Referencia y aplicaciones del Puerto Serie (TIA/EIA-232) en entornos Linux y win

¿El puerto serie aun existe?

Actualmente es **infrecuente** encontrar una PC destinada a hogar u oficina con un conector para puerto serie: en los 90 se usaban para conectar el mouse...

Sin embargo se fabrican y venden adaptadores de USB a 232 que son comunes, baratos y fáciles de obtener...

¡El puerto serie aun existe!

¿Para qué se usa actualmente?

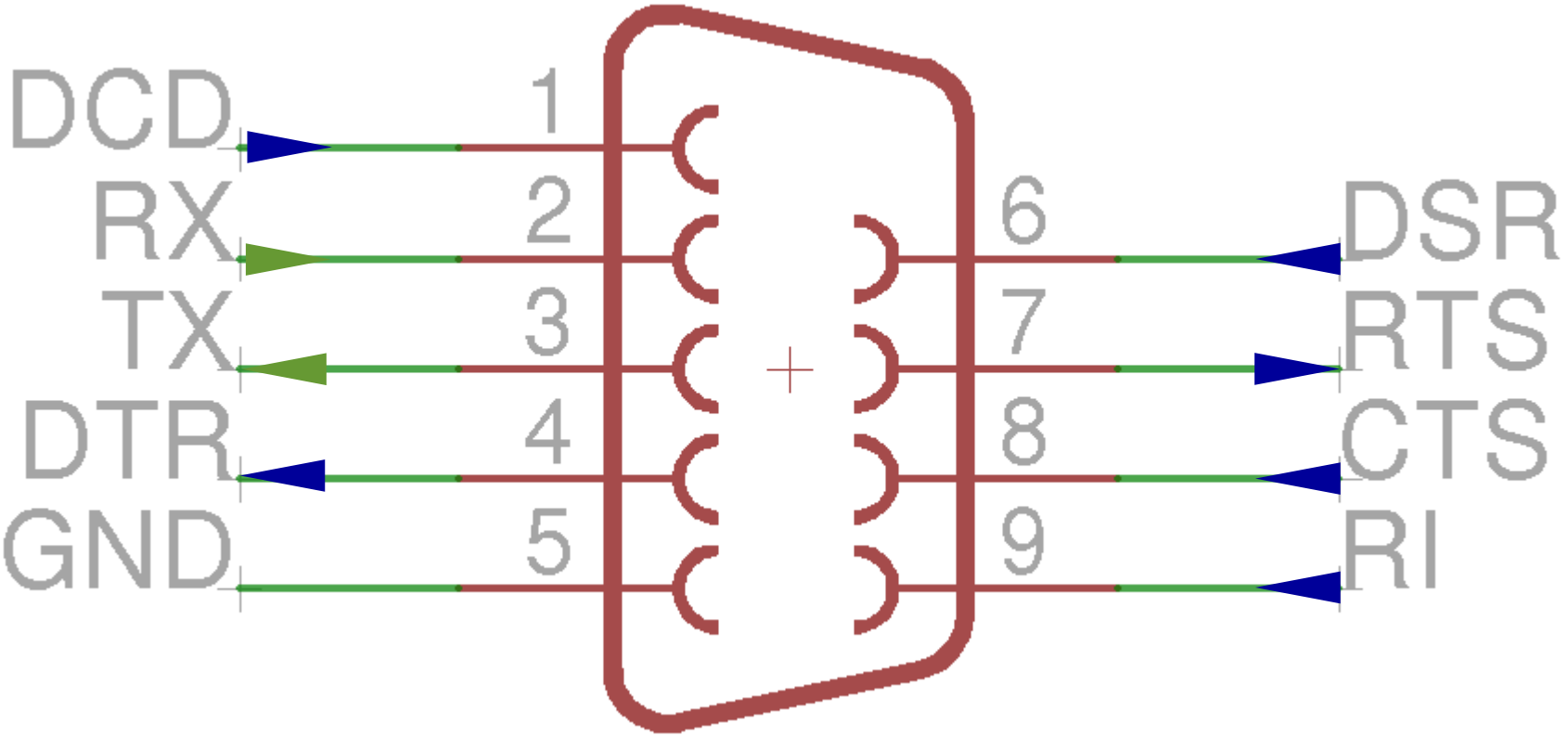
Para comunicarse con muchos dispositivos:

- GPS
- Módulos GPRS / GSM ... (comunicación celular)
- PLC (Controlador Lógico Programable)
- Microcontroladores (Ej. “Arduino”)
- Sistemas embebidos (Ej. “Raspberry Pi”)
- Lectores de códigos de barras
- Lectores-grabadores de tarjetas magnéticas y tags RFID
- Impresoras
- Sensores

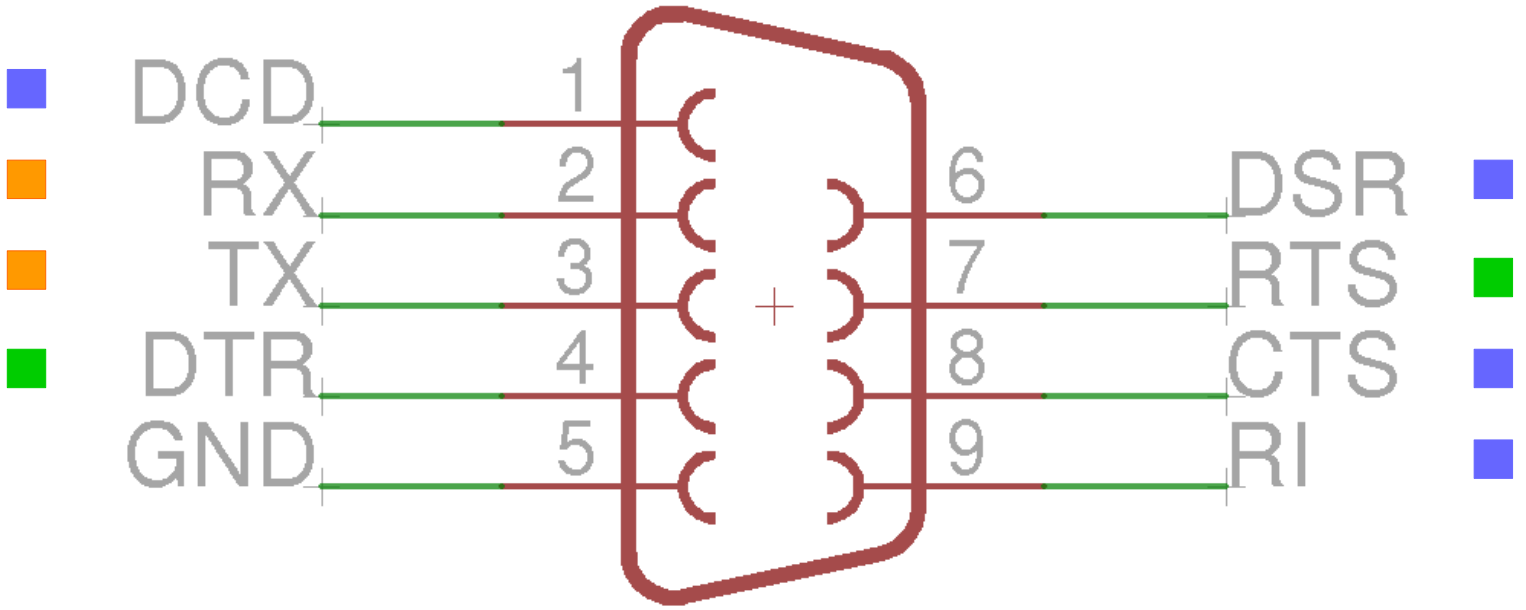
ETC.

Conector DB 9 – Aspecto y pines

El conector más utilizado actualmente es el DB9 / D-sub 9. Tiene 9* conexiones:

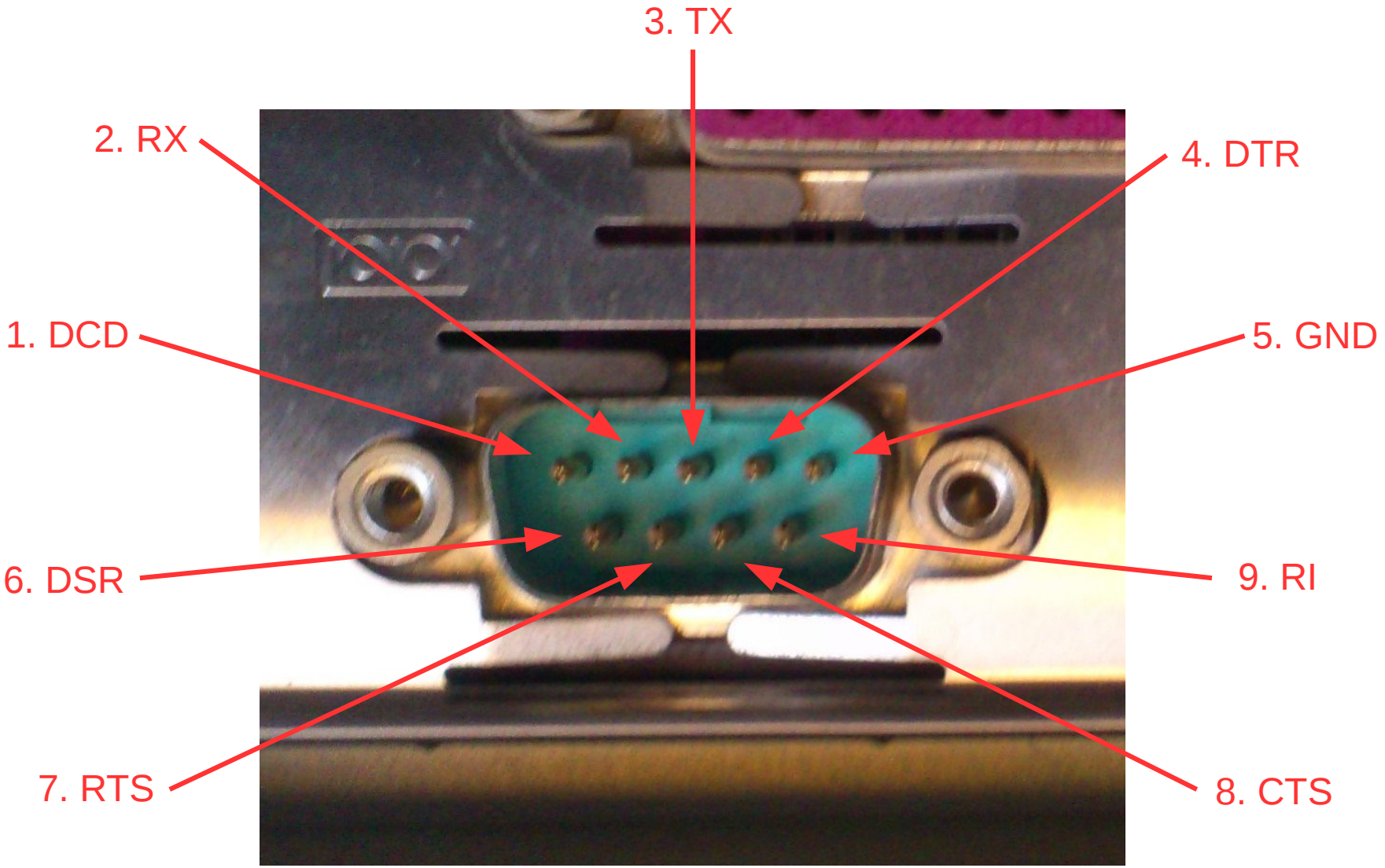


Conector DB 9 – Aspecto y pines



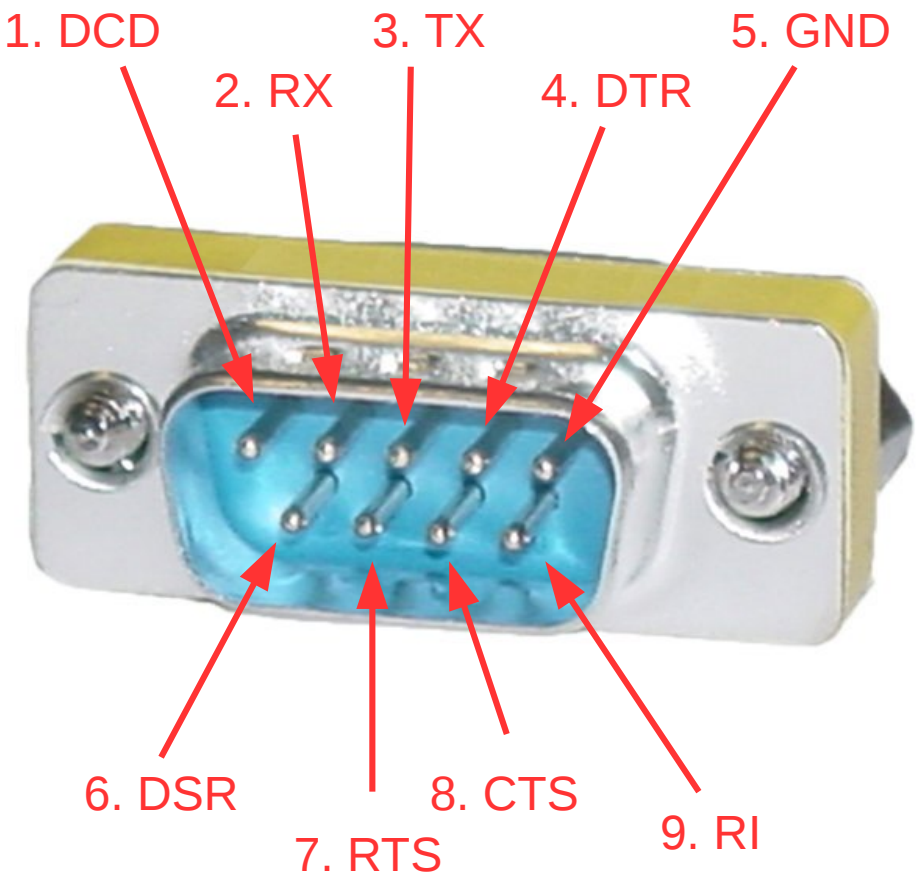
- Comunicación serie: RX recepción, TX transmisión
- Handshaking – salidas: DTR (Data Terminal Ready), RTS (Request To Send)
- Handshaking – entradas: DCD (Data Carrier Detect), DSR (Data Set Ready), CTS (Clear To Send), RI (Ring Indicator)

Conector D-Sub 9 – Aspecto y pines

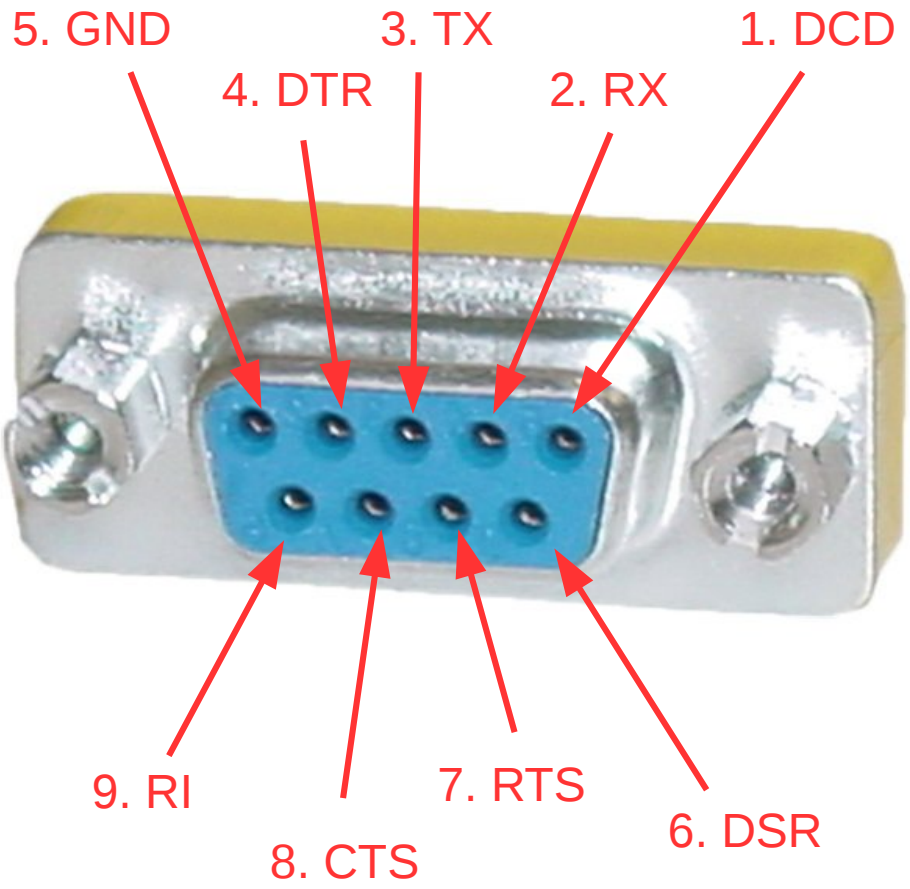


DB-9 Macho en la parte posterior de una PC de escritorio

Conector D-Sub 9 – Aspecto y pines



DB-9 Macho



DB-9 Hembra

Características eléctricas

- Máxima V en pines: ±25 V
- “Cero” lógico emisor: entre 5 V y 15 V / receptor: 3V y 15 V
- “Uno” lógico emisor: entre -5 V y -15 V / receptor: -3V y -15 V

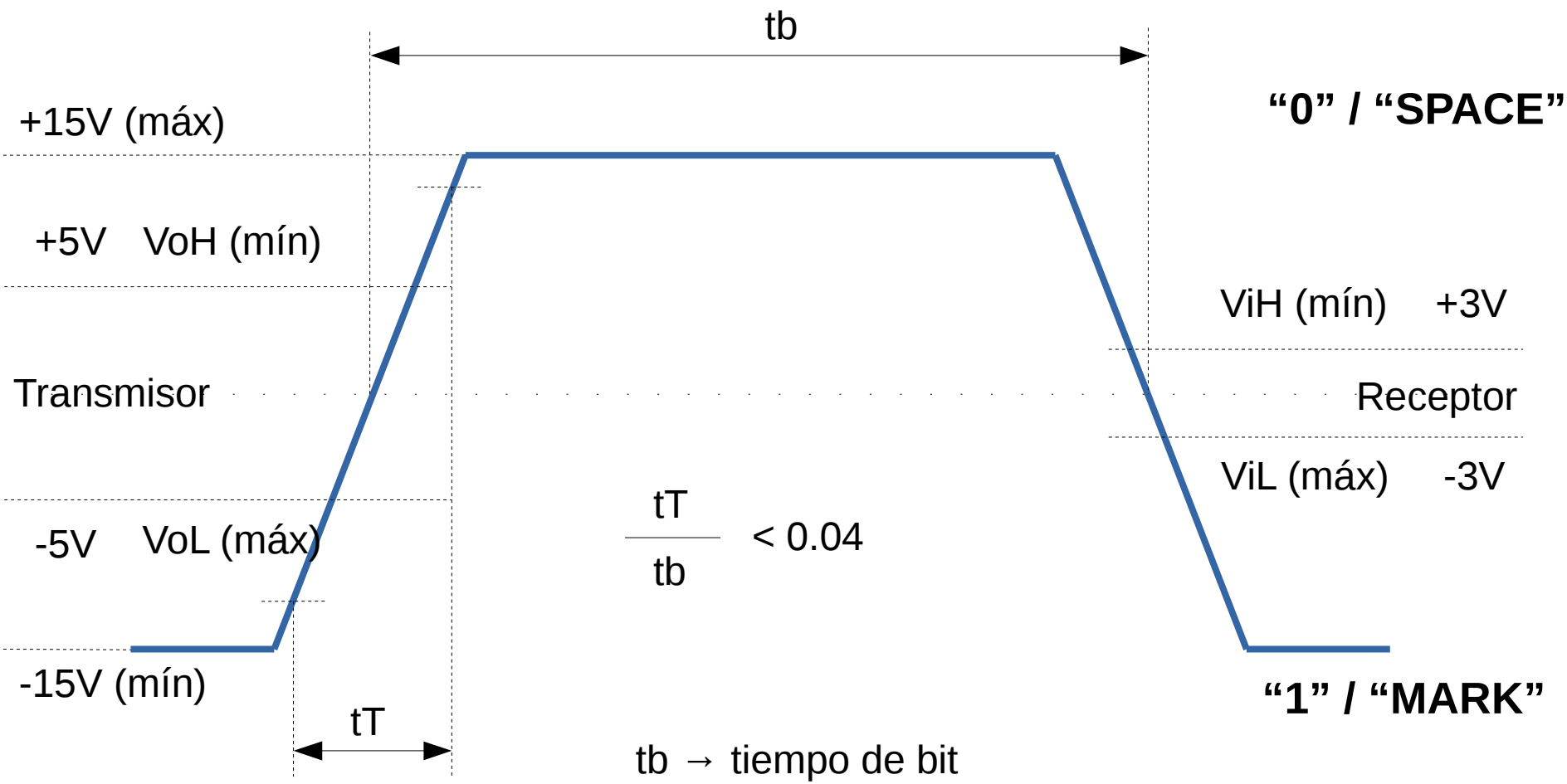
2V de margen de ruido (mínimo)

- Zi del receptor entre 3 KΩ y 7KΩ
- CL < 2500 pF (máx.) → Cables + receptor: Determina longitud máxima del cable*
- Slew rate 30 V/µs (máx.) → Para limitar crosstalk
- Velocidad máxima de transmisión: 20 kbit/s. (Sin exceder el Slew Rate máximo)
- Relación t de transición / t bit = 0,04 (4%)

* La longitud máxima del cable depende de la C del cable (calidad, si es blindado o no)
A mayor C, se necesita mayor Io en el driver para mantener la relación tt/tb de 4% por lo que también la C de la línea limita la velocidad a la que se desea transmitir

(Fuente: Texas Instruments - SLLA037A)

Características eléctricas



tb → tiempo de bit

Depende de la velocidad de transmisión.

Características eléctricas

- En la práctica los puertos de las PC “hogareñas” y adaptadores USB-232 presentan tensiones máximas < a 12V y mínimas > a -12V sin carga.

- La Vo+ baja (Y la Vo- sube) al aumentar la Io.

- Por ejemplo, en el adaptador USB-232 utilizado para realizar este documento:
    - Sin carga (con Io = 0) la Vo+ es 9V y Vo- es -8.9V
    - Con Io = 7.3 mA, la Vo es 7.3 V

(Fuente: Texas Instruments - SLLA037A)

Características eléctricas

- La velocidad de transmisión se expresa en bits / s o baud (Bd)*
- 1 baud = 1bit / s *
- tb, duración de símbolo, intervalo de unidad = 1 / velocidad Bd

Ej.:

Velocidad 9600 Bd, un bit dura 1 / 9600 = 104.167 µs

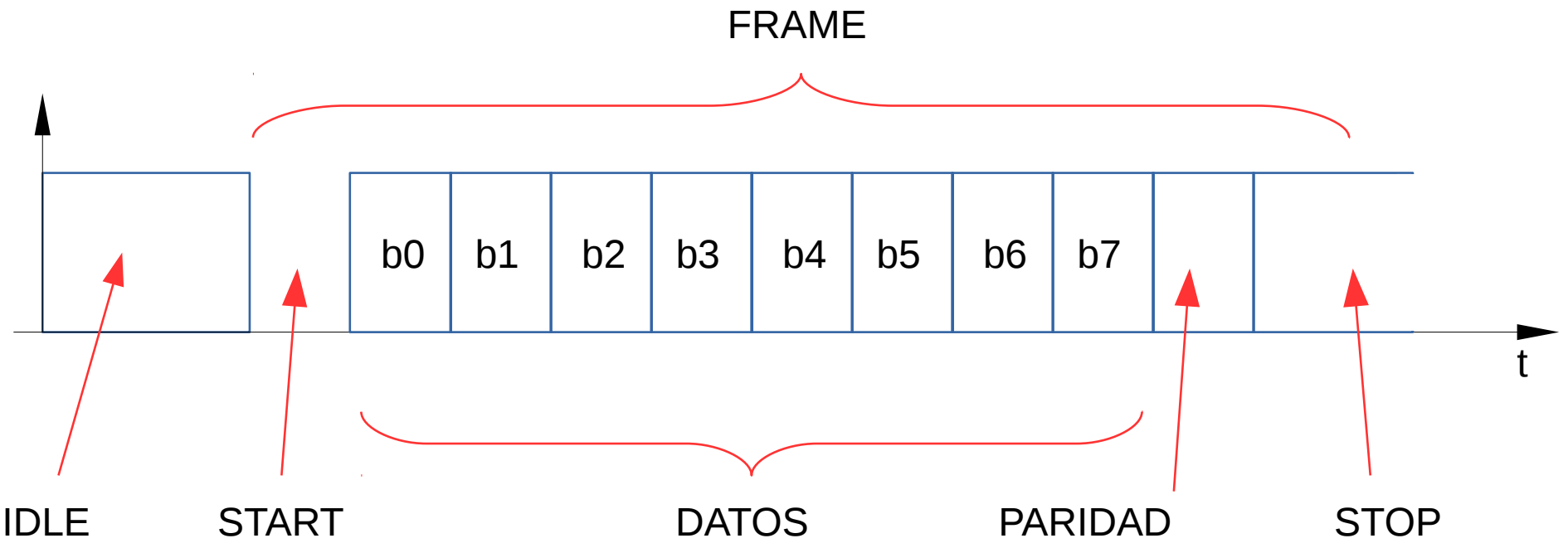
Velocidad 50 Bd, un bit dura 1 / 50 = 20ms

* válido siempre que, como en este caso, coincida la cantidad de símbolos con la cantidad de bits.

¿Cómo se transmite la información?

- Un bit tras otro: por eso es “serie”
- En grupos de 7 u 8 bits de **datos** (El estándar soporta de 5 a 8)
- El bit menos significativo (Lsb) va primero
- Si no hay actividad la línea está ociosa: “1” lógico (“Idle”)
- Se indica “Break” dejando la línea en “0”
- Se inicia la transmisión con un bit de **start**: “0” lógico
- Se puede incluir un bit para verificar la integridad: **paridad**
- Finaliza con “bit(s) de **stop**” en que la línea queda en “1” lógico
- Un “frame” está formado por: start + datos + paridad + stop

¿Cómo se transmite la información?



Verificación básica de integridad: Detecta **hasta** que hubo un bit erróneo (no cuál)

Tiempo de bit mínimo en que la línea debe permanecer en “1” lógico

*niveles lógicos → En 232 El “1” es negativo y el “0” es positivo!!!!

¿Cómo se transmite la información?

Paridad

- Es un método de detección de errores.
- Permite saber si hubo errores, no corregirlos.
- Un bit de paridad detecta solo un bit erróneo.
- Si hay dos o más bits erróneos pueden pasar desapercibidos.
- Receptor y emisor deben trabajar coincidir en el tipo de paridad con la que trabajan: Par, Impar, Mark, Space

¿Cómo se transmite la información?

Paridad

- Paridad par (o impar):

  Se agrega un bit para que la cantidad de “1” sea par (o impar)

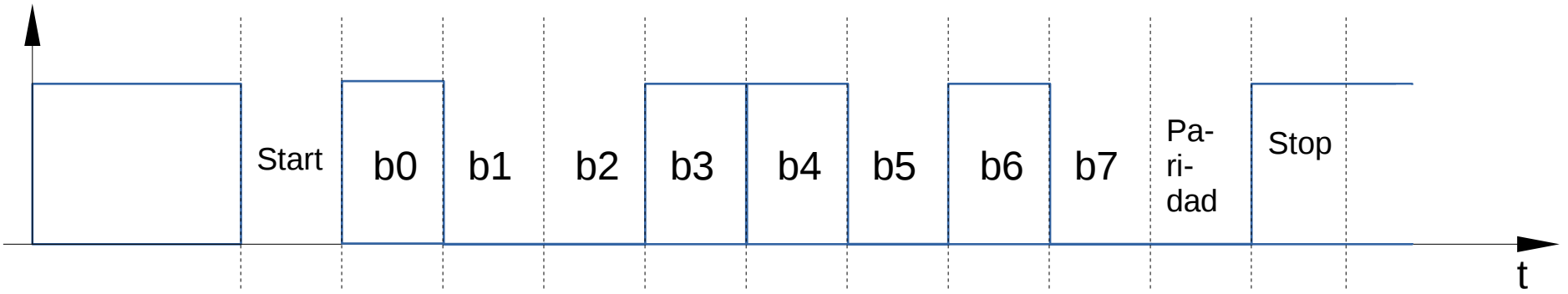
  Ej.: Datos → 10101011 → el bit de paridad par es 1 (para que la cantidad de unos sea par)
  Para el mismo grupo de bits de datos, paridad impar → 0 (para que la cantidad de unos sea impar)

- Mark (o Space): el bit de paridad es siempre “1” (o “0”) → No se utiliza

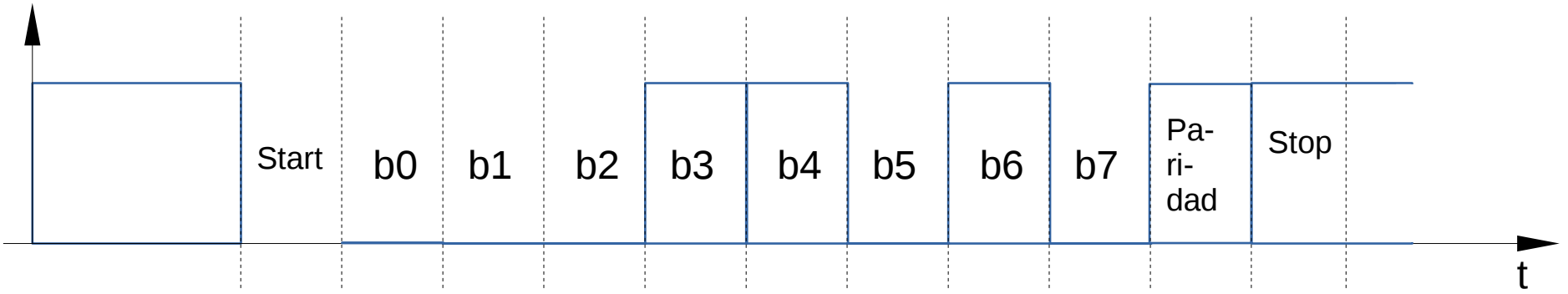
¿Cómo se transmite la información?

Ejemplos

- Dato: 'Y' 01011001 – Paridad par – 1 bit de stop – 9600 Bd



- 'X' 01011000 – Paridad par – 1 bit de stop – 9600 Bd



*niveles lógicos → En 232 El “1” es negativo y el “0” es positivo!!!!

¿Cómo se transmite la información?

Ejemplos

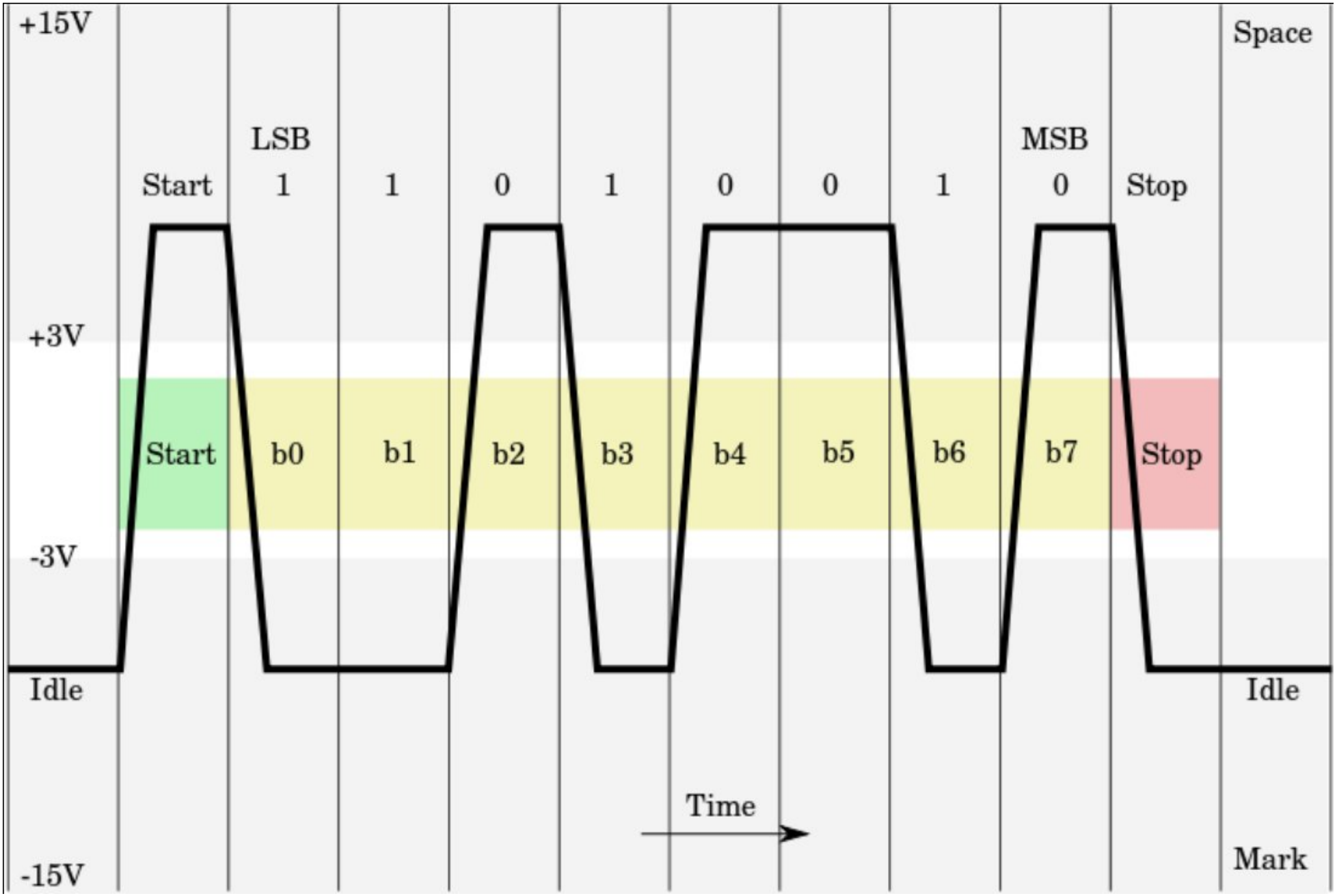
En 232 el cero es +V!!!!

En la PC

Nivel TTL (En un µC)

1 0 1 0 0 0 1 0

¿Cómo se transmite la información?



¿Y el resto de las patitas?

- Además de TX, RX y GND, el conector más común (DB9) tiene otros 6 terminales: 4 entradas digitales y 2 salidas digitales
- Estas entradas y salidas extra se usan como “handshaking” en comunicaciones serie sincrónicas.
- No se utilizan en las comunicaciones asincrónicas, que son las que acabamos de mostrar y.... son las más comunes.
- Estas se pueden utilizar de forma independiente de la transmisión y recepción serie.
- Y para otros usos que no tengan que ver con la comunicación serie...

¿Y el resto de las patitas?

- Un uso bastante común es usar una de las salidas (RTS) como control de transmisión-recepción para adaptadores 232-485

- Algunos adaptadores de este tipo se pueden alimentar del propio puerto serie (de la otra salida, DTR)

- Por supuesto, esas entradas y salidas se pueden utilizar para cualquier otra aplicación que involucre control y/o monitoreo (encender LEDs, accionar Relés, conectar sensores, etc.)

Abrir el puerto Serie

Identificando el puerto serie

- Antes de utilizar un puerto hay que abrirlo.
- Como con un archivo.
- De hecho, en GNU/Linux los puertos nos aparecen como archivos “comunes”
- ...y en windows... no. Pero las funciones que se utilizan son las mismas que para archivos, así que...

- En GNU/Linux, los puertos series aparecen como archivos en el directorio /dev y su nombre comienza con `tty:`

  - `/dev/tty0`
  - `/dev/ttyACM1`
  - `/dev/ttyUSB0`

- Una forma de saber qué puertos tenemos disponibles es utilizar el programa sloopback

Problemas abriendo el puerto serie

- Al intentar abrir un puerto serie (en GNU/Linux) tal vez se nos informe de errores de **permisos**

- Cómo solucionarlo:

- Le cambiamos el permiso al archivo correspondiente:

    - Ej. Al intentar usar un Arduino, nos da error. Su puerto normalmente es `/dev/ttyACM0`
    - Entonces hacemos:

```
chmod 666 /dev/ttyACM0
```

...y listo.

Problemas abriendo el puerto serie

- Para evitar cambiar el permiso de los puertos cada sesión (o cada vez que se desenchufa y enchufa u ndispositivo)

- Nos hacemos mienbros del grupo `dialout`

    - Hacemos:

```
sudo usermod -a -G dialout mi_usuario
```

En sucesivas sesiones podremos usar los puertos serie sin inconvenientes de permisos.

Enviar y **recibir**

Información con el

Puerto Serie

Utilizando Python + PySerial

- “Importar librería” serial

```
import serial
```

- Como se mencionó, para comenzar a utilizar un puerto serie hay que abrirlo y configurarlo (Recordar: emisor y receptor deben tener los mismos parámetros de comunicación: velocidad, paridad, bits de datos, bits de stop)

```
ser = serial.Serial('/dev/ttyUSB0',
                    baudrate=9600,
                    timeout=1,
                    bytesize=serial.EIGHTBITS,
                    parity=serial.PARITY_NONE,
                    stopbits=STOPBITS_ONE)
```

Utilizando Python + PySerial

- port → nombre del puerto serie: '/dev/ttyUSB0', “COM6”, etc.

- Baudrate → 50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200 (estos son valores estándar POSIX, hay más opciones...)

- timeout → (float) - tiempo máximo de espera al hacer read() - Si se omite timeout read() bloquea hasta que se leen los caracteres especificados.

Utilizando Python + PySerial

- bytesize → entre 5 y 8 (serial.FIVEBITS, serial.SIXBITS, serial.SEVENBITS, serial.EIGHTBITS)

- parity → serial.PARITY_NONE, serial.PARITY_EVEN, serial.PARITY_ODD, serial.PARITY_MARK, serial.PARITY_SPACE

- stopbits → serial.STOPBITS_ONE, serial.STOPBITS_ONE_POINT_FIVE, serial.STOPBITS_TWO

Utilizando Python + PySerial

- Una vez abierto el puerto, se puede enviar datos con:

```
ser.write(“uno dos tres”)
```

- Y se puede obtener lo recibido por el puerto con:

`a = ser.read(n)` n es la cantidad de bytes a leer (default: 1)

- Al terminar, cerrar el puerto:

```
ser.close()
```

Utilizando Python + PySerial

Ejemplo rx-01.py

```
#!/usr/bin/
# -*- coding: utf-8 -*-

import serial
import sys

# se abre port serie
ser = serial.Serial('/dev/ttyACM0',
              baudrate=9600,
              timeout=1,
              bytesize=serial.EIGHTBITS,
              parity=serial.PARITY_NONE,
              stopbits = serial.STOPBITS_ONE)


while True:
    rec = ser.read()
    print rec
```

- Se abre el puerto ttyACM0.
- Configuración 9600 8-N-1 y timeout 1 segundo
- Se muestra en pantalla todo lo recibido por el port serie.

Utilizando Python + PySerial

Ejemplo rx-02.py

```
#!/usr/bin/
# -*- coding: utf-8 -*-

import serial
import sys

try:
    # se abre port serie
    ser = serial.Serial('/dev/ttyACM0',
                        baudrate=9600,
                        timeout=1,
                        bytesize=serial.EIGHTBITS,
                        parity=serial.PARITY_NONE,
                        stopbits = serial.STOPBITS_ONE)

except serial.serialutil.SerialException, mensaje:
    print mensaje
    print "No se puede continuar con la ejecución"
    raise SystemExit


while True:
    rec = ser.read()
    sys.stdout.write(rec)
```

- Se abre el puerto ttyACM0.
- Configuración 9600 8-N-1 y timeout 1 segundo
- Se muestra en pantalla todo lo recibido por el port serie.

Try … except se usa en Python para manejar errores...
Si la sección try falla, se ejecuta la sección except correspondiente...

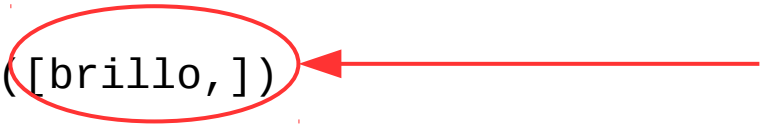
Si se usa print no se puede mostrar u carácter al lado de otro

Utilizando Python + PySerial

Ejemplo tx-01.py

```
#!/usr/bin/
# -*- coding: utf-8 -*-

import serial
import sys
import time

# se abre port serie
ser = serial.Serial('/dev/ttyACM0',
                    baudrate=9600,
                    timeout=1,
                    bytesize=serial.EIGHTBITS,
                    parity=serial.PARITY_NONE,
                    stopbits = serial.STOPBITS_ONE)

brillo = 0
while True:
    ser.write([brillo,])

    if brillo < 255:
        brillo = brillo + 1
    else:
        brillo = 0

    time.sleep(0.1)
```

- Se abre el puerto ttyACM0.
- Configuración 9600 8-N-1
- Envía un byte cada 100ms
- Funciona con el ejemplo Dimmer de Arduino



Así se envían bytes, en lugar de caracteres ASCII

Utilizando Python + PySerial

Ejemplo tx-rx-01.py

```
#!/usr/bin/
# -*- coding: utf-8 -*-

import serial

# se abre port serie
ser = serial.Serial('/dev/ttyACM1',
          baudrate=9600,
          timeout=2,
          bytesize=serial.EIGHTBITS,
          parity=serial.PARITY_NONE,
          stopbits = serial.STOPBITS_ONE)

linea = ""
while True:
    cant = raw_input("cuántos asteriscos? ")
    ser.write( cant);

    while True:
        rec = ser.read()
        linea = linea + rec
        if rec == '\n':
            print linea
            linea = ""
            break
```

- Se abre el puerto ttyACM1.
- Configuración 9600 8-N-1
- Envía un carácter ASCII entre '0' y '9' y recibe esa cantidad de asteriscos.

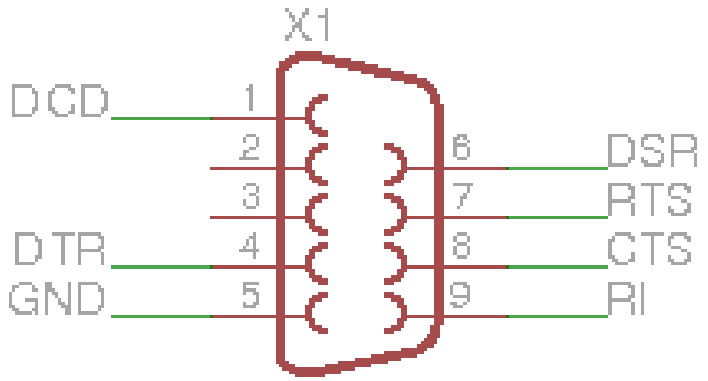
(Ver código Arduino en ejemplos)

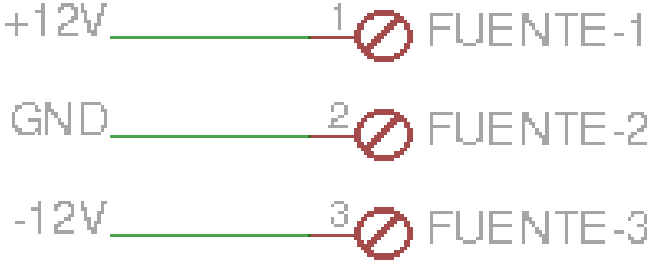
Solo muestra una línea completa.

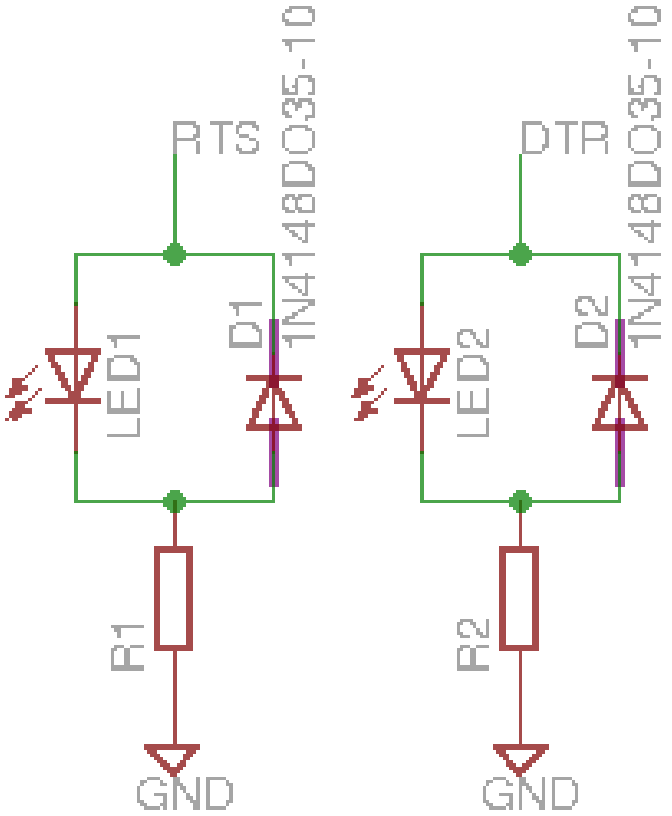
(Al recibir \n)

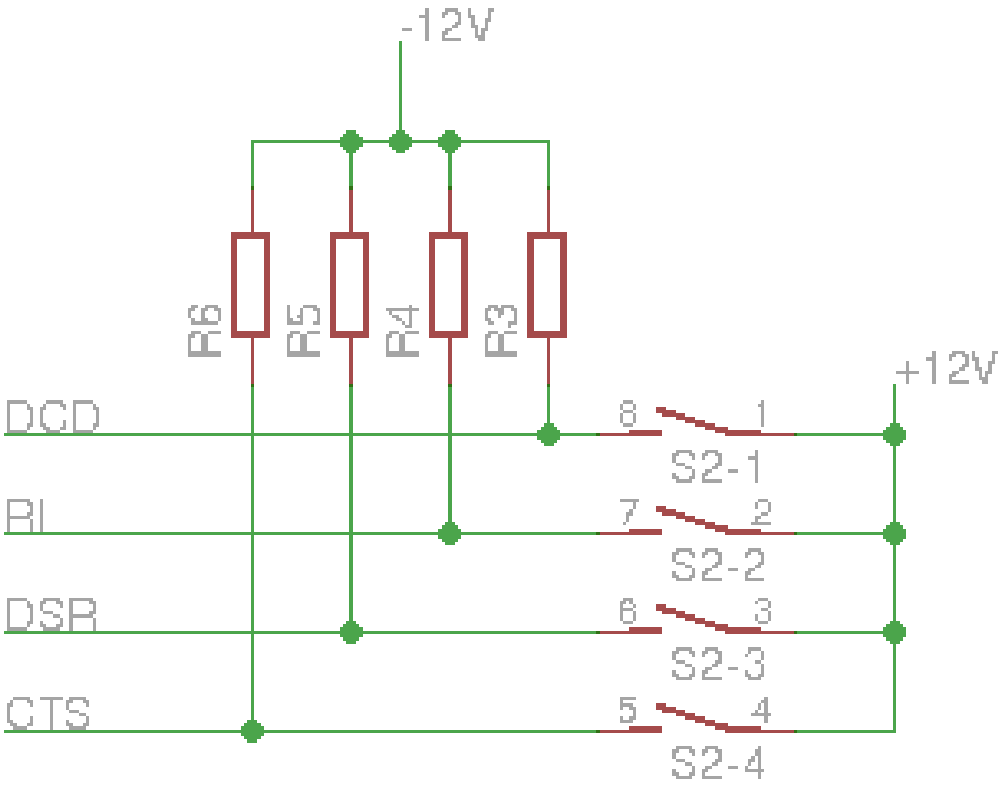
Control y monitoreo de dispositivos Utilizando las líneas de Handshaking del Puerto Serie

Un circuito para empezar...









Un circuito para empezar...

- D1 y D2 se colocan para que la V inversa no deteriore a los LED.

- Usarlos también con otros dispositivos: transistores, optoacopladores (que en definitiva son LED...)

- En la práctica se puede usar una fuente simple de 5V. A pesar del estándar muchos puertos reconocen tensiones menores a 3V como estado válido...

Serie con Python + PySerial

- Se importa **PySerial**

```
import serial
```

- Para comenzar a utilizar un puerto serie hay que **“abrirlo”**

```
puerto = serial.Serial("/dev/ttyUSB1")
```

- Con puerto, un objeto clase **`Serial`**, vamos a hacer referencia a ese puerto abierto...

Serie con Python + PySerial

- Para mostrar el estado de las entradas:

```
puerto.cd  puerto.ri  puerto.dsr  puerto.cts
```

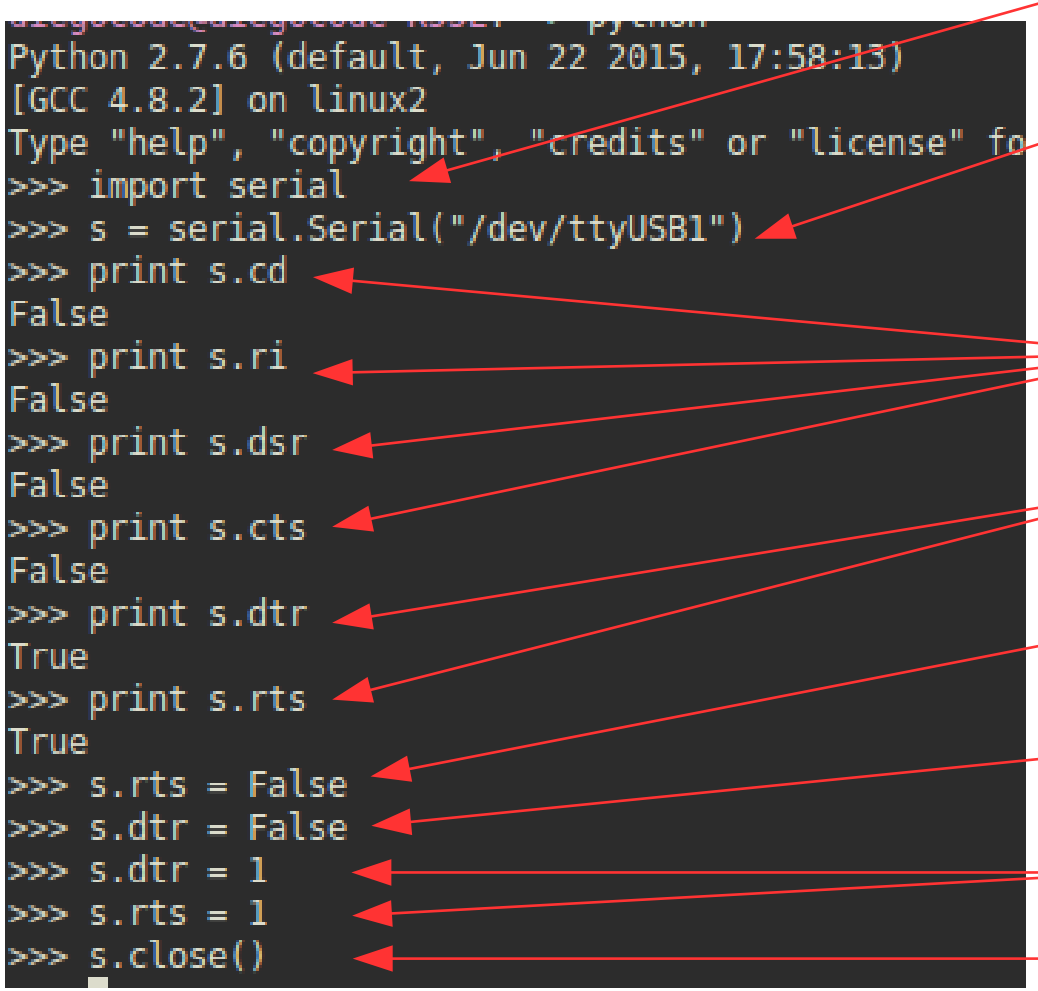
- Para mostrar y fijar estados en las salidas:

```
puerto.dtr puerto.rts
```

- Para cerrar el puerto:

```
puerto.close()
```

Serie con Python + PySerial



Importamos PySerial

Abrimos el puerto
(si solo vamos a usar líneas de handshaking no hace falta configurar nada)

Mostramos el estado de CD, RI, DSR, CTS

Mostramos el estado de DTR y RTS

“Apagamos” RTS (-V)

“Apagamos” DTR (-V)

“Encendemos” DTR y RTS (+V)

Cerramos el puerto

Serie con Python + PySerial

- LED1 (conextado en RTS) titila - invertimos su estado cada 1 segundo

```
>>> s.rts = 0
>>> import time
>>> while True:
...     time.sleep(1)
...     s.rts = not s.rts
...
```

RTS “Apagado”

Importamos time
(para usar sleep())

Esperamos 1s

Invertimos el estado de RTS

- En la imagen está realizado en el shell
- script en hndsh-01.py

Serie con Python + PySerial

Ejemplo hndsh-02.py

```
#!/usr/bin/
# -*- coding: utf-8 -*-

import serial
import time

s = serial.Serial("/dev/ttyUSB1")

s.rts = 0
while True:
    if s.cd:
        s.rts = not s.rts

    time.sleep(0.5)

    if s.ri:
        break
```

- Si CD se conecta a una tensión positiva, LED1 titila.
- Si CD se conecta a una tensión negativa LED1 queda fijo.
- Si RI se conecta a una tensión positiva, finaliza el programa.

Serie con Python + PySerial

Ejemplo hndsh-03.py

```
#!/usr/bin/
# -*- coding: utf-8 -*-

import serial
import timer

def led_verde(puerto):
    puerto.rts = not puerto.rts

def led_rojo(puerto):
    puerto.dtr = not puerto.dtr

s = serial.Serial("/dev/ttyUSB1")
s.dtr = 0
s.rts = 0

tim_v = timer.tick_timer(1, led_verde, [s,])
tim_v.start()

tim_r = timer.tick_timer(0.25, led_rojo, [s,])
tim_r.start()

while True:
    pass
```

- LED1 y LED2 titilan a diferentes velocidades.
- No se usa time, sino clase Timer, con threads, no bloqueante. (disponible en directorio de ejemplos)