

Comunicación serie

Informática II - R2004
2021

Tipos de comunicación entre 2 o más dispositivos

Cuando conectamos varios dispositivos “inteligentes” (que son capaces de recibir varios comandos o instrucciones, y actuar de forma distinta en función de lo que se reciba), aparecen distintas maneras de comunicarnos.


La primera, que vimos cuando trabajamos con displays 7 segmentos, es la comunicación **PARALELA**. En ella, se envía una gran cantidad de datos **al mismo tiempo** (en el caso del display, se comunican 8 líneas de segmentos y la selección del dígito todo en líneas separadas). Esto hace que la comunicación sea más veloz, pero requiere de un bus muy grande, y por lo tanto las posibilidades de errores en la comunicación son muchas cuando las distancias son largas (todas las líneas pueden empezar a interferir entre sí, y un defasaje de tiempos entre la escritura de las líneas puede interferir). Se usa este tipo de comunicación para dispositivos que están MUY cerca físicamente, y en donde se prioriza la velocidad (memorias RAM, periféricos del microcontrolador, placas de video, etc.)

Tipos de comunicación entre 2 o más dispositivos

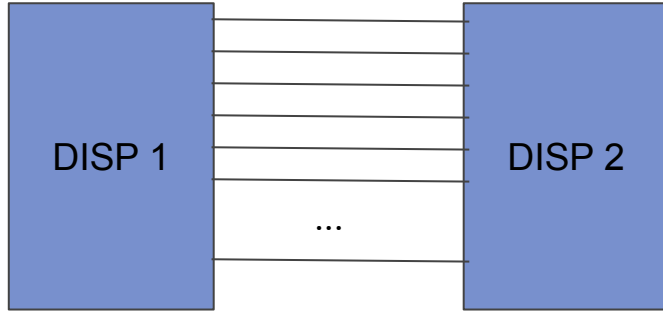
El segundo tipo de comunicación posible se basa en utilizar pocas líneas para mandar un dato, e ir mandando los bits de ese dato de a uno por vez. Este tipo de comunicación se la denomina comunicación **SERIE**, y dada la facilidad con que puede implementarse, ha ido creciendo cada vez más en los últimos 30 años, generando estándares como el TCP/IP, USB, RS485, RS232, CAN, I2C, I2S, SPI, Bluetooth, entre muchos otros.

Todos estos estándares tienen en común que los datos se van mandando de a un bit por vez, por lo que se necesita un mínimo de cables para comunicar 2 dispositivos (llegando a compartir únicamente 2 cables, como en el caso del RS485 o el 1-wire, por ejemplo), y esto lo hace una comunicación muy fácil de implementar y muy robusta.

Sin embargo, se necesita de algún tipo de estándar o acuerdo entre las partes para poder codificar y decodificar un mensaje correctamente.



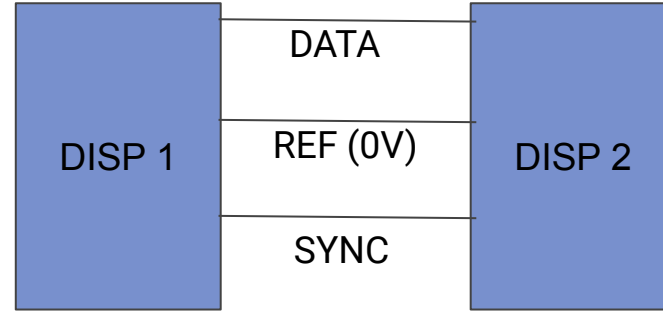
Comunicación paralelo vs. serie



- Comunicación más veloz
(n bits al mismo tiempo)
- Distancias cortas
(por posibles interferencias entre líneas)

Ejemplos:

Impresoras viejas
Memorias (RAM)
Placas PCI
Discos rígidos IDE



- Pocas líneas de datos
- Hacen falta algunas líneas para “ponerse de acuerdo” entre los dispositivos (en el ejemplo hay una línea de sincronía)

Ejemplos:

Impresoras USB
Memorias SD
Placas USB
Discos rígidos SATA

Comunicación serie - Estándares

De acuerdo a estas definiciones podremos establecer una comunicación de múltiples maneras: utilizando solamente un cable para enviar la información (y uno de referencia de tensiones), o tener dos líneas (una para envío y otra para recepción), o establecer una línea de “sincronismo” (que mande pulsos a una determinada velocidad para que se pongan de acuerdo los 2 dispositivos), etc.

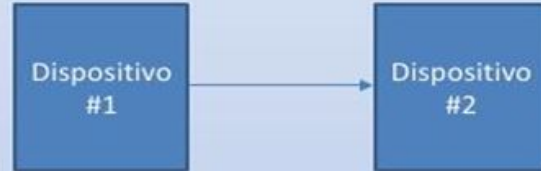
Un estándar define todos los aspectos relacionados con la comunicación: Cuántas líneas se usan, que niveles de tensión, que orden y longitud van a tener los datos, si hay algún chequeo de errores, etc. A estos aspectos se los conoce como “capa física” (cables, tensiones, etc.) y “capa lógica” (cuantos bits conforman una palabra, chequeo de errores, etc.)

El LPC1769 implementa varios estándares de comunicación serie: RS232, RS485, SPI, I2C, Ethernet y USB. De todos estos utilizaremos RS232, por ser el más útil para la comunicación con módems (wifi, bluetooth, GPRS, etc.). Posteriormente se podrán extrapolar algunos conceptos a otros estándares de comunicación, agregando a cada uno ciertas características particulares.

Tipos de comunicación serie:

Simplex

El envío de información es en un único sentido. El transmisor y el receptor están bien identificados. Se utiliza un solo canal de datos.



Half-Duplex

El envío de información es bidireccional pero multiplexada en el tiempo.

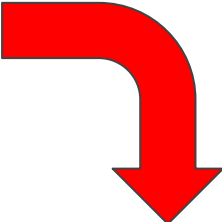
Como se muestra en el esquema, primero se transmite en un sentido (1) y luego en el otro sentido (2).

Se utiliza un solo canal de datos.



Full-Duplex

El envío de información es bidireccional simultáneo. Se utilizan dos canales de datos.

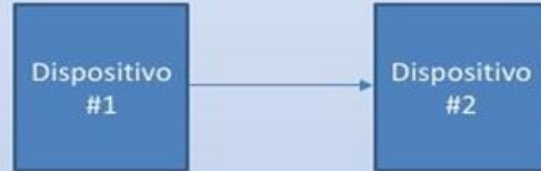


Casi no se utiliza, porque generalmente el dispositivo #2 tiene alguna respuesta, aunque sea un ACK o NACK

Tipos de comunicación serie:

Simplex

El envío de información es en un único sentido. El transmisor y el receptor están bien identificados. Se utiliza un solo canal de datos.



Half-Duplex

El envío de información es bidireccional pero multiplexada en el tiempo.

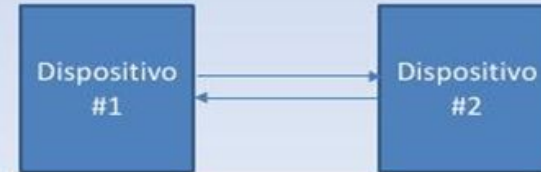
Como se muestra en el esquema, primero se transmite en un sentido (1) y luego en el otro sentido (2).

Se utiliza un solo canal de datos.

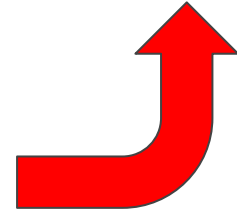


Full-Duplex

El envío de información es bidireccional simultáneo. Se utilizan dos canales de datos.



En este tipo de comunicaciones generalmente hay algún dispositivo que inicia las comunicaciones (master) y otro que responde (slave). Algunos ejemplos pueden ser la I2C, SPI, RS485



Tipos de comunicación serie:

Simplex

El envío de información es en un único sentido. El transmisor y el receptor están bien identificados. Se utiliza un solo canal de datos.



Half-Duplex

El envío de información es bidireccional pero multiplexada en el tiempo.

Como se muestra en el esquema, primero se transmite en un sentido (1) y luego en el otro sentido (2).

Se utiliza un solo canal de datos.

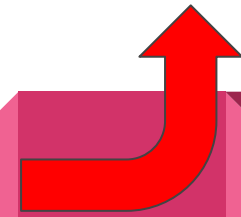


Full-Duplex

El envío de información es bidireccional simultáneo. Se utilizan dos canales de datos.

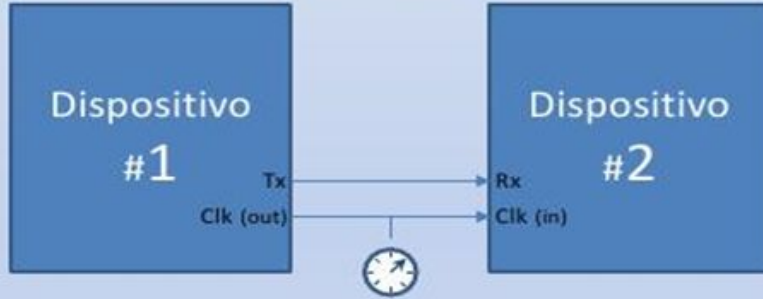


Es la que utilizaremos en el LPC845. Hay un cable para envío de datos y otro para recepción, por lo que la comunicación se puede dar en los dos sentidos en forma simultánea.



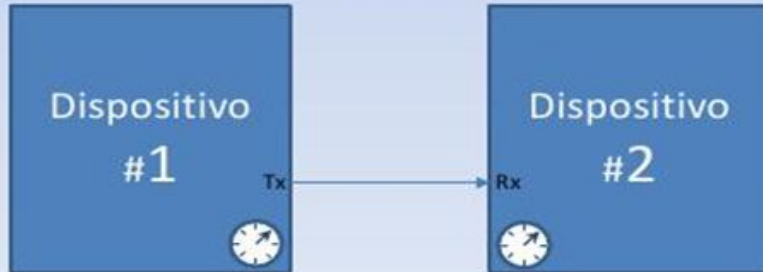
Tipo de comunicación serie: Sincrónica/Asincrónica

Comunicación Serie Sincrónica



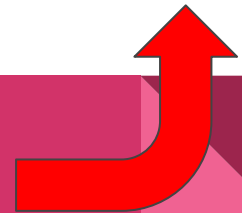
- Es necesario una señal de Clock común a ambos dispositivos.
- Es necesario un hilo extra para sincronizar la comunicación.

Comunicación Serie Asincrónica

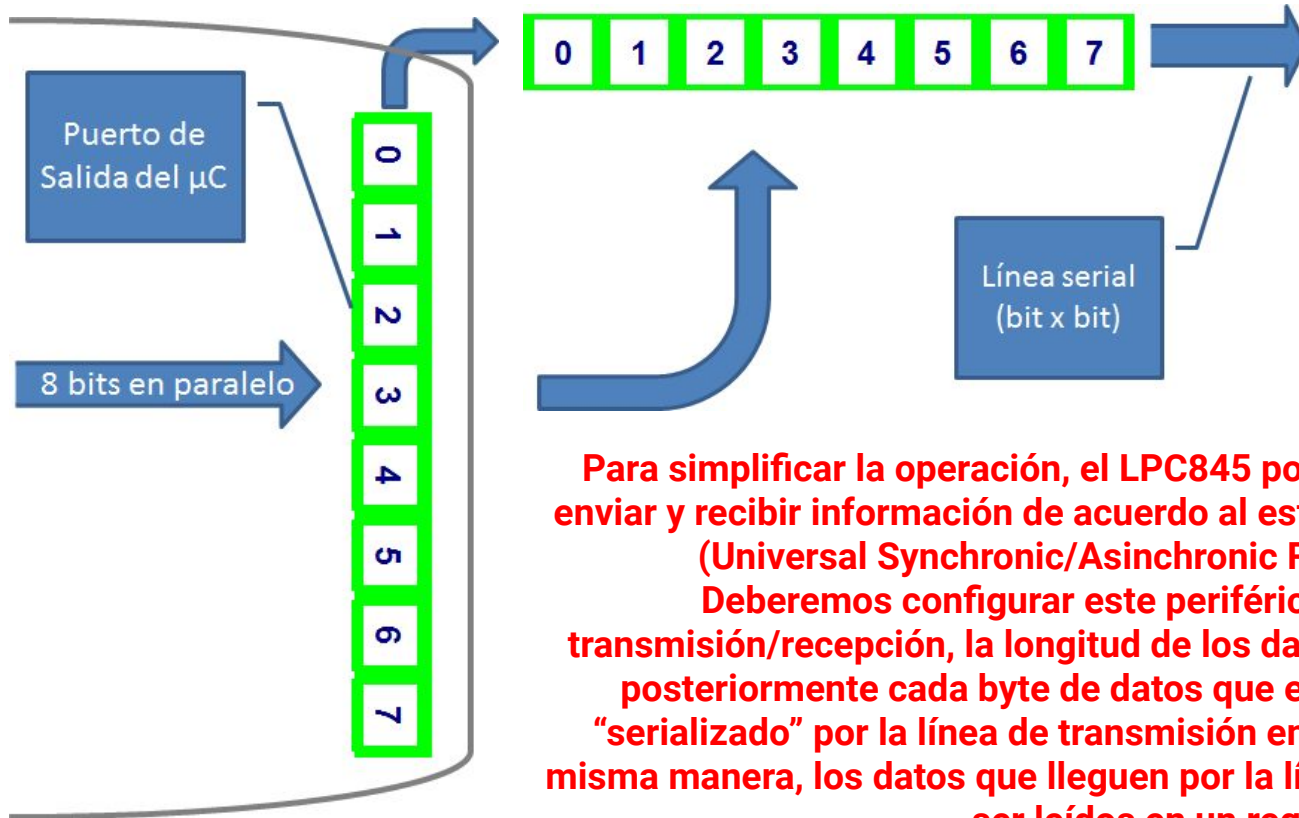


- Cada dispositivo posee su propia señal de clock.
- Los dispositivos deben conocer la velocidad de transmisión.
- La transmisión se sincroniza mediante los bits de Start y Stop.

El estándar RS-232 no prevee una señal para el sincronismo, por lo que tendremos que conocer previamente a que velocidad funciona el dispositivo con el que nos queremos comunicar.



Proceso de “serialización” de la información

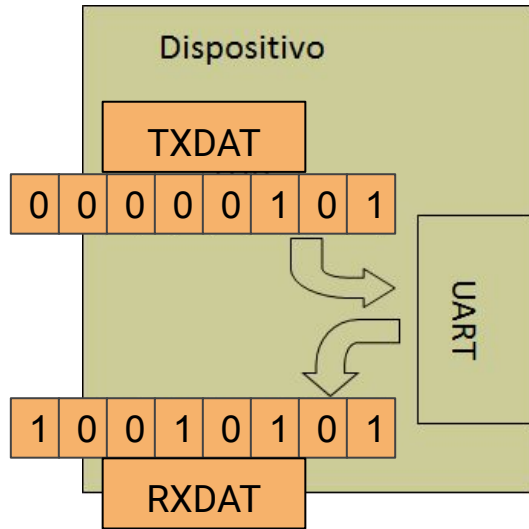


Para simplificar la operación, el LPC845 posee un periférico dedicado a enviar y recibir información de acuerdo al estándar RS232, llamado USART (Universal Synchronic/Asynchronous Receiver Transmitter).

Deberemos configurar este periférico con la velocidad de transmisión/recepción, la longitud de los datos y el chequeo de errores, y posteriormente cada byte de datos que enviemos al periférico será “serializado” por la línea de transmisión en el formato adecuado. De la misma manera, los datos que lleguen por la línea serán “paralelizados” para ser leídos en un registro.

Universal Synchronic/Asinchronic Receiver Transmitter (USART)

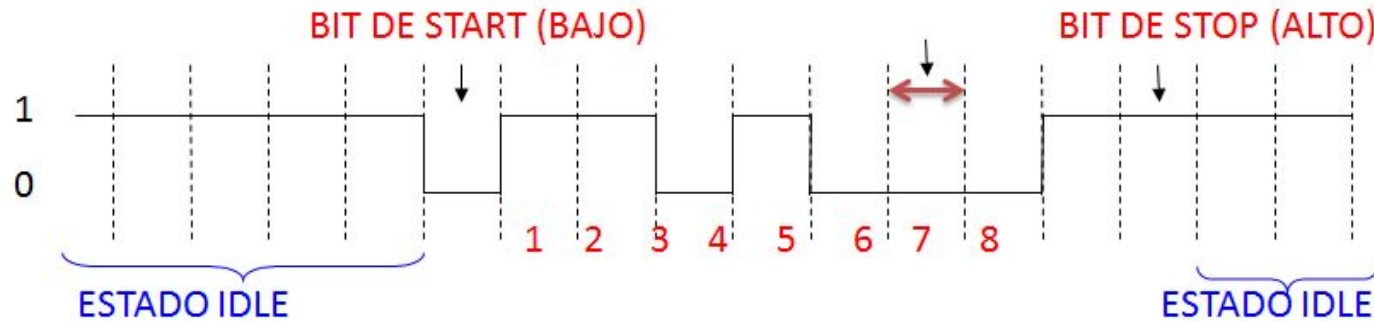
Este dispositivo se encarga entonces de la parte física de la comunicación, y nuestra tarea será configurarlo para que se comporte de la manera deseada en nuestro sistema.



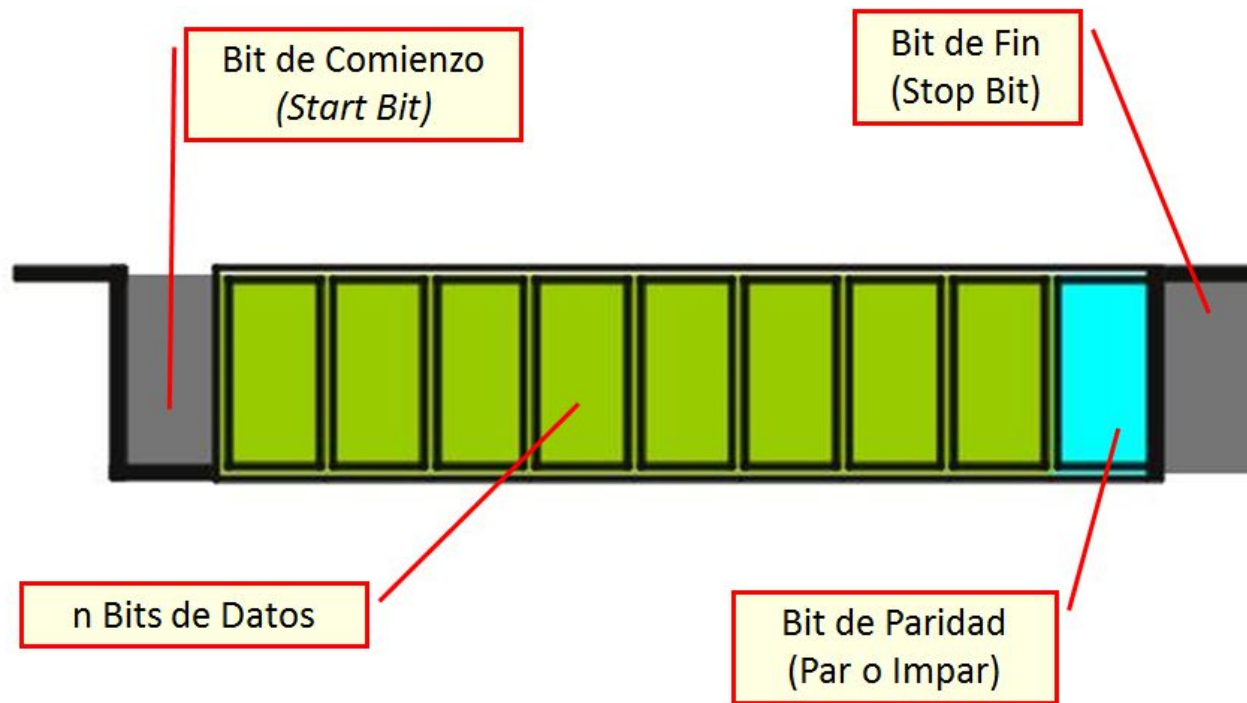
Parámetros involucrados en la comunicación

Para poder establecer una comunicación (asíncrona) se deben tener en cuenta las siguientes consideraciones:

1. La velocidad de trabajo (baudios, o cantidad de bits por segundo)
2. Se debe definir una longitud de palabra que permita identificar cuando se terminó cada conjunto de bits.
3. Debe haber un bit adicional que indique que va a comenzar la comunicación y otro que indique el estado inactivo (IDLE) de la línea.



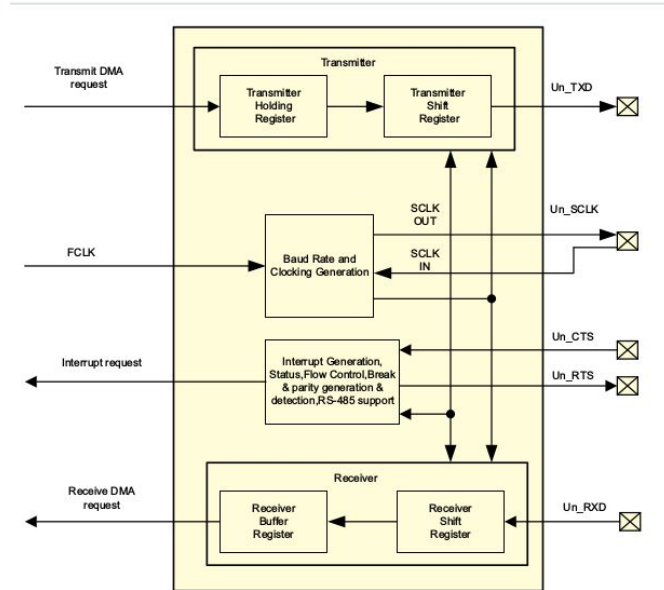
Capa lógica: cantidad de bits de información



El estándar RS232 indica que por cada bloque de información (que puede tener 5, 6, 7 u 8 bits de largo) deberá haber un bit de comienzo (un cambio de estado con respecto al estado de “reposo” de la línea), un bit de stop (un período de duración con la línea en estado de reposo), y puede haber o no un bit de paridad (para que la cantidad de 1s total sean pares o impares). Este último bit me permite saber si hubo algún error sencillo en la transmisión.

USARTs LPC845

El LPC845 tiene 5 USARTs, con registros independientes para configurar todos los parámetros anteriores, sumadas a algunas características adicionales que no veremos en Informática II. Asimismo pueden configurarse las interrupciones de estos periféricos ante el envío o recepción de datos por cualquiera de los 5 canales.



APB peripherals

31-30	(reserved)	0x4007 FFFF
29	UART4	0x4007 8000
28	UART3	0x4007 4000
27	UART2	0x4007 0000
26	UART1	0x4006 C000
25	UART0	0x4006 8000
24	CapTouch	0x4006 4000
23	SPI1	0x4006 0000
22	SPI0	0x4005 C000

Configurando la UART: Habilitación del clock

Como en otros periféricos, primero debemos habilitar el clock del dispositivo (registro SYSAHBCLKCTRL0)

Table 146. System clock control 0 register (SYSAHBCLKCTRL0, address 0x4004 8080) bit description ...continued

Bit	Symbol	Value	Description	Reset value
13	CRC		Enables clock for CRC.	0
		0	Disable	
		1	Enable	
14	UART0		Enables clock for USART0.	0
		0	Disable	
		1	Enable	
15	UART1		Enables clock for USART1.	0
		0	Disable	
		1	Enable	
16	UART2		Enables clock for USART2	0

```
// HABILITO LA UART0  
SYSCON->SYSAHBCLKCTRL0 |= (1 << 14);
```

Configurando la UART: Habilidad del clock

En el caso de la UART, como veremos en otros periféricos, conviene “resetearla” antes de usarla. Para esto se pone un 0 y luego un 1 en el registro PRESETCTRL0:

Table 148. Peripheral reset control 0 register (PRESETCTRL0, address 0x4004 8088) bit description

Bit	Symbol	Value	Description	Reset value
3:0	-		Reserved	1
4	FLASH_RST_N		Flash controller reset control	1
		0	Assert the flash controller reset.	
		1	Clear the flash controller reset.	
5	I2C0_RST_N		I ² C0 reset control	1
		0	Assert the I ² C0 reset.	
		1	Clear the I ² C0 reset.	
14	UART0_RST_N		UART0 reset control	1
		0	Assert the UART0 reset.	
		1	Clear the UART0 reset.	
15	UART1_RST_N		UART1 reset control	1
		0	Assert the UART1 reset.	
		1	Clear the UART1 reset.	
16	UART2_RST_N		UART2 reset control	1
		0	Assert the UART2 reset.	
		1	Clear the UART2 reset.	

```
// RESETEO LA UART0  
SYSCON->PRESETCTRL0 &= ~(1 << 14);  
SYSCON->PRESETCTRL0 |= (1 << 14);
```


Configurando la UART: Selección de pines

Seleccionaremos los pines que utilizaremos para el envío y recepción de bits (registros PINASSIGNx). En caso de que no haya sido activada la matriz de selección de pines, debemos activarla (habilitarle el CLK en SYSAHBCLKCTRL0)

Table 180. Pin assign register 0 (PINASSIGN0, address 0x4000 C000) bit description

Bit	Symbol	Description
7:0	U0_TXD_O	U0_TXD function assignment. The value is the pin number to be assigned to this function. The following pins are available: PIO0_0 (= 0) to PIO0_31 (= 0x1F) and from PIO1_0 (= 0x20) to PIO1_21(= 0x35).
15:8	U0_RXD_I	U0_RXD function assignment. The value is the pin number to be assigned to this function. The following pins are available: PIO0_0 (= 0) to PIO0_31 (= 0x1F) and from PIO1_0 (= 0x20) to PIO1_21(= 0x35).
23:16	U0_RTS_O	U0_RTS function assignment. The value is the pin number to be assigned to this function. The following pins are available: PIO0_0 (= 0) to PIO0_31 (= 0x1F) and from PIO1_0 (= 0x20) to

```
// Selecciono en que pines quiero
// la transmision (TX) (P0.8)
// y la recepción (RX) (P0.9)

//Habilito el CLK del la SWM:
SYSCON->SYSAHBCLKCTRL0 |= 1<<7;
//Pongo en 0 todos los bits:
PINASSIGN0 &= ~(0xFFFF)
//Pongo 8 y 9 en los primeros 16
bits:
PINASSIGN0 |= (8 << 0) | (9 << 8);
```

Configurando las características de la comunicación: Registro CFG (primeros bits)

Table 324. USART Configuration register (CFG, address 0x4006 4000 (USART0), 0x4006 8000 (USART1), 0x4006 C000 (USART2), 0x4007 0000 (USART3), 0x4007 4000 (USART4)) bit description

Bit	Symbol	Value	Description	Reset Value
0	ENABLE		USART Enable.	0
		0	Disabled. The USART is disabled and the internal state machine and counters are reset. While Enable = 0, all USART interrupts and DMA transfers are disabled. When Enable is set again, CFG and most other control bits remain unchanged. For instance, when re-enabled, the USART will immediately generate a TXRDY interrupt (if enabled in the INTENSET register) or a DMA transfer request because the transmitter has been reset and is therefore available.	
		1	Enabled. The USART is enabled for operation.	
1	-		Reserved. Read value is undefined, only zero should be written.	NA
3:2	DATALEN		Selects the data size for the USART.	00
		0x0	7 bit Data length.	
		0x1	8 bit Data length.	
		0x2	9 bit data length. The 9th bit is commonly used for addressing in multidrop mode. See the ADDRDET bit in the CTL register.	
		0x3	Reserved.	

```
//Configuro la comunicación 8,N,1:
    USART0->CFG =
// 0=DISABLE 1=ENABLE
        (0 << 0)
// 0=7BITS 1=8BITS 2=9BITS
        |    (1 << 2)
// 0=NOPARITY 2=PAR 3=IMPAR
        |    (0 << 4)
// 0=1BITSTOP 1=2BITSTOP
        |    (0 << 6)
// 0=NOFLOWCONTROL 1=FLOWCONTROL
        |    (0 << 9)
// 0=ASINCRONICA 1=SINCRONICA
        |    (0 << 11) ;
```

Configurando las características de la comunicación: Registro CFG (siguientes bits)

5:4	PARITYSEL		Selects what type of parity is used by the USART.	00
		0x0	No parity.	
		0x1	Reserved.	
		0x2	Even parity. Adds a bit to each character such that the number of 1s in a transmitted character is even, and the number of 1s in a received character is expected to be even.	
6	STOPLEN	0x3	Odd parity. Adds a bit to each character such that the number of 1s in a transmitted character is odd, and the number of 1s in a received character is expected to be odd.	0
			Number of stop bits appended to transmitted data. Only a single stop bit is required for received data.	
		0	1 stop bit.	
8:7	-	1	2 stop bits. This setting should only be used for asynchronous communication.	NA
			Reserved. Read value is undefined, only zero should be	

```
//Configuro la comunicación 8,N,1:
    USART0->CFG =
// 0=DISABLE 1=ENABLE
    (0 << 0)
// 0=7BITS 1=8BITS 2=9BITS
    |    (1 << 2)
// 0=NOPARITY 2=PAR 3=IMPAR
    |    (0 << 4)
// 0=1BITSTOP 1=2BITSTOP
    |    (0 << 6)
// 0=NOFLOWCONTROL 1=FLOWCONTROL
    |    (0 << 9)
// 0=ASINCRONICA 1=SINCRONICA
    |    (0 << 11) ;
```

Configurando las características de la comunicación: Registro CFG (siguientes bits)

9	CTSEN		CTS Enable. Determines whether CTS is used for flow control. CTS can be from the input pin, or from the USART's own RTS if loopback mode is enabled.	0
		0	No flow control. The transmitter does not receive any automatic flow control signal.	
		1	Flow control enabled. The transmitter uses the CTS input (or RTS output in loopback mode) for flow control purposes.	
10	-		Reserved. Read value is undefined, only zero should be written.	NA
11	SYNCEN		Selects synchronous or asynchronous operation.	0
		0	Asynchronous mode is selected.	
		1	Synchronous mode is selected.	

```
//Configuro la comunicación 8,N,1:
    USART0->CFG =
// 0=DISABLE 1=ENABLE
    (0 << 0)
// 0=7BITS 1=8BITS 2=9BITS
    |    (1 << 2)
// 0=NOPARITY 2=PAR 3=IMPAR
    |    (0 << 4)
// 0=1BITSTOP 1=2BITSTOP
    |    (0 << 6)
// 0=NOFLOWCONTROL 1=FLOWCONTROL
    |    (0 << 9)
// 0=ASINCRONICA 1=SINCRONICA
    |    (0 << 11);
```

CLK del periférico: ¿Con qué frecuencia lo alimento?

Antes de configurar la velocidad de la comunicación, tengo que saber a qué frecuencia está alimentado el periférico.

Esto se configura con el registro **UARTnCLKSEL**, donde puedo seleccionar varias fuentes para alimentar el mismo:

```
//Selecciono que el periférico  
//funcione a 30MHz
```

```
UART0CLKSEL = 1;
```

- UART0 clock source select register (UART0CLKSEL, address 0x4004 8090).
- UART1 clock source select register (UART1CLKSEL, address 0x4004 8094).
- UART2 clock source select register (UART2CLKSEL, address 0x4004 8098).
- UART3 clock source select register (UART3CLKSEL, address 0x4004 809C).
- UART4 clock source select register (UART4CLKSEL, address 0x4004 80A0).
- I²C0 clock source select register (I2C0CLKSEL, address 0x4004 80A4).
- I²C1 clock source select register (I2C1CLKSEL, address 0x4004 80A8).
- I²C2 clock source select register (I2C2CLKSEL, address 0x4004 80AC).
- I²C3 clock source select register (I2C3CLKSEL, address 0x4004 80B0).
- SPI0 clock source select register (SPI0CLKSEL, address 0x4004 80B4).
- SPI1 clock source select register (SPI1CLKSEL, address 0x4004 80B8).

Table 150. Peripheral clock source select registers

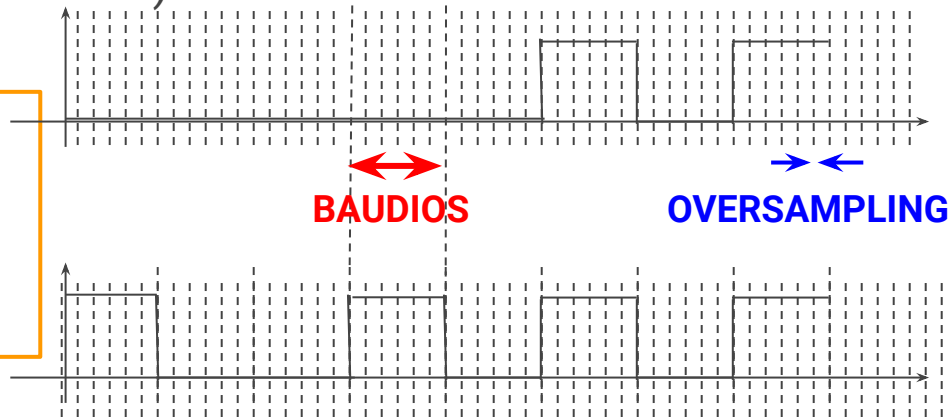
Bit	Symbol	Value	Description	Reset value
2:0	SEL		Peripheral clock source	0x7
		0x0	FRO	
		0x1	Main clock	
		0x2	FRG0 clock	
		0x3	FRG1 clock	
		0x4	FRO_DIV = FRO / 2	
		0x5	Reserved	
		0x6	Reserved	
		0x7	None	
31:3	-	-	Reserved	-

Configuración de la velocidad

Para setear el baudrate de la USART, tengo que dividir el CLK que recibe por un divisor (registro BRG) para que funcione N veces más rápido que la baudrate seleccionada. Este factor N (típicamente 16) se lo conoce como **oversampling**, y sirve para compensar por ruidos o demoras que haya en la línea de comunicación. Pongo un bit en un estado, dejo pasar unos ciclos de clock, y luego leo el estado del bit (una vez que la señal se “asentó”):

```
//Si el baudrate = 9600 baudios  
//Oversampling = 16  
//CLK de la UART = 30MHz:
```

```
USART0->BRG= 30000000/(9600*16);
```



Configurando las interrupciones: Registro INTEN

Cada USART tiene varias fuentes de interrupciones. Puede interrumpir cuando recibe un dato nuevo (los N bits + bit de stop + paridad), cuando se termina de transmitir un dato (y el periférico está listo para mandar un byte nuevo), o cuando se produce alguna condición de error en la transmisión (error por bit de paridad, entre otros). Cada una de estas fuentes se activa con el registro **INTENSET**, y luego su activación se ve reflejada en el registro **STAT**:

b31	b30	29-19	b18	b17	b16	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
		~			ABERR	NOISE	PARITY	FRAME	START	DELTA	RXBR FAK	--	OVERRU NINT	--	TXDISS TAT	DELTA CTS	CTS	TXIDLE	TXRDY	RXIDLE	RXRDY

RECEPCION

ERROR

TRANSMISIÓN

```
//Configuro interrupciones por RX:  
USART0->INTENSET = 0x01;  
  
//Habilito interrupciones por TX:  
USART0->INTENSET |= 1<<2;
```

NO LAS HABILITO AL INICIO PORQUE SINO SIEMPRE ME ESTARÍA INTERRUPIENDO, YA QUE ME AVISA CUANDO ESTÁ LISTO PARA ENVIAR UN NUEVO DATO. LA HABILITARÉ CUANDO TENGO QUE MANDAR MUCHOS DATOS TODOS JUNTOS

Configurando las interrupciones: NVIC

Como ya vimos, además de habilitar la interrupción del periférico, tenemos que decirle al controlador de interrupciones que cuando se active la interrupción del dispositivo, se interrumpa el programa en el microcontrolador. Esto lo hacemos en el registro ISER0:

Table 110. Interrupt Set Enable Register 0 register (ISER0, address 0xE000 E100) bit description

Bit	Symbol	Description	Res
0	ISE_SPI0	Interrupt enable.	0
1	ISE_SPI1	Interrupt enable.	0
2	ISE_DAC0	Interrupt enable.	0
3	ISE_UART0	Interrupt enable.	0
4	ISE_UART1	Interrupt enable.	0
5	ISE_UART2	Interrupt enable.	0
6		Reserved	0

```
//Habilito interrupciones de la UART0 en el NVIC:  
ISER0 |= 1<<3;
```


Iniciando la UART

```
void UART0_Init(void)
```

```
{  
    // HABILITO LA UART0  
    SYSCON->SYSAHBCLKCTRL0 |= (1 << 14);
```

```
    //Reseteo la UART0:  
    SYSCON->PRESETCTRL0 &= ~(1<<14);  
    SYSCON->PRESETCTRL0 |= (1<<14);
```

```
    //Habilito el CLK a la SWM:  
    SYSCON->SYSAHBCLKCTRL0 |= (1<<7);
```

```
    // CONFIGURO LA MATRIX TX0 en P0.25 y RX0 en P0.24  
    PINASSIGN0 &= ~(0x0000FFFF);  
    PINASSIGN0 = (25 << 0) | (24 << 8);
```

```
    // CONFIGURACION GENERAL  
    USART0->CFG = (0 << 0)           // 0=DISABLE 1=ENABLE  
    | (1 << 2)           // 0=7BITS 1=8BITS 2=9BITS  
    | (0 << 4)           // 0=NOPARITY 2=PAR 3=IMPAR  
    | (0 << 6)           // 0=1BITSTOP 1=2BITSTOP  
    | (0 << 9)           // 0=NOFLOWCONTROL 1=FLOWCONTROL  
    | (0 << 11);        // 0=ASINCRONICA 1=SINCRONICA
```

```
    // CONFIGURACION INTERRUPTIONES  
    USART0->INTENSET = (1 << 0);    //RX
```

```
    // CONFIGURACION DEL BAUDRATE  
    UART0CLKSEL = 0; //CLK = FRO = 30MHz  
    USART0->BRG = (FREQ_PRINCIPAL / (BAUDRATE* 16));
```

```
    //Habilito interrupcion USART0 en el NVIC  
    ISER0 |= (1 << 3);
```

```
    // ENABLE LA UART  
    USART0->CFG |= 1;
```

```
    return;
```

Una vez finalizada la configuración, habilito la UART

SYSAHBCLKCTRL

Habilito el CLK del periférico

PINASSIGN0

Configuro el pin P0.8 para la transmisión y el P0.9 para la recepción del puerto serie

CFG - Configuración de la comunicación

Selecciono una palabra de 8 bits, 1 bit de stop, sin control de paridad, configuro que la comunicación sea solo con 2 pines, en formato asincrónico.

INTENSET - Configuración de interrupciones

Configuro el periférico para que interrumpa cada vez que llega un nuevo byte por el puerto serie

Baudrate - Velocidad de la comunicación

Configuro el clk del periférico como el clk principal, y el divisor de la velocidad para alcanzar la velocidad deseada

NVIC - Habilidad de interrupción

Configuro el controlador programable de interrupciones para que la interrupción de la USART0 interrumpa el programa

Identificando la interrupción - USARTn_IRQHandler

```
void USART0_IRQHandler (void)
```

```
{
```

```
    int16_t auxTemporal;
```

```
    uint32_t stat = USART0->STAT;
```

```
    // CASO RECEPCION
```

```
    if(stat & (1 << 0))
```

```
    {
```

```
    }
```

```
    //CASO TRANSMISION
```

```
    if(stat & (1 << 2))
```

```
    {
```

```
    }
```

```
    //CASO ERRORES
```

```
    else
```

```
    {
```

```
    }
```

```
}
```

Una vez que se genera una interrupción, debo saber si la misma se generó por recepción (llegó un nuevo dato), por transmisión (se terminó de enviar un dato), o por error (en caso de que haya activado este tipo de interrupción). Esto puedo verlo en el registro **STAT**

b16	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
ABERR	NOISE	PARITY	FRAME	START	DELTA	RXBR FAK	--	OVERRU NINT	--	TXDISS TAT	DELTA CTS	CTS	TXIDLE	TXRDY	RXIDLE	RXRDY

Ejemplo: Configuración UART1

A partir de lo anterior, generar la función de inicialización de la UART1, utilizando 2 pines cualesquiera para comunicarse al exterior. Realizar un programa que ante la recepción de un dato (recibe por interrupción), reenvíe el dato recibido al emisor (El programa debe hacer un eco). Para probarlo, se pueden conectar entre sí las patas de emisión y recepción (todo lo enviado será luego recibido nuevamente), o se puede conectar a la PC mediante un conversor USB-serie. Con una terminal (como el putty, gtkterm u otros), se puede abrir el puerto serie virtual generado y enviar datos, y visualizar la recepción.



UARTs en el kit y para
nuestros proyectos



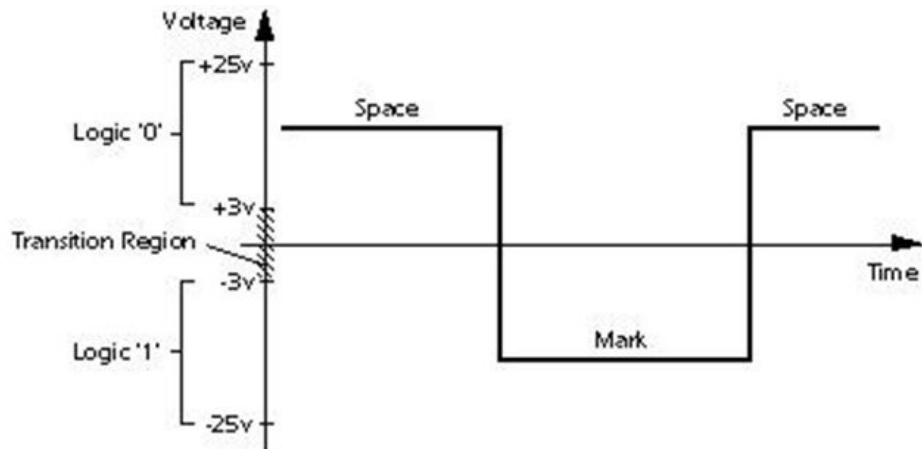
La norma eléctrica RS-232

Las especificaciones eléctricas del puerto serial están contenidas en el estándar RS232 del EIA que establece muchos parámetros, entre ellos:

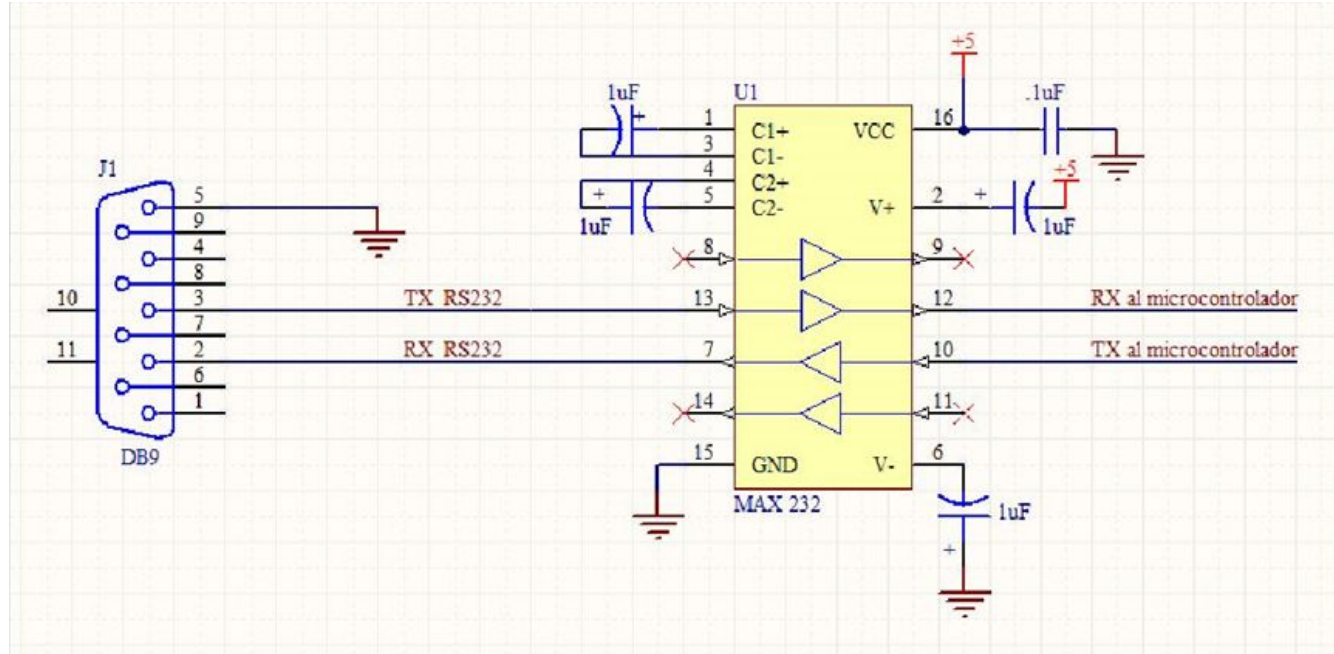
Un “Espacio” (0 lógico) está entre +3 y +25 volts.

Una “Marca” (1 lógico) estará entre -3 y -25 Volts

La región entre +3 y -3 volts es indefinida.

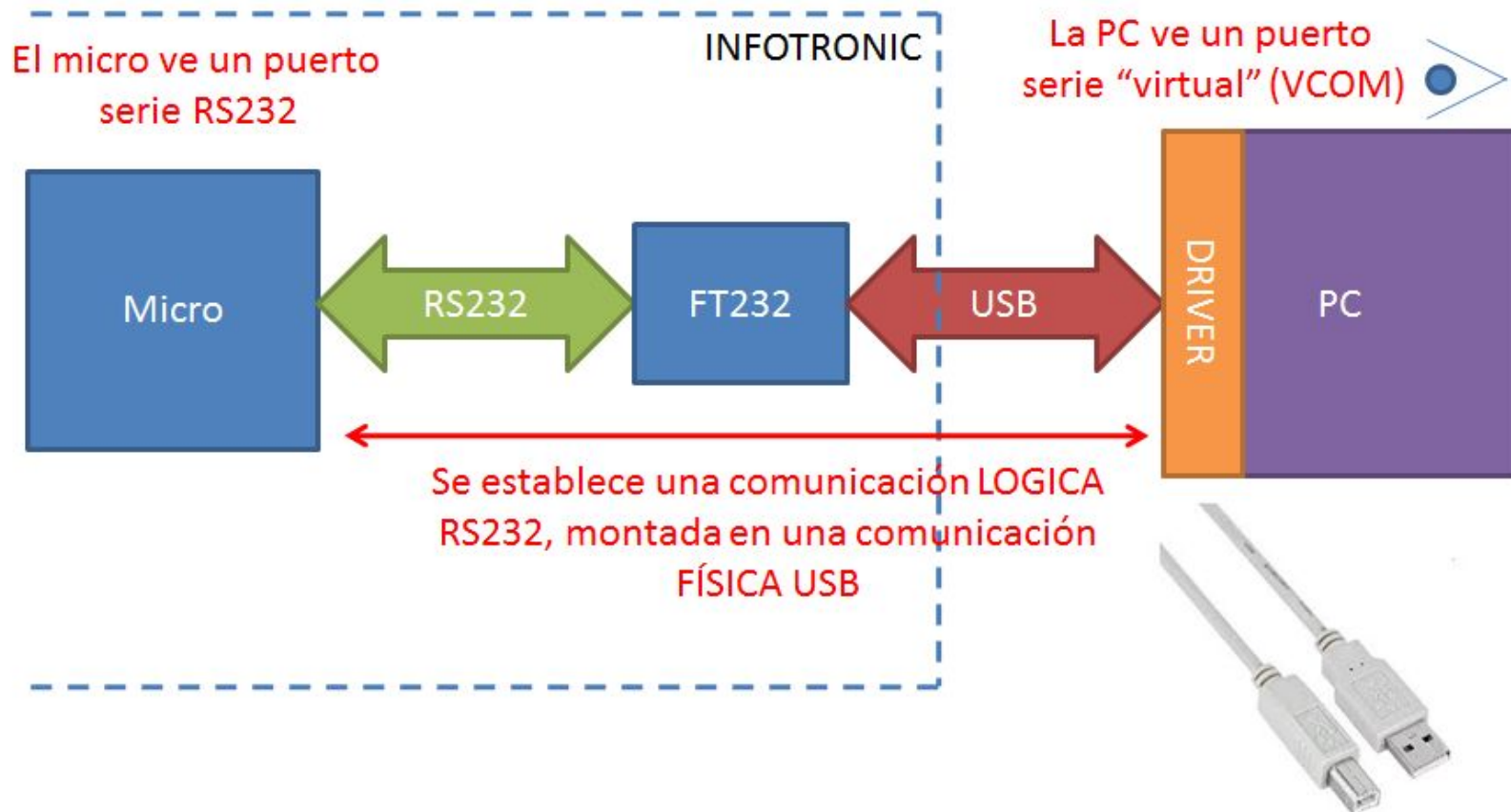


MAX 232 - Para comunicarme con puertos series “viejos” (en alguna PC de escritorio por ejemplo)



Adapta niveles RS232 y TTL

Bridge RS232/USB - FT232



Bridge RS232/USB - CP2102



Al conectarse, el FT232 (o CP2102) se “presentará” como un dispositivo USB, de tipo “USB to UART Bridge”. Por defecto, los SO no reconocen este tipo de dispositivos como estándar, por lo que hay que descargar el driver correspondiente para que el Sistema Operativo reconozca el dispositivo y se cree la “capa de software” que presentará a las aplicaciones un “puerto serie virtual”, con iguales características que un puerto serie común.

Link a la página de FTDI Chip con el driver:

<http://www.ftdichip.com/Support/Documents/InstallGuides.htm>