description is for UNIX. Each serial port has a file associated with it in the /dev directory. It isn't really a file but it seems like one. For example, /dev/ttyS0. Other serial ports have similar names, like /dev/ttyS1. Ports on USB bus, multiport cards have different names. On PCs running DOS or Microsoft Windows, there are reserved device names like COM1, which don't appear in the file system hierarchy as a file or directory name (you can't name a file with the name of a reserved device name).

The common specification for the conventional serial port is RS-232 (changed to EIA-232). So it is often called a RS-232 serial port. One pin is used to send out data bytes and another to receive.

At the EIA-232 serial port, voltages are bipolar (positive or negative with respect to ground) and should be about 12 volts in magnitude (some are 5 or 10 volts). A lot of the pins on 232 are no longer used.

For the transmit and receive pins, +12 volts is a 0-bit (sometimes called ***space***) and -12 volts is a 1-bit (sometimes called ***mark***). This is known as **inverted logic** since normally a 0-bit is both false and negative while a 1 is normally both true and positive.

Other pins (modem control lines) are normal logic with a positive voltage being true/on and negative voltage being false/off. Zero voltage has no meaning except it usually means that the unit is powered off.

A range of voltages is allowed. The specs say the magnitude of a transmitted signal should be between 5 and 15 volts but must never exceed 25 volts.

RS-170A is putting TV on a serial line. A is modified slightly for color.

Any voltage received under 3 volts is undefined.

EIA-422 on a Mac or EIA-423: voltage only 5 V.

Normal computer logic normally just a few volts so if you try to use test equipment designed for testing 3-5 volt computer logic (TTL) on the12 volts of a serial port, it may damage the test equipment.

Voltage Sequence for a Byte

Transmit pin (sometimes called TxD) held at -12 volts (mark) at idle when nothing is being sent.

To start a byte it jumps to +12 volts (space) for the start bit and remains at +12V for the duration (period) of the start bit. What if a start bit is different sizes? If it is smaller, the frequency is higher and thus the bit rate is higher.

Baud rate is bit rate. Baud from last name Baudot. Started out at 110 baud. 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 76800. Somewhere up to 119000. Most stuff we will see is at 9600 or 19200.

Next comes the low-order bit of the data byte. Also known as least significant bit (LSB). If it's a zero, nothing changes and the line remains at +12V for another bit-period. If it's a one, the voltage jumps from +12 to -12 volts. After the last data bit, a parity bit may be sent, and then a -12V (mark) stop bit.

Specifying to the UART the parameters, see: speed, number of data bits, number of stop bits, parity code (N (no), O (odd), E (even), I (ignore)). For example, 9600,8,0,N.

For example, hyperterminal on Windows communicates with the serial port. Communicate with router by connecting it to the serial port of a computer.

Asynchronous is not clock-less. The data is clocked at a constant speed within the data. Asynchronous refers to that the clocks on the receiver and transmitter are not synchronized; the receiver has to figure out the clock timing on the fly as the data comes in from the transmitter.

Then, the line remains at -12V (idle) until the next start bit. Note that there is no return to 0 volts and thus there is no simple way (except by a synchronizing signal) to tell where one bit ends and the next one begins for the case where two consecutive bits are the same polarity (both zero or both one).

A second stop bit would also be -12V. Since there is no signal to mark the boundaries between these bits, the only effect of the second stop bit is that the line must remain at -12V idle twice at long.

The receiver has no way of detecting the difference between a second stop bit and a longer idle time between bytes. Thus communications works OK if one end uses one stop bit and the other end uses two stop bits, but using only one stop bit is obviously faster. In rare cases 1 ½ stop bits are used. This means that the line is kept at -12V for 1 ½ time periods (like a stop bit 50% wider than normal).

Parity Explained

Characters are normally transmitted with either 7 or 8 bits of data. An additional parity bit may or may not be appended to this resulting in a byte length of 7, 8, or 9 bits. Some terminal emulators and older terminals do not allow 9 bits. Some prohibit 9 bits if 2 stop bits are used since this would make the total number of bits too large: 12 bits total after adding the start bit. The parity may be set to odd, even or none (mark and space parity may be options on some terminals or other serial devices).

Odd Parity

With odd parity, the parity bit is selected so that the number of 1 bits in a byte, including the parity bit, is odd. For example, for a capital A (41 hex or 65 decimal or 01000001 binary), would have 100000100 as the bit sequence for odd parity (from left to right),since the parity bit would have to be at zero to make the bit count odd including the parity. Recall that we are using inverted logic, so a 0-bit indicates a logical 1 and a 1-bit indicates a logical 0.

If such a byte gets corrupted by a bit being inverted, the result is an illegal byte of even parity. May ask for a resend or might throw it away. This error will be detected and if it is an incoming byte to the terminal an error-character symbol will appear on the screen. Symbol might be a star/asterisk.

Even Parity

Even parity works in a similar manner with all legal bytes (including the parity bit) having an even number of 1-bits. During setup, the number of bits per character usually means only the number of data bits per byte (7 for true ASCII and 8 for various ISO character sets). ASCII is American Standard Code for Information Interchange, ISO is International Standards Organization.

A mark is a 1 bit or logic 1 and a space is a 0 bit or logic 0. For mark parity, the parity bit is always a 1 bit. For space parity it is always a 0 bit. Mark or space parity (also known as sticky parity) only wastes bandwidth and should be avoided if feasible.

Forming a Byte (Framing)

In serial transmission of bytes via EIA-232 ports, the low-order bit is always sent first (the bit order). Serial ports on PCs use asynchronous communication where there is a start bit and a stop bit to mark the beginning and end of a byte. This is called framing and the framed byte is sometimes called a frame. A total of 9, 10, or 11 bits are sent per byte with 10 being the most common. 8-N-1 means 8 data bits, no parity, 1 stop bit; this adds up to 10 bits total when counting the start bit. One stop bit is almost universally used.

At 110 bits/sec (and sometimes at 300 bits/sec) 2 stop bits were once used but today the second stop bit is used rarely.

How Asynchronous is Synchronized

The EIA-232 serial port as implemented on PCs is asynchronous which means that there is no clock signal sent with ticks to mark when each bit is sent. No transitions within the signal to tell where the clock is.

There are only two states of the transmit/receive wire: mark (-12V) or space (+12V). There is no state of 0V.

For a receiver to detect individual bits it must always have a clock signal in synchronization with the transmitter clock. The synchronization is the start bit.

Why doesn't noise usually affect RS-232? The voltage level. Noise runs around zero volts. RS-232 signal is either 12V or -12V, which is way above the noise. The voltage level will drop fairly fast so there is a line length limitation.

For asynchronous transmission, synchronization is achieved by framing each byte with a start bit and a stop bit (done by hardware). The receiver listens on the negative line for a positive start bit and when it detects one it starts its clock ticking. It uses this clock tick to time the reading of the next 7, 8, or 9 bits. It actually is more complex since several samples of a bit are normally taken and this requires additional timing ticks. Might sample in the center in what it thinks is the data in order to avoid sampling on the edges. Then the stop bit is read, the clock stops and the receiver waits. Asynchronous is synchronized during the reception of a single byte but there is no synchronization between one byte and the next byte.

FIFOs

First In, First Out queue discipline. Versus dumb.

When a UART sends/receives a byte. For dumb UARTs, the CPU gets an interrupt from the serial device every time a byte has been sent or received. The CPU then moves the received byte out of the UART's buffer and into memory somewhere, or gives the UART another byte to send.

Older UARTs only have a 1 byte buffer; that means that the UARTinterrupts the CPU for each byte. At high transfer rates, the CPU is so busy handling the UART it doesn't have time to adequately tend to other tasks.

In some cases the CPU doesn't get around to servicing the interrupt in time, and the byte is overwritten because the bytes are coming in so fast. This is called an **overrun** or **overflow**.

FIFO UARTs help solve this problem. The 16550A FIFO chip comes with 16 byte FIFO buffers. This means it can receive up to 14 bytes before it has to interrupt the CPU. Not only can it want for more bytes, but the CPU then can transfer all (14) bytes at a time. The CPU receives fewer interrupts and it is more free to do other things. Data is rarely lost. Note that the interrupt threshold of FIFO buffers (trigger level) may be set at less than 14. 1, 4, and 8 are other possible choices of interrupt thresholds. Other UARTs have even larger buffers.

Note the interrupt issued before buffer full; allows room for a couple more bytes to be received before interrupt service routine able to fetch the bytes. Trigger level set by kernel software to various permitted value (1 is like the old style 1-byte buffer UART).

Consider case where sent short text through the serial port. If send number of characters that is fewer than the threshold level, the FIFO buffer might be waiting to fill up further before interrupting the CPU. There is a timeout to prevent this problem; the UART will interrupt the CPU to clear the FIFO buffer of received data even if the buffer isn't full. There is a timeout for the transmit buffer as well.

Why FIFO Buffers So Small

Flow control. Stops the flow of data when necessary. If stop signal sent to serial port, stop request handled by software (even if flow control is "hardware"). Serial port hardware knows nothing about flow control. If serial port buffer contains 64 bytes ready to send when it receives a flow control signal to stop sending, it will send out the 64 bytes anyway. There is not stopping it since it doesn't know about flow control. If buffer was large, mistake would be even worse (in violation of flow control's request to stop).

Pins and Wires

Old PCs used 25 pin connectors but only about 9 pins used. Voltage on any wire measured with respect to the signal ground on one of the wires. Minimum number of wires to use for two-way transmission of data is 3. Other wires for control purposes. All of these signals could have been shared on a single wire but there is a separate/dedicated wire for each signal. Some called modem control lines. These are either in asserted state (on) of +12 volts or in negated state (off) of -12 volts.

One wire to signal computer to stop sending bytes out serial port cable. Conversely, another wire signals the device attached to the serial port to stop sending bytes to the computer.

Not used a lot. Used for printers (impact printers like daisy wheel printers) that had to move mechanically and that took time, so they used this flow control.

If attached device is modem, other wires may tell modem to hang up line, tell computer connection made, telephone line ringing.

RS-232 or EIA-232, etc.

Serial port usually RS-232-C, EIA-232-D, or EIA-232-E. They are all about the same.

Original RS (Recommended Standard) prefix became EIA (Electronics Industries Association) and later EIA/TIA after EIA merged with TIA (Telecommunications Industries Association). The EIA-2332 spec also provides for synchronous communication.

I/O Address and IRQ

Since computer needs to communicate with serial port, operating system must know that each serial port exists and where it is (its I/O address). It also needs to know which wire (IRQ number) the serial port must use to request service from the computer's CPU. It requests service by sending an interrupt voltage on this wire. Every serial port device must store in its non-volatile memory both its I/O address and its IRQ number (interrupt request).

The PCI bus has its own system of interrupts. Peripheral Component Interconnect. This is the socket into which you plug in hardware boards.

PCI bus has its own system of interrupts. Since PCI-aware BIOS sets up these PCI interrupts to map to IRQs, seemingly behaves just as described above. Except that sharing of PCI interrupts is allowed (2 or more devices may use the same IRQ number).

I/O addresses are not the same as memory addresses. When an I/O address is put into the computer's address bus, another wire is energized. Tells main memory to ignore the address and tells all devices which have I/O addresses (such as the serial port) to listen to the address sent on the bus to see if it matches the device's. If the address matches then the I/O device reads the data on the data bus.

The I/O address of a certain device will actually be a range of addresses. The lower address in this range is the **base address**.

Interrupts

When serial port receives a number of bytes into FIFO, signals CPU to fetch them by sending an electrical signal known as an interrupt on a certain wire normally used only by that port. Also may send interrupt if timeout waiting for additional bytes.

Some UART chips. If 4 bytes in a row could have been received in a time interval, but none show up, then port gives up waiting for more bytes and issues interrupt to fetch the bytes from FIFO. No interrupt if FIFO empty.

Each interrupt conductor in computer has a number (IRQ).

Small buffer may overflow if not serviced by CPU soon enough, resulting in loss of data.

There is no flow control to prevent this.

Interrupts also issued when serial port just sent out all of its bytes from small transmit FIFO buffer out the external cable. Notifies CPU so that it may put more bytes in the small transmit buffer. Also, when a modem control line changes state, issues interrupt.

Buffers mentioned above are all hardware buffers. Serial port also has large buffers in main memory.

Interrupts convey a lot of information but only indirectly. Interrupt tells chip called interrupt controller that a certain serial port needs attention. Interrupt controller then signals CPU, which then runs a special program to service the serial port. That program is called an interrupt service routine (ISR). It tries to find out what happened at the serial port and then deals with the situation. Program can easily find out what has happened since the serial port has registers at I/O addresses known to the serial driver software. Registers contain status information about the serial port. Software reads registers and finds out what has happened.

Names: ttyS0, ttyS1, etc.

Serial ports named COM1, COM2 in DOS/Windows. Serial driver software maintains table showing which I/O address corresponds to which COM. This mapping of names to I/O does address the I/O address and IRQ in the hardware itself which is set by jumpers (now rarely), or by plug-and-play software.

Thus, which physical port corresponds to ttyS1 depends both on what the serial driver thinks and what is set in the hardware. If mistake made, physical port may not correspond to any name such as ttyS2 and thus it can't be used.

Flow Control

Flow control is the ability to slow down the flow of bytes in a wire. Serial port: stopping and restarting the flow without any loss of bytes. Needed for modems and other hardware to allow a jump in instantaneous flow rates.

Example of Flow Control

33.6K external modem. Flow from computer to modem is 115,200 bits/second. Flow from modem out phone line is only 33.6 kbps. Modem stores excess flow (115.2 –33.6 = 81.6 kbps) in its buffers, which will soon overrun unless computer flow stopped. Modem sends stop signal to serial port when modem's buffer almost full. Serial port passes stop signal on to device driver and the 115.2 kbps flow is halted. Then modem continues sending out data at 33.6kbps drawing on the data it previously accumulated in its buffer. Since nothing is coming into this buffer, the number of bytes in it starts to drop. When almost no bytes left in buffer, modem sends start signal to serial port and the 115.2 kbps flow from computer to modem continues.

Flow control creates an average flow rate in the cable that is much less than the full flow rate of 115.2 kbps. This is *start-stop* flow control.

In thi example it was assumed that the modem did no data compression. Now consider where modem is compressing data at high compression ratio. Then, modem may be able to handle the 115.2 kbps since it is compressing to 33.6 kbps. Compression ratio of 3.43. No need for flow control here since compressing and transmitting as fast as receiving it.

Still a speed limit on PC-to-modem speed even though flow doesn't take place over external cable. This makes internal modems compatible with external modems.

Flow from modem to computer also.

Each direction of flow involves 3 buffers:

1. in the modem
2. in the UART chip (called FIFOs)
3. in main memory managed by the serial driver

Flow control protects all buffers (except the FIFOs) from overflowing. Small UART FIFOs not protected by flow control but instead rely on fast response to interrupts they issue. Some UART chips can be set to do hardware flow control to protect their FIFOs.

Symptoms of No Flow Control

Chunks of data missing from files sent without benefit of flow control. When overflow happens, bytes lost in contiguous chunks.

Hardware vs. Software Flow Control

Best to use hardware flow control using two dedicated modem controls to send stop and start signals.

Hardware flow control works like this. Two pins: RTS (request to send) and CTS (clear to send) are used. When computer ready to receive data it asserts RTS by putting positive voltage on RTS pin. When not able to receive any more bytes, negates RTS by putting negative voltage on pin.

RTS pin connected by serial cable to another pin on the external device. This other pin's only function is to receive this signal. For modem, will be modem's RTS pin. For printer, another PC, non-modem device, it's usually a CTS pin so a **crossover** or **null-modem** cable is required. For a modem, a straight-through cable is used.

Example: two computers have Tx, Rx, and ground lines. If hook up Tx to Tx and Rx to Rx, won't work since one computer's Rx must be hooked to the other computer's Tx line. This means there will be a cross-over and that's how this gets the name.

For the opposite direction. For a modem, the CTS pin used to send flow contrd to the CTS pin on the PC. For a non-modem, the RTS pin sends the signal. Thus modems and non-modems have roles of RTS and CTS interchanged. Some non-modems may use other pins for flow control.

Software flow control uses the main receive and transmit data wires to send stop and start signals. Uses ASCII control characters DC1 and DC3. DC1 is 17, DC3 is 19. Inserted into regular stream of data. Software flow control slower in reacting and also does not allow sending of binary data unless special precautions are taken (distinguish between data and DC1/DC3 control characters).

Data Flow Path; Buffers

Three buffers: FIFO buffers in UART, main memory for the device driver, the modem memory.

When application program sends bytes to serial port, first kept in transmit serial port buffer in main memory. Also have a receive buffer for opposite direction.

Example diagram for browsing Internet with browser. Transmit is left to right, receive right to left.

Application	8k-byte	16-byte	1k-byte	telephone
Browser Program	Memory buffer	FIFO buffer	Modem buffer	line

- Application Browser Program
- 8k-byte Memory buffer
- 16-byte FIFO buffer
- 1k-byte Modem buffer
- telephone line