Receive algorithm, executed for each "symbol" received.

(*Symbol* usually means *byte* but could mean other numbers of bits.)

1.) Assume the line is idle. Wait for MARK to SPACE (1 to 0) transition (a rising edge in voltage)

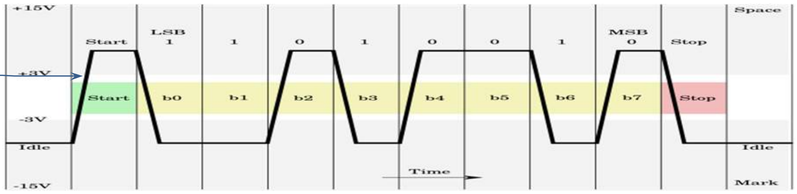


Illustration from Wikimedia Commons, used by permission CC SA 1.0

1

Receive algorithm, executed for each "symbol" received.

(*Symbol* usually means *byte* but could mean other numbers of bits.)

1.) Assume the line is idle. Wait for MARK to SPACE (1 to 0) transition (a rising edge in voltage)

2.) Upon reception of MARK to SPACE transition, start a counter synchronized to the receiver clock system. Count enough clock pulses to represent 1.5 bit times.

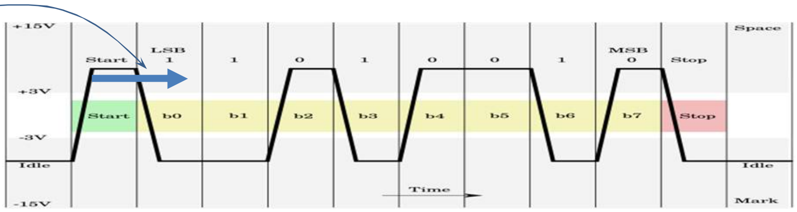


Illustration from Wikimedia Commons, used by permission CC SA 1.0

2

Receive algorithm, executed for each "symbol" received.

(*Symbol* usually means *byte* but could mean other numbers of bits.)

1.) Assume the line is idle. Wait for MARK to SPACE (1 to 0) transition (a rising edge in voltage)

2.) Upon reception of MARK to SPACE transition, start a counter synchronized to the receiver clock system. Count enough clock pulses to represent 1.5 bit times.

3.) Reset the counter to zero and read a bit. (Resetting the counter marks this point in time.)

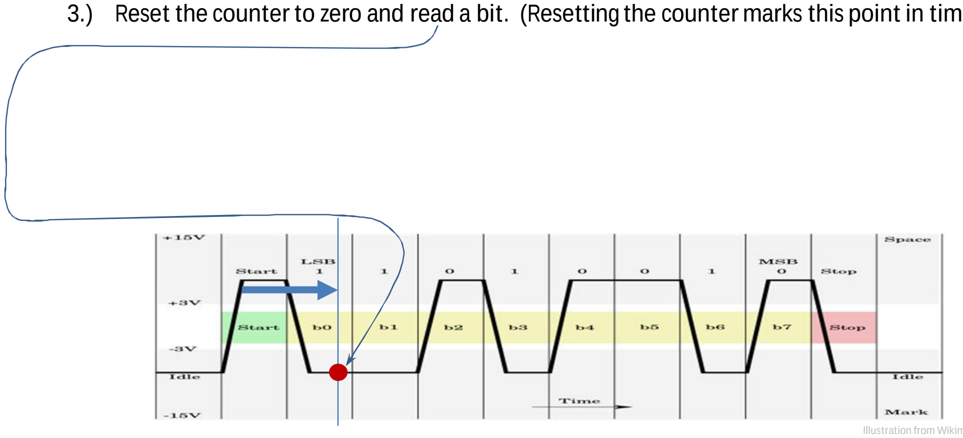


Illustration from Wikimedia Commons, used by permission CC SA 1.0

3

Receive algorithm, executed for each "symbol" received.

(*Symbol* usually means *byte* but could mean other numbers of bits.)

1.) Assume the line is idle. Wait for MARK to SPACE (1 to 0) transition (a rising edge in voltage)

2.) Upon reception of MARK to SPACE transition, start a counter synchronized to the receiver clock system. Count enough clock pulses to represent 1.5 bit times.

3.) Reset the counter to zero and read a bit. (Resetting the counter marks this point in time.)

4.) Wait for the counter to get to one bit time

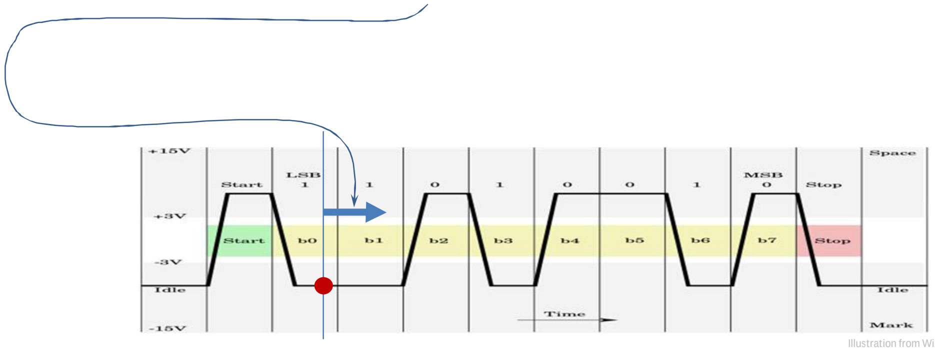


Illustration from Wikimedia Commons, used by permission CC SA 1.0

4

Receive algorithm, executed for each "symbol" received.

(*Symbol* usually means *byte* but could mean other numbers of bits.)

1.) Assume the line is idle. Wait for MARK to SPACE (1 to 0) transition (a rising edge in voltage)

2.) Upon reception of MARK to SPACE transition, start a counter synchronized to the receiver clock system. Count enough clock pulses to represent 1.5 bit times.

3.) Reset the counter to zero and read a bit. (Resetting the counter marks this point in time.)

4.) Wait for the counter to get to one bit time

5.) Repeat steps 3 and 4 until the correct number of data bits have been read.

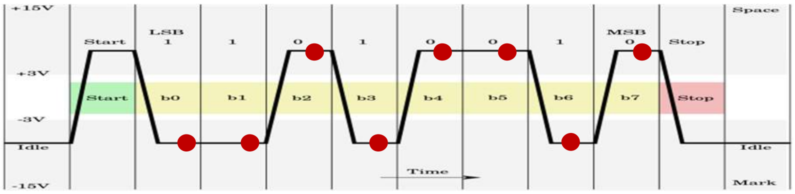


Illustration from Wikimedia Commons, used by permission CC SA 1.0

5

Receive algorithm, executed for each "symbol" received.

(*Symbol* usually means *byte* but could mean other numbers of bits.)

1.) Assume the line is idle. Wait for MARK to SPACE (1 to 0) transition (a rising edge in voltage)

2.) Upon reception of MARK to SPACE transition, start a counter synchronized to the receiver clock system. Count enough clock pulses to represent 1.5 bit times.

3.) Reset the counter to zero and read a bit. (Resetting the counter marks this point in time.)

4.) Wait for the counter to get to one bit time

5.) Repeat steps 3 and 4 until the correct number of data bits have been read.

If rx clock is running fast

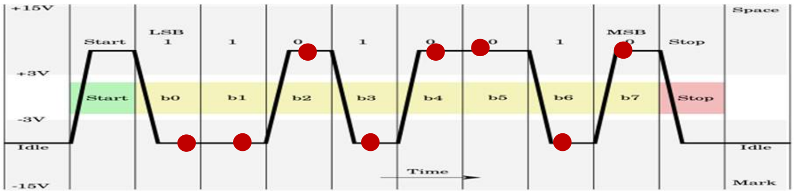


Illustration from Wikimedia Commons, used by permission CC SA 1.0

6

Receive algorithm, executed for each “symbol” received.

(*Symbol* usually means *byte* but could mean other numbers of bits.)

1.) Assume the line is idle. Wait for MARK to SPACE (1 to 0) transition (a rising edge in voltage)

2.) Upon reception of MARK to SPACE transition, start a counter synchronized to the receiver clock system. Count enough clock pulses to represent 1.5 bit times.

3.) Reset the counter to zero and read a bit. (Resetting the counter marks this point in time.)

4.) Wait for the counter to get to one bit time

5.) Repeat steps 3 and 4 until the correct number of data bits have been read.

6.) Send buffered word, then return to step one. (In preparation to read the next word of data.)

Word illustrated here is 01001011

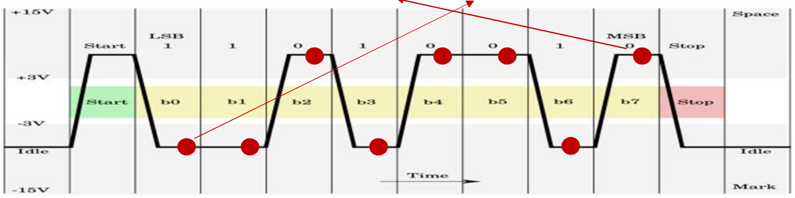


Illustration from Wikimedia Commons, used by permission CC SA 1.0

7

Using a serial port in a particular set of hardware usually involves borrowing or purchasing “Intellectual Property.”

Here is one example for the Raspberry Pi

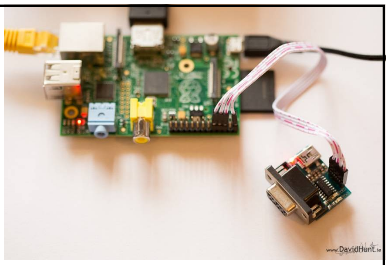
http://www.elinux.org/Serial_port_programming

```
import serial
import time

def readlineCR(port):
    rv = ""
    while True:
        ch = port.read()
        rv += ch
        if ch=='\r' or ch=='':
            return rv

port = serial.Serial("/dev/ttyAMA0", baudrate=115200, timeout=3.0)

while True:
    port.write("\r\nSay something:")
    rcv = readlineCR(port)
    port.write("\r\nYou sent:" + repr(rcv))
```

8

UART hardware vs. Software *Bit-Banged*, serial ports

Serial protocols such as RS-232 usually rely on **hardware support to load a register in parallel** (like any register in a CPU) and then automatically, under **hardware-state-machine control, shift out one bit at a time to a pin for serial transmission.** Or, vice-versa, shift in one bit at a time and then allow the CPU to read a whole word as one parallel I/O operation. This hardware support for parallel-to-serial and serial-to-parallel conversion is call a **UART** (Pronounced as a word: "You art.") The initialism stands for "Universal Asynchronous Receiver-Transmitter." In addition to the UART, hardware level shifters may be needed at the GPIO pins. For example RS232 uses ±12 V signaling but a Raspberry Pi uses 0 to 3.3 V signaling.

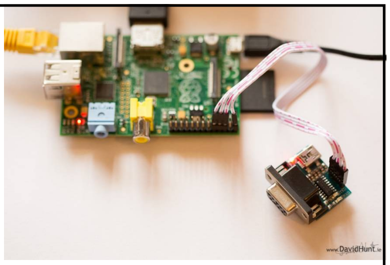


http://www.davidhunt.ie/add-a-9-pin-serial-port-to-your-raspberry-pi-in-10-minutes/

9

UART hardware vs. Software *Bit-Banged*, serial ports

Serial protocols such as RS-232 usually rely on **hardware support to load a register in parallel** (like any register in a CPU) and then automatically, under **hardware-state-machine control, shift out one bit at a time to a pin for serial transmission.** Or, vice-versa, shift in one bit at a time and then allow the CPU to read a whole word as one parallel I/O operation. This hardware support for parallel-to-serial and serial-to-parallel conversion is call a **UART** (Pronounced as a word: "You art.") The initialism stands for "Universal Asynchronous Receiver-Transmitter." In addition to the UART, hardware level shifters may be needed at the GPIO pins. For example RS232 uses ±12 V signaling but a Raspberry Pi uses 0 to 3.3 V signaling.

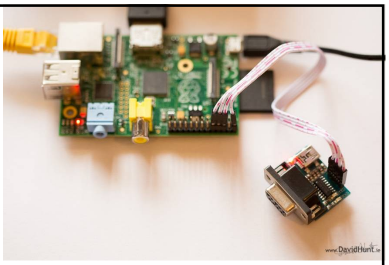


When no UART hardware is available, the function of a UART can be programmed using a pair of digital I/O pins. (The level-shifting of course must be done with hardware. **Using software to replace UART hardware is called creating a *bit-banged* serial port**. The phrase "bit-banged" is colloquial, but fairly well recognized.

http://www.davidhunt.ie/add-a-9-pin-serial-port-to-your-raspberry-pi-in-10-minutes/

10

UART hardware vs. Software *Bit-Banged*, serial ports



Serial protocols such as RS-232 usually rely on **hardware support to load a register in parallel** (like any register in a CPU) and then automatically, under **hardware-state-machine control, shift out one bit at a time to a pin for serial transmission.** Or, vice-versa, shift in one bit at a time and then allow the CPU to read a whole word as one parallel I/O operation. This hardware support for parallel-to-serial and serial-to-parallel conversion is call a **UART** (Pronounced as a word: "You art.") The initialism stands for "Universal Asynchronous Receiver-Transmitter." In addition to the UART, hardware level shifters may be needed at the GPIO pins. For example RS232 uses ±12 V signaling but a Raspberry Pi uses 0 to 3.3 V signaling.

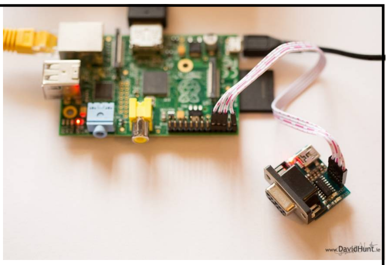
When no UART hardware is available, the function of a UART can be programmed using a pair of digital I/O pins. (The level-shifting of course must be done with hardware. **Using software to replace UART hardware is called creating a *bit-banged* serial port**. The phrase "bit-banged" is colloquial, but fairly well recognized.

There are libraries for bit-banged serial I/O for the Arduino, the Raspberry Pi, and most other platforms. However, these **libraries usually take over several internal chip resources (timers, etc) that cannot then be used for other purposes.** Although it is often considered to be something of a hack, bit banging does allow the same device to use different protocols with minimal or no hardware changes required. Also, most platforms only have one or a few UARTs. Some have none. By using bit-banging one can have more serial interfaces than there are UARTs.

http://www.davidhunt.ie/add-a-9-pin-serial-port-to-your-raspberry-pi-in-10-minutes/

11

UART hardware vs. Software *Bit-Banged*, serial ports



Serial protocols such as RS-232 usually rely on **hardware support to load a register in parallel** (like any register in a CPU) and then automatically, under **hardware-state-machine control, shift out one bit at a time to a pin for serial transmission.** Or, vice-versa, shift in one bit at a time and then allow the CPU to read a whole word as one parallel I/O operation. This hardware support for parallel-to-serial and serial-to-parallel conversion is call a **UART** (Pronounced as a word: "You art.") The initialism stands for "Universal Asynchronous Receiver-Transmitter." In addition to the UART, hardware level shifters may be needed at the GPIO pins. For example RS232 uses ±12 V signaling but a Raspberry Pi uses 0 to 3.3 V signaling.

When no UART hardware is available, the function of a UART can be programmed using a pair of digital I/O pins. (The level-shifting of course must be done with hardware. **Using software to replace UART hardware is called creating a *bit-banged* serial port**. The phrase "bit-banged" is colloquial, but fairly well recognized.

There are libraries for bit-banged serial I/O for the Arduino, the Raspberry Pi, and most other platforms. However, these **libraries usually take over several internal chip resources (timers, etc) that cannot then be used for other purposes.** Although it is often considered to be something of a hack, bit banging does allow the same device to use different protocols with minimal or no hardware changes required. Also, most platforms only have one or a few UARTs. Some have none. By using bit-banging one can have more serial interfaces than there are UARTs.

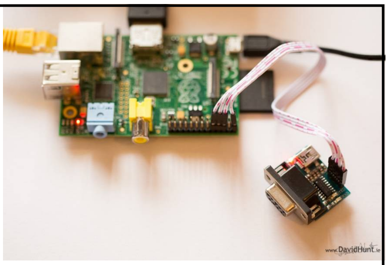
There are some drawbacks to bit banging. The software emulation process **consumes more processing power** than does supporting dedicated hardware. The microcontroller spends much of its time reading or sending samples to and from the port, at the expense of other tasks. The signal produced usually has more jitter (timing uncertainty), especially if the processor is also executing other tasks while communicating. However, **if the bit-banging software is interrupt-driven by the signal, this may be of minor importance.**
–Paraphrased from Wikipedia

http://www.davidhunt.ie/add-a-9-pin-serial-port-to-your-raspberry-pi-in-10-minutes/

12

SUMMARY SLIDE

UART hardware vs. Software *Bit-Banged*, serial ports



Serial protocols such as RS-232 usually rely on **hardware support to load a register in parallel** (like any register in a CPU) and then automatically, under **hardware-state-machine control, shift out one bit at a time to a pin for serial transmission.** Or, vice-versa, shift in one bit at a time and then allow the CPU to read a whole word as one parallel I/O operation. This hardware support for parallel-to-serial and serial-to-parallel conversion is call a **UART** (Pronounced as a word: "You art.") The initialism stands for "Universal Asynchronous Receiver-Transmitter." In addition to the UART, hardware level shifters may be needed at the GPIO pins. For example RS232 uses ±12 V signaling but a Raspberry Pi uses 0 to 3.3 V signaling.

When no UART hardware is available, the function of a UART can be programmed using a pair of digital I/O pins. (The level-shifting of course must be done with hardware. **Using software to replace UART hardware is called creating a *bit-banged* serial port**. The phrase "bit-banged" is colloquial, but fairly well recognized.

There are libraries for bit-banged serial I/O for the Arduino, the Raspberry Pi, and most other platforms. However, these **libraries usually take over several internal chip resources (timers, etc) that cannot then be used for other purposes.** Although it is often considered to be something of a hack, bit banging does allow the same device to use different protocols with minimal or no hardware changes required. Also, most platforms only have one or a few UARTs. Some have none. By using bit-banging one can have more serial interfaces than there are UARTs.

There are some drawbacks to bit banging. The software emulation process **consumes more processing power** than does supporting dedicated hardware. The microcontroller spends much of its time reading or sending samples to and from the port, at the expense of other tasks. The signal produced usually has more jitter (timing uncertainty), especially if the processor is also executing other tasks while communicating. However, **if the bit-banging software is interrupt-driven by the signal, this may be of minor importance.**
–Paraphrased from Wikipedia

http://www.davidhunt.ie/add-a-9-pin-serial-port-to-your-raspberry-pi-in-10-minutes/

13

Configurable link set-up parameters with RS-232

Full Duplex vs. Half Duplex

Full—can transmit in both directions at once on separate Tx and Rx wires.
RS-232 is designed to be full-duplex
Sometimes one wire is omitted, link is a one-way link, which is a type of half-duplex.
Sometimes the link has both Tx and Rx lines but other shared buffers or resources can only transfer data in one direction at a time. The link is then half-duplex.
Half-Duplex—either one-way or it works like a walk-talkie in that only one direction may be used at a time.

14

Configurable link set-up parameters with RS-232

Full Duplex vs. Half Duplex

Full—can transmit in both directions at once on separate Tx and Rx wires.

RS-232 is designed to be full-duplex

Sometimes one wire is omitted, link is a one-way link, which is a type of half-duplex.

Sometimes the link has both Tx and Rx lines but other shared buffers or resources can only transfer data in one direction at a time. The link is then half-duplex.

Half-Duplex—either one-way or it works like a walk-talkie in that only one direction may be used at a time.

Echo or Local Echo

Echo—this is the intended mode. The data must make a round-trip before being printed (echoed) at the source end. This is done so that any corruption can be observed by the sender.

Local Echo—Message is sent but never returned to the sender. The message is copied directly from Tx to Rx at the source.

if Tx is set to Local Echo but Rx is set to Echo, then "HHeelloo WWoorrlllldd" at Tx end (Rx end is OK)

if Tx is set to Echo but Rx is set to Local Echo, then "" at Tx end (Rx end is OK)

15

Configurable link set-up parameters with RS-232

Full Duplex vs. Half Duplex

Full—can transmit in both directions at once on separate Tx and Rx wires.

RS-232 is designed to be full-duplex

Sometimes one wire is omitted, link is a one-way link, which is a type of half-duplex.

Sometimes the link has both Tx and Rx lines but other shared buffers or resources can only transfer data in one direction at a time. The link is then half-duplex.

Half-Duplex—either one-way or it works like a walk-talkie in that only one direction may be used at a time.

Echo or Local Echo

Echo—this is the intended mode. The data must make a round-trip before being printed (echoed) at the source end. This is done so that any corruption can be observed by the sender.

Local Echo—Message is sent but never returned to the sender. The message is copied directly from Tx to Rx at the source.

if Tx is set to Local Echo but Rx is set to Echo, then "HHeelloo WWoorrlllldd" at Tx end (Rx end is OK)

if Tx is set to Echo but Rx is set to Local Echo, then "" at Tx end (Rx end is OK)

Local Editing

Message goes into a buffer until the "return" key is struck or buffer is full. If buffer is full, then send oldest character in buffer. If return is struck, then the buffer is streamed out until empty. There is also provision to edit that which is still in the buffer. Usually the provision is the "backspace" key only.

16

SUMMARY SLIDE

Configurable link set-up parameters with RS-232

Full Duplex vs. Half Duplex

Full—can transmit in both directions at once on separate Tx and Rx wires.
RS-232 is designed to be full-duplex
Sometimes one wire is omitted, link is a one-way link, which is a type of half-duplex.
Sometimes the link has both Tx and Rx lines but other shared buffers or resources can only transfer data in one direction at a time. The link is then half-duplex.
Half-Duplex—either one-way or it works like a walk-talkie in that only one direction may be used at a time.

Echo or Local Echo

Echo—this is the intended mode. The data must make a round-trip before being printed (echoed) at the source end. This is done so that any corruption can be observed by the sender.
Local Echo—Message is sent but never returned to the sender. The message is copied directly from Tx to Rx at the source.
if Tx is set to Local Echo but Rx is set to Echo, then “HHeelloo WWoorrlllldd” at Tx end (Rx end is OK)
if Tx is set to Echo but Rx is set to Local Echo, then “” at Tx end (Rx end is OK)

Local Editing

Message goes into a buffer until the “return” key is struck or buffer is full. If buffer is full, then send oldest character in buffer. If return is struck, then the buffer is streamed out until empty. There is also provision to edit that which is still in the buffer. Usually the provision is the “backspace” key only.

17

Configurable link set-up parameters with RS-232

←More parameters!
A continuation of the previous slide.

DTE or DCE

Does this device act like a terminal (DTE) or a modem (DCE)? Essentially configures the data direction.

18

Configurable link set-up parameters with RS-232

DTE or DCE

Does this device act like a terminal (DTE) or a modem (DCE)? Essentially configures the data direction.

Baud rate

Baud rate for RS-232 is actually the clock rate that sends individual bits.
(DDB thinks this is an abuse of the word "baud," which should refer to information rate. Call it Tx clock rate.)
Standard rates are 30, 60, 75, 110, 150, 300, 330, 512, 600, 1200, 2400, 4800, 9600, 19200, 38400, 56000
There are some higher rates occasionally used. 115200, 128000, etc up to 9216000 but these are outside the standard. Actually, practically any rate can be found since many applications of RS-232 deviate from the standard.

19

Configurable link set-up parameters with RS-232

DTE or DCE

Does this device act like a terminal (DTE) or a modem (DCE)? Essentially configures the data direction.

Baud rate

Baud rate for RS-232 is actually the clock rate that sends individual bits.
(DDB thinks this is an abuse of the word "baud," which should refer to information rate. Call it Tx clock rate.)
Standard rates are 30, 60, 75, 110, 150, 300, 330, 512, 600, 1200, 2400, 4800, 9600, 19200, 38400, 56000
There are some higher rates occasionally used. 115200, 128000, etc up to 9216000 but these are outside the standard. Actually, practically any rate can be found since many applications of RS-232 deviate from the standard.

Bits per word

5, 7, or 8 bits per word are allowed. (Almost always 8 bits/word in modern applications.)

Parity Even or odd or none.
ASCII is a 7-bit/word code. When sending ASCII, fill the 8th bit with parity or logic-0 (none).
If "even" is selected, fill the 8th bit such that there is an even number of "1" bits in the entire 8-bit word.

Stop bits 1, 1.5, or 2 bits long. Longer stop bits speed acquisition of synchronism. If 1 bit, must wait for idle time.

20

Configurable link set-up parameters with RS-232

DTE or DCE

Does this device act like a terminal (DTE) or a modem (DCE)? Essentially configures the data direction.

Baud rate

Baud rate for RS-232 is actually the clock rate that sends individual bits.
(DDB thinks this is an abuse of the word "baud," which should refer to information rate. Call it Tx clock rate.)
Standard rates are 30, 60, 75, 110, 150, 300, 330, 512, 600, 1200, 2400, 4800, 9600, 19200, 38400, 56000
There are some higher rates occasionally used. 115200, 128000, etc up to 9216000 but these are outside the standard. Actually, practically any rate can be found since many applications of RS-232 deviate from the standard.

Bits per word

5, 7, or 8 bits per word are allowed. (Almost always 8 bits/word in modern applications.)

Parity Even or odd or none.
ASCII is a 7-bit/word code. When sending ASCII, fill the 8th bit with parity or logic-0 (none).
If "even" is selected, fill the 8th bit such that there is an even number of "1" bits in the entire 8-bit word.

Stop bits 1, 1.5, or 2 bits long. Longer stop bits speed acquisition of synchronism. If 1 bit, must wait for idle time.

Conventional notation for the above 3: bits/parity/stop, e.g. 8/n/1 is common, aka 8N1 (←sometimes called a *configuration word*.)

21

SUMMARY SLIDE

Configurable link set-up parameters with RS-232

DTE or DCE

Does this device act like a terminal (DTE) or a modem (DCE)? Essentially configures the data direction.

Baud rate

Baud rate for RS-232 is actually the clock rate that sends individual bits.
(DDB thinks this is an abuse of the word "baud," which should refer to information rate. Call it Tx clock rate.)
Standard rates are 30, 60, 75, 110, 150, 300, 330, 512, 600, 1200, 2400, 4800, 9600, 19200, 38400, 56000
There are some higher rates occasionally used. 115200, 128000, etc up to 9216000 but these are outside the standard. Actually, practically any rate can be found since many applications of RS-232 deviate from the standard.

Bits per word

5, 7, or 8 bits per word are allowed. (Almost always 8 bits/word in modern applications.)

Parity Even or odd or none.
ASCII is a 7-bit/word code. When sending ASCII, fill the 8th bit with parity or logic-0 (none).
If "even" is selected, fill the 8th bit such that there is an even number of "1" bits in the entire 8-bit word.

Stop bits 1, 1.5, or 2 bits long. Longer stop bits speed acquisition of synchronism. If 1 bit, must wait for idle time.

Conventional notation for the above 3: bits/parity/stop, e.g. 8/n/1 is common, aka 8N1 (←sometimes called a *configuration word*.)

22

Configurable link set-up parameters with RS-232

←R U telling me there are MORE parameters!?
A continuation of the previous slide.

Flow Control

XON/XOFF—software flow control by sending special ASCII characters that are intercepted and used to control flow. E.g. in a full-duplex link, remote terminal sends "XOFF" and local terminal stops sending. Later remote terminal sends XON and local terminal resumes sending. Requires at least a 3-wire connection.

Hardware—uses RTS and CTS. Requires at least a 5-wire connection.

23

Configurable link set-up parameters with RS-232

←R U telling me there are MORE parameters!
A continuation of the previous slide.

Flow Control

XON/XOFF—software flow control by sending special ASCII characters that are intercepted and used to control flow. E.g. in a full-duplex link, remote terminal sends "XOFF" and local terminal stops sending. Later remote terminal sends XON and local terminal resumes sending. Requires at least a 3-wire connection.

Hardware—uses RTS and CTS. Requires at least a 5-wire connection.

OK, that's it for configurable link set-up parameters.

24

Configurable hardware with RS-232

←R U telling me there are HARDWARE VARIATIONS TOO!?

25, 9, 5, 3, 2 wire connection

25

Configurable hardware with RS-232

←R U telling me there are HARDWARE VARIATIONS TOO!?

25, 9, 5, 3, 2 wire connection

“Straight through” or “null modem” cable.
(Null modem crosses Rx and Tx, and RTS CTS, and DTR, DSR if present)

26

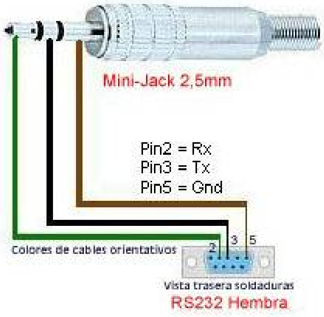
Configurable hardware with RS-232

←R U telling me there are HARDWARE VARIATIONS TOO!?

25, 9, 5, 3, 2 wire connection

“Straight through” or “null modem” cable.
(Null modem crosses Rx and Tx, and RTS CTS, and DTR, DSR if present)

Type of connector (DB-9, RJ-11, DIN, etc. Practically everything has been used!)



27

SUMMARY SLIDE

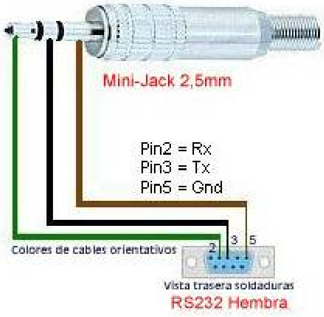
Configurable hardware with RS-232

←R U telling me there are HARDWARE VARIATIONS TOO!?

25, 9, 5, 3, 2 wire connection

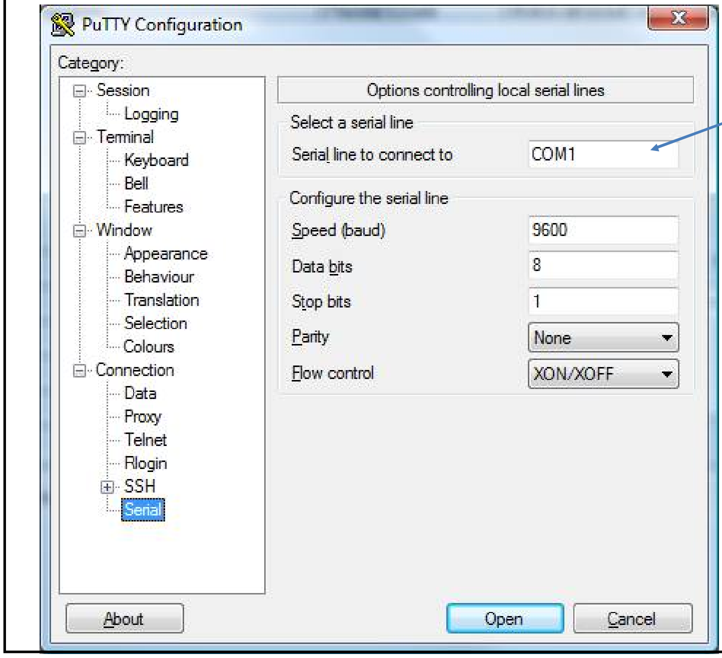
“Straight through” or “null modem” cable.
(Null modem crosses Rx and Tx, and RTS CTS, and DTR, DSR if present)

Type of connector (DB-9, RJ-11, DIN, etc. Practically everything has been used!)



28

A typical RS-232 configuration screen



Options controlling local serial lines

Select a serial line

Serial line to connect to: COM1

Configure the serial line

Speed (baud): 9600

Data bits: 8

Stop bits: 1

Parity: None

Flow control: XON/XOFF

Hardware address of the UART. Something equivalent to this is present on every system. This ties the port to the OS. It is part of the OS (Windows here) not RS-232.

Most OS's can support several RS-232 ports. Windows, up to about 256, depending on your version of Windows. On windows the first address available is COM1

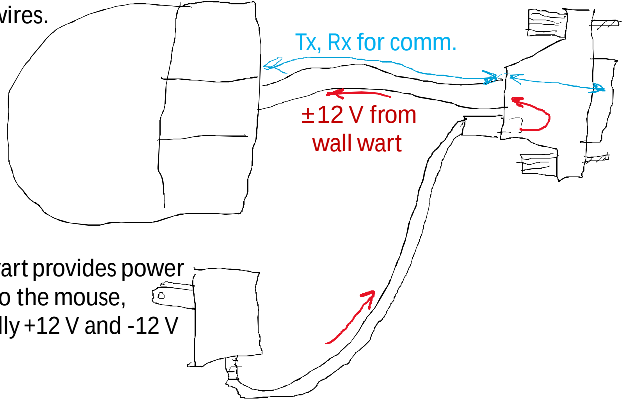
On Linux (Raspbian) these are called ttyS0, ttyS1, etc. On Raspbian, the first address available is ttyS0. Raspberry pi has only one UART, 40 GPIO pins, so there will not be 256 of these ports!

29

Power over RS-232

Example: A mouse that interfaces to the computer via RS-232
The mouse needs power. Early mouse used a wall-wart power supply to power the mouse.
The wall-wart plugged into the back of the RS-232 plug that attached to the computer.

Mouse receives power and communicates via RS-232 via one cable having several wires.



RS-232 connected to a COM port of the computer.

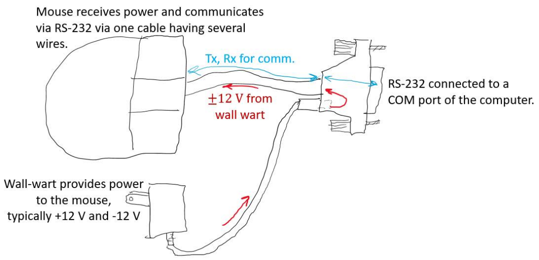
Wall-wart provides power to the mouse, typically +12 V and -12 V

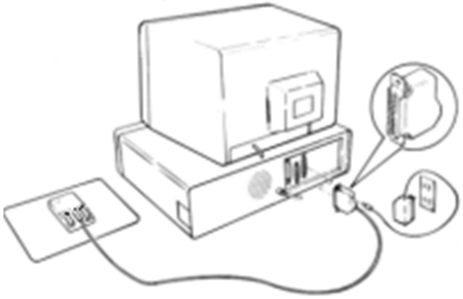
30

Power over RS-232

Example: A mouse that interfaces to the computer via RS-232
The mouse needs power. Early mouse used a wall-wart power supply to power the mouse.
The wall-wart plugged into the back of the RS-232 plug that attached to the computer.

Mouse systems model M-2 optical mouse, 1982.


Mouse receives power and communicates via RS-232 via one cable having several wires.
Tx, Rx for comm.
±12 V from wall wart
RS-232 connected to a COM port of the computer.
Wall-wart provides power to the mouse, typically +12 V and -12 V

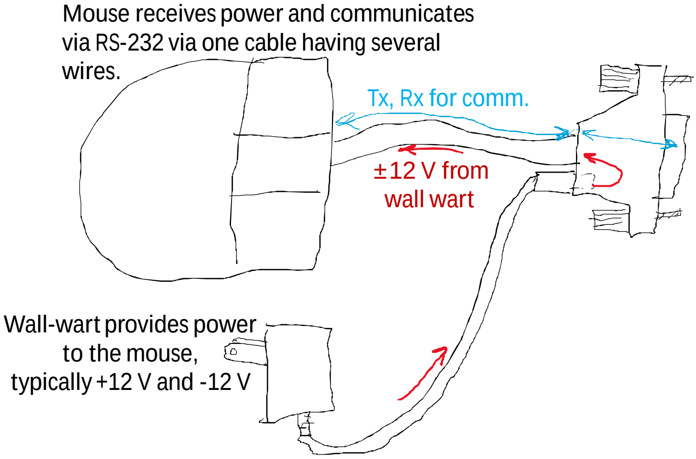


https://www.oldmouse.com/mouse/mousesystems/m2.shtml

31

Power over RS-232

Example: A mouse that interfaces to the computer via RS-232
The mouse needs power. Early mouse used a wall-wart power supply to power the mouse.
The wall-wart plugged into the back of the RS-232 plug that attached to the computer.


Mouse receives power and communicates via RS-232 via one cable having several wires.
Tx, Rx for comm.
±12 V from wall wart
RS-232 connected to a COM port of the computer.
Wall-wart provides power to the mouse, typically +12 V and -12 V

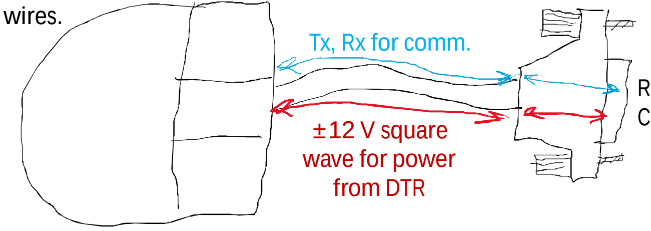
This era did not last long.
The wall-wart was quite an annoyance.

32

Power over RS-232

Example: A mouse that interfaces to the computer via RS-232
The mouse needs power. Have the mouse driver in the computer toggle the DTR line periodically between MARK and SPACE. If the computer is acting as a DTE (terminal) device with ±12 V signaling this creates a square wave that is sent to the mouse. Rectify and filter it to +12 V and −12 V to power the mouse.

Mouse receives power and communicates via RS-232 via one cable having several wires.



Tx, Rx for comm.

±12 V square wave for power from DTR

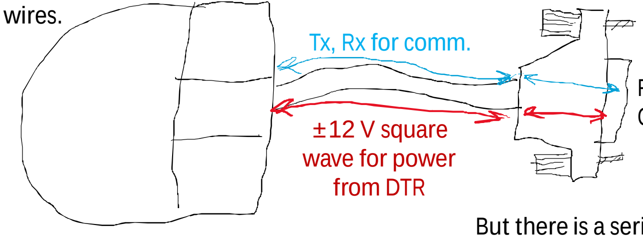
RS-232 connected to a COM port of the computer.

33

Power over RS-232

Example: A mouse that interfaces to the computer via RS-232
The mouse needs power. Have the mouse driver in the computer toggle the DTR line periodically between MARK and SPACE. If the computer is acting as a DTE (terminal) device with ±12 V signaling this creates a square wave that is sent to the mouse. Rectify and filter it to +12 V and −12 V to power the mouse.

Mouse receives power and communicates via RS-232 via one cable having several wires.



Tx, Rx for comm.

±12 V square wave for power from DTR

RS-232 connected to a COM port of the computer.

A new era in RS-232 signaling was born! This is totally outside the standard. The DTR signal is not being used in the way the standard specifies.

But there is a serious limitation. The DTR driver output impedance is nominally 300 Ω. For a less than 1 V loss the current delivered must be less than about 3.3 mA. At ±11 V delivered that is about a 35 mW power limit. The mouse (or other device) must require less than 35 mW of power—not much more than is required to send signals back to the computer! Never-the-less, “power over RS-232 is pretty common.”

34