E/S Serie

Ángel Pérez Muñoz[1], Manuel Nieto[2]

Informática Industrial

14 de diciembre de 2023

[1] angel.perez.munoz@upm.es - D4202

[2] mnieto@fi.upm.es – D4106

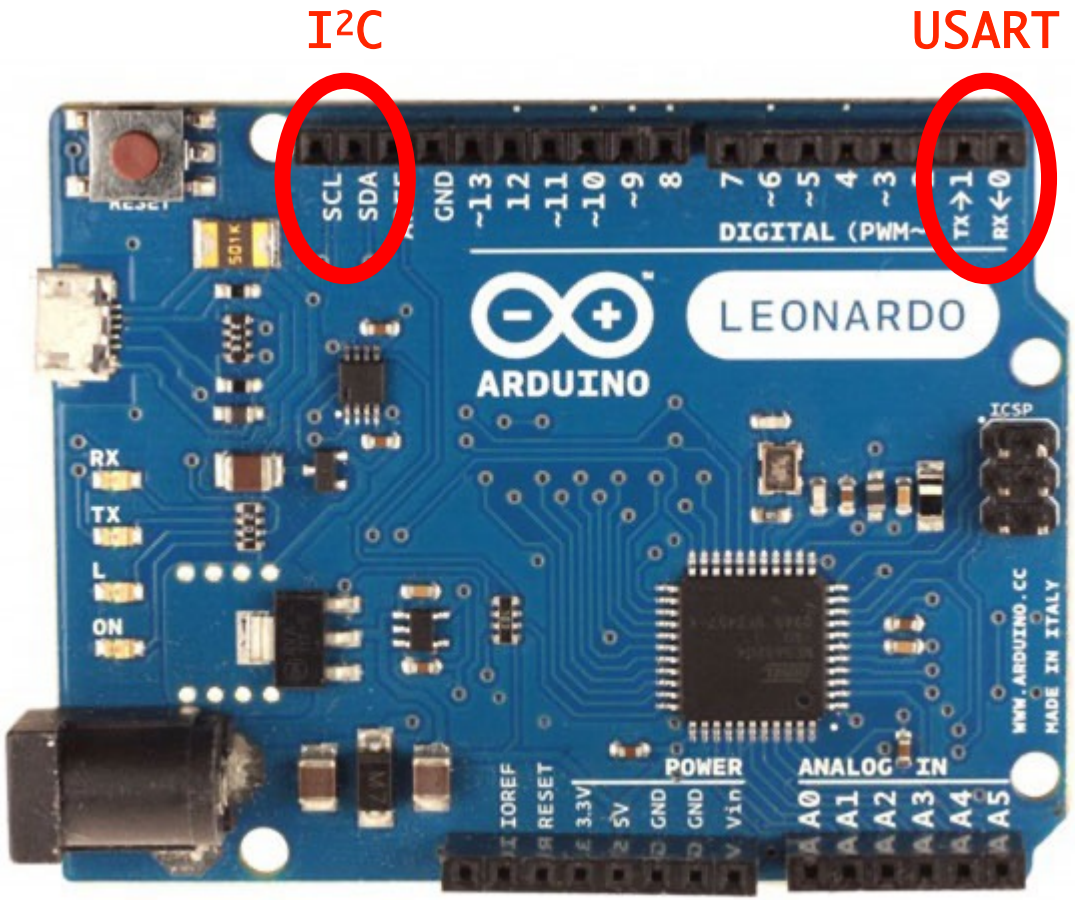
datsi

Contenido original del Prof. Manuel Nieto

Introducción

Generador de señal de reloj

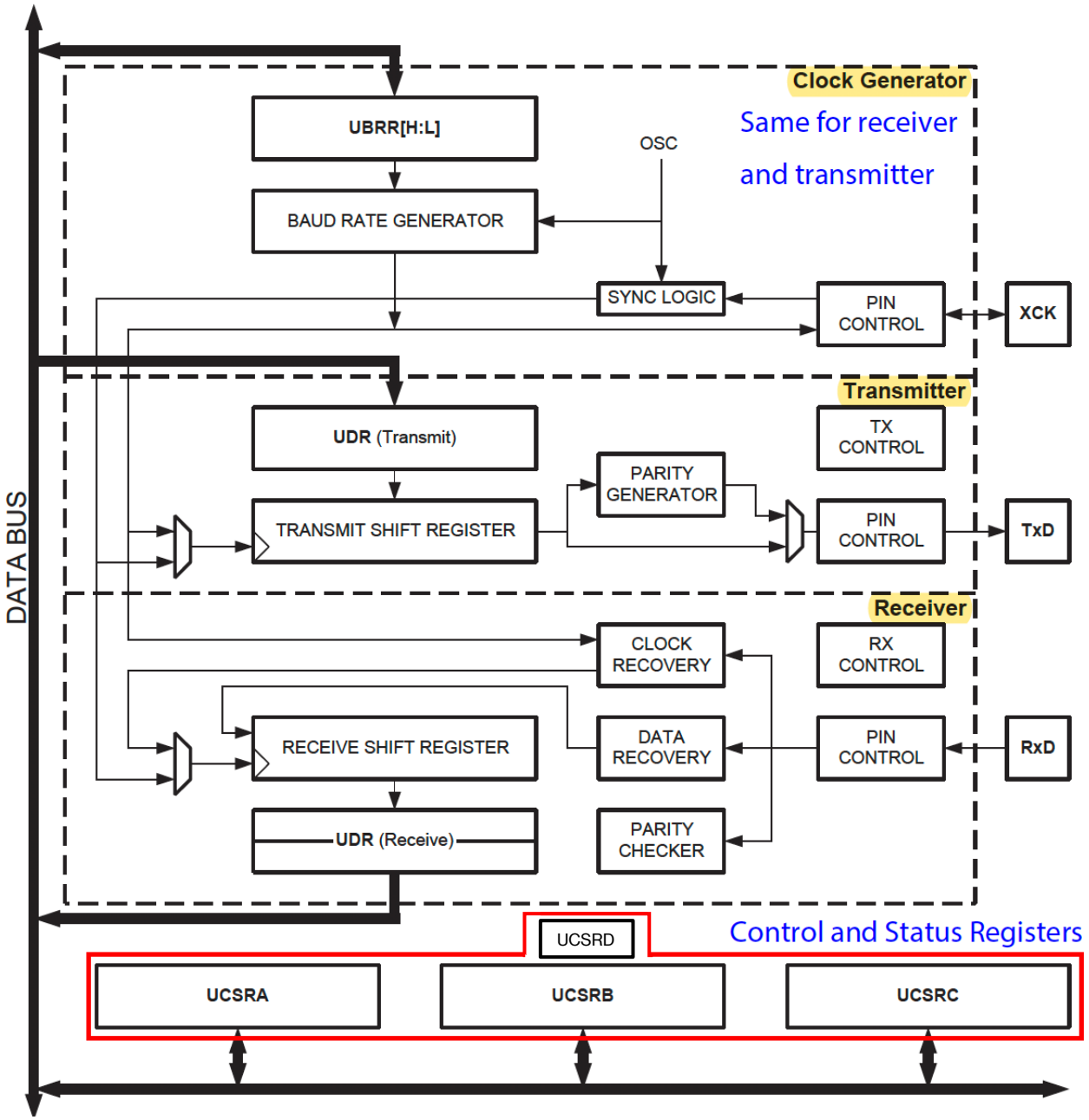
- El µControlador ATMega32U4 incluye una interfaz I^2C y una USART.
- I^2C se es una interfaz de E/S serie con reloj.
- USART es una interfaz de E/S serie sin reloj y puede ser configurada en modo síncrona o asíncrona.
- Veremos su programación:
  - Registros.
  - Modos de configuración.



USART en ATMega32U4

Características

- Comunicación *full-dúplex*
- Control de flujo:
  - *CTS (Clear To Send),*
  - *RTS (Request to Send).*
- Longitud de datos:
  - *Datos: 5, 6, 7, 8, 9.*
  - *Parada: 1, 2.*
- Bit de paridad “par” e “impar” verificado por Hardware.
- Generación de 3 IRQs:
  - *TX Complete, TX Data Register Empty, RX Complete.*
- 4 Modos de comunicación:
  - *Syncrhonous: **Master** o **Slave**.*
  - *Asynchronous: **Normal** o **Double Speed**.*



Transmisor

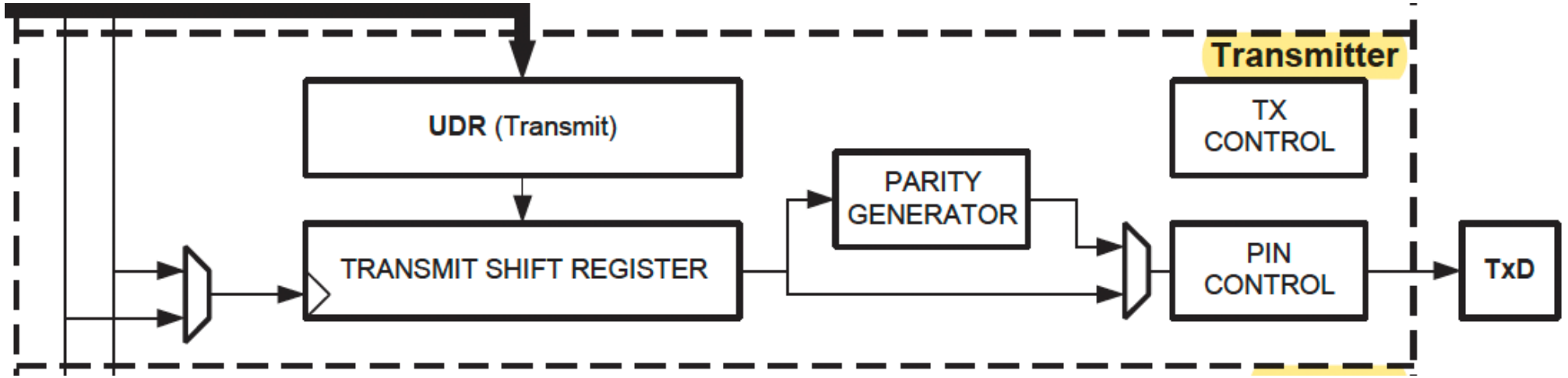
- Estructura: Buffer de escritura, Shift Register, Generador de paridad.
- Registro UDR de solo escritura para la transmisión.

  Contiene los datos que se enviarán por Tx.

- Proceso de transmisión:
  1. Escritura de caracteres en UDR.
  2. Copia en el registro de transmisión.
  3. Se arma el mensaje y se envía:

START	DATOS (UDR)	PARIDAD	STOP

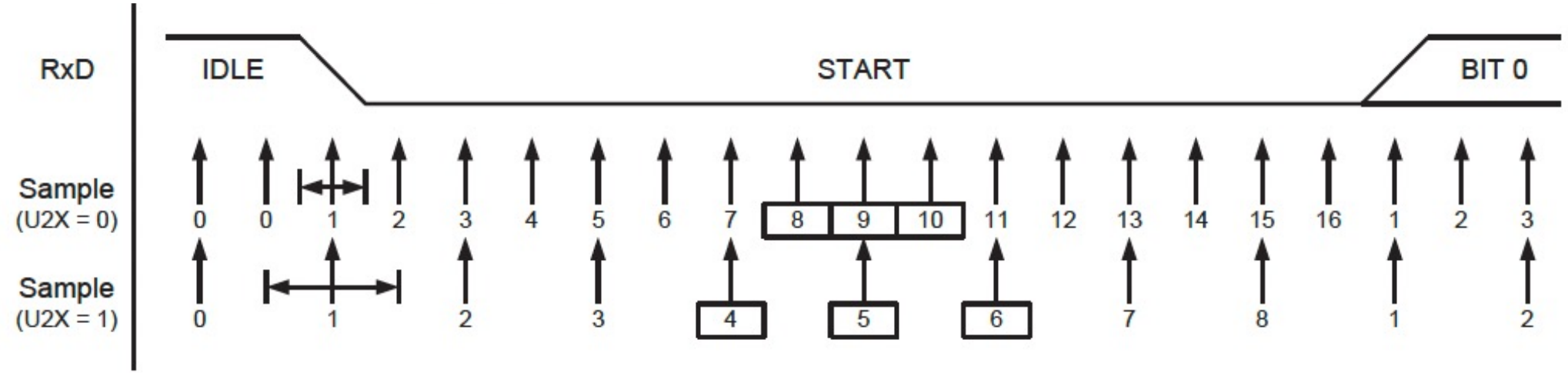
  4. Se notifica disponibilidad de transmisión de datos mediante el flag: *"RTS – Ready To Send"*.



Receptor: Recepción de bit de START

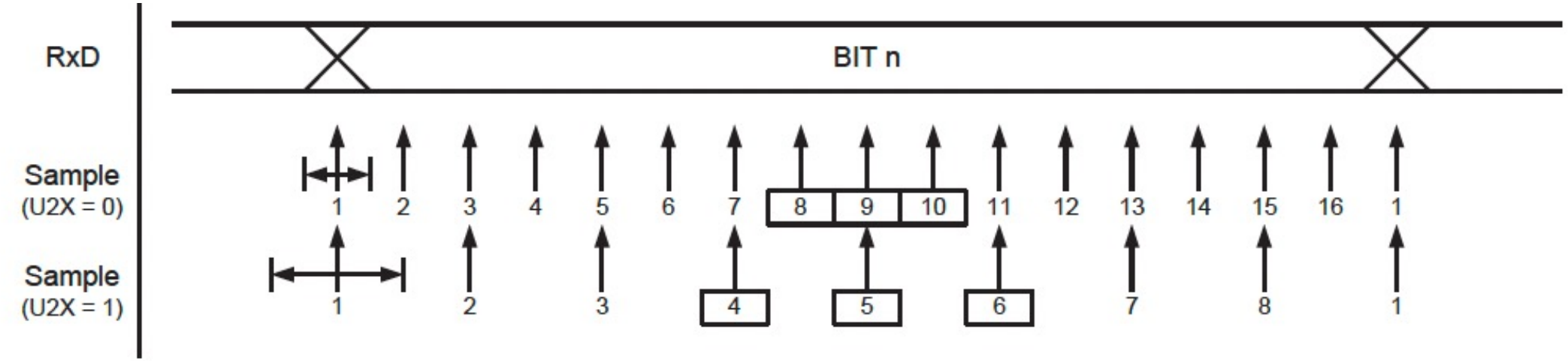
Nyquist: En este caso, la frecuencia de muestreo es 16 veces superior al baudrate.

1. RxD IDLE, a nivel ALTO:
   - Contador interno a 0.
2. RxD cambia de nivel ALTO a BAJO:
   - Empieza la secuencia para detectar el bit START.
3. RxD START bit, a nivel BAJO:
   - Contador interno incrementa 16 veces.
4. Las muestras 8, 9 y 10 definen el valor del bit Start por mayoría:
   - Al menos dos a nivel ALTO: Ruido.
   - Al menos dos a nivel BAJO: START bit.

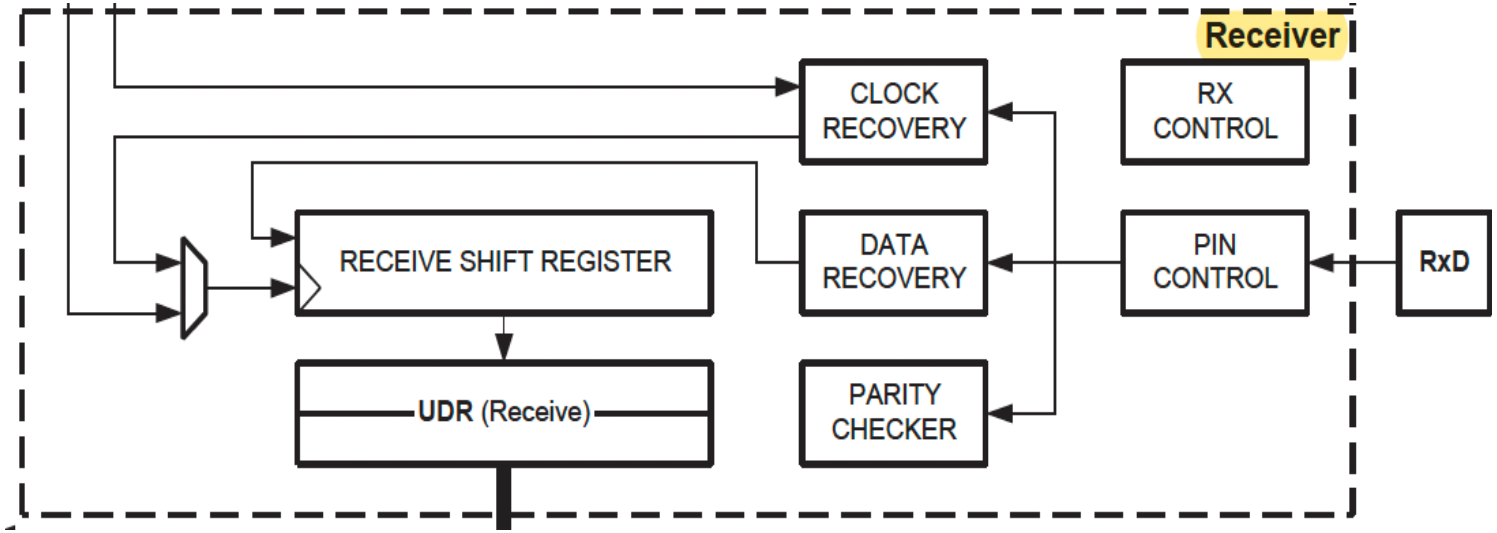


Receptor: Recepción de bit de DATOS

1. Tras la muestra nro. 16 del bit de START, empieza la detección del 1er bit de datos.
2. Se muestrean 16 veces el pin RxD.
3. Las muestras 8, 9 y 10 definen el valor del bit de datos por mayoría:
   - Al menos dos a nivel ALTO: Bit ALTO.
   - Al menos dos a nivel BAJO: Bit BAJO.
4. Tras la muestra nro. 16 del 1er bit de datos, empieza la detección del 2do bit de datos…
   - Y así sucesivamente hasta el bit 5, 6, 7, o 8; dependiendo de la configuración.



Receptor



1. Detecta el bit de START.
2. Detecta los bits de Datos y los va introduciendo en el registro de desplazamiento.
3. Cuanto se detecta el tamaño suficiente, estos bits se introducen en el registro UDR para su lectura
4. Evalúa la paridad de los datos y lo compara con el bit de paridad.

- Más complejo que el transmisor:
  - *En su modo asíncrono debe sincronizar los datos que le llegan por el pin RxD.*
- Detección de errores y disponibilidad:
  - *Los datos vienen acompañados de unos biestables sobre **error de formateo** y otro para el **error de paridad**.*
  - *Si no hay espacio en el buffer (> 3 datos): el dato nuevo se pierde y se activa un biestable sobre perdida de datos: **Over Run**.*

Registro UCSR1A

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

Biestables de petición de IRQs

- **RXC1, Receive Complete**: Puesto a 1 si hay datos disponibles para leer.
- **UDRE1, Data Register Empty**: Puesto a 1 si se pueden transmitir datos, es decir, el buffer de escritura UDR1 está vacío y puedo escribir en él.
- **TXC1, Transmit Complete**: Puesto a 1 cuando se han transmitido todos los datos.
- **FE1, Frame Error**: Puesto a 1 si hubo un error de formateo, p.ej.: bit de STOP es 0.
- **DOR1, Data Over Run**: Puesto a 1 si el buffer de recepción está lleno
- **UPE1, Parity Error**: Puesto a 1 si el bit de paridad es incorrecto. La comprobación de paridad se activa si el bit UPM11 del registro UCSR1C está puesto a 1.

Registro UCSR1B

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **RXEN1, Enable Receiver**: Puesto a 1 para habilitar la recepción.
- **TXEN1, Enable Transmitter**: Puesto a 1 para habilitar la transmisión.

Biestables de máscara de IRQs

- **RXCIE1**: Puesto a 1 para habilitar la generación de interrupciones si hay datos recibidos en UDR que aún no se han leído. Genera la IRQ hasta que se vacíe.
- **UDRIE1**: A 1 para habilitar la generación de interrupciones si puedo transmitir datos, es decir, el buffer de transmisión tiene espacio y puedo escribir en UDR1.
- **TXCIE1**: A 1 para habilitar la generación de interrupciones si se completó la transmisión de un dato.

Registro UCSR1B

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **RXB81, Receiver Data Bit 8**: Contiene el noveno bit para mensajes recibidos de 9-bits. Antes de leerlo, se debe leer el buffer de recepción UDR1 que contienes los 8 bits (desde 0 hata el 7).
- **TXB81, Transmitter Data Bit 8**: Contiene el noveno bit para mensajes transmitidos de 9-bits. Antes de escribir en él, se debe escribir los bits del buffer UDR1 que contiene los 8 bits (desde el 0 hasta el 7).
- **UCZ12**: Junto con UCZ10 y UCZ11 ubicados en el registro UCSR1C, define la longitud de bits de datos a transmitir y recibir.

Registro UCSR1C

Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

- **UMSEL10 y UMSEL11**: Su valor define el modo de operación de la USART:

UMSEL11	UMSEL10	Modo de la USART
0	**0**	**USART en modo asíncrono (para vuestra práctica).**
0	1	USART en modo síncrono.
1	1	USART en modo Maestro de Serial Peripheral Interface (SPI).

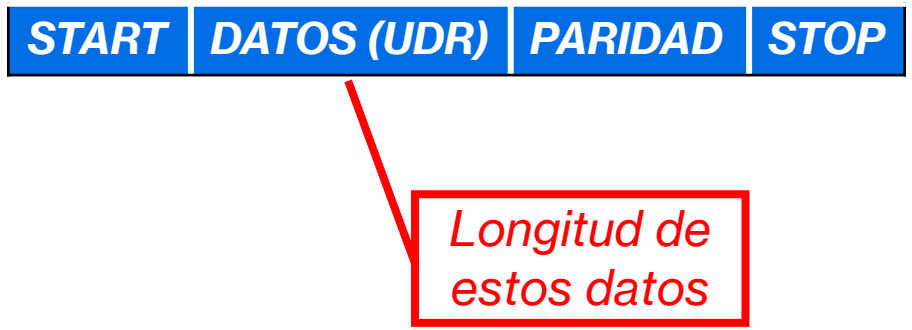
- **UPM10 y UPM11**: Su valor activa y define el control de paridad:

UMSEL11	UMSEL10	Paridad
0	0	Desactivada
1	0	**Paridad par**, el bit de paridad se establece de forma que la cantidad de 1s (sin contar START y STOP) sea par.
1	1	**Paridad impar**, el bit de paridad se establece de forma que la cantidad de 1s (sin contar START y STOP) sea impar.

Registro UCSR1C

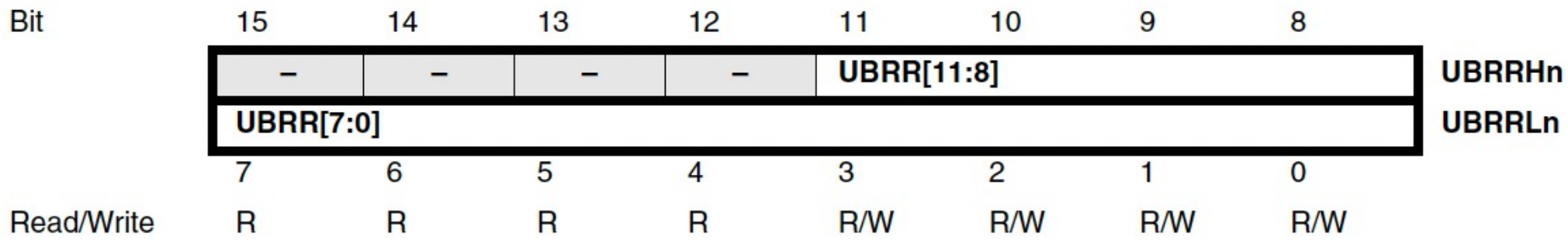
Bit	7	6	5	4	3	2	1	0	
	UMSELn1	UMSELn0	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn	UCSRnC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	1	0	

- **USBS1**: Codifica el número de bits de STOP a transmitir. A 0 para 1 bit de STOP y a 1 para 2 bits de STOP.
- **UCSZ10 y UCSZ11**: Junto a UCZ12 ubicado en el registro UCSR1B, define la longitud de bits de datos a transmitir y recibir.



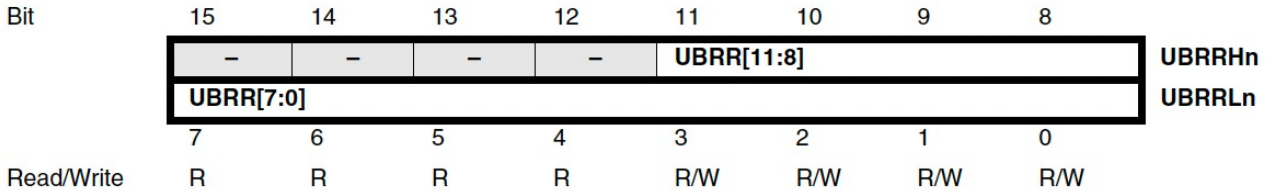
UCSZ12	UCSZ11	*UCSZ10*	Longitud de datos
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	**1**	**1**	**8-bit (práctica)**
1	1	1	9-bit

Registro UBRRL1 y UBRRH1



- **USART Baud Rate Register** de 12 bits dividido en dos registros, uno para la parte alta y otro para la parte baja:
  - UBRRH contiene los 4 bits más significativos, es decir, la parte alta.
  - UBRRL contiene los 8 bits menos significativos, es decir, la parte baja.
  - Se debe escribir primero en la parte alta (UBRRH) y luego en la parte baja (UBRRL).

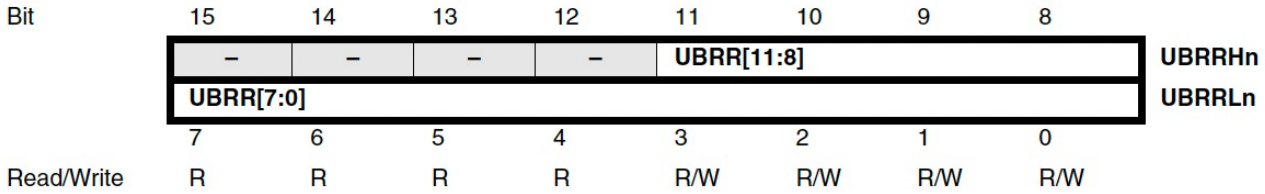
Registro UBRRL1 y UBRRH1



Ejemplo: Necesito configurar una velocidad 28800 bps (28.8 Kbps)

Baud Rate (bps)	f_{osc} = 16.0000 MHz			
	U2Xn = 0		U2Xn = 1	
	UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%
4800	207	0.2%	416	-0.1%
9600	103	0.2%	207	0.2%
14.4k	68	0.6%	138	-0.1%
19.2k	51	0.2%	103	0.2%
28.8k	34	-0.8%	68	0.6%
38.4k	25	0.2%	51	0.2%
57.6k	16	2.1%	34	-0.8%
76.8k	12	0.2%	25	0.2%
115.2k	8	-3.5%	16	2.1%
230.4k	3	8.5%	8	-3.5%
250k	3	0.0%	7	0.0%
0.5M	1	0.0%	3	0.0%
1M	0	0.0%	1	0.0%
Max. (1)	1 Mbps		2 Mbps	

Registro UBRRL1 y UBRRH1

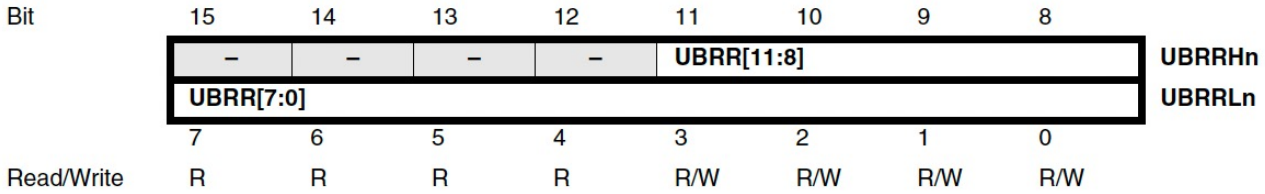


Ejemplo: Necesito configurar una velocidad 28800 bps (28.8 Kbps)

- Tengo dos opciones con U2X1=0 y U2X1=1.
  - Elijo la que menos error tenga, es decir, U2X1=1 ya que obtendré un error del 0.6%.
  - Luego, UBRR debe valer 68.
- Contenido de UBRRL1 y UBRRH1 para que UBR valga 68:
  - $68_{10} = 0000\ 01000100_2$

Baud Rate (bps)	f_{osc} = 16.0000 MHz			
	U2Xn = 0		U2Xn = 1	
	UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%
4800	207	0.2%	416	-0.1%
9600	103	0.2%	207	0.2%
14.4k	68	0.6%	138	-0.1%
19.2k	51	0.2%	103	0.2%
28.8k	34	-0.8%	68	0.6%
38.4k	25	0.2%	51	0.2%
57.6k	16	2.1%	34	-0.8%
76.8k	12	0.2%	25	0.2%
115.2k	8	-3.5%	16	2.1%
230.4k	3	8.5%	8	-3.5%
250k	3	0.0%	7	0.0%
0.5M	1	0.0%	3	0.0%
1M	0	0.0%	1	0.0%
Max. (1)	1 Mbps		2 Mbps	

Registro UBRRL1 y UBRRH1



Ejemplo: Necesito configurar una velocidad 28800 bps (28.8 Kbps)

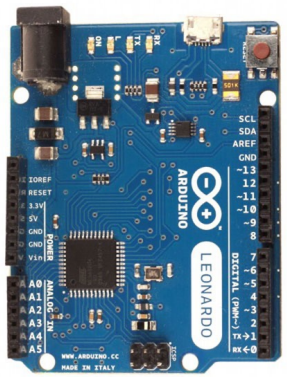
- Tengo dos opciones con U2X1=0 y U2X1=1.
  - Elijo la que menos error tenga, es decir, U2X1=1 ya que obtendré un error del 0.6%.
  - Luego, UBRR debe valer 68: UBRR = 68.

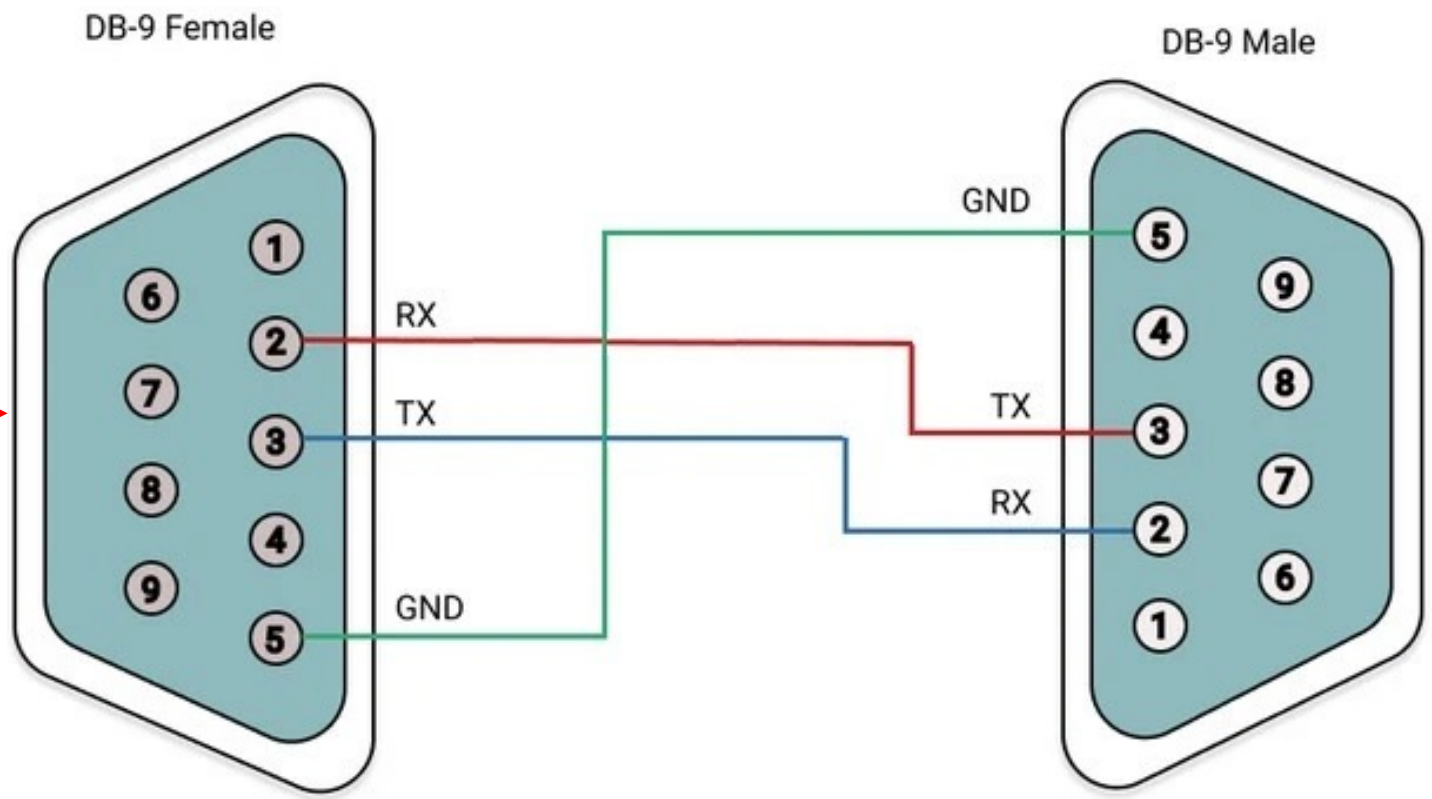
Baud Rate (bps)	f_{osc} = 16.0000 MHz			
	U2Xn = 0		U2Xn = 1	
	UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%
4800	207	0.2%	416	-0.1%
9600	103	0.2%	207	0.2%
14.4k	68	0.6%	138	-0.1%
19.2k	51	0.2%	103	0.2%
28.8k	34	-0.8%	68	0.6%
38.4k	25	0.2%	51	0.2%
57.6k	16	2.1%	34	-0.8%
76.8k	12	0.2%	25	0.2%
115.2k	8	-3.5%	16	2.1%
230.4k	3	8.5%	8	-3.5%
250k	3	0.0%	7	0.0%
0.5M	1	0.0%	3	0.0%
1M	0	0.0%	1	0.0%
Max. (1)	1 Mbps		2 Mbps	

Programación en ATmega32U4

Montaje

La conexión de un DTE (Arduino) a otro DTE (PC)
implica cruzar los cables de recepción y transmisión

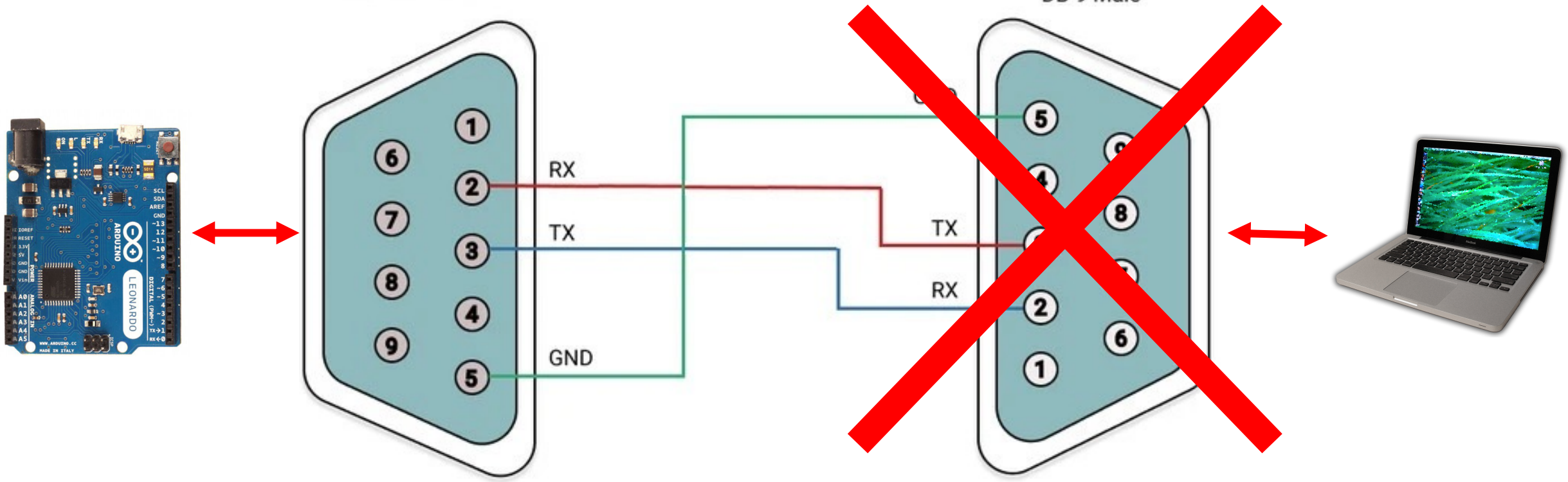




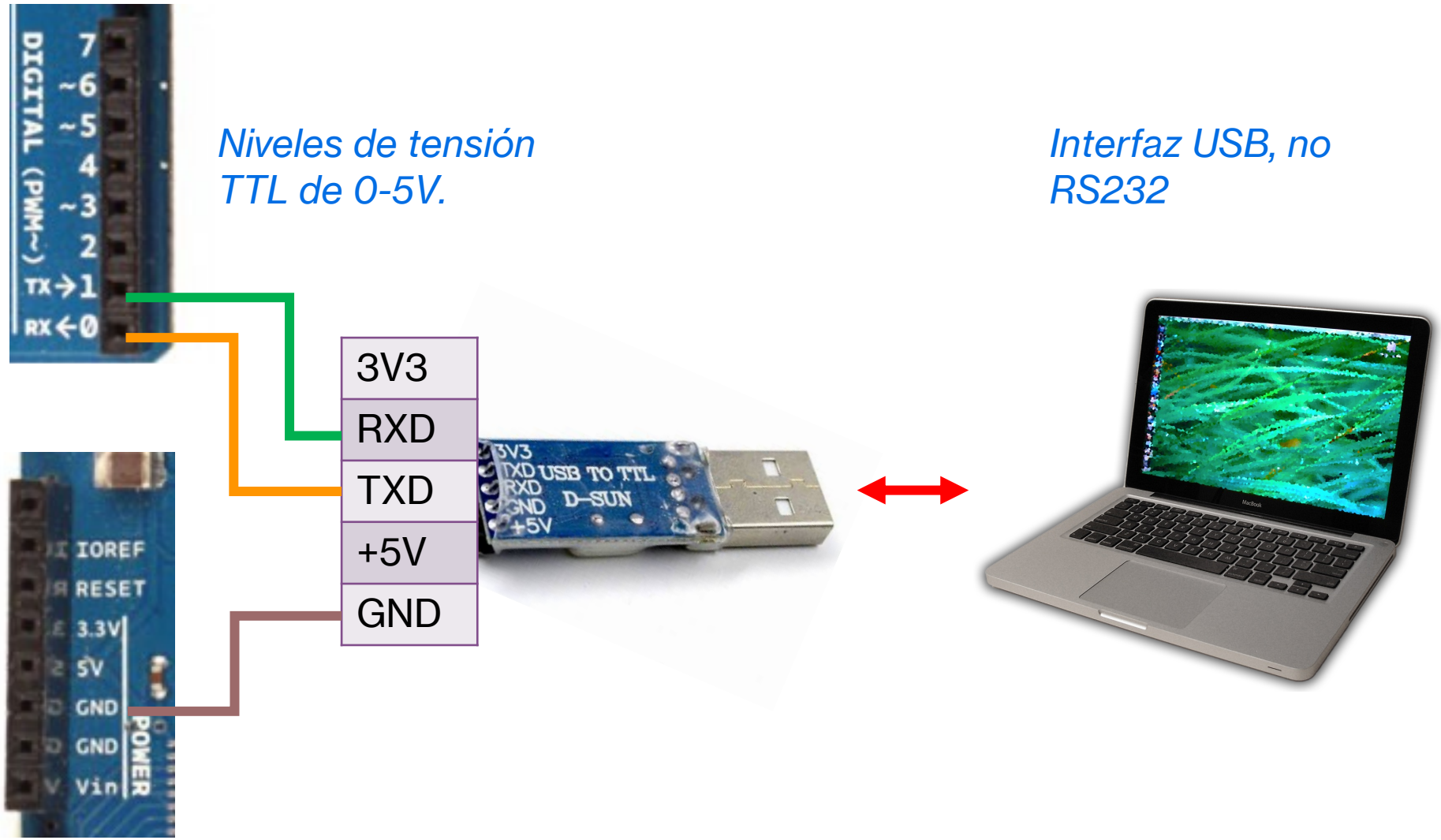
Montaje

Niveles de tensión TTL de 0-5V.

Interfaz USB, no RS232



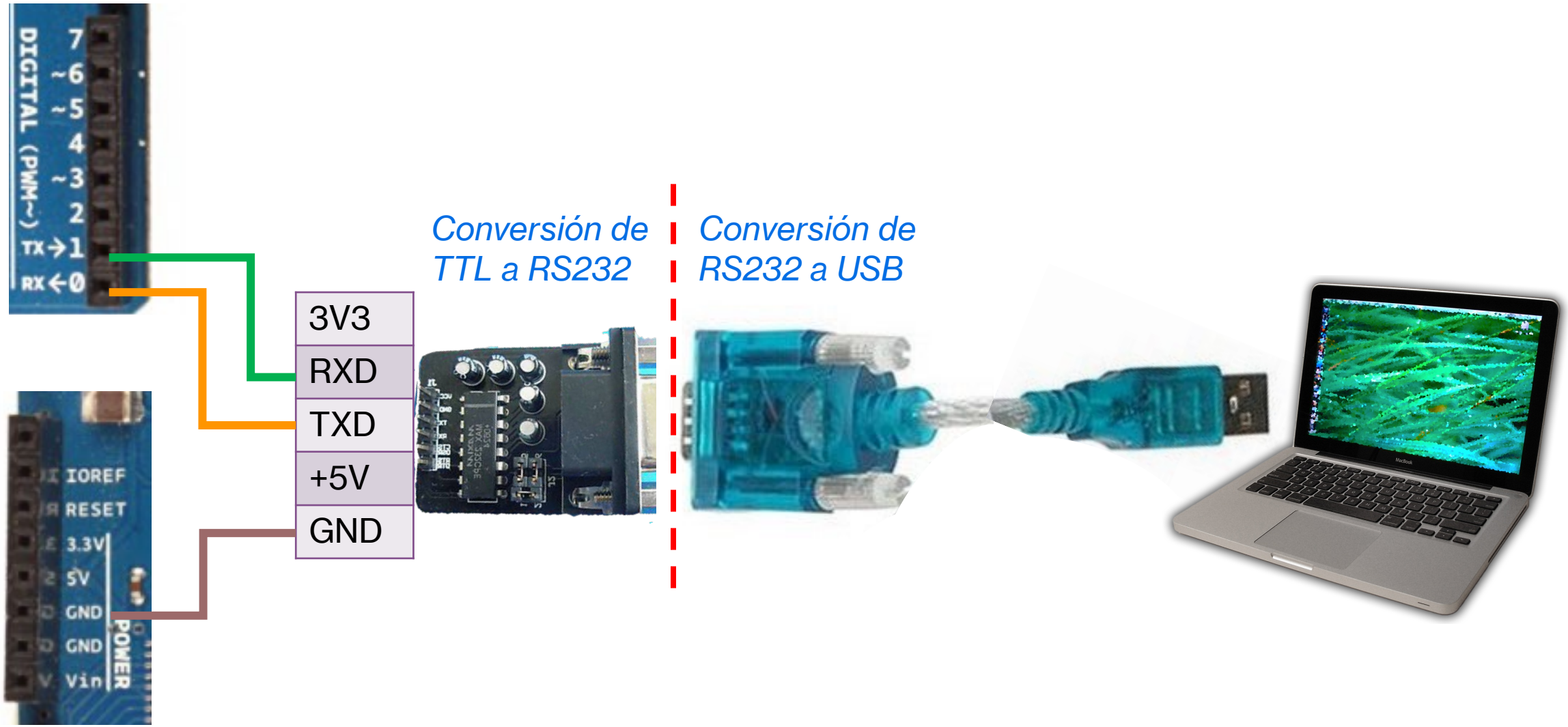
Montaje para RS232-TTL Sencillo y Completo



Niveles de tensión TTL de 0-5V.

Interfaz USB, no RS232

Montaje mediante cable DSub-9



Control de flujo

- Mediante Hardware:
  - Emplear las señales Clear To Send (CTS) y Request to Send (RTS).
  - Alternativa: emplear las señales Data Set Ready (DSR) y Data Terminal Ready (DTR).
- Mediante Software:
  - No se usan las señales de control de flujo, solo las de transmisión y recepción.
  - Se usan dos caracteres especiales:
    - <XON>: Transmisión ON pulsando CTRL + Q.
    - <XOFF>: Transmisión OFF pulsando CTRL + S.
    - Ejemplo: Editor VIM.

Simulación por VMLab *[4]*

- Simulado sobre un microcontrolador Atmega128.
- 3 ejemplos que emplean la UART0 del microcontrolador.
  - El ejemplo usart.c implementa todas las funciones de bajo nivel sin librerías externas.
  - Gestiona la recepción mediante interrupciones, de forma no bloqueante a través de un buffer lineal.
  - Gestiona la transmisión mediante interrupciones, de forma bloqueante a través de un buffer circular.
- De modo que para adaptar dicho ejemplo (usart.c) al microcontrolador ATMega32U4 del Arduino Leonardo:
  - Se deben cambiar el nombre de las ISRs, y los bits y registros de control y datos, etc.
  - Por ejemplo: UCSR1A en vez de UCSR0A, ISR(USART1_RX_vect) en vez de SIGNAL(SIG_UART0_RECV), UDR1 en vez de UDR0, mover la configuración de main() a setup() y el bucle infinito (for(;;)) a loop, entro otros…

Simulación por VMLab *[4]*

Temario

1. **Introducción al diseño de sistemas basados en microcontroladores. Microcontrolador AVR Atmega32U4.**
   - Arquitectura.
   - Juego de instrucciones.
   - Módulos de E/S.
2. **Arduino Leonardo.**
   - IDE de Arduino.
   - Programación en C.
   - Prácticas propuestas.
3. **Entrada/Salida digital (Presentación).**
   - Activación de LED's.
   - Lectura de pulsadores.
   - Teclado (lectura digital y analógica).
   - Control de dispositivos.
5. **Gestión de interrupciones en AVR ATmega.**
   - Ejemplo interfaz centronics para el simulador vmlab.
6. **Temporizadores programables (Presentación, Ejemplos).**
   - Pulse Width Modulation (PWM).
   - Filtros PID (Vídeo ilustrativo).
   - Control de motores y servomotores.
   - Detección de paso por cero (ZCD). Gestión de potencia.
7. **Entrada/Salida serie.**
   - USART (RS-232).
     - Ejemplos para el simulador vmlab.
   - I2C.
   - SPI.

Recepción

- Los caracteres se reciben por RX y según van llegando se van almacenando en el buffer rbuf con un tamaño de 64 bytes (definido en la macro RBUFSIZE).
- Cuando llega un <CR> se completa la recepción de una línea. En C/C++ un string termina por '\0' y no por <CR>
  - Se sustituye el <CR> por '\0'.
  - Se vacía todo el buffer, por eso es lineal.
  - Como no es un buffer circular, solo se necesita un índice para insertar los caracteres.
- Si el buffer rbuf se llena porque la línea es más grande de 64 bytes:
  - Todo el contenido de rbuf se trata como una línea aunque no haya llegado el <CR>.
  - Se añade al final el carácter '\0'.

Transmisión

- Los caracteres se envían por TX se van almacenando en el buffer tbuf con un tamaño circular de 64 bytes (definido en la macro TBUFSIZE).
- Condiciones del buffer circular:
  - Si in – out == 0 → El buffer está vacío.
  - Si in – out == TBUFSIZE → Está lleno.
  - Si in – out < 64 → Hay huecos.
- Cuando llega un <CR> se completa la recepción de una línea. En C/C++ un string termina por '\0' y no por <CR>
  - Se sustituye el <CR> por '\0'.
  - Se vacía todo el buffer, por eso es lineal.
- Si el buffer rbuf se llena porque la línea es más grande de 64 bytes:
  - Todo el contenido de rbuf se trata como una línea aunque no haya llegado el <CR>.
  - Se añade al final el carácter '\0'.

Procesado de mensajes

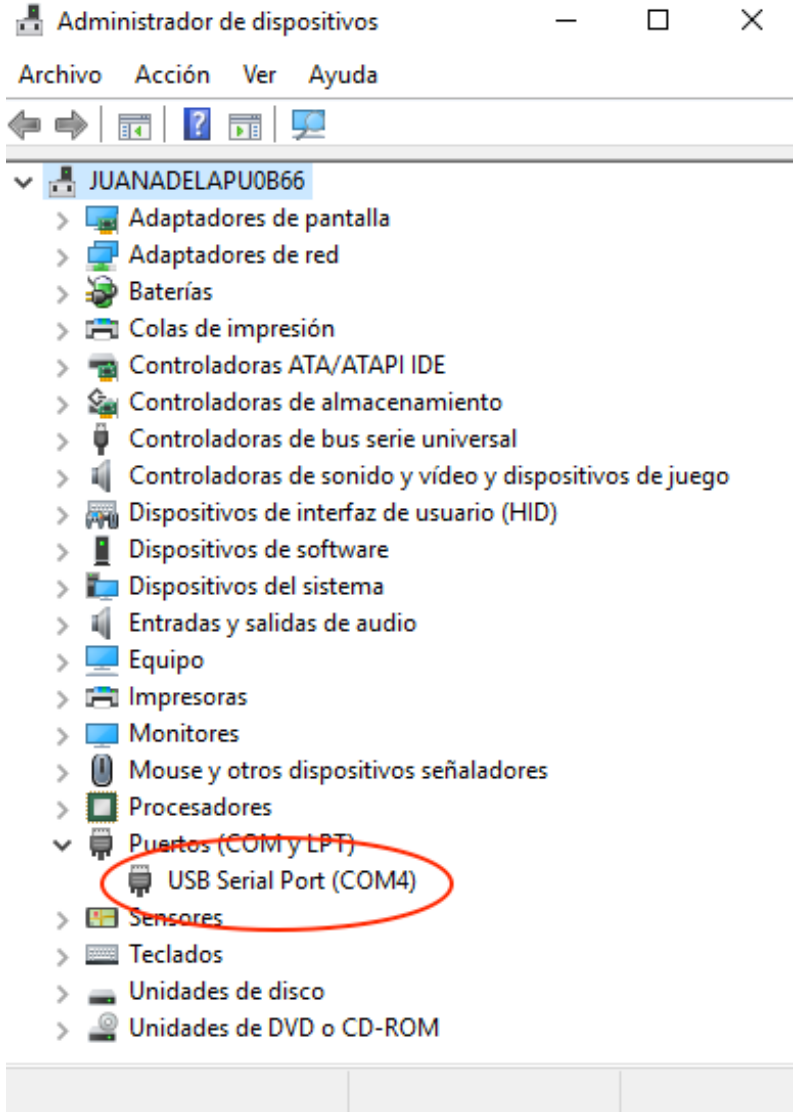
- El ejemplo implementa la función de “eco”. De forma que los caracteres recibidos por RX son retransmitidos por TX.
- Se ofrece un único comando “led” para hacer controlar un LED.
- Puede tomar los valores START, STOP y números enteros del 1 hasta el 9.
  - USO: led <START|STOP|1..9>
    - START: Inicia la secuencia
    - STOP: Para la secuencia
    - 1..9: Niveles de velocidad de encendido de LEDs.

Arranque del terminal

- Linux/MacOS: Mediante línea de mandatos con TERM

```
TERM=rxvt-256color screen
/dev/tty.usbserial-0001 9600,cs8,ixoff
```

- Windows/Linux: Con el cliente Putty
  - Download and install https://www.putty.org
  - Find the the USB port in the device manager (e.g. COM4)
  - Open PutTTY and configure as serial connection, 9600 bps, 8N1

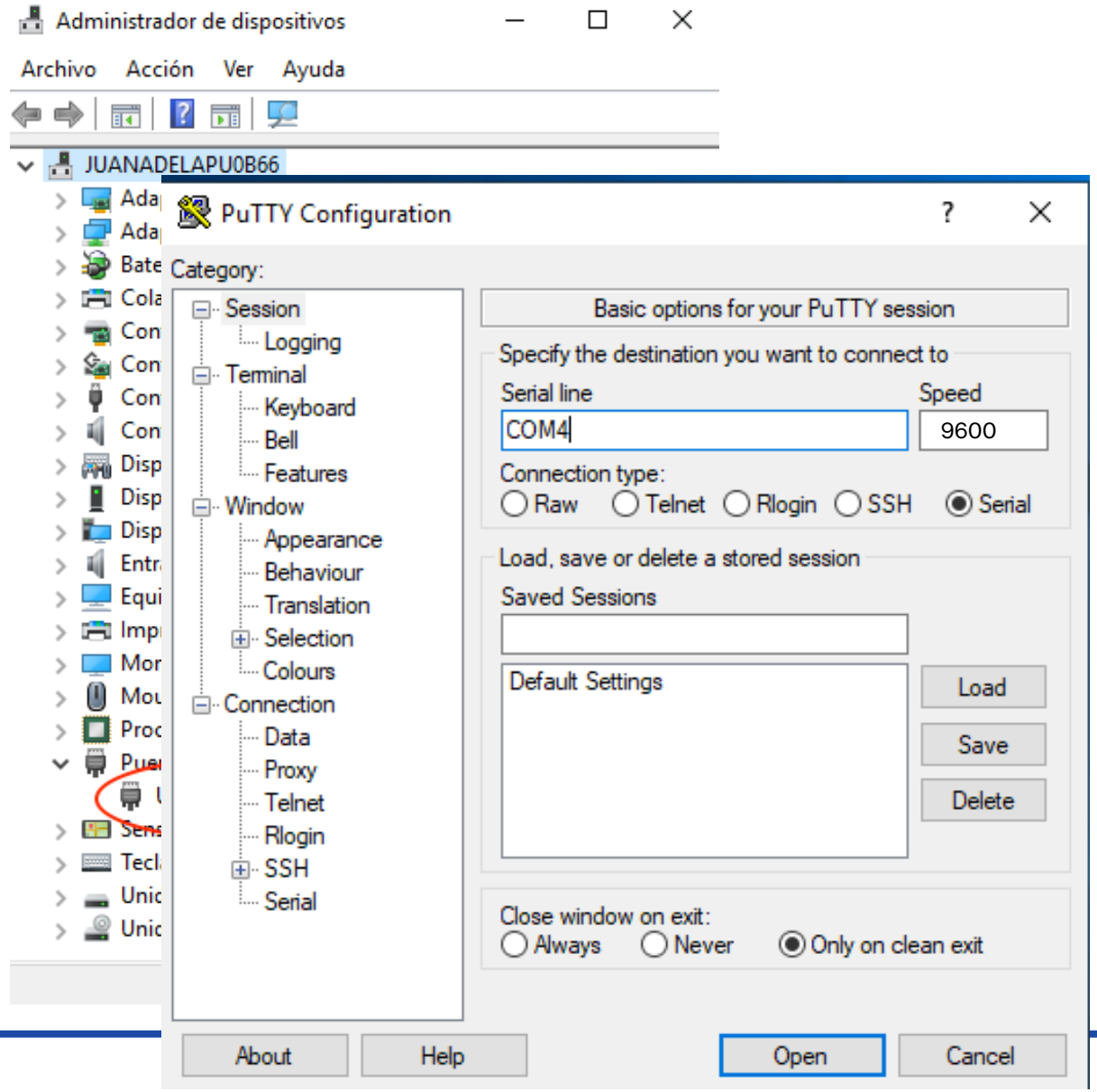


Arranque del terminal

- Linux/MacOS: Mediante línea de mandatos con TERM

```
TERM=rxvt-256color screen
/dev/tty.usbserial-0001 9600,cs8,ixoff
```

- Windows/Linux: Con el cliente Putty
  - Download and install https://www.putty.org
  - Find the the USB port in the device manager (e.g. COM4)
  - Open PutTTY and configure as serial connection, 9600 bps, 8N1



Referencias

Referencias

- **[1]** ATMEL. ATmega16/32U4 datasheet. https://www.datsi.fi.upm.es/docencia/Informatica_Industrial/DMC/pdf/atmega32u4.pdf
- **[2]** Elecia White. *Making Embedded Systems, 2nd Edition*. O'Reilly Media, Inc. https://learning.oreilly.com/library/view/making-embedded-systems/9781098151539/
- **[3]** Putty. Terminal Client. https://www.putty.org
- **[4]** DATSI. *Ejemplos para programar la USART*. Página web de la asignatura https://www.datsi.fi.upm.es/docencia/Informatica_Industrial/DMC/eje_usart_vmlab.rar
- **[5]** DATSI. *Resumen y guía para programar el modulo LCD por sobre I2C.* https://www.datsi.fi.upm.es/docencia/Informatica_Industrial/DMC/pdf/lcd_i2c.pdf
- **[6]** Arduino. *API de la librería I²C Wire*. https://www.arduino.cc/reference/en/language/functions/communication/wire/