Diseño con microcontroladores

Entrada/Salida Serie

Marco Xavier Rivera González
marco.rivera@upm.es

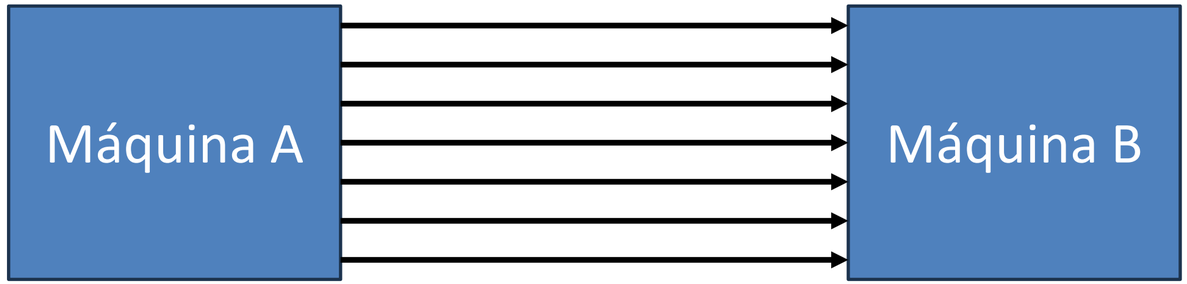
Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

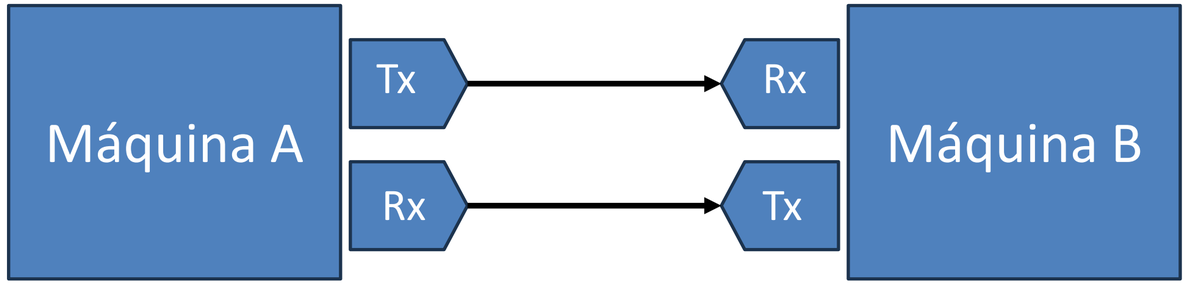
UNIVERSIDAD
POLITÉCNICA
DE MADRID

Comunicación Serial

- El protocolo de comunicación entre dispositivos es crucial para la interpretación de la información.



- Comunicación Paralela



- Comunicación Serial

Comunicación Serial

- La comunicación Serial USART (Universal Synchronous Asynchronous Receiver Transmitter) es la evolución de las comunicaciones para dar la posibilidad a que las maquinas se comuniquen entre si.

Comunicación Serial

- La historia de USART no comienza con las computadoras, sino con máquinas de escribir a distancia.

Teletype Model 37

- El teletype: Era una máquina de escribir que, al presionar una tecla, convertía esa letra en un código estandarizado (como el código Ascii) y lo enviaba como una serie de pulsos eléctricos.

- En el otro extremo, una máquina receptora decodificaba esos pulsos e imprimía la letra.

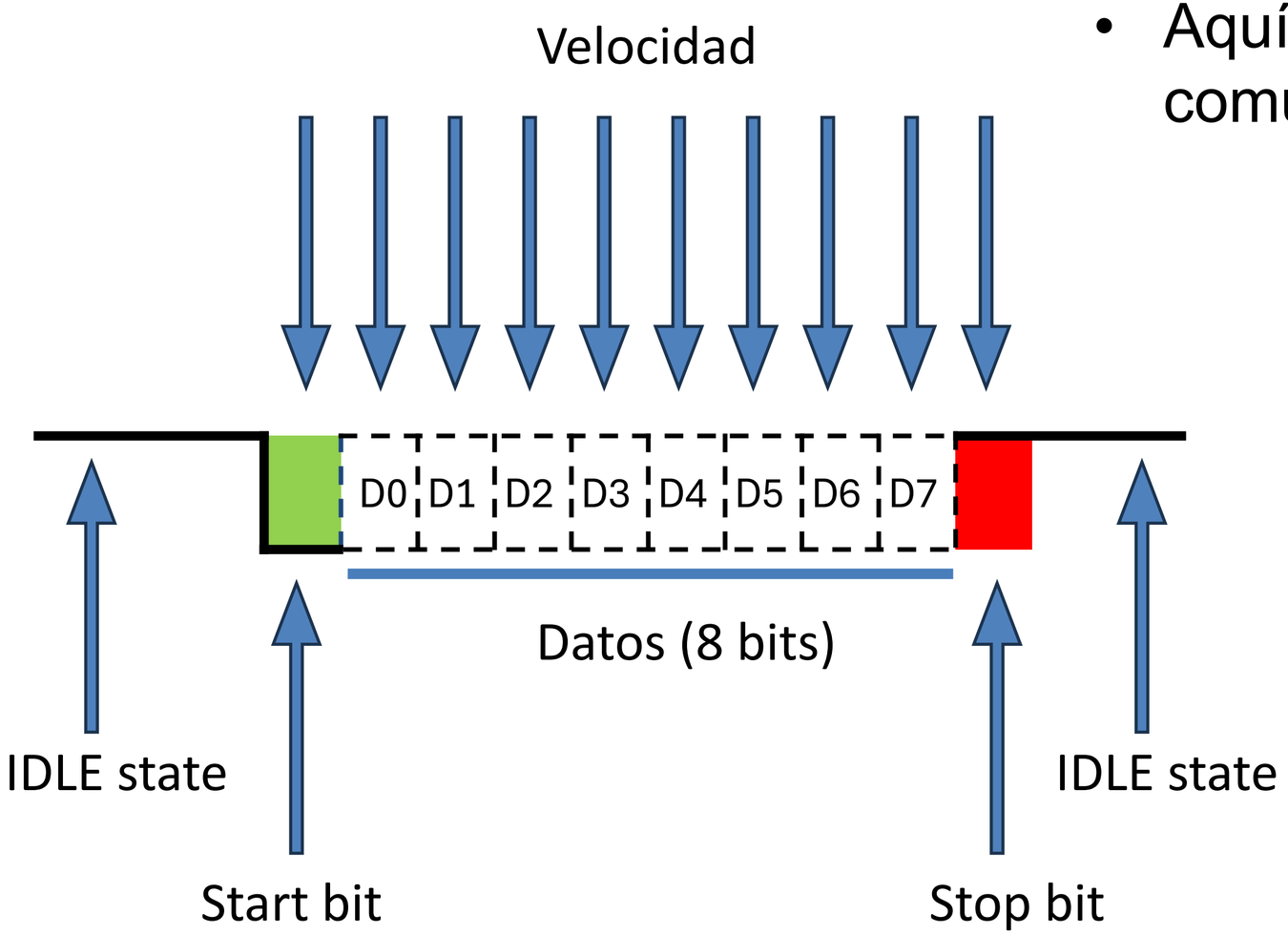
Comunicación Serial

- La historia de USART no comienza con las computadoras, sino con máquinas de escribir a distancia.

Teletype Model 37

- Aquí nació el concepto de la comunicación asíncrona:
  - No había señal de reloj.
  - Sincronización.
  - Velocidad Fija

Comunicación Serial



- Aquí nació el concepto de la comunicación asíncrona:
  - **No había señal de reloj:** señal que indique en que momento leer los datos
  - **Sincronización**: Bit de inicio (start bit) y el bit de parada (stop bit)
  - **Velocidad Fija:** velocidad en que se comunican (baudios)

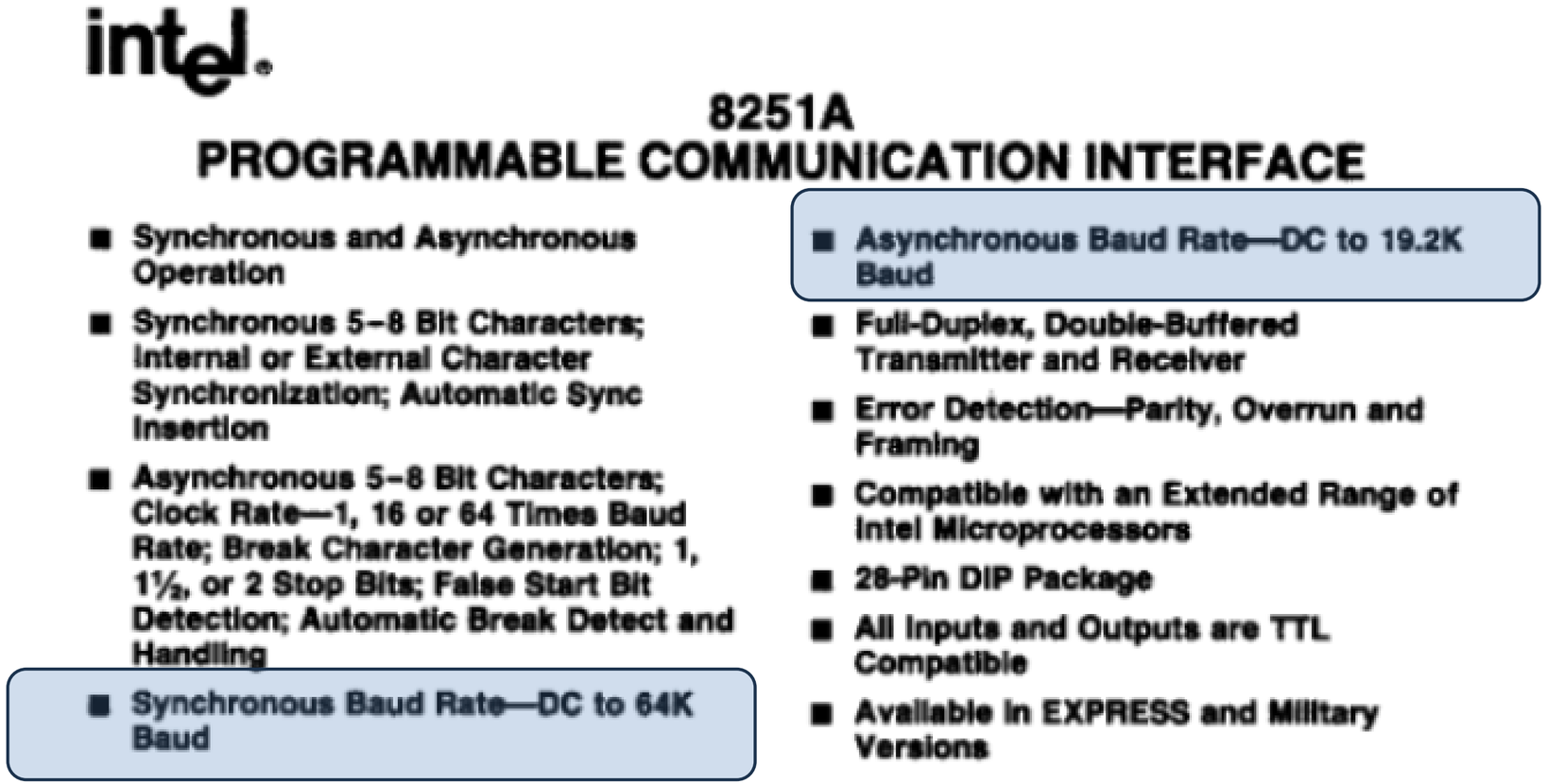
Comunicación Serial

- Al principio, el microprocesador (CPU) tenía que hacer todo el trabajo de "enmarcar" los datos: añadir el bit de inicio, enviar los bits de datos uno por uno a la velocidad correcta y añadir el bit de parada. Pero consumía muchísimo tiempo de la CPU.

- El CPU trabaja únicamente para mantener la lenta temporización del serial.
  - Solución, crear el Hardware UART (Universal Asynchronous Receiver Transmitter) que actúa como un asistente independiente.
    - La CPU entrega un byte, y este se encarga de la temporización y añadir los bits de inicio y parada.
    - Al mismo tiempo espera un bit de inicio, para leer los datos y enviar al CPU.

Comunicación Serial

- En 1971, se lanza la interface Intel 8251 para el procesador 8008. Pionero en comunicación Serial, primer hardware **USART**.



intel

8251A
PROGRAMMABLE COMMUNICATION INTERFACE

- Synchronous and Asynchronous Operation
- Synchronous 5–8 Bit Characters; Internal or External Character Synchronization; Automatic Sync Insertion
- Asynchronous 5–8 Bit Characters; Clock Rate—1, 16 or 64 Times Baud Rate; Break Character Generation; 1, 1½, or 2 Stop Bits; False Start Bit Detection; Automatic Break Detect and Handling
- Synchronous Baud Rate—DC to 64K Baud
- Asynchronous Baud Rate—DC to 19.2K Baud
- Full-Duplex, Double-Buffered Transmitter and Receiver
- Error Detection—Parity, Overrun and Framing
- Compatible with an Extended Range of Intel Microprocessors
- 28-Pin DIP Package
- All Inputs and Outputs are TTL Compatible
- Available in EXPRESS and Military Versions

Comunicación Serial

- En 1971, se lanza la interface Intel 8251 para el procesador 8008. Pionero en comunicación Serial, primer hardware **USART**.

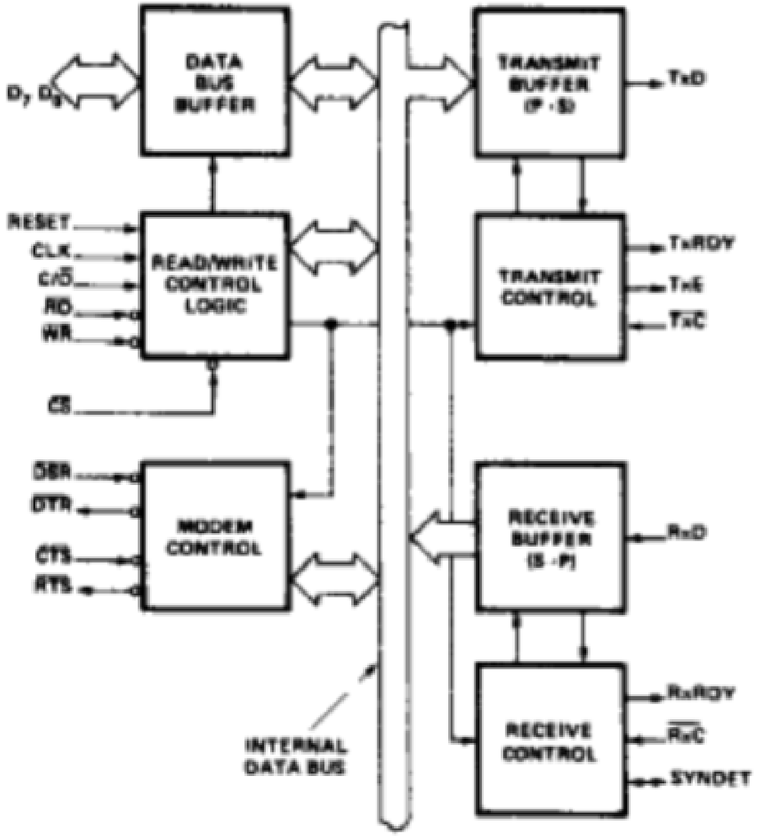


Figure 1. Block Diagram

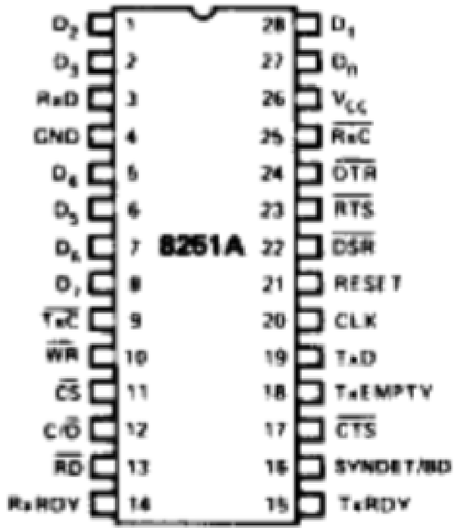
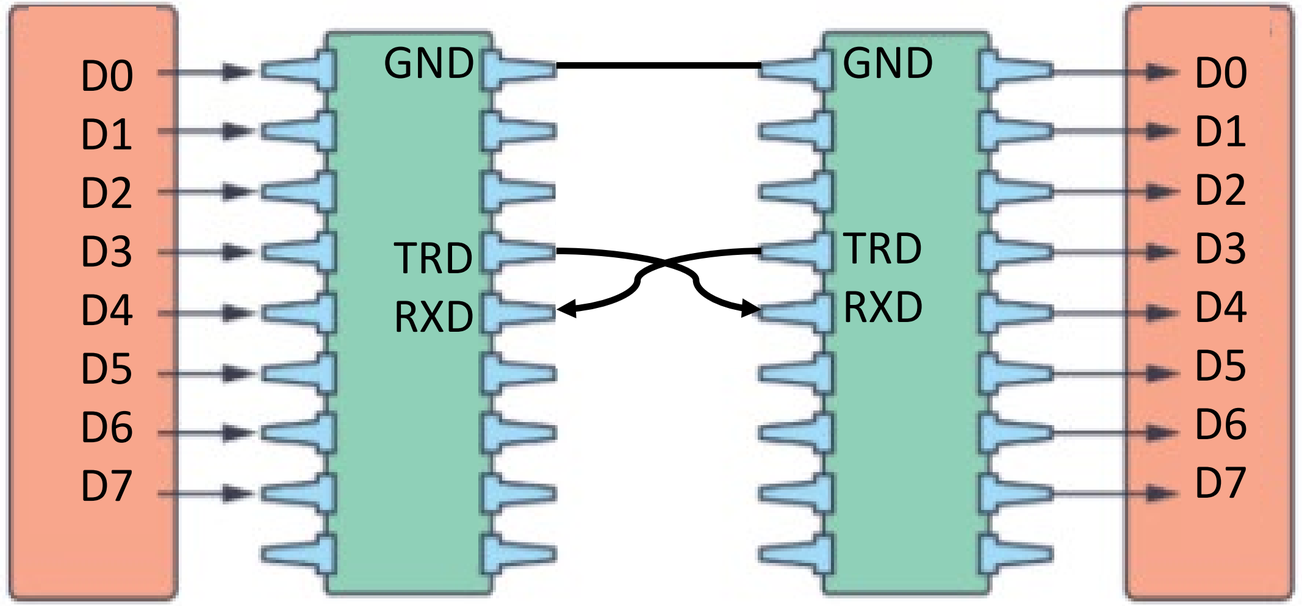


Figure 2. Pin Configuration

Comunicación Serial: Estandarización RS-232 y puerto COM

- El RS-232 (Recommended Standard 232) fue el estándar industrial y se definió para ser robusto y fiable en comunicaciones a larga distancia.



Comunicación Serial: Estandarización RS-232 y puerto COM

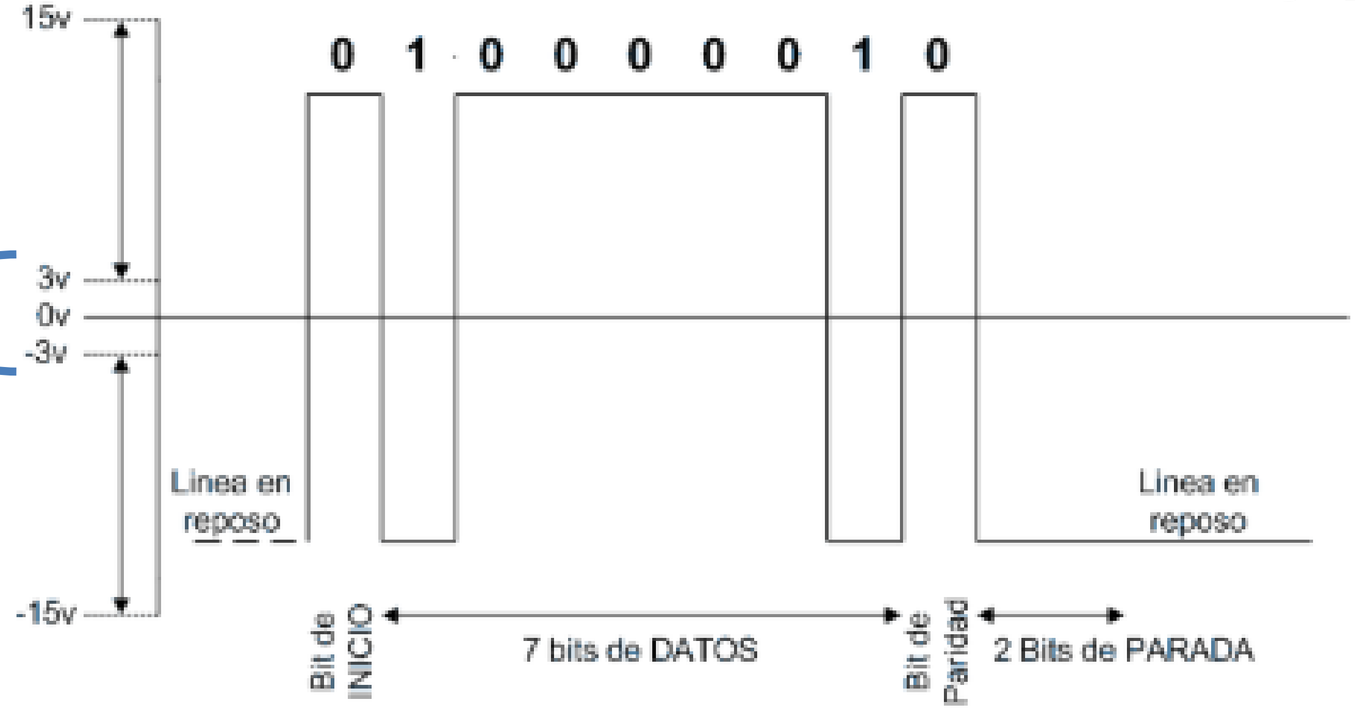
- **Voltajes:** Un '1' lógico era un voltaje negativo (ej. -12V) y un '0' lógico era un voltaje positivo (ej. +12V). (Genera robustes)

- **Conectores:** Definió los conectores DB-25 y, más tarde, el **DB-9** o **"Puerto Serial" (COM Port)**.

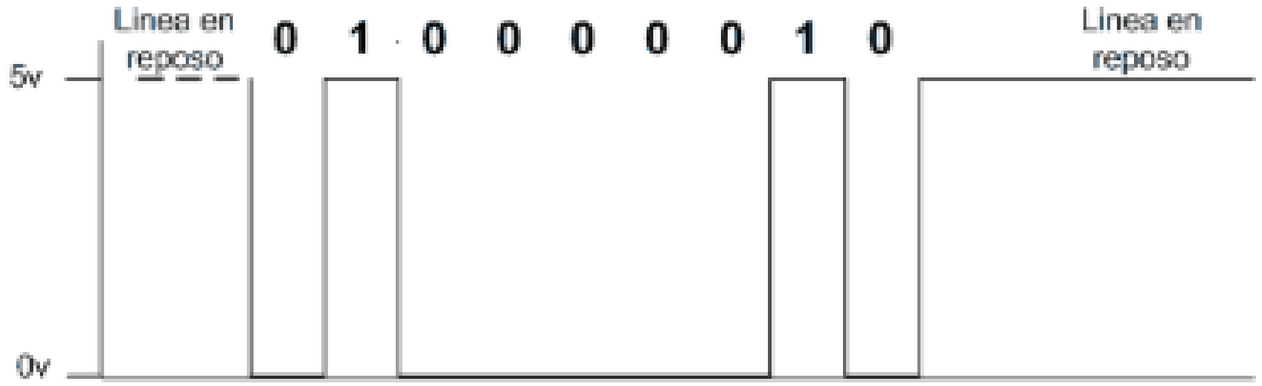
- En los años 80 y 90 fue el estándar para la conexión de periféricos al ordenador (mouse, impresoras, módems, etc)

Comunicación Serial: Estandarización RS-232 y puerto COM

Zona muerta



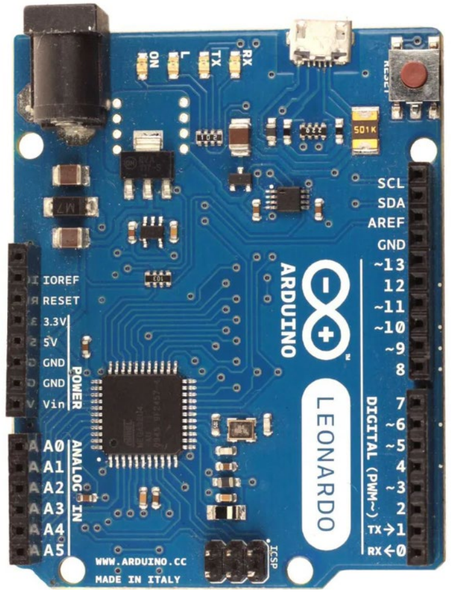
RS-232

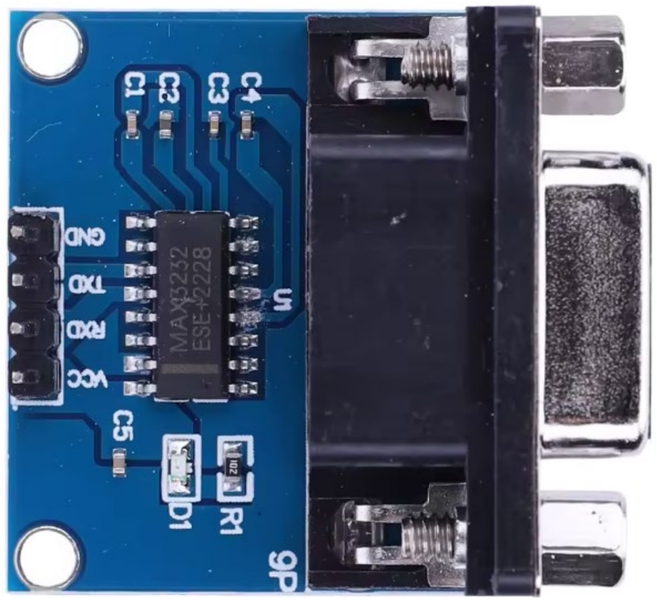


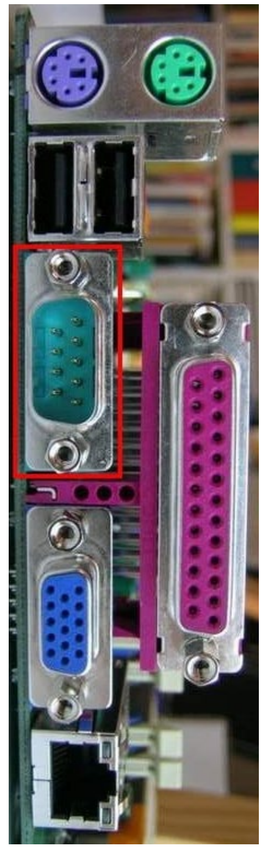
TTL

Comunicación Serial: MAX 232

- Es un traductor de TTL a RS-232, es decir cambia los voltajes de +/- 12 V a 0/5 V para conectar con cualquier microcontrolador



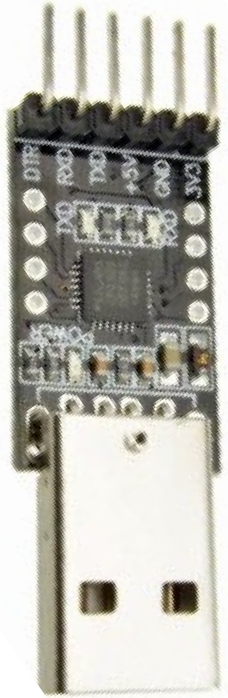




Comunicación Serial: USB/RS-232/USART

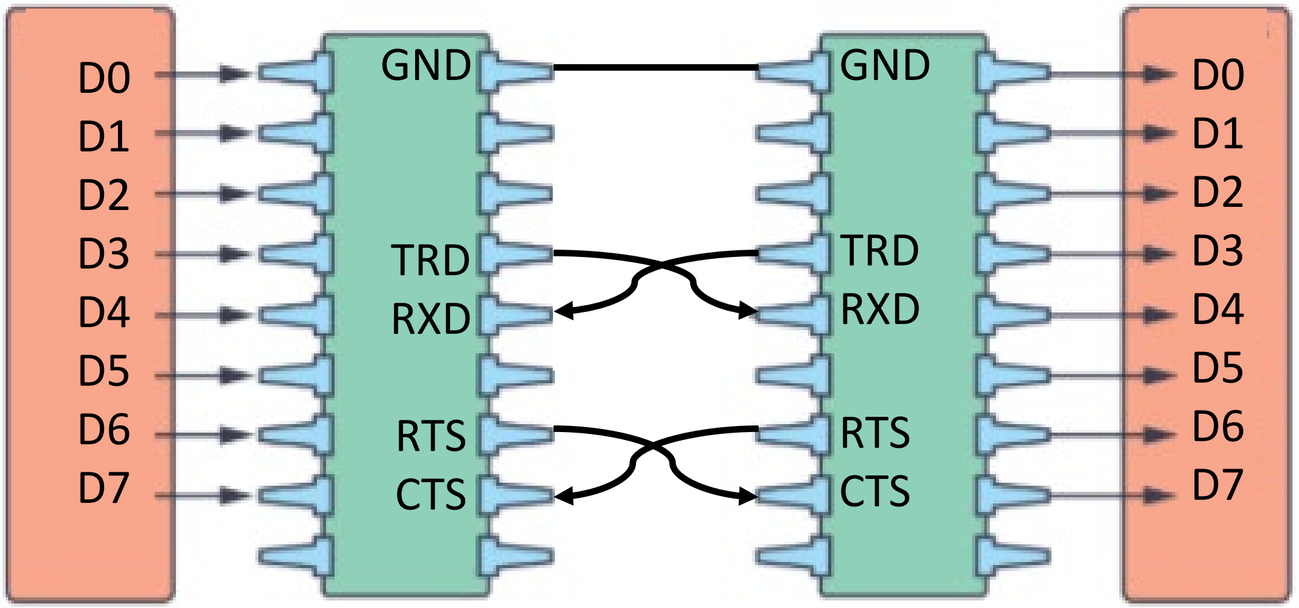
- A finales de los 90, apareció el USB (Universal Serie Bus) el cual era mas rápido, proporcionaba alimentación, y era mas pequeño.

- El conector RS-232 (DB9) desapareció, dejo de ser un chip externo, y se convirtió en un bloque dentro del microcontrolador.



Característica	USART (TTL)	RS-232
1 Lógico	3.3V o 5V	-3V a -15V
O Lógico	0V	+3 a +15 V
Conector	Conexión directa en un chip	DB-9 (Puerto COM)
Control de fuljo	Por software	Hardware (RTS/CTS)
Uso Típico	Comunicación entre chips.	Módems, impresoras, equipos industriales

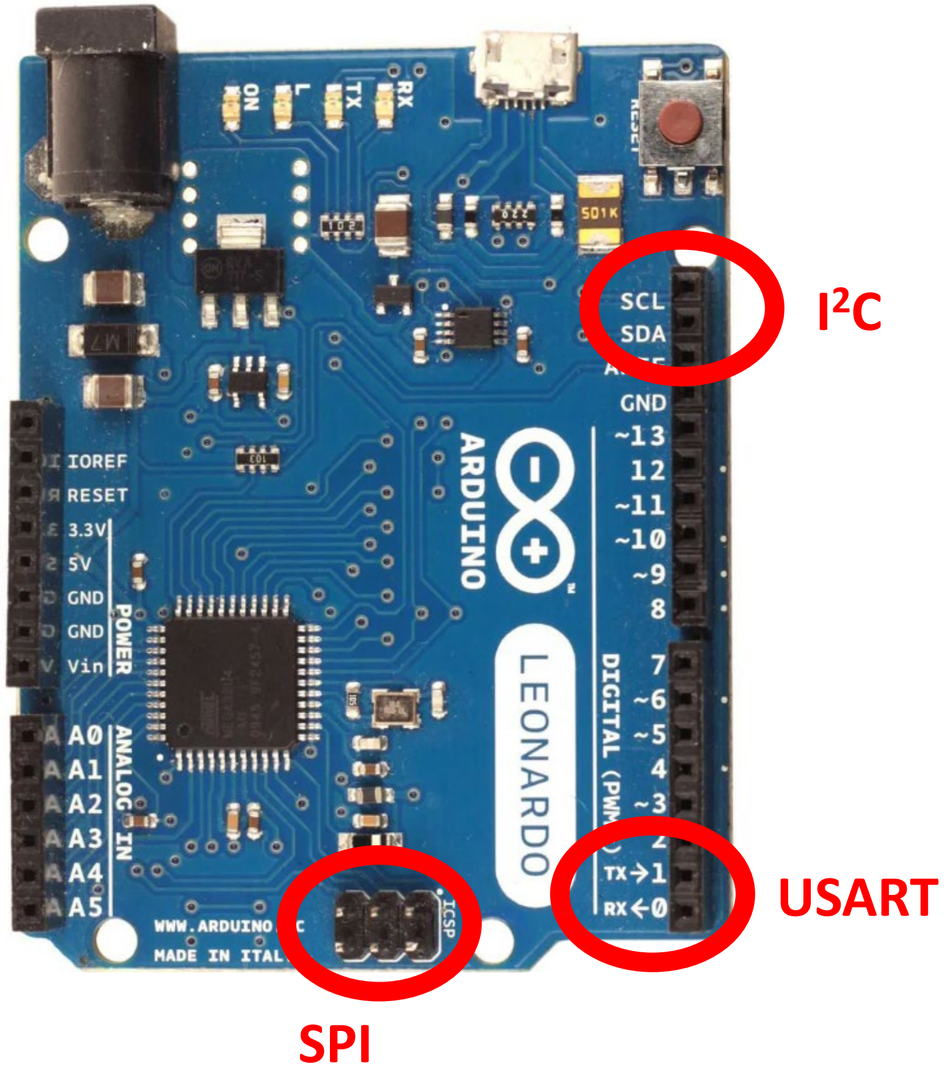
Comunicación Serial: USB/RS-232/USART



- RTS (salida): Request To Send: El receptor utiliza esta señal para indicar que esta listo para recibir.

- CTS (Entrada): Clear To Send: El transmisor mira constantemente y pausa la transmisión si esta señal esta activa.

USART: en ATMega32U4 (Arduino Leonardo)



El microcontrolador incluye:

- Interface I²C x1
- Interface SPI x1

} Con reloj

- Interface USART x1

} Sin reloj

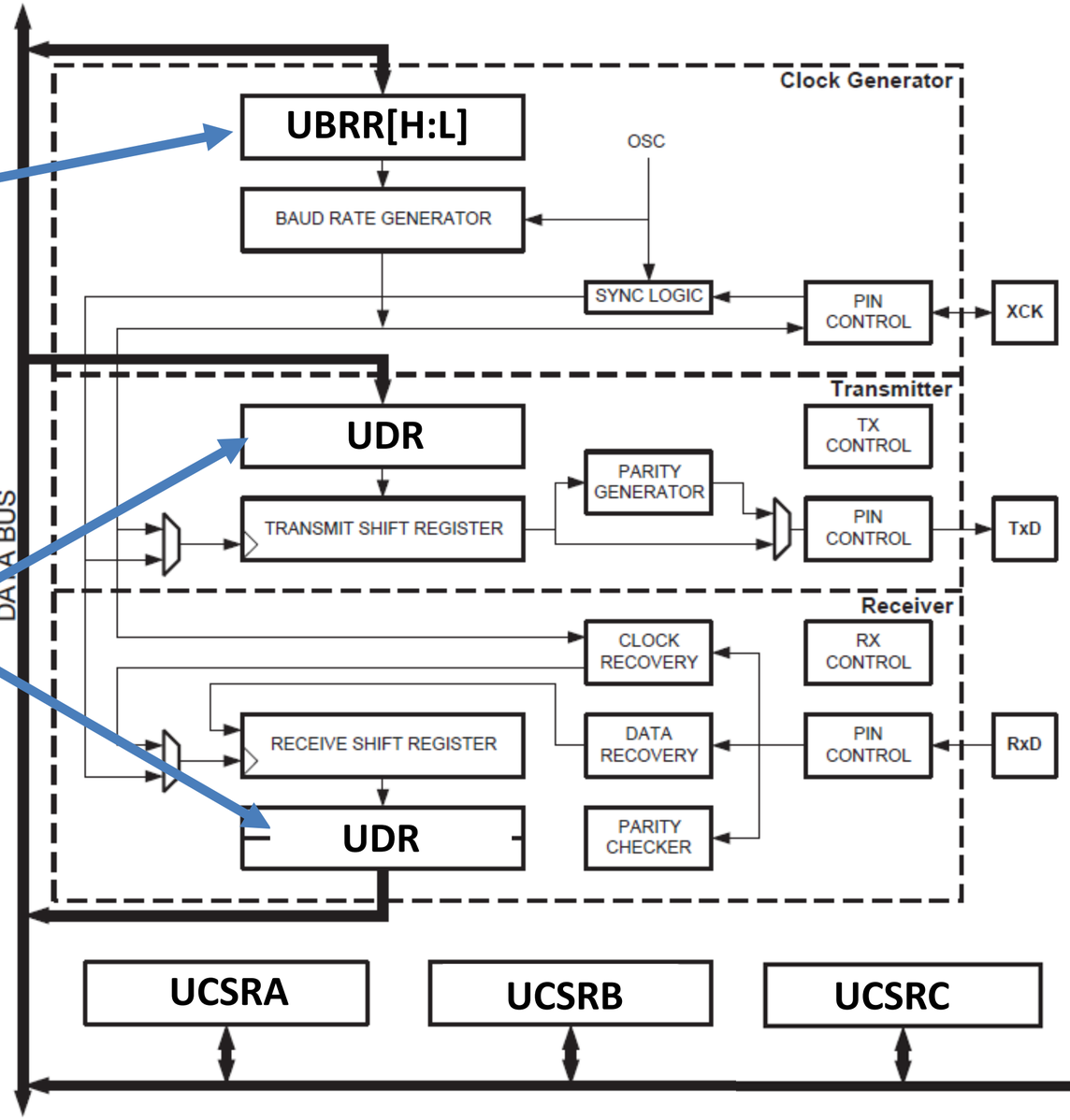
Características del USART en ATMega32U4

- **Comunicación full-dúplex**
- **Control de flujo:**
    - CTS (Clear To Send),
    - RTS (Request to Send).
- **Longitud de datos:**
    - Bits de inicio: 1.
    - Datos: 5, 6, 7, 8, 9.
    - Bits de Parada: 1 o 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.*

Características del USART en ATMega32U4

Baud Rate Register

Buffer de Datos UDR



Generador de Reloj

Transmisor

Receptor

Registros de control y de estatus

Registros del USART en ATMega32U4

- El atmega32U4 solo tiene un USART de hardware: USART1
- Todos los registros llevan el sufijo ‘1’:
    - ***UDR1:*** Registro de Datos (Buffer TX/RX).
    - ***UCSR1A:*** Control y Estado A (Banderas).
    - ***UCSR1B:*** Control y Estado B (Habilitación, Interrupciones).
    - ***UCSR1C:*** Control y Estado C (Formato de Trama).
    - ***UCSR1D:*** Control y Estado D (Habilitación de señales de control de flujo).
    - ***UBRRH1 y UBRRL1***: Registros de Baud Rate.

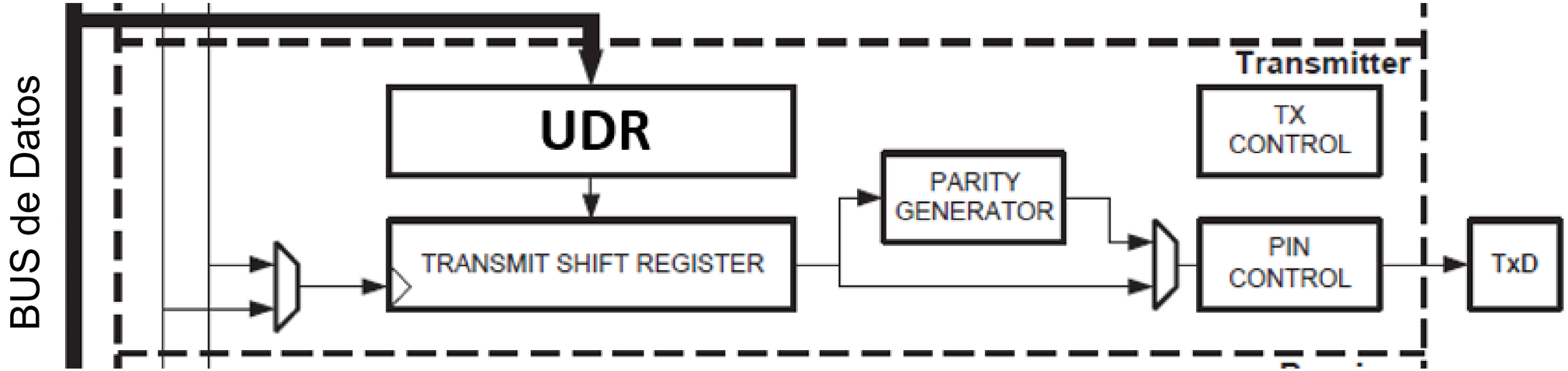
Transmisor

- Registro UDR: de solo escritura para la **escritura**.

1. Mira el estado de CTS:
   a) Activo (nivel bajo, 0V): se puede enviar
   b) Inactivo (nivel alto, 5V): dejar de enviar
2. Escritura de caracteres en UDR.
3. Copia en el registro de transmisión.
4. Se arma el mensaje y se envía:

INICIO	DATA (UDR)	PARIDAD	PARADA

5. Notificar disponibilidad por el RTS:



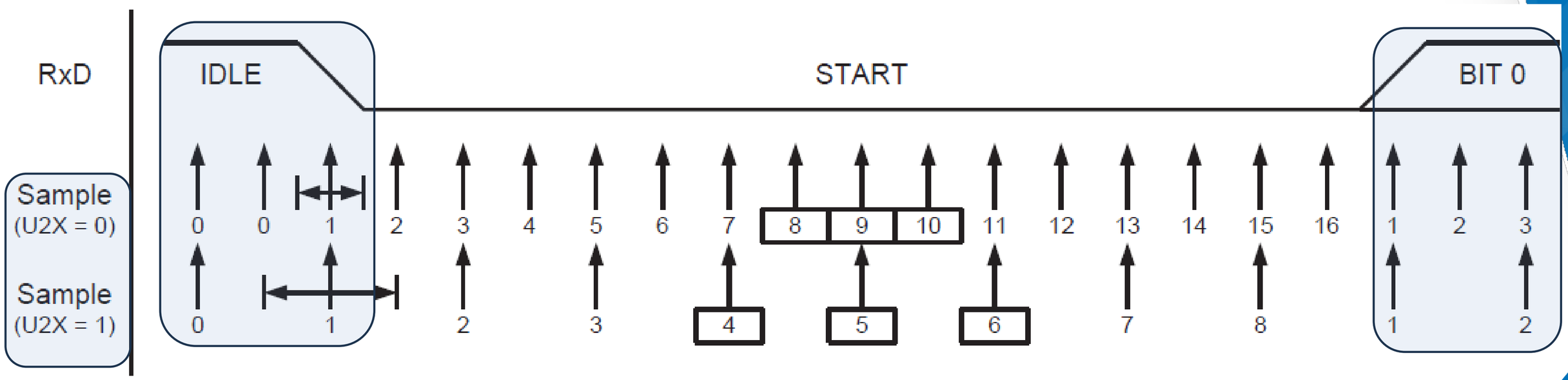
Receptor: Recepción de bit START

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

1. RxD → IDLE (nivel alto):
   - Contador interno a 0.

2. RxD → Cambia de estado a nivel Bajo
   - Inicia la detección del bit de INICIO

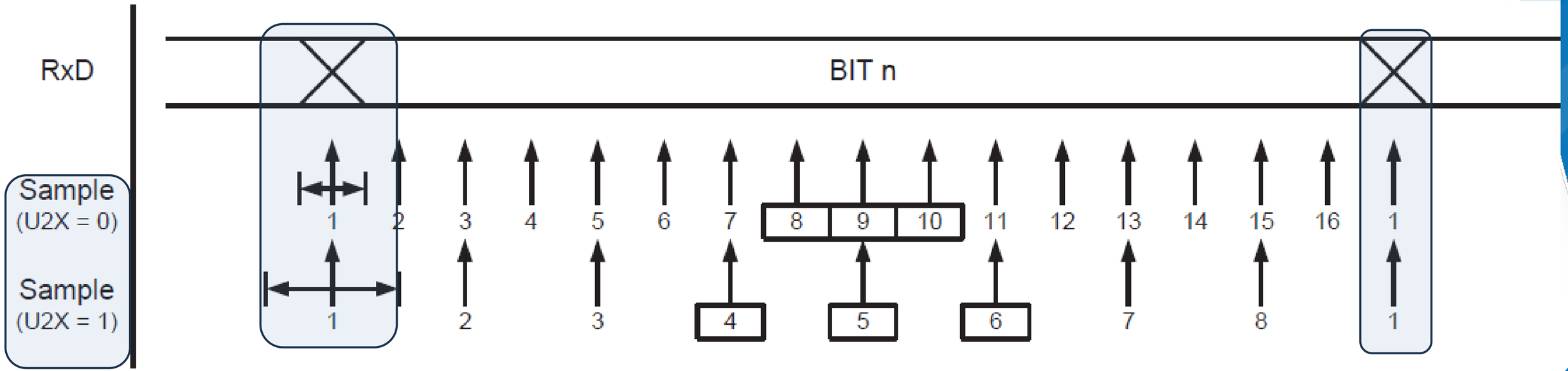
3. RxD → Bit INICIO (nivel bajo):
   - Contador interno llega a 16
4. Las muestras 8, 9 y 10 definen el estado del bit por mayoría:
   - Al menos 2 a nivel ALTO → Ruido
   - Al menos 2 a nivel BAJO → Bit de inicio



Receptor: Recepción de bit de DATOS

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

1. Tras detectar el bit de inicio, 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 estado del bit por mayoría:
   - Al menos 2 a nivel ALTO → detecta un '1'
   - Al menos 2 a nivel BAJO → detecta un '0'

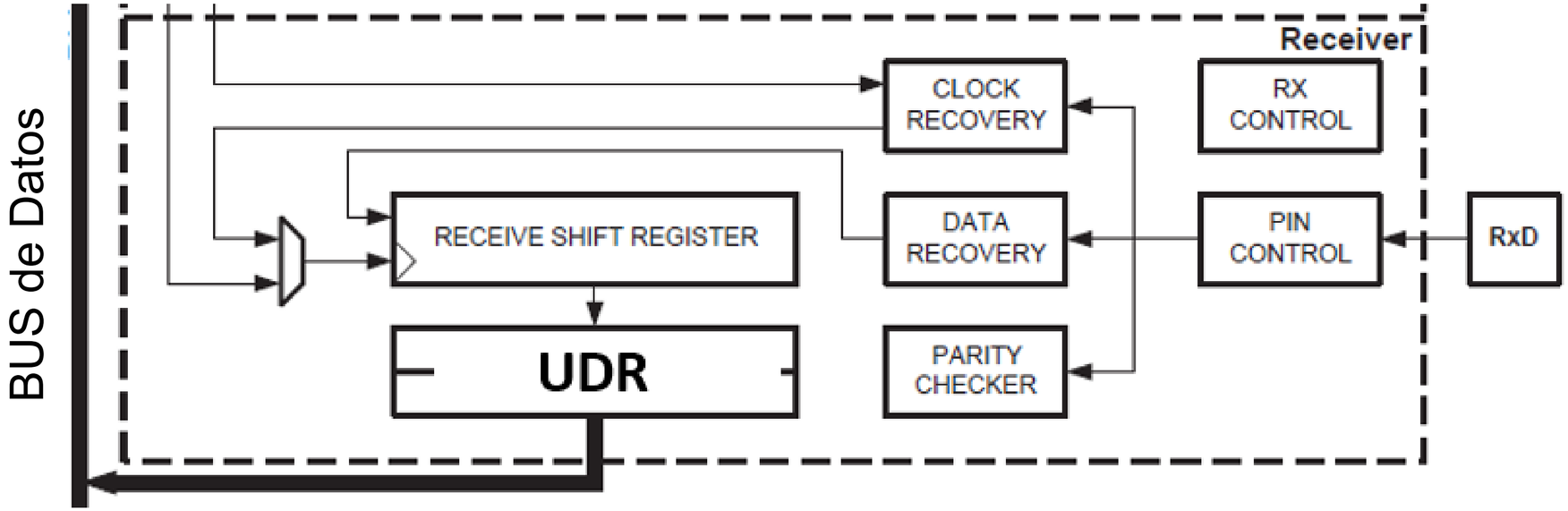


Receptor

- Registro UDR: de solo escritura para la **lectura**

1. Detecta el bit de INICIO
2. Detecta los bits de DATOS y carga en el registro de desplazamiento
3. Se introducen los bits de DATOS en el registro UDR
4. Evalúa el bit PARIDAD para detección de errores

- **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:**
  - Error de Formateo
  - Error de Paridad
  - Over Run



Registros del USART en ATMega32U4

- El atmega32U4 solo tiene un USART de hardware: USART1
- Todos los registros llevan el sufijo ‘1’:
    - ***UDR1:*** Registro de Datos (Buffer TX/RX).
    - ***UCSR1A:*** Control y Estado A (Banderas).
    - ***UCSR1B:*** Control y Estado B (Habilitación, Interrupciones).
    - ***UCSR1C:*** Control y Estado C (Formato de Trama).
    - ***UCSR1D:*** Control y Estado D (Habilitación de señales de control de flujo).
    - ***UBRRH1 y UBRRL1***: Registros de Baud Rate.

UDR1: Registro de Datos (Buffer TX/RX).

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDRn (Read)
	TXB[7:0]								UDRn (Write)
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	0	0	0	

El registro UDR1 apunta a dos registros físicos diferentes:

Transmisión:

UDRE1 (Data Register Empty) (en UCSR1A) debe estar en ‘1’para que el registro TXB esté libre para recibir un nuevo byte.

- Si se intenta escribir en UDR1 cuando UDRE1 es ‘0’,el dato se ignora. Esto asegura que la CPU no sobrescriba un byte que el hardware aún no ha tenido tiempo de mover al Registro de Desplazamiento.

- Al terminar de enviar la trama completa, UDRE1 automáticamente se pone a ‘1’, indicando al CPU que se puede enviar el siguiente DATO.

UDR1: Registro de Datos (Buffer TX/RX).

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDRn (Read)
	TXB[7:0]								UDRn (Write)
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	0	0	0	

El registro UDR1 apunta a dos registros físicos diferentes:

Recepción:

RXC1 (Receive Complete) (en UCSR1A) se coloca automáticamente a ‘1’ cuando un byte ha sido ensamblado por el hardware y esta listo en el registro de lectura.

- Al leer UDR1, la bandera RXC1 se borra automáticamente a ‘0’.

UCSR1A: Control y Estado A (Banderas).

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	

Banderas para 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.

Estados de error

- **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.

UCSR1B: Control y Estado B (Habilitación, Interrupciones).

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	

Habilitación de Rx/Tx

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

Habilitación 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.

UCSR1B: Control y Estado B (Habilitación, Interrupciones).

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	

9no Bit de DATO

- **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).

Longitud de Datos

- **UCZ12:** Junto con UCZ10 y UCZ11 ubicados en el registro UCSR1C, define la longitud de bits de datos a transmitir y recibir.

UCSR1C: Control y Estado C (Formato de Trama).

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 operación del USART
0	0	**USART en modo Asíncrono (para la práctica).**
0	1	USART en modo Síncrono
1	0	(No se utiliza) /reservado
1	1	USART en modo Maestro para comunicación SPI

UCSR1C: Control y Estado C (Formato de Trama).

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	

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

UPM11	UPM10	Modo de operación de PARIDAD
0	0	**Desactivado (Estándar) (Se utiliza para la práctica)**
0	1	(No se utiliza) /reservado
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.

UCSR1C: Control y Estado C (Formato de Trama).

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.

USBS1	Cantidad de bits de PARADA
0	**1-bit (estándar) (Se utiliza para la práctica)**
1	2-bit

UCSR1C: Control y Estado C (Formato de Trama).

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	

- **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 (Estándar) (Para la práctica)**
1	0	0	(No se utiliza) /reservado
1	0	1	(No se utiliza) /reservado
1	1	0	(No se utiliza) /reservado
1	1	1	9-bit

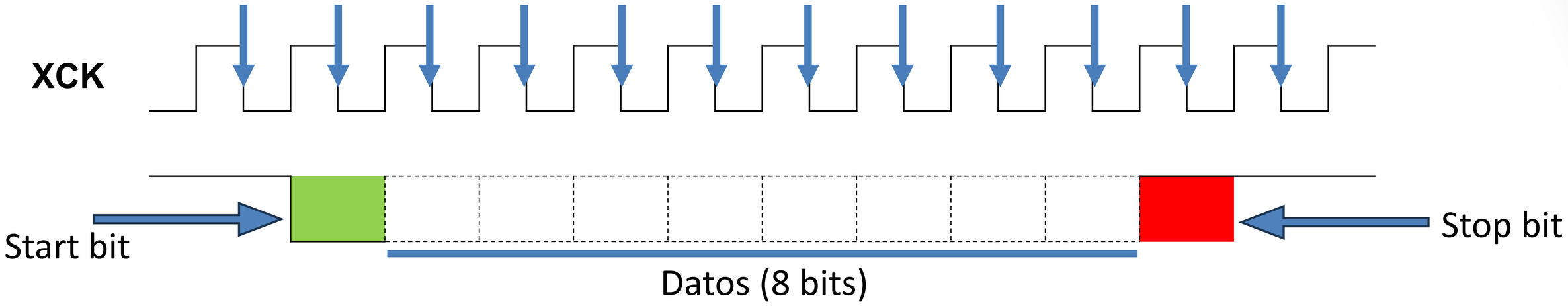
UCSR1C: Control y Estado C (Formato de Trama).

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	

- **CPOL1,** relaciona los datos de salida y de entrada con la señal de reloj (XCK)

CPOL1	Data a transmitir (TxD pin)	Data a recibir (RxD pin)
0	**Flanco de subida en XCK**	**Flanco de bajada en XCK**
1	Flanco de bajada en XCK	Flanco de subida en XCK

Estándar (práctica)



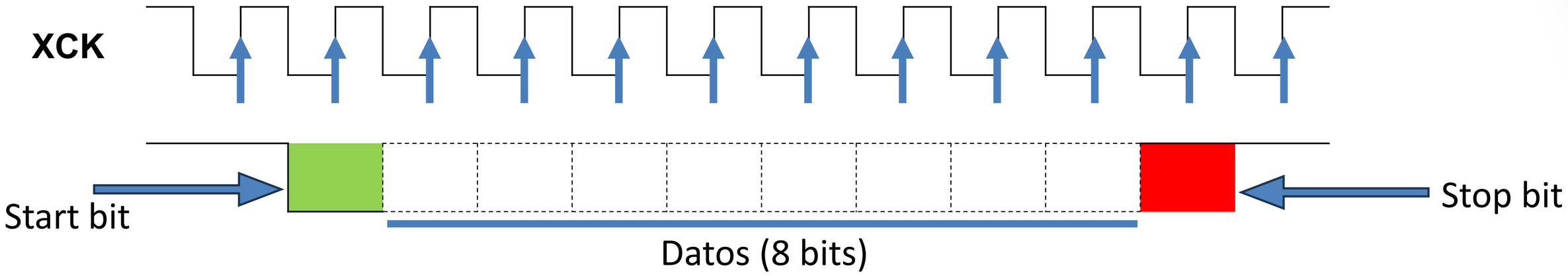
UCSR1C: Control y Estado C (Formato de Trama).

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	

- **CPOL1,** relaciona los datos de salida y de entrada con la señal de reloj (XCK)

CPOL1	Data a transmitir (TxD pin)	Data a recibir (RxD pin)
0	**Flanco de subida en XCK**	**Flanco de bajada en XCK**
1	Flanco de bajada en XCK	Flanco de subida en XCK

Estándar (práctica)



UCSR1D: Control y Estado D (Habilitación de señales de control de flujo).

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	–	–	CTSEN	RTSEN	UCSRnD
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7:2:** No se utiliza) /reservado.

Habilitación de Señales de control de flujo

- **CTSEN:** habilita la señal de control de fuljo CTS. Se puede transmitir solo si en el pin de entrada de CTS hay un nivel bajo ‘0’.
- **RTSEN:** Habilita la señal de control de flujo RTS. En el pin RTS coloca automáticamente un nivel lógico alto ‘1’, si el buffer de entrada esta lleno.

UBRRH1 y UBRRL1: Registros de Baud Rate.

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRR[11:8]				UBRRHn
	UBRR[7:0]								UBRRLn
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

- **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).

$$UBBR1 = \frac{F_{CPU}}{16 \text{ x } BAUD} - 1$$

UBRRH1 y UBRRL1: Registros de Baud R

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRR[11:8]				UBRRHn
	UBRR[7:0]								UBRRLn
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

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

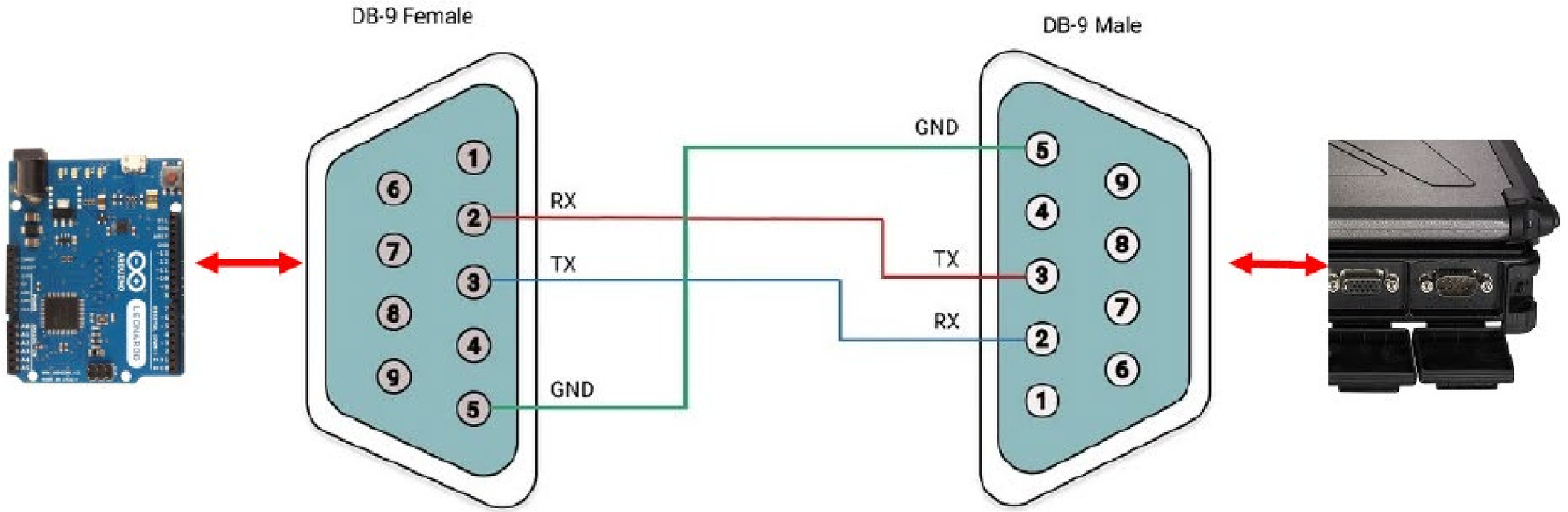
- **Hay dos opciones con U2X1=0 y U2X1=1.**
  - Elijo la que menos error tenga, es decir, U2X1=1 ya que obtendré un error del 0.6%.
  - Luego, UBRR debe valer 68.
- **Contenido de UBRRL1 y UBRRH1 para que UBR valga 68:**
- $68_{10} = 0000\ 01000100_2$

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	

Programación en Atmega32U4

Montaje

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

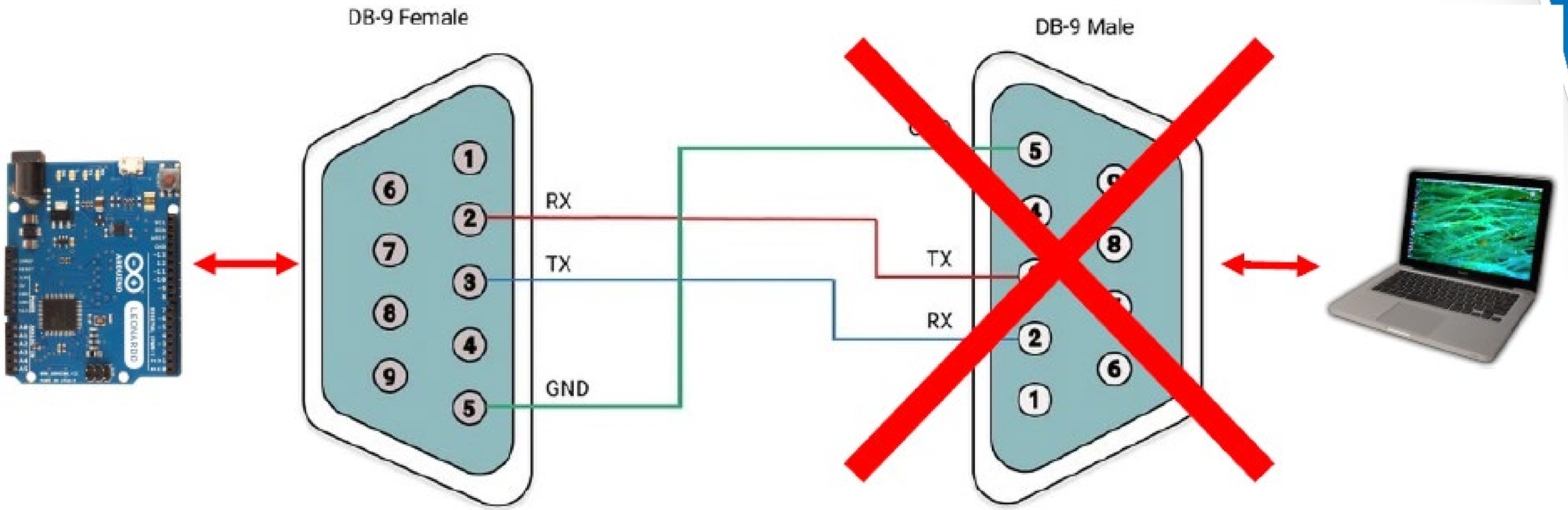


Programación en Atmega32U4

Montaje

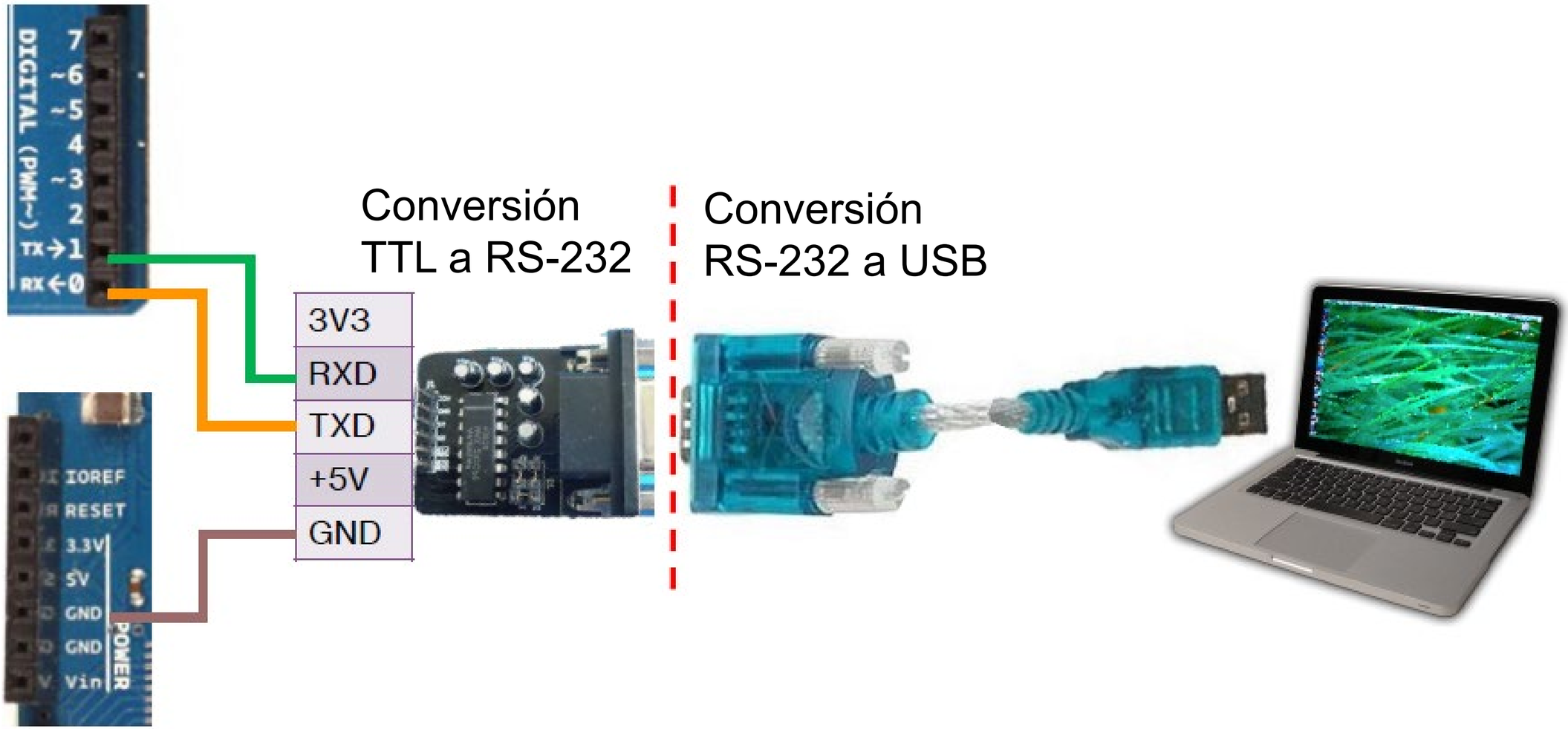
Niveles TTL

Interfaz USB, no RS-232



Programación en Atmega32U4

Montaje mediante cable DB-9

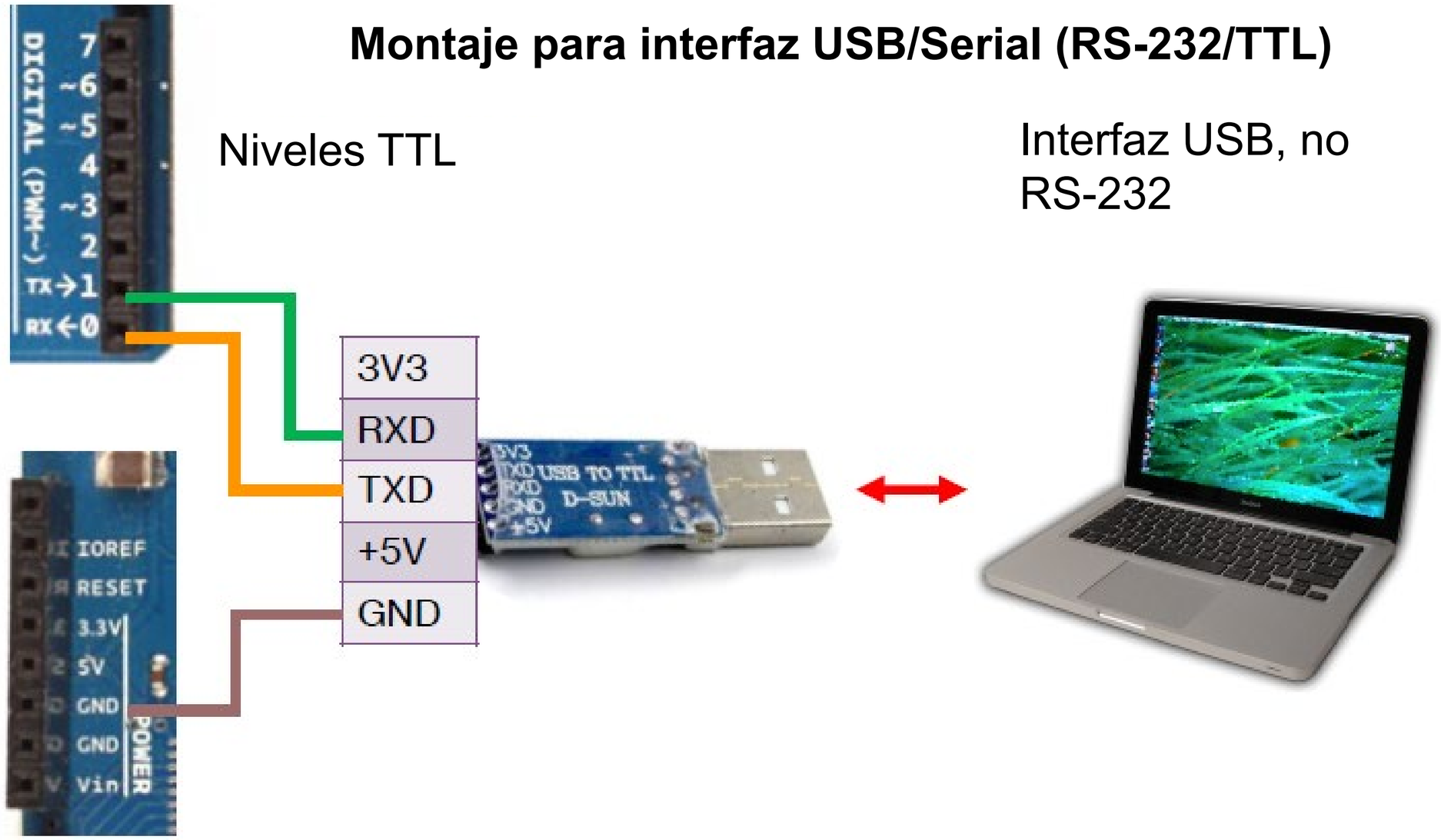


Programación en Atmega32U4

Montaje para interfaz USB/Serial (RS-232/TTL)

Niveles TTL

Interfaz USB, no RS-232



Simulación por VMLab

- **Simulado sobre un microcontrolador Atmega128.**
- **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 deSIGNAL(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

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.

Simulación por VMLab: 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'.

Simulación por VMLab: 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 El buffer está lleno.
  - Si in – out < 64 Hay huecos en el buffer.

Simulación por VMLab: Procesado de mensajes

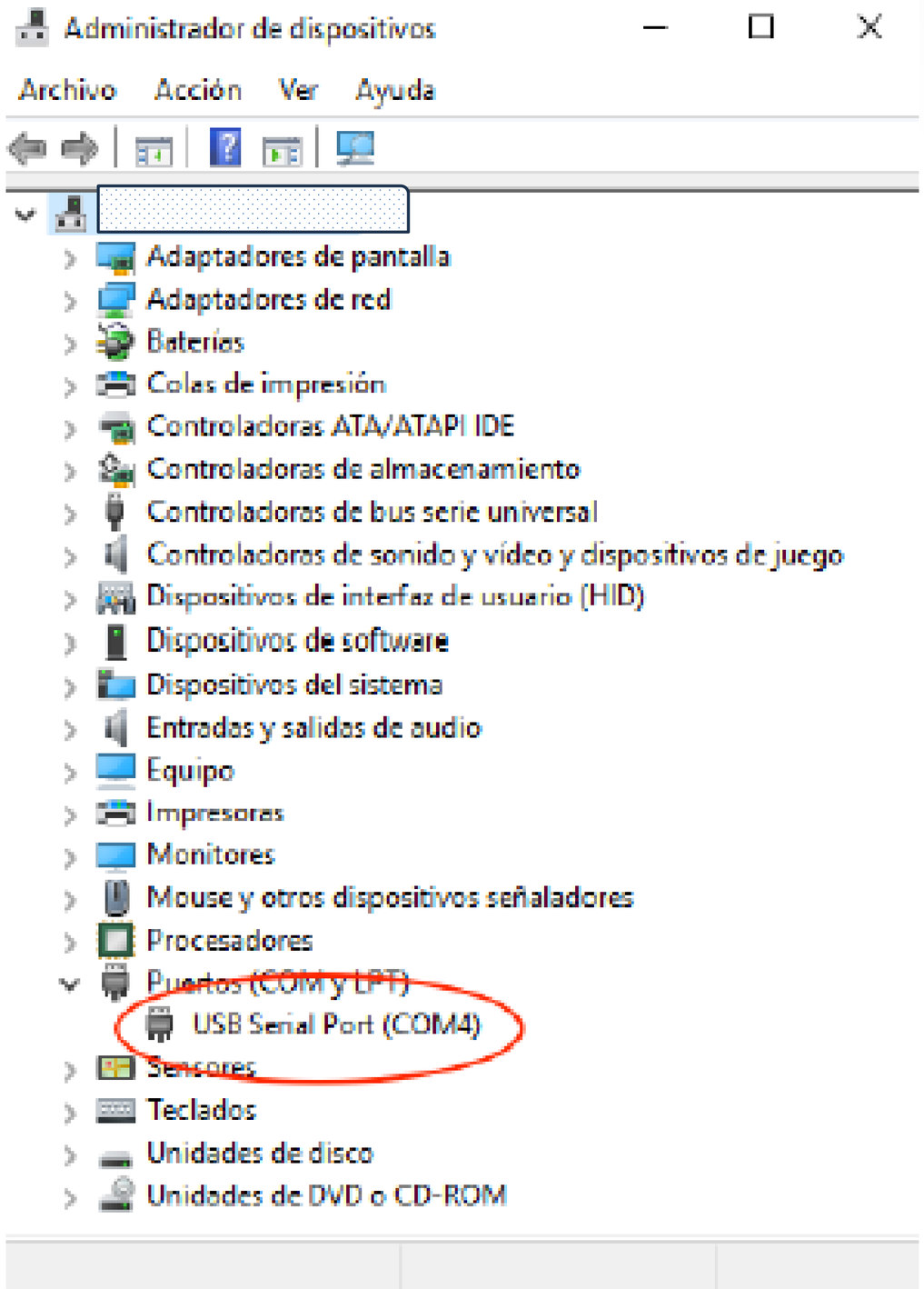
- **El ejemplo implementa la función de “eco”. De forma que los caracteres recibidos por RX son retransmitidos por TX.**

- **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.

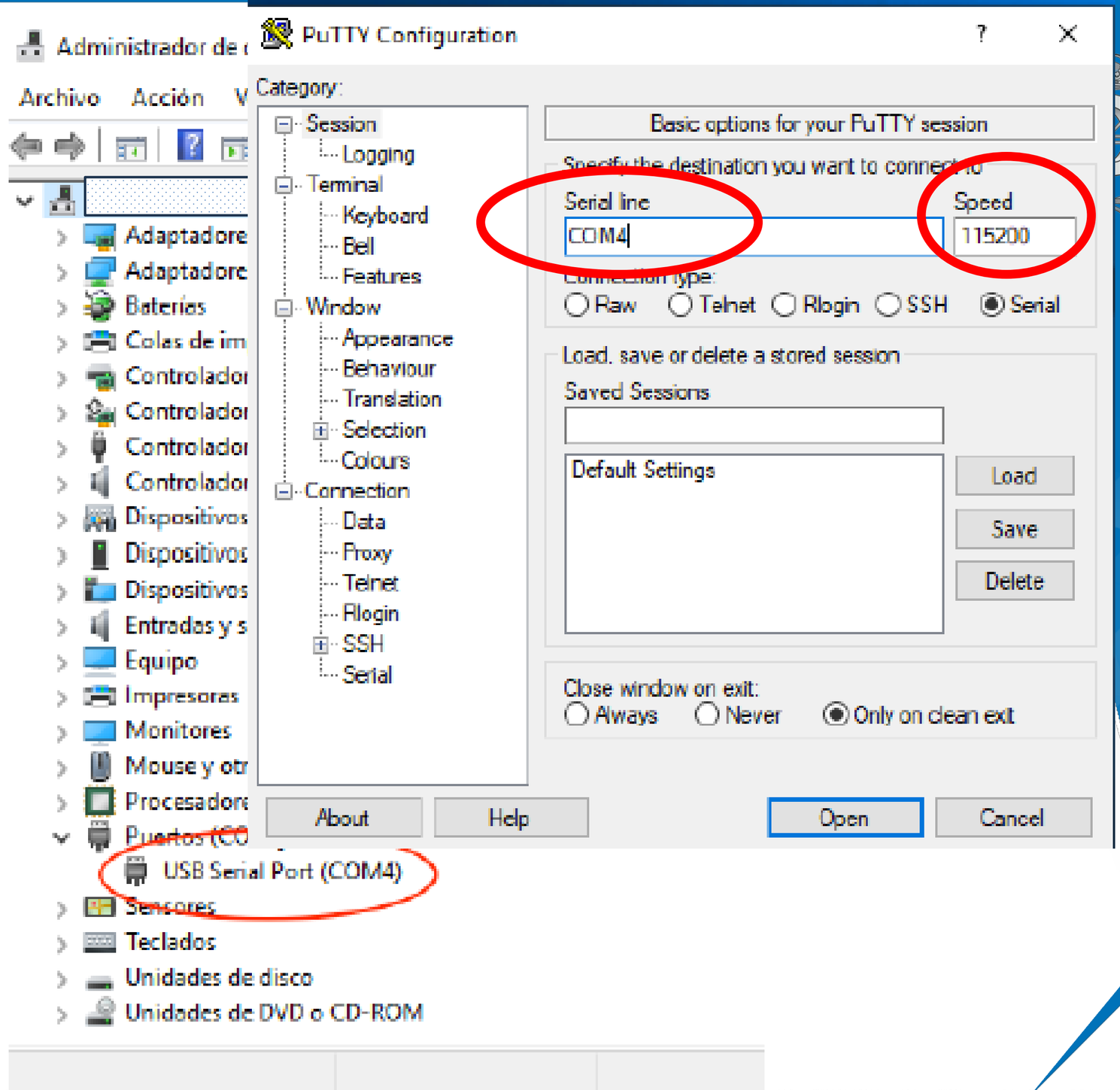
Arranque del terminal

- **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

- **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:

- [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 I^2C Wire. https://www.arduino.cc/reference/en/language/functions/communication/wire/