

Primer parcial - 33.33%

20% prácticas

80% examen

Segundo parcial - 33.33%

20% prácticas

80% examen

Robot sumo (obligatorio)

Tercer parcial - 33.33%

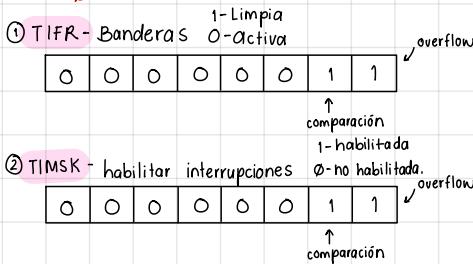
20% prácticas

80% proyecto final (la rúbrica se define antes de salir de vacaciones de abril)

Nota: En caso de que no llevemos un promedio aprobatorio hasta ese momento, se necesita presentar examen final

```
#define PINT PINA
#define PORTT PORTA
#define DDRT DDRA
int main(void)
{ DDRT=0b0000 1111
  PORTT=0b 1111 1111
  //teclado ...copiar ↗
  uint8_t tecla;
  while(1){
```

## Timer 0

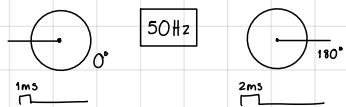
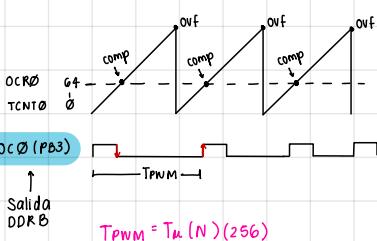
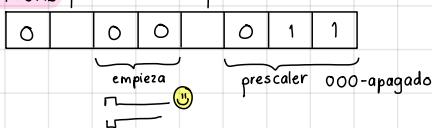


③ TCNT0 - Lleva la cuenta (de los ciclos de reloj)

④ OCR0 - "Tope" → int por comparación } Excel

$$\text{quiero} = T_u \cdot N(OCR0+1)$$

⑤ TCCR0 modo (11=fast PWM)



→ Puedo escribir en lugares que no se ven

-Inicializando I/D=1, S=0 (sin shift cursor a la derecha)

\$00	\$01	\$02	\$03	\$04	\$05	\$06	\$07	\$08	\$09	\$0A	\$0B	\$0C	\$0D	\$0E	\$0F	...	27
\$40	\$41	\$42	\$43	\$44	\$45	\$46	\$47	\$48	\$49	\$4A	\$4B	\$4C	\$4D	\$4E	\$4F	...	67

-Inicializando I/D=0 S=0 (sin shift cursor a la izquierda) también aplica la misma distribución

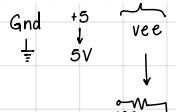
\$00	\$01	\$02	\$03	\$04	\$05	\$06	\$07	\$08	\$09	\$0A	\$0B	\$0C	\$0D	\$0E	\$0F	...	27
\$40	\$41	\$42	\$43	\$44	\$45	\$46	\$47	\$48	\$49	\$4A	\$4B	\$4C	\$4D	\$4E	\$4F	...	67

15/02/2021

LCD → Las letras están en puntos

Para que las letras (regular contraste)  
se vean más negritas  
Potenciómetro

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Gnd	+5V	Vee	RS	R/W	E	D0	D1	D2	D3	D4	D5	D6	D7	+	-



Para que brille más (se pueden quedar desconectados)

## Instructions

Instruction	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution Time (max) (when f <sub>osc</sub> is 27 kHz)
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.	1.52 ms
Return home	0	0	0	0	0	0	0	0	1	0	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	
Entry mode set	0	0	0	0	0	0	1	I/D	S	0	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 µs
Display on/off control	0	0	0	0	0	1	D	C	B	0	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 µs
Cursor or display shift	0	0	0	0	1	S/C	R/L	0	0	0	Moves cursor and shifts display without changing DDRAM contents.	37 µs
Function	0	0	0	1	DL	N	F	0	0	0	Sets interface data length (DL), number of display lines (N), and character font (F).	37 µs
Set CGRAM address	0	0	1	ACG	ACG	ACG	ACG	ACG	ACG	0	Sets CGRAM address. CGRAM data is sent and received after this setting.	37 µs
Set DDRAM address	0	0	1	ADD	ADD	ADD	ADD	ADD	ADD	0	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 µs
Read busy flag & address	0	1	BF	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 µs						

→ Puedo escribir en lugares que no se ven

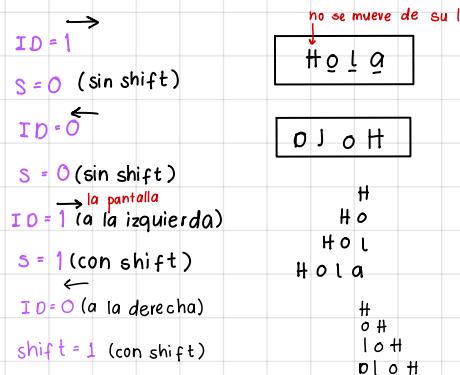
→ 8 bits vs. 4 bits  
+ Más rápido  
+ Más fácil  
- Más espacio  
- Más puertos (1.5 puertos)

- Más lento  
- MÁS difícil  
+ Menos puertos (7 pines)

→ Para mandar de 4, debo de mandar 2 paquetes (1 más significativos (7-4))

## Register Selection

RS	R/W	Operation
0	0	IR write as an internal operation (display clear, etc.) → borrar, cambiar de posición
0	1	Read busy flag (DB7) and address counter (DB0 to DB6) → bandera de ocupado
1	0	DR write as an internal operation (DR to DDRAM or CGRAM) → letras
1	1	DR read as an internal operation (DDRAM or CGRAM to DR) → memoria



**RS = 0**      **RW = 1**  $\Rightarrow$  el LSD va a ser puerto de entrada  
**Instrucciones**      **leer**      aunque solo te interese el 7

(siempre hay que recibir los 8 bits, primero los más significativos y luego los menos significativos)

se checa 1 vez al final  
 $\Rightarrow$  Busy flag DB7

Vida real	1 → ocupado
0 → ya terminó	Proteus
	1 → libre
	0 → ocupado

### MANDAR INSTRUCCIONES

Para mandar cualquiera de estas instrucciones al LCD lo que debemos hacer es:

- 1) "Escribimos" los bits de la instrucción en los pines DB7..DB0
  - 2) Configuramos RS=0
  - 3) Configuramos R/W=0
  - 4) Configuramos E=1  $\rightarrow$  cuando leemos paquetes de 4 bits
  - 5) Esperamos un tiempo activo/desactivado Enable
  - 6) Deshabilitamos E (E=0)
  - 7) Verificamos la bandera "Busy" en DB7 (o bien esperamos un tiempo para garantizar que ya no esté ocupado)
- Cambiando a entrada

### DATOS

**RS  $\rightarrow$  1; R/W  $\rightarrow$  0**

Los bits correspondientes a RS y R/W se configuran de esta forma para indicarle al LCD que lo que se mandará es un dato (Para escribir algo en el LCD)

Cuando conectamos el LCD a un microprocesador, podemos indicarle el dato que se va a enviar como un string, es decir enviándole directamente la letra que deseamos, usualmente los textos que se envían al LCD se

almacenarán en tablas (utilizando el comando .db) sin embargo también existe la posibilidad de enviar los caracteres que se desean desplegar mediante su código binario. A continuación, se muestra la tabla correspondiente.

### BANDERA DE OCUPADO

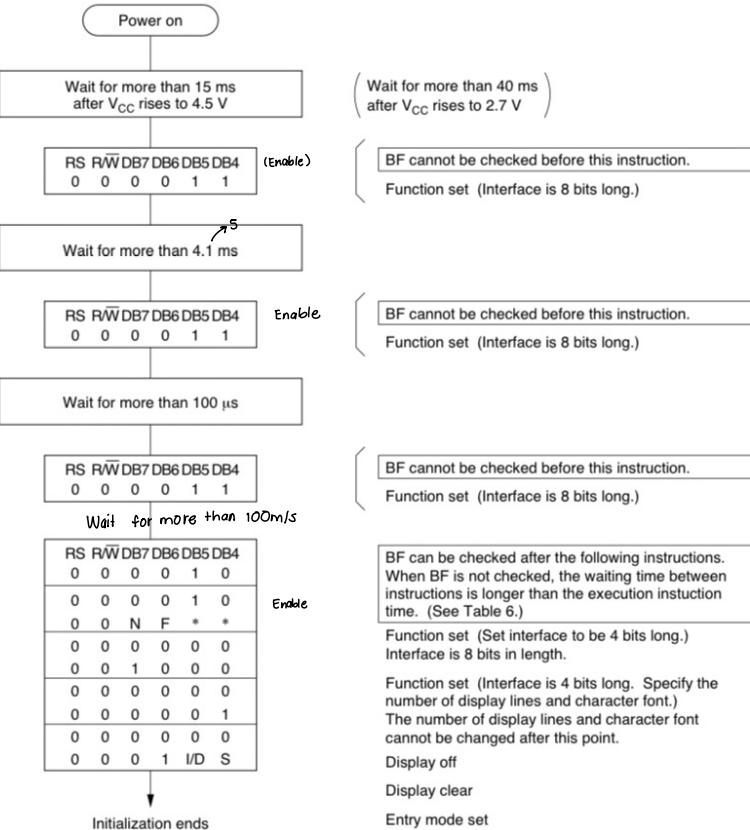
**RS  $\rightarrow$  0; R/W  $\rightarrow$  1**

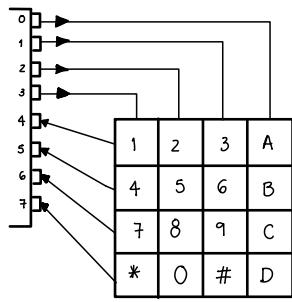
Esta configuración es utilizada cuando deseamos leer la bandera de ocupado (correspondiente al pin DB7 del LCD). La **bandera de ocupado del LCD** (es decir el pin DB7) se pone en alto cuando una instrucción está siendo ejecutada y regresa a 0 cuando la instrucción se termina de ejecutar. Ninguna otra instrucción debe enviarse al LCD hasta que esta bandera cambia de estado.

### MANDAR DATOS

Para mandar cualquier dato al LCD lo que debemos hacer es:

- 1) "Escribimos" los bits de la instrucción en los pines DB7..DB0 o bien enviamos al puerto correspondiente la letra deseada
- 2) Configuramos RS=1
- 3) Configuramos R/W=0
- 4) Configuramos E=1
- 5) Esperamos un tiempo
- 6) Deshabilitamos E (E=0)
- 7) Verificamos la bandera "Busy" en DB7 (o bien esperamos un tiempo para garantizar que ya no esté ocupado)







## Práctica 01.

dato  $\leftarrow$  16 bits  $\leftarrow$  (0-1023)

## ① Sin interrupciones

```
ADMUX=0b01000111;
SFIOR=0b00000000; //puedo omitir
```

```
ADCSRA=0b10010101; //para 4MHz con factor 32
```

```
DDRA=0b00000000; //puedo omitir
```

```
PORATA=0b00000000; //puedo omitir
```

```
while(1){
```

↓

```
    ADCSRA = ADCSRA | (1<<ADSC); //inicia conversión
```

```
    while(until_en_bit(&ADCSRA, ADSC)){ } //trabaja, mientras tenga 1 → no ha acabado  
    (el 0 se pone solo)
```

```
    uint16_t res=ADC;
```

```
    ==  
}
```

```
while(
```

```
    dato = ADC
```

1023 - 5V  
dato - X

importante  
hacer cast  $\Rightarrow$  float x = dato(500)  
para que guarde  
el decimal

1023  
uint8\_t c=x/100  
wr\_char(c+48)

para que no imprima el  
ASCII

## ② Con interrupciones

```
ADMUX=0b01000111;
```

```
SFIOR=0b00000000; //puedo omitir
```

```
ADCSRA=0b1001101; //para 4MHz con factor 32
```

```
DDRA=0b00000000; //puedo omitir
```

```
PORATA=0b00000000; //puedo omitir
```

```
sei();
```

```
ADCSRA |= (1<<ADSC); //inicia
```

```
while(1) { }
```

```
ISR(ADC_vect){
```

```
    uint16_t res=ADC;
```

```
    ==
```

```
    ADCSRA |= (1<<ADSC);
```

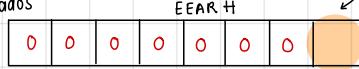
```
}
```

\* especiales porque si se apaga el sistema, se quedan guardados

**EEPROM** : 512 bytes de memoria almacenados

**EEAR** (dirección → 0 → 511)

¿dónde?



Se le pone 1 si se quiere leer o escribir

**EEAR L**

dirección en donde queremos escribir el dato

uint16\_t

**EEDR** (dato a escribir o aquí queda el dato leído)

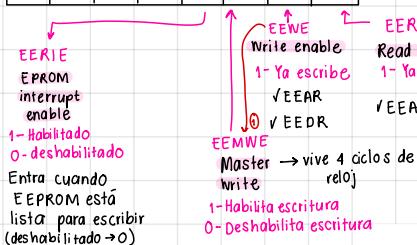


tal cual el dato que quiero guardar

**EECR** (control de la memoria)



inicializa con 0 para asegurarte que no está escribiendo, cuando termina de escribir, se regresa a 0 automáticamente



### PROCESO DE LECTURA EN LA MEMORIA EEPROM

El siguiente diagrama de flujo muestra los pasos necesarios para llevar a cabo una operación de lectura en la memoria EEPROM del microcontrolador.



Vale la pena hacer notar que una vez que el dato se leído se encuentra en el registro EEDR, este deberá almacenarse en un registro de uso temporal a través de la instrucción `IN R__, EEDR` para posteriormente darle el uso que se requiera.

### Función de lectura

```
uint8_t EEPROM_read(uint16_t dir)
{
    while(uno_en_bit(&EECR, Eewe)){}

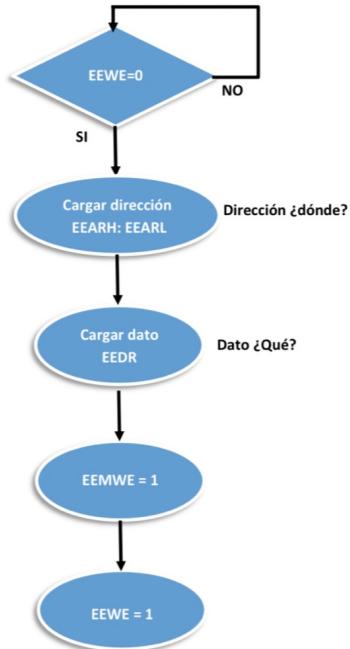
    EEAR = dir;
    EECR |= (1<< EERE);
    return EEDR;
}
```

**EEPROM\_write(dirección, dato)**  
→ cuando quiero guardar un dato

**EEPROM\_read (dirección)**  
→ cuando quiero saber qué es lo que está guardado en ese dato

### PROCESO DE ESCRITURA EN LA MEMORIA EEPROM

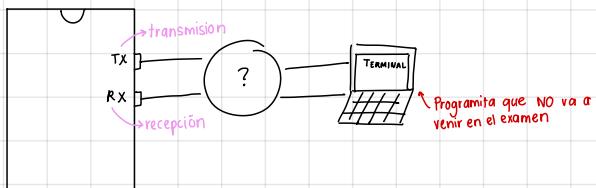
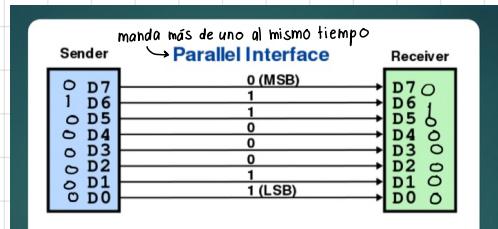
El siguiente diagrama de flujo muestra los pasos necesarios para llevar a cabo una operación de escritura en la EEPROM del microcontrolador



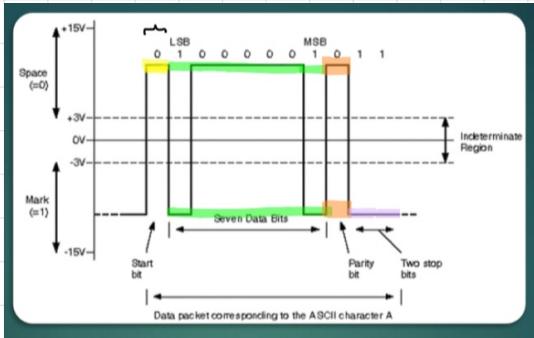
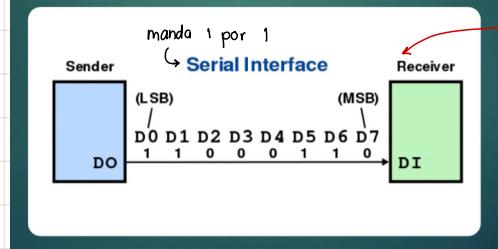
### Función de escritura

```
void EEPROM_write(uint16_t dir, uint8_t dato)
{
    while(uno_en_bit(&EECR, Eewe)){}

    EEAR = dir;
    EEDR = dato;
    cli();
    EECR |= (1<< EEMWE); // Pone 1 en EEMWE
    EECR |= (1<< Eewe); // Pone 1 en Eewe
    sei();
}
```



→ Lo más común es mandar paq. de 8 bits aunque lo puedo configurar de 5, 6, 7, 8 o 9 bits



### USART (Universal Synchronous Asynchronous serial receiver and transmitter)

ATMega 16 lo puede trabajar en distintos modos

- Normal asincrono } asincrona
- Asincrono de doble velocidad }
- Sincrono maestro } sincrono
- Sincrono esclavo

UCSRC en el bit UMSel

0 = asincrono

1 = sincrono

→ en el caso del UDR, como es una pila, en cuanto consulte un valor, lo saca

→ Casi siempre que recibo datos, usamos las interrupciones

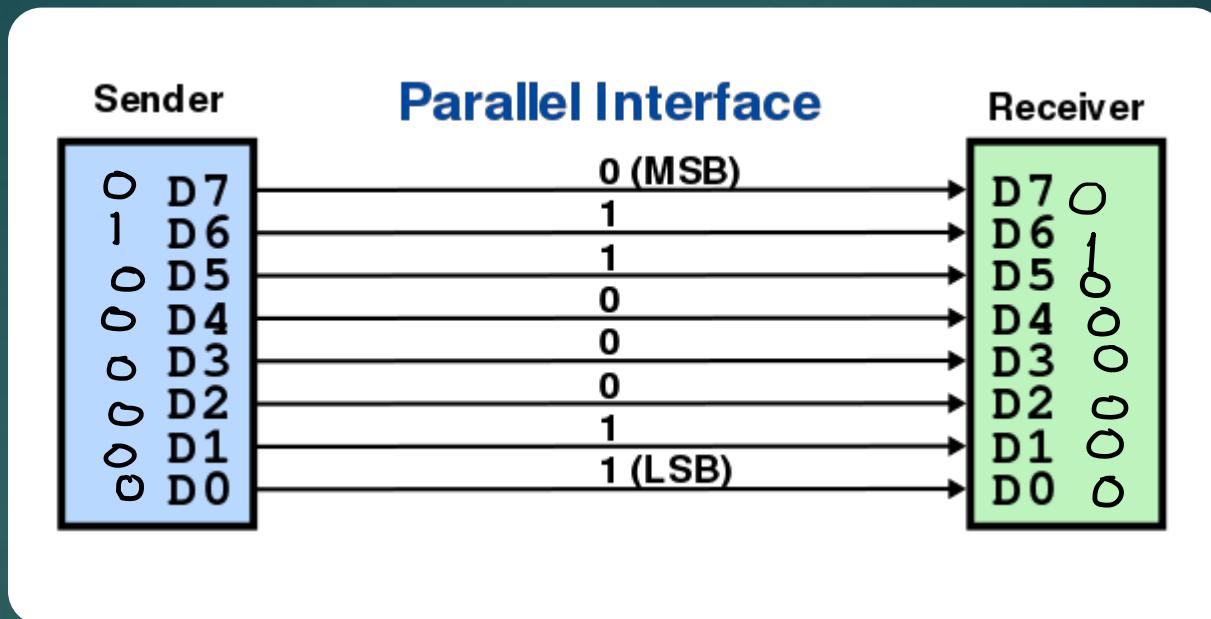
Revisar libro para ver registros

Libro página 186

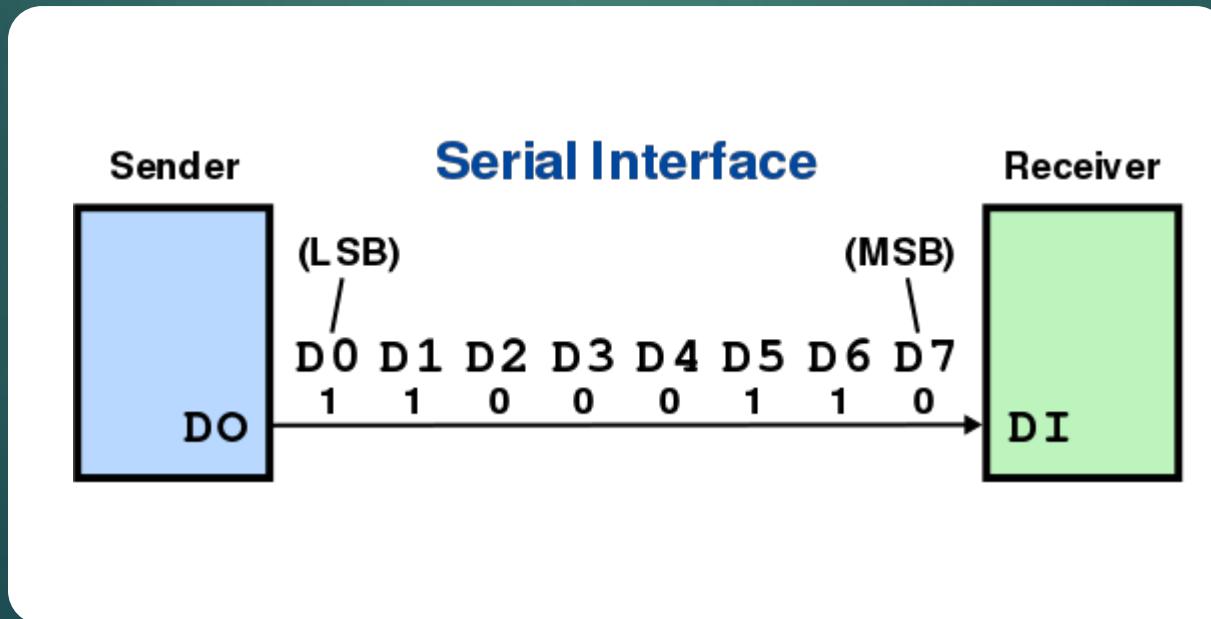
# Fundamentos de Comunicación serial

*Teresa Orvañanos Guerrero*

→mando más de uno al mismo tiempo



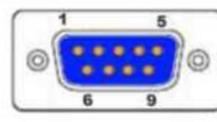
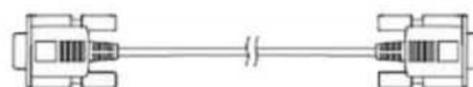
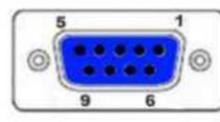
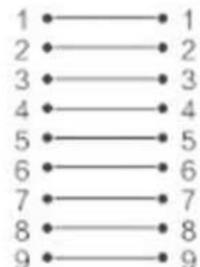
→mando de uno por uno



cables que se usaban antes para el puerto serial (NO es lo mismo que el VGA)

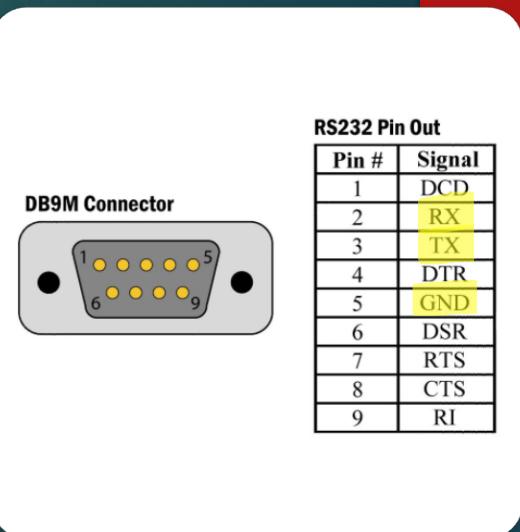
## Serial RS232 DB9 Male to Female Cable

The length: 1.5m/4.92ft ,3m/9.84ft ,5m/16.4ft (optional)



DB9 Female

DB9 Male



RS232 Pin Out

Pin #	Signal
1	DCD
2	RX
3	TX
4	DTR
5	GND
6	DSR
7	RTS
8	CTS
9	RI

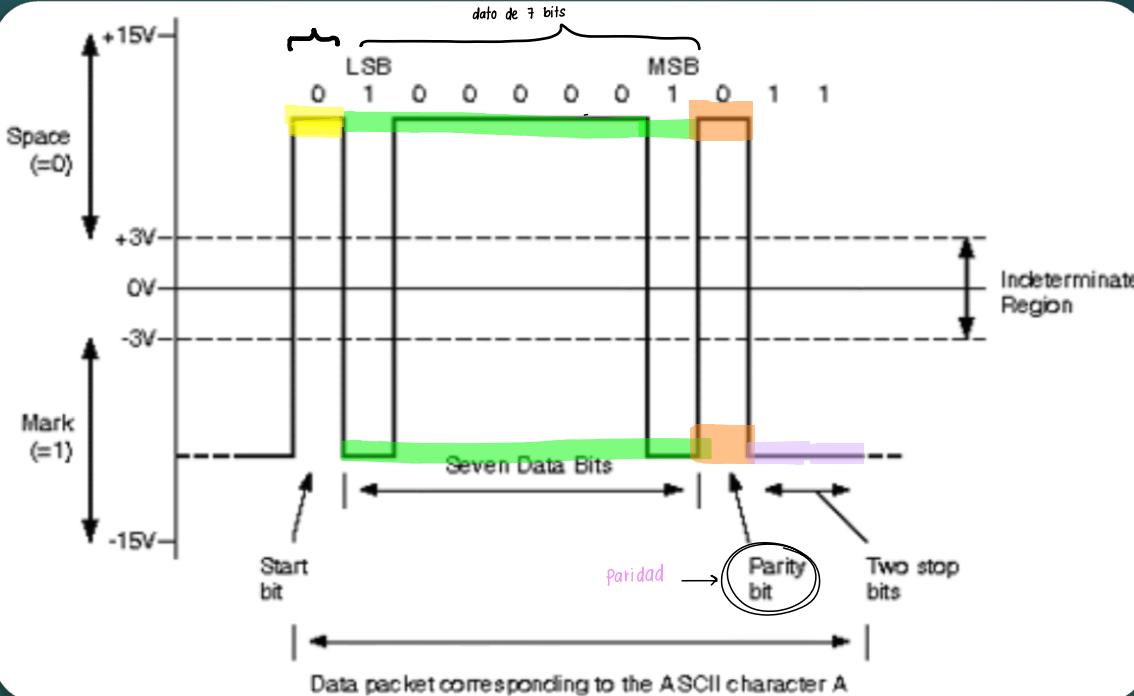
Cada PIN servía para configurar algo en serial (ahora ya solo importan los 3 amarillos)



cable actual para terminal -serial

→ USB: Universal Serial Port

→ La computadora para la comunicación serial ocupa de -15V a 15V



! Configurar igual de los 2 lados

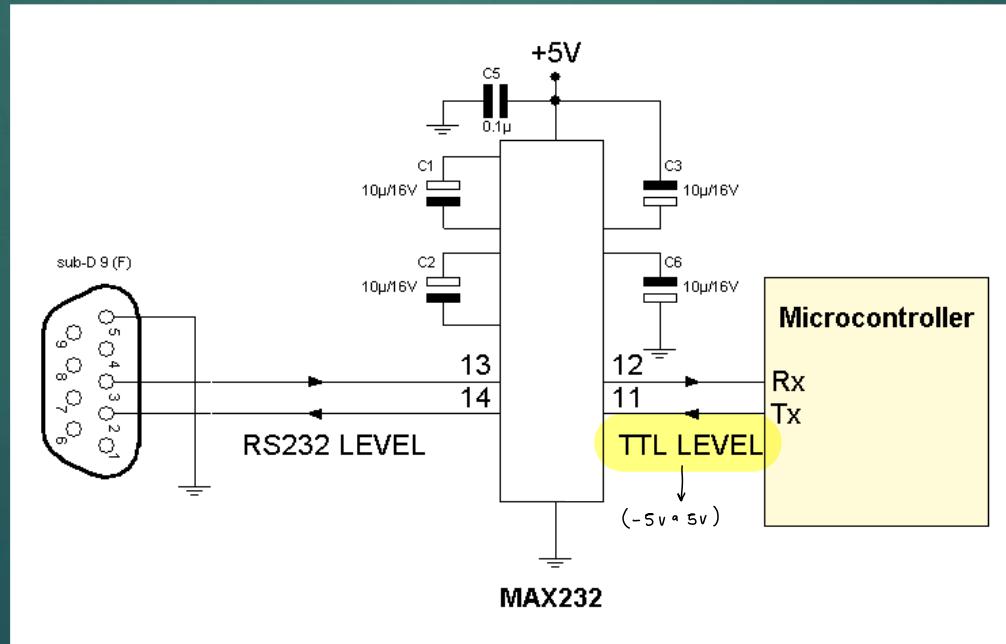
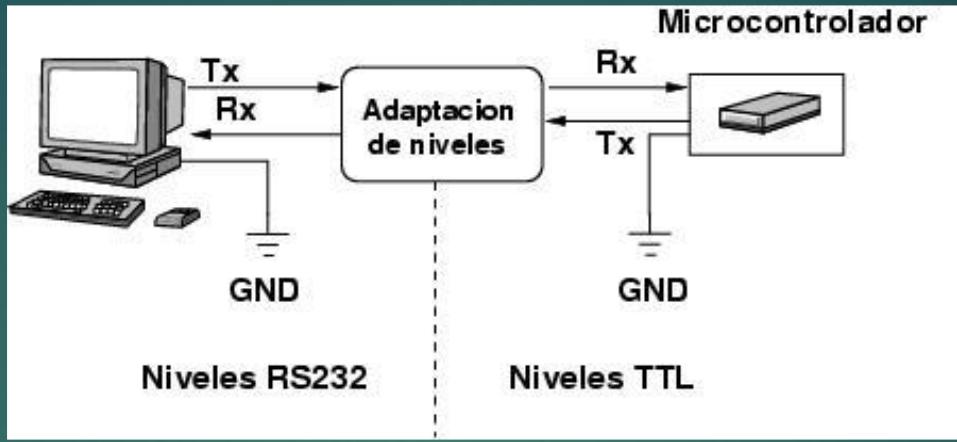
## Frame Formats

→ paquetes de datos para saber en donde empieza y en donde termina

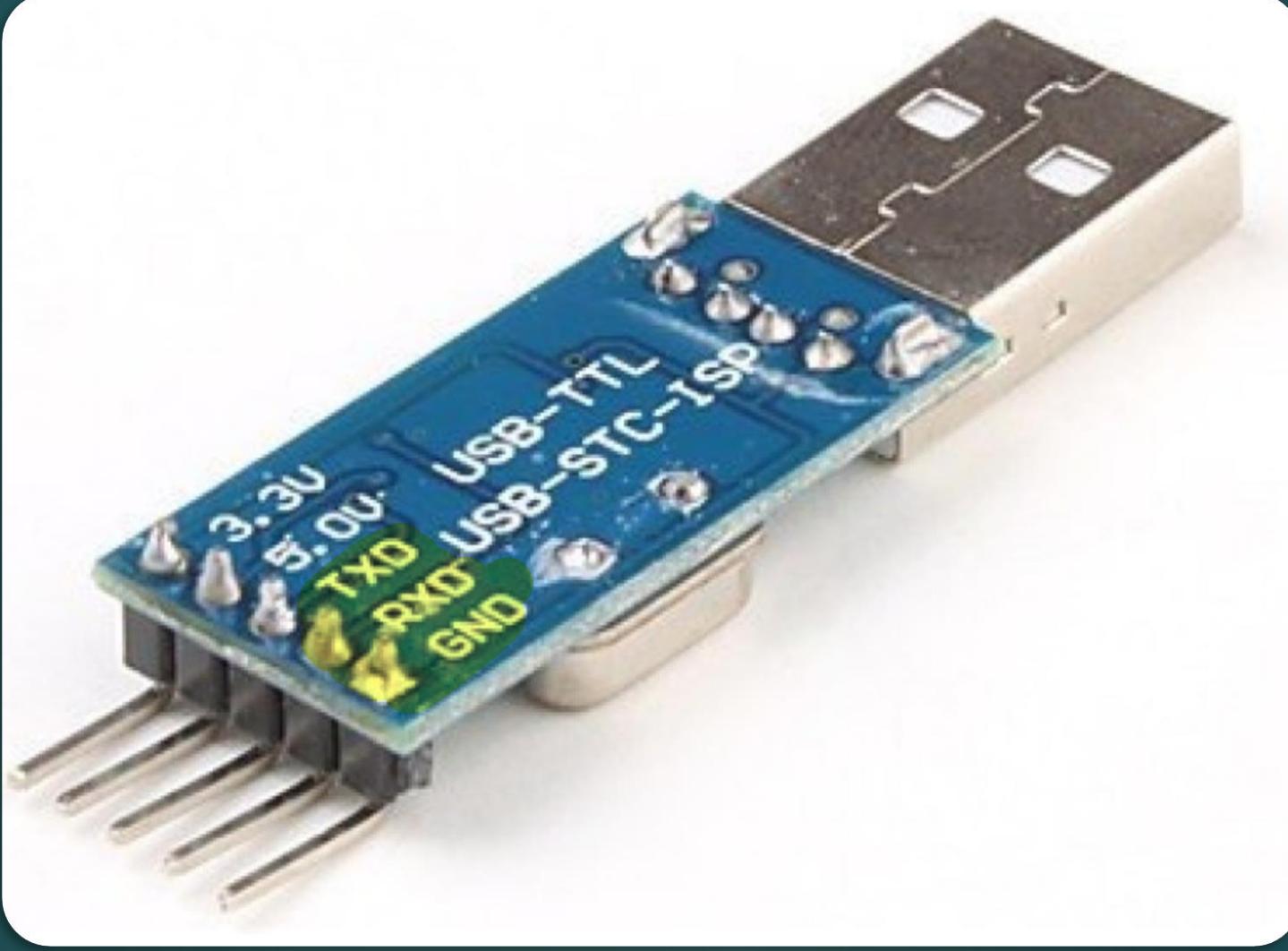
A serial frame is defined to be one character of data bits with synchronization bits (start and stop bits), and optionally a parity bit for error checking. The USART accepts all 30 combinations of the following as valid frame formats:

- 1 start bit
- 5, 6, 7, 8, or 9 data bits
- no, even or odd parity bit
- 1 or 2 stop bits

Paridad cuenta la cantidad de 1  
(para pegarlo al final del paq. y  
ver si no perdió info)



→ es un chip que me sirve para pasar del rango de -15 a 15v que necesita la computadora a -5 a 5v



→ adaptador que usaremos

lo que se va a usar del lado del microcontrolador

Datasheet página 140 / Libro página 187

# USART (Universal Synchronous and Asynchronous serial receiver and transmitter)

Teresa Orvañanos Guerrero

En ATmega16A el USART puede trabajar en diferentes modos

- - Normal asíncrono el que vamos a usar
  - Asíncrono de doble velocidad
  - Síncrono maestro
  - Síncrono esclavo
- } asíncrona
- } síncrono
- Parecido a las distintas maneras de configurar el TIMER0

  - Asíncrono: sin clock
  - Síncrono: con clock

En UCSRC en el bit UMSEL

→ 0 = asíncrona  
1 = síncrona

UCSRB  
\*

UDR — dato

cargarlo (cuando transmito)  
equivale a "ya lo mandé"  
leerlo (cuando recibo)  
una vez que lo leo, lo saco y ya no está ahí

UDR

cola de 2 niveles  
FIFO

dónde guardo el dato  
que quiero transmitir

Bit	8	7	6	5	4	3	2	1	0	UDR (Read)	UDR (Write)
Read/Write	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0	0	0	0

Buffer de transmisión y de recepción

sbis / sbis X if(cero\_en\_bit(&UDR,2)) { } // 1a vez sacu UDR \*  
2a vez " UDR \*

→ aunque solo saque un bit del  
UDR con cero\_en\_bit, ya una  
vez que lo lee, lo borra

sbi / cbf

? UDR |= (1<<0); // envío

UDR |= (1<<3); // envío

} aunque solo cambie un bit, se  
manda y se modifica todo

!!!

leer 1º en una variable  
uint8\_t dato = UDR;

/ cargar UDR completo  
UDR = dato;  
0b11110000;

con eso ya mandé  
el dato

UDR = (1<<1) | (1<<3)



Flag Flag Flag

Bit	7	6	5	4	3	2	1	0	UCSRA
Read/Write	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	
Initial Value	0	0	1	0	0	0	0	0	

UCSRA

"timbre"  
ya llegó  
algo

→ RXC (recepción)

- 1 - Hay datos sin leer esperando
- 0 - El buffer vacío (lectura)

Transmisión TXC

- Terminó ☺
- 1 - Ya se transmit. todos los datos
- 0 - "Limpio" en automático cuando entra a la int. correspond.

(1, lista para transmitir)

- 0 - No listo para transmisión
- 1 - Buffer vacío listo para transm.

FE

- data overrun → trató de entrar más de 2 veces
- 0 - Ponerlo! \*
- 1 - Hay un dato OVR RUN (recepción) + válido antes de leer UDR

(trato de llegar más de lo que corre)

PE

- 0 - Ponerlo \*
- 1 - Hay error de paridad + válido antes de leer al UDR

Error de paridad

Simple o doble vel. i

- si síncrona = 0
- si asíncrona
- 0 - normal
- 1 - doble vel.

MPCM

- 0 - normal
- 1 - ignora datos recibidos sin info de la dir

para usar interrupciones

\* Errores en Recepción

configuración personales



**UCSRB**

Bit	7	6	5	4	3	2	1	0	UCSRB
ReadWrite	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	

**RXCIE**

recepción  
1 - activa la int. cuando reciba un dato  
0 - No int

!!!

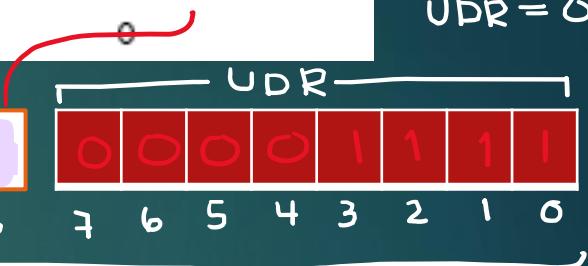
con este decido si quiero que mi programa lea, transmite o haga las 2 cosas (+)  
Casi siempre cuando uso recepción, necesito interrupción

**TXCIE**

transmisión  
1 - activa la int. cuando terminó la trans.  
0 - No int

**UDRIE**

activa la int.  
1 - Int. cuando listo para trans.  
0 - No. int



! 9 bits en total !

Leer o escribir ANTES de leer o escribir al UDR

Table 67. UCSZ Bits Settings

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

**RXEN**

0 - Deshabilita Recepción  
1 - Habilitar Recepción

**TX EN**

0 - Deshabilita Transmision  
1 - Habilitar transm...

Transmitir 9 bits



① primero mando el de transmisión (el 9no)  
UCSRB |= (1<<0); ←

② Luego mando los 8 bits  
UDR = 0b 0101 0101 ; ←

→ Leer  
Recibir 9 bits

// 1º leer bit 1 del UCSR<sub>B</sub>

① uint16\_t dato = 0;  
if (uno\_en\_bit(8 UCSR<sub>B</sub>, 1)) { dato = (1<<8); }

// leer UDR

② uint8\_t temp = UDR; ~~UDR~~  
dato |= temp;

Recibí 0b 10101 0101  
bit 1 de UCSR<sub>B</sub>  
UDR  
dato = 0b 10000 0000 0000

1 0000 0000  
0101 0101  
-----  
10101 0101 !!

No paríada  
1 bit stop  
8 bits



UCSRC

Bit	7	6	5	4	3	2	1	0	UCSRC
Read/Write	R/W								
Initial Value	1	0	0	0	0	1	1	0	

Table 64. UMSEL Bit Settings

UMSEL	Mode
0	Asynchronous Operation <i>asíntrona vel. normal</i>
1	Synchronous Operation

Table 66. USBS Bit Settings

USBS	Stop Bit(s)
0	1-bit
1	2-bit

Table 65. UPM Bits Settings

UPM1	UPM0	Parity Mode	
0	0	Disabled	No!
0	1	Reserved	
1	0	Enabled, Even Parity	Par
1	1	Enabled, Odd Parity	Impar

Paridad

para revisar la cant.  
de 1 en un byte  
(revisa que no se pierdan valores)

igual q' en el otro  
dispositivo (comp)

Igual q' en el otro  
dispositivo (comp)

\*

Table 67. UCSZ Bits Settings

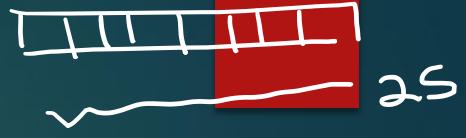
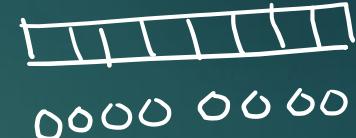
UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

UBRR

16 bits → UBRRH &amp; UBRL

# para velocidad de transmisión

UBRRH : UBRL



siempre que  
trabajo con 1MHz  
uso vel. normal  
(no doble) en  
el U2X

Solo tengo 1MHz, 2MHz, 4MHz u  
8MHz cuando uso el reloj interno,  
pero podría usar 1.84MHz en los  
fusibles si quiero usar un  
cristal (y no tener error en la  
transmisión)

Table 69. Examples of UBRR Settings for Commonly Used Oscillator Frequencies

Baud Rate (bps)	f <sub>osc</sub> = 1.0000 MHz				f <sub>osc</sub> = 1.8432 MHz				f <sub>osc</sub> = 2.0000 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4k	3	8.5% <small>igual de mal que -7%</small>	8	-3.5%	7	0.0%	15	0.0%	8	-3.5%	16	2.1%
19.2k	2	8.5%	6	-7.0%	5	0.0%	11	0.0%	6	-7.0%	12	0.2%
28.8k	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	-3.5%
38.4k	1	-18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	-7.0%
57.6k	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%
76.8k	-	-	1	-18.6%	1	-25.0%	2	0.0%	1	-18.6%	2	8.5%
115.2k	-	-	0	8.5%	0	0.0%	1	0.0%	0	8.5%	1	8.5%
230.4k	-	-	-	-	-	-	0	0.0%	-	-	-	-
250k	-	-	-	-	-	-	-	-	-	-	0	0.0%
Max <sup>(1)</sup>	62.5 kbps		125 kbps		115.2 kbps		230.4 kbps		125 kbps		250 kbps	

\* formula para establecer bps  
↑ bits x seg.

UBRR = 103;

Table 70. Examples of UBRR Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	$f_{osc} = 3.6864 \text{ MHz}$				$f_{osc} = 4.0000 \text{ MHz}$				$f_{osc} = 7.3728 \text{ MHz}$			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	95	0.0%	191	0.0%	103	0.2%	207	0.2%	191	0.0%	383	0.0%
4800	47	0.0%	95	0.0%	51	0.2%	103	0.2%	95	0.0%	191	0.0%
9600	23	0.0%	47	0.0%	25	0.2%	51	0.2%	47	0.0%	95	0.0%
14.4k	15	0.0%	31	0.0%	16	2.1%	34	-0.8%	31	0.0%	63	0.0%
19.2k	11	0.0%	23	0.0%	12	0.2%	25	0.2%	23	0.0%	47	0.0%
28.8k	7	0.0%	15	0.0%	8	-3.5%	16	2.1%	15	0.0%	31	0.0%
38.4k	5	0.0%	11	0.0%	6	-7.0%	12	0.2%	11	0.0%	23	0.0%
57.6k	3	0.0%	7	0.0%	3	8.5%	8	-3.5%	7	0.0%	15	0.0%
76.8k	2	0.0%	5	0.0%	2	8.5%	6	-7.0%	5	0.0%	11	0.0%
115.2k	1	0.0%	3	0.0%	1	8.5%	3	8.5%	3	0.0%	7	0.0%
230.4k	0	0.0%	1	0.0%	0	8.5%	1	8.5%	1	0.0%	3	0.0%
250k	0	-7.8%	1	-7.8%	0	0.0%	1	0.0%	1	-7.8%	3	-7.8%
0.5M	-	-	0	-7.8%	-	-	0	0.0%	0	-7.8%	1	-7.8%
1M	-	-	-	-	-	-	-	-	-	-	0	-7.8%
Max <sup>(1)</sup>	230.4 kbps		460.8 kbps		250k bps		0.5 Mbps		460.8 kbps		921.6 kbps	

## Resumen

UDR — dato (8 bits) → para leer o transmitir

UCSRA — banderas e indicadores

-----

UCSRB — Interrupciones? / Transm y/o Recep / # bits\* / Svo bit.

UCSRC — Asíncrona / Paridad / # bits\* / bit parada

UBRR (UBRRH:UBRRL) — Velocidad (tabla & formula)

# INICIALIZAR / CONFIGURAR

## C Code Example<sup>(1)</sup>

FCPU

```
#define FOSC 1843200 // Clock Speed ← lo quité porque en el datasheet usaron un cristal
#define BAUD 9600 ← cuantos bps
#define MYUBRR FOSC/16/BAUD-1 ← en caso de que ya no quiera usar la tabla
void main( void ) FCPU
{
    .. función para inicializar el USART
    USART_Init ( MYUBRR );
    ..
}

void USART_Init( uint16_t ubrr )
{
    /* Set baud rate */ uint8_t
    UBRRH = ( unsigned char ) ( ubrr >> 8 ); para que quede en la parte alta
    UBRRL = ( unsigned char ) ubrr; se queda con la parte menos significativa ← dependiendo de lo que necesites
    /* Enable receiver and transmitter */
    UCSRB = ( 1<<RXEN) | ( 1<<TXEN) | ( 1<<RXCIE ); (Recap, trans, Int)
    /* Set frame format: 8data, 2stop bit */
    UCSRC = ( 1<<URSEL) | ( 1<<USBS ) | ( 3<<UCSZ0 ); dependiendo de lo que necesites // 8 bits
    Siempre Paridad en 1 → 0
    g 1 en UCSZ0
    g 1 en UCSZ1
}
2 stop sin paridad
```



```

void main(void)
{
    USAR_Transmit(1);
}

```

Para transmitir

valor que quiero transmitir

$\emptyset = \text{FALSO}$

NO  $\emptyset = \text{VERDADERO} (1)$

#### C Code Example<sup>(1)</sup>

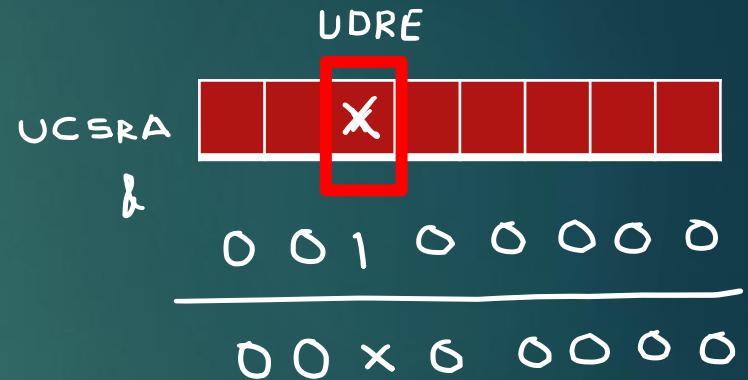
```

void USART_Transmit( unsigned char data )
{
    /* Wait for empty transmit buffer */
    while ( !( UCSRA & (1<<UDRE) ) ) {}

    /* Put data into buffer, sends the data */
    UDR = data;
}

```

O es falso + ! = True  
La condición va a ser verdadera mientras no esté listo para transmitir y se va a quedar trabado



Si UDRE=0 : \* = 0b00000000  
! ( falso )  
No USART  
VERDAD

Si UDRE=1 : \* = 0b00100000  
! ( verdadero )  
LISTO  
FALSO

```
void main(void) {
```

≡

```
    uint8_t dato = USART_Receive();
```

≡

}

Para recibir

se usa cuando no uso INT y uso traba

∅ falso

NO ∅ verdad

NO USAR SI NECESITO UN TECLADO

C Code Example<sup>(1)</sup>

```
unsigned char USART_Receive( void )
{
    uint8_t
        /* Wait for data to be received */
        while ( !(_UCSRA & (1<<RXC)) ) {} ; // TRABA !!! cuando NO se
                                                // hasta que le llegue un dato
                                                // (como la del
                                                // teclado que se
                                                // trabaja hasta presionar
                                                // una tecla)
        /* Get and return received data from buffer */
        return UDR;
}
```



TRABA

VS

Interrup Recip

RXC

UCSRA

8

1 0 0 0 0 0 0 0

X 0 0 0 0 0 0 0

si RXC = 0 ∴ \* = 0b00000000

!(falso)

VERDAD

no hay

datos

si RXC = 1 ∴ \* = 0b10000000

!(verdad)

FALSO

si hay

datos

# INTERRUPCIONES

\* Inicializar serial con RXCIE=1

```
volatile uint8_t dato;
```

```
int main(void)
{
    *
    /* Replace with your application code */
    while (1)
    {
    }
}
```

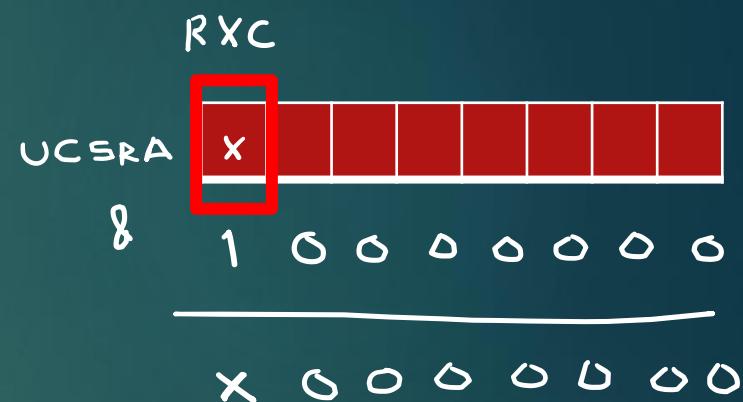
en lugar de la diapositiva anterior

```
ISR(USART_RXC_vect)
{
    dato = UDR
}
```

NO SE USA

C Code Example<sup>(1)</sup>

```
void USART_Flush( void )
{
    uint8_t
    unsigned char dummy;
    while ( UCSRA & (1<<RXC) ) dummy = UDR;
}
```



si RXC = 0 ∴ \* = 0b00000000  
no hay datos

si RXC = 1 ∴ \* = 0b10000000  
si hay datos

## Octava 4

Límite para sacar el OCRO y prescaler en el EXCEL

Do octava 4  $\Rightarrow F = 201,626$

$$T = \frac{1}{F} = 0.000000382$$

$$t = \frac{T}{2} = 0.000000191$$

Límite 2 para sacar OCRO y prescaler en excel

Si octava 4  $\Rightarrow F = 493,883$

$$T = \frac{1}{F} = 0.000000202$$

$$t = \frac{T}{2} = 0.000000101$$

Tiempo que se desea obtener (seg.)

0.00000191

Frecuencia	Periodo	Prescaler	OCRO con dec	OCRO	t(real)
8Mhz	0.000000125	1	14.28	14	0.000001875
		8	0.91	1	0.000002
		64	-0.76125	-1	0
		256	-0.9403125	-1	0
		1024	-0.985078125	-1	0

4Mhz	0.00000025	1	6.64	7	0.000002
		8	-0.045	0	0.000002
		64	-0.880625	-1	0
		256	-0.97015625	-1	0
		1024	-0.992539063	-1	0

2Mhz	0.0000005	1	2.82	3	0.000002
		8	-0.5225	-1	0
		64	-0.9403125	-1	0
		256	-0.985078125	-1	0
		1024	-0.996269531	-1	0

1Mhz	0.000001	1	0.91	1	0.000002
		8	-0.76125	-1	0
		64	-0.97015625	-1	0
		256	-0.992539063	-1	0
		1024	-0.998134766	-1	0

Tiempo que se desea obtener (seg.)

0.00000101

8Mhz	0.000000125	1	7.08	7	0.000001
		8	0.01	0	0.000001
		64	-0.87375	-1	0
		256	-0.9684375	-1	0
		1024	-0.992109375	-1	0

4Mhz	0.00000025	1	3.04	3	0.000001
		8	-0.495	0	0.000002
		64	-0.936875	-1	0
		256	-0.98421875	-1	0
		1024	-0.996054688	-1	0

2Mhz	0.0000005	1	1.02	1	0.000001
		8	-0.7475	-1	0
		64	-0.9684375	-1	0
		256	-0.992109375	-1	0
		1024	-0.998027344	-1	0

1Mhz	0.000001	1	0.01	0	0.000001
		8	-0.87375	-1	0
		64	-0.98421875	-1	0
		256	-0.996054688	-1	0
		1024	-0.999013672	-1	0

Sacamos t de las demás notas

Re octava  $\Rightarrow F = 293,665$

$$T = \frac{1}{F} = 0.0000034052$$

$$t = \frac{T}{2} = 0.0000017026$$

Mi octava 4  $\Rightarrow F = 329,628$

$$T = \frac{1}{F} = 0.0000030337$$

$$t = \frac{T}{2} = 0.0000015169$$

Fa octava 4  $\Rightarrow F = 349,228$

$$T = \frac{1}{F} = 0.0000028635$$

$$t = \frac{T}{2} = 0.0000014318$$

Sol octava 4  $\Rightarrow F = 391,995$

$$T = \frac{1}{F} = 0.0000025511$$

$$t = \frac{T}{2} = 0.0000012756$$

La Octava 4  $\Rightarrow F = 440,000$

$$T = \frac{1}{F} = 0.0000022727$$

$$t = \frac{T}{2} = 0.0000011364$$

