

Tema 11: Comunicación Serie Asíncrona (USART)

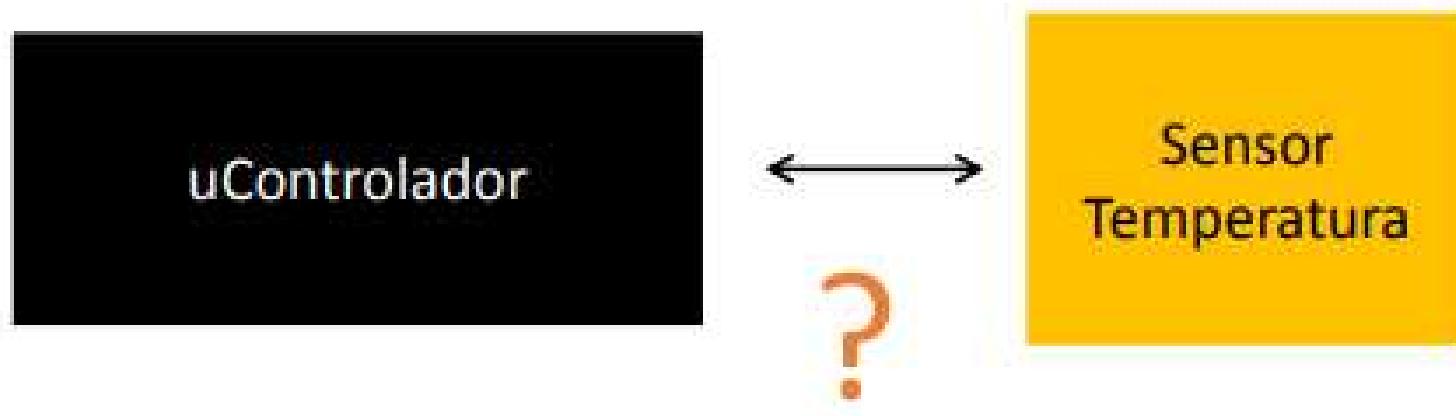
Índice

- 1 - Conceptos Generales
 - Comunicaciones Serie
 - Comunicación Serie Asíncrona
- 2 - Arquitectura, Configuración y Uso
- 3 - ¿Cómo utilizar y probar la USART?
- 4 - Ejemplos
 - ¿Cómo utilizar la comunicación serie y cómo probarla?
 - Tx por Espera Activa
 - Rx y Tx por Espera Activa
 - Rx por IRQ y Tx por Espera Activa
- 5 - Ejercicios Propuestos
- 6 - Uso de la USART a nivel de Registros:
 - Registros de Control
 - Registros de Datos
 - Registros de Estado
- 7 - Ejemplos a nivel de Registros

1 - Conceptos Generales

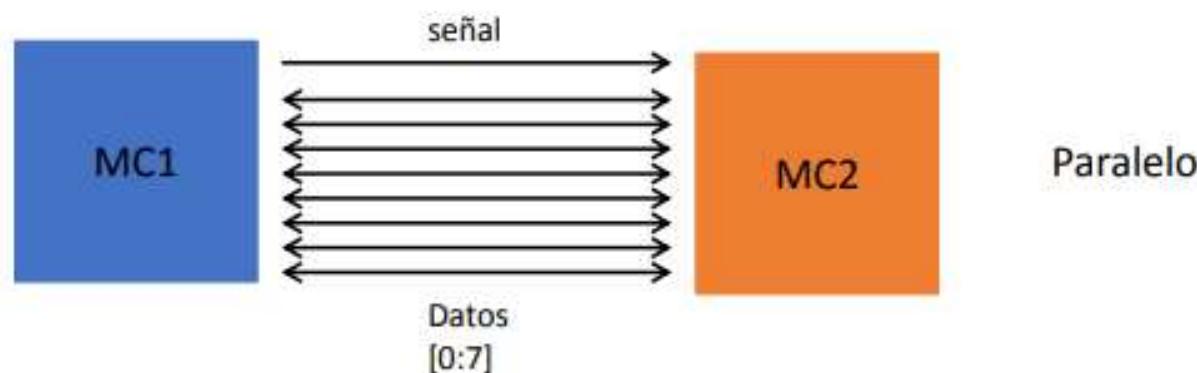
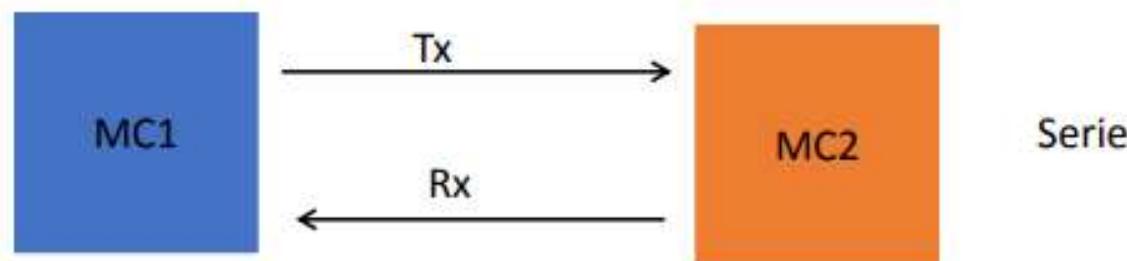
Conceptos Generales

- ¿Cuándo necesitamos interfaces de comunicación?
- ¿Por qué no es suficiente con la interfaz GPIO o la interfaz ADC/DAC?



Conceptos Generales

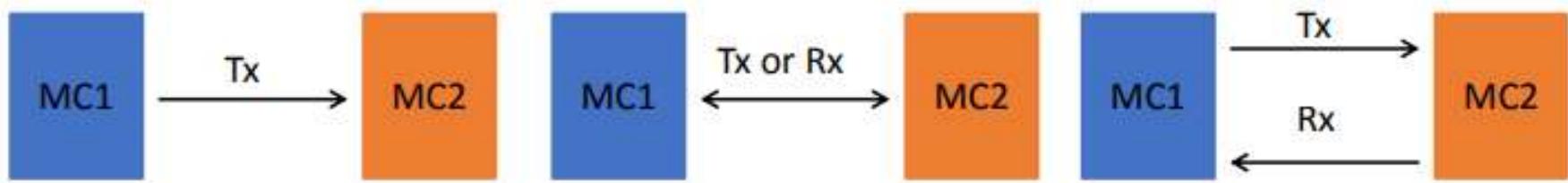
- Hay posibilidad de comunicación serie y comunicación paralelo
- ¿Por qué la mayoría de las interfaces de comunicación son serie y no paralelo?



Conceptos Generales

- En la comunicación serie se trata de transmitir la información bit a bit:
 - Se divide la información en palabras
 - Cada palabra se transmite bit a bit
- Hay que buscar métodos y puntos de sincronismo
 - Para indicar la existencia de cada bit
 - Para indicar el inicio de una palabra
- Atendiendo al sincronismo de bit, se tendrá:
 - **Comunicación Síncrona:** cuando una señal de reloj indique la validez del bit transmitido
 - **Comunicación Asíncrona:** cuando no existe dicha señal de reloj, por lo que la temporización de emisor y receptor se realiza por medios independientes

Conceptos Generales: Terminología



- **Simplex mode**

La transmisión solo es posible en una dirección.

- **Half-duplex mode**

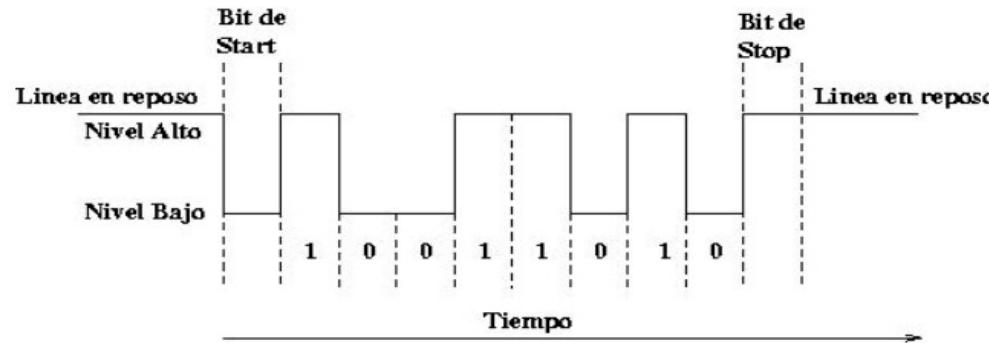
Se puede enviar y recibir información pero no simultáneamente.

- **Full-duplex mode**

Se puede enviar y recibir datos simultáneamente.

Conceptos Generales

- La comunicación asíncrona se realiza de la siguiente manera:
 - En estado de reposo, la línea de transferencia de datos tiene que estar a nivel alto
 - El inicio de la transmisión de un carácter se indica poniendo, **durante el tiempo de 1 bit**, la línea a 0V (**bit de arranque**)
 - Se transmiten uno por uno cada uno de los bits de la palabra. Se mantiene el valor de cada bit durante el **tiempo de 1 bit**.
 - Se transmite primero el bit menos significativo del carácter
 - Si el bit es un ‘1’ se colocan 5V en la línea, si es un ‘0’ se pone la línea a 0V
 - Después del último bit del carácter, se pone la línea a 5V, para indicar el final del carácter (**bit de parada**) durante el tiempo de 1 bit

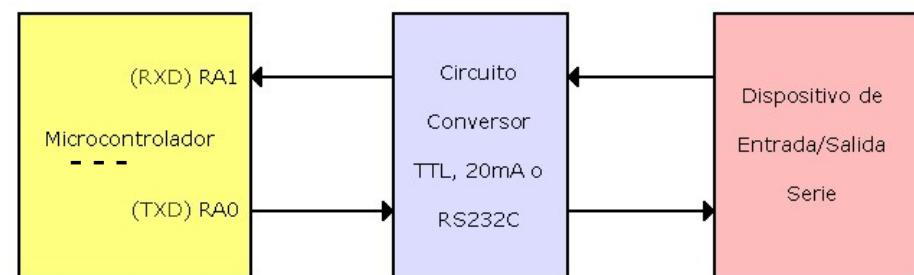


Conceptos Generales

- Esta forma de comunicación está regida por la selección previa y común (entre receptor y transmisor) de una serie de parámetros:
 - Velocidad de transmisión
 - Los baudios a los que se transmite (9600, 19200, 33600, etc.)
 - Esto determina el tiempo de bit: 9600bps → $t_{bit} = 0.1ms$
 - Longitud del carácter
 - Número de bits por carácter
 - Típicamente suelen ser 7 u 8 (últimamente siempre 8)
 - Tipo de Paridad (par / impar / ninguna)
 - Se trata de un bit que se añade al final del carácter para que se cumpla que en todo carácter transmitido, el número de 1s es un número par o impar (dependiendo de la paridad escogida)
 - Si la paridad es “ninguna” entonces no se transmite ese bit
 - Número de bits de parada (típicamente 1)

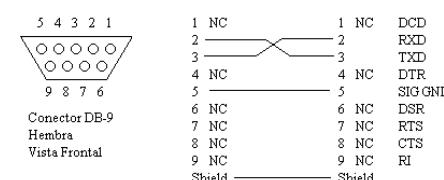
Conceptos Generales

- La comunicación serie asíncrona se suele especificar con expresiones del tipo:
9600,8,N,1
 - 9600bps
 - 8 bits de datos
 - N = sin paridad (E – paridad par / O – paridad impar)
 - 1 bit de parada
- Además hay que tener en cuenta que la comunicación serie está pensada para transmitir información a distancia
 - Si los bits se envían en TTL (0V – 3V) se atenuarán en la línea y no llegarán al destino si el cable es largo
 - Por tanto se utilizan transductores que convierten a otra señal eléctrica
 - RS-232: los 1s se convierten a una tensión negativa entre -5 y -15V, mientras que los 0s se convierten a una tensión positiva entre +5 y +15V. La conversión la hace un chip (p. ej. MAX232)
 - RS-485: se transmite mediante tensiones diferenciales, sobre pasando una distancia máxima de 1km
 - Etc.



Conceptos Generales

- La comunicación serie asíncrona suele ser **full-duplex**, ya que se contempla una línea de Tx y otra de Rx
 - En concreto, se utiliza frecuentemente la conexión **null-modem** (o módem nulo):
 - Tx del dispositivo 1 conectado al Rx del dispositivo 2
 - Rx del dispositivo 1 conectado al Tx del dispositivo 2
 - Una señal de GND común a los dos dispositivos
 - Aunque se pueden hacer conexiones más complejas, con control de flujo hardware
 - Hay un conjunto de señales (DCE, RTS, DTE, etc.) que se utilizaban antiguamente para poner en contacto los dos dispositivos, al estilo de una comunicación telefónica
 - A día de hoy casi nunca se utiliza. Se plantean o controles de flujo software (Xon – Xoff), o directamente (lo más habitual) sin **ningún control de flujo** (salvo en la capa de aplicación)



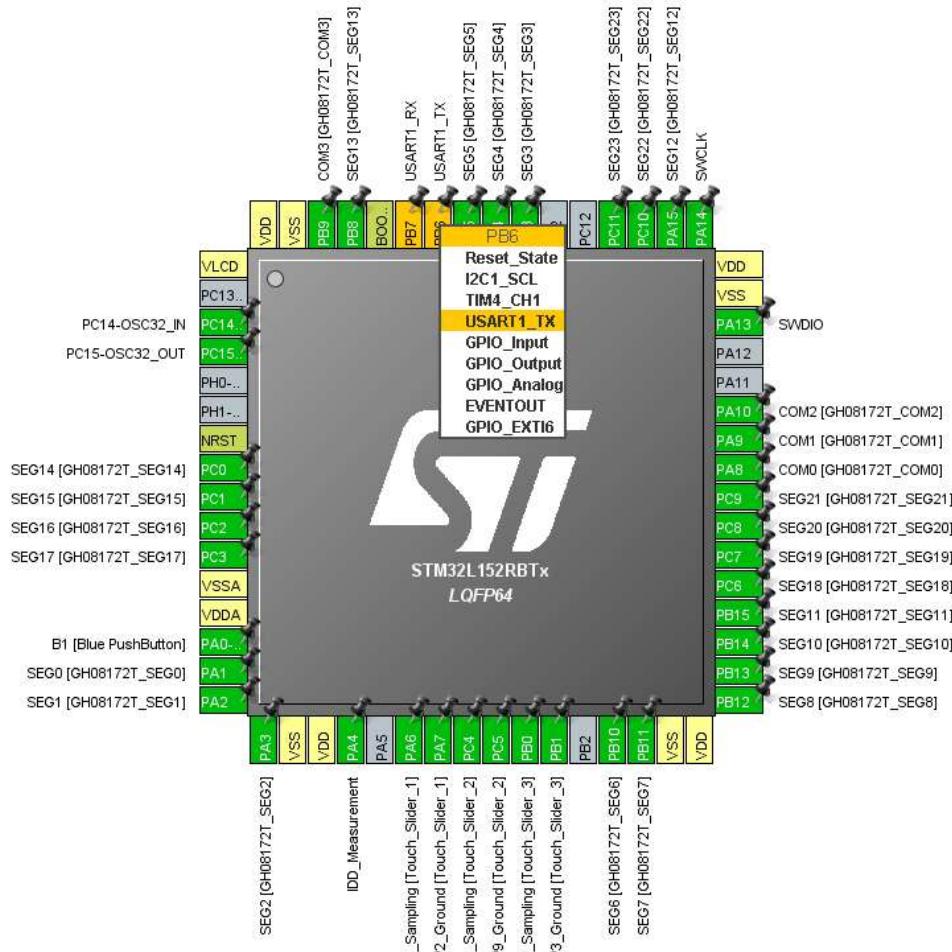
2 - Arquitectura, Configuración y Uso

Arquitectura

- La USART del STM32L152RB tiene las siguientes características principales:
 - Comunicación síncrona (que no se va a utilizar)
 - Comunicación asíncrona full duplex
 - Formato de codificación estándar NRZ
 - Generador programable de tasas de comunicación (baudrate)
 - Común a recepción y transmisión
 - Puede llegar a 4Mbps -> En los programas usaremos 9600 bps
 - Tamaño de datos programable (8 o 9 bits) -> En los programas usaremos 8 bits
 - Número de bits de parada configurables (1 o 2) -> En los programas usaremos 1 bit
 - Codificador IrDA
 - Capacidad de emulación de comunicación con tarjetas inteligentes (ISO/IEC 7816-3)
 - Habilitación independiente para recepción y transmisión
 - Control de estado de buffers de Rx y de Tx
 - Control de paridad -> En los programas no usaremos el bit de paridad

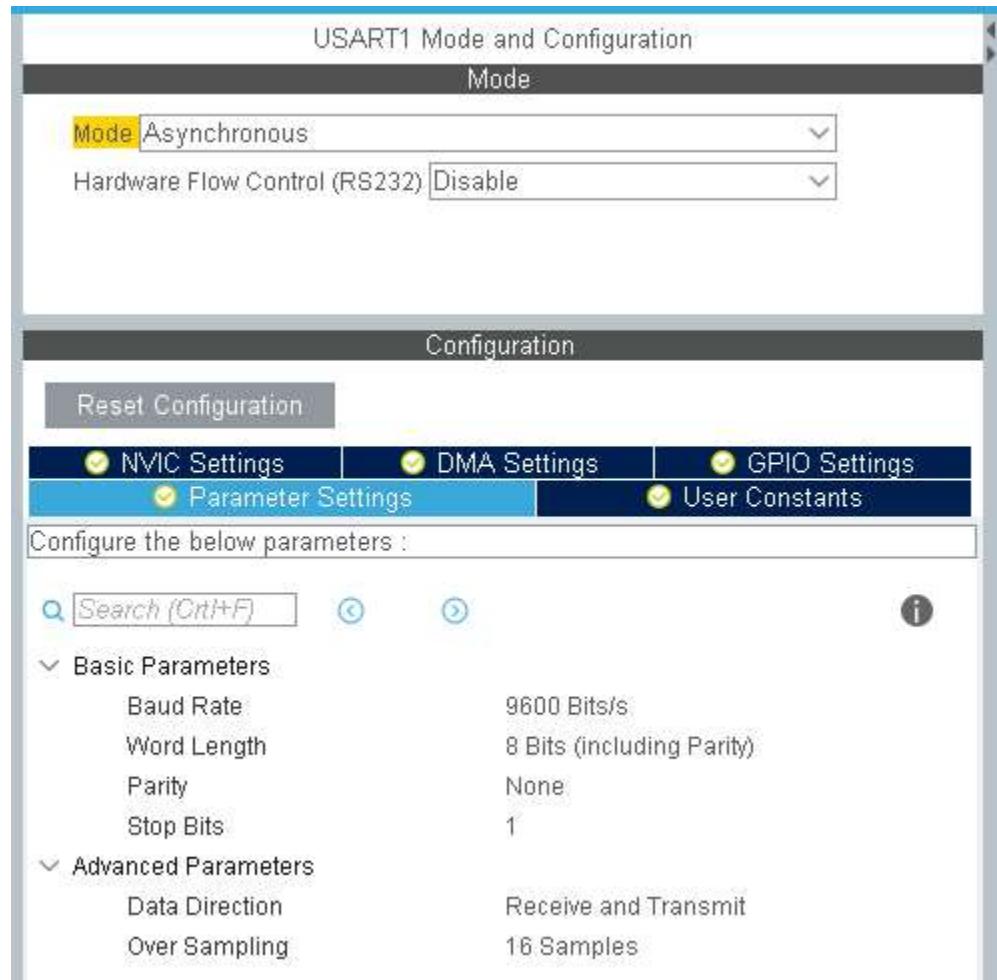
Configuración en CUBE

- Se configuran los pines (PB6 y PB7) como funcionalidad USART (USART1_TX y USART1_RX)



Configuración en CUBE

- Configure la USART en modo Asíncrono
- Deshabilite el control de flujo hardware (disable)
- Seleccione lo que hay en la imagen (para los ejemplos):
 - Baud Rate: 9800
 - Número de bits de datos: 8
 - Paridad: None
 - Bits de parada: 1
- Decida si va a comunicar en un sentido o en ambos. Para los ejemplos se puede poner ambos, aunque luego se use sólo un sentido (por comodidad)
- Decida el sobre-muestreo (8 ó 16)
 - Si no está seguro, déjelo como recomienda el programa (16)



Uso de la USART en µVision

Por Espera Activa

- Inicialización:
 - `HAL_UART_Init(h)`
- Transmisión:
 - `HAL_UART_Transmit(h, b, s, t)`
- Recepción:
 - `HAL_UART_Receive(h, b, s, t)`

Por Interrupciones

- Inicialización:
 - `HAL_UART_Init(h)`
- Transmisión:
 - `HAL_UART_Transmit_IT(h, b, s)`
 - `HAL_UART_TxCpltCallback(h)`
- Recepción:
 - `HAL_UART_Receive_IT(h, b, s)`
 - `HAL_UART_RxCpltCallback(h)`
- Gestión de Errores:
 - `HAL_UART_ErrorCallback(h)`

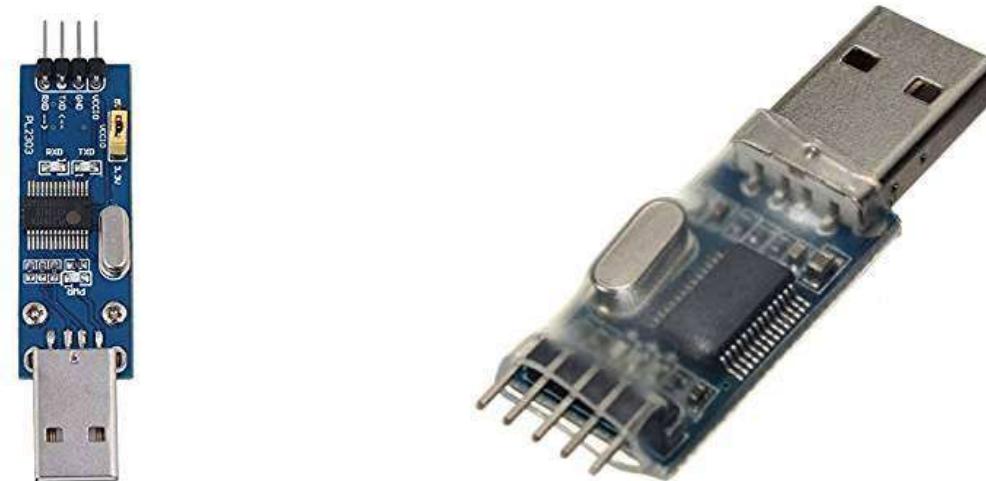
Parámetros:

h – handler: `UART_HandleTypeDef *`
b – buffer: `uint8_t *`
s – tamaño buffer: `uint16_t`
t – timeout en ms: `uint32_t`

3 - ¿Cómo utilizar y probar la USART?

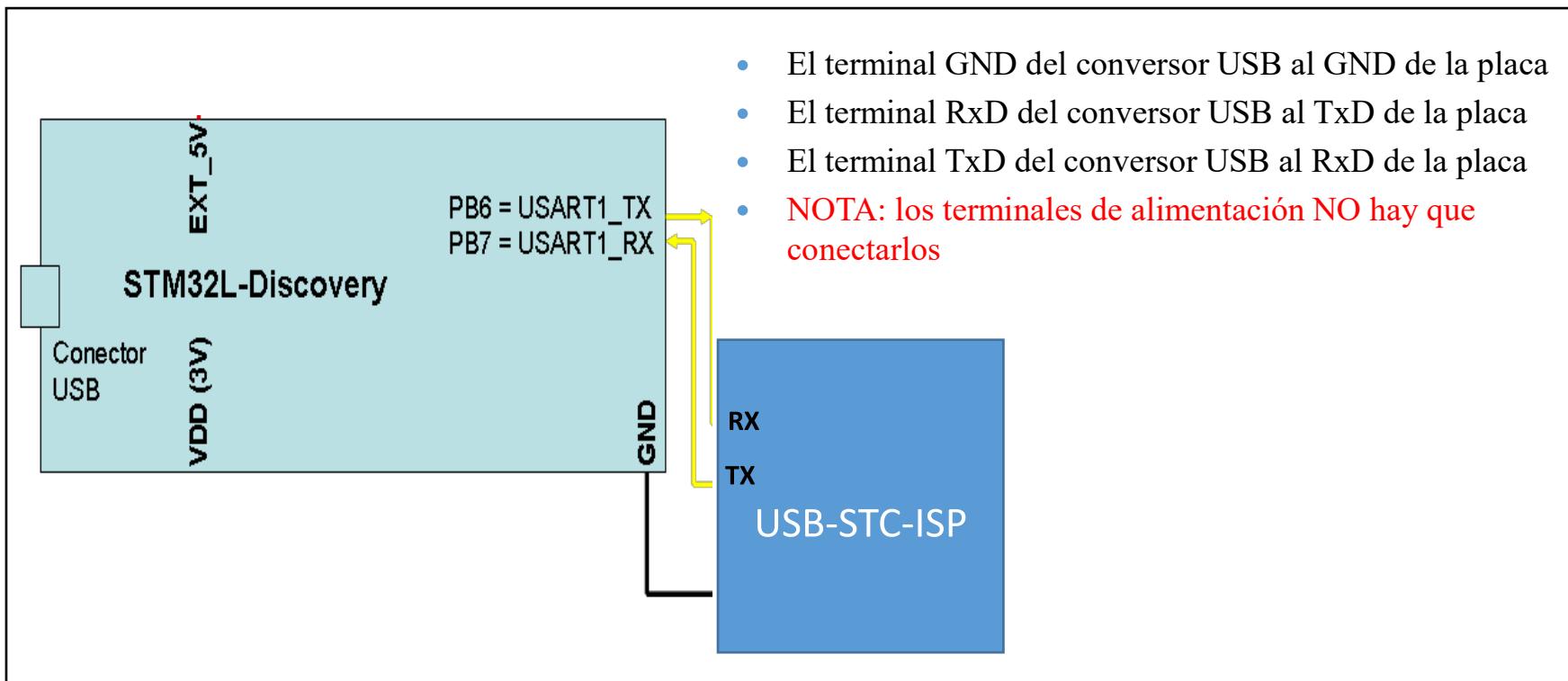
¿Cómo utilizar y probar la USART?

- La mayoría de los ordenadores actuales no poseen puertos serie (COM), sino solamente puertos USB
 - La opción más sencilla es utilizar un módulo de conversión TTL-USB
 - Por ejemplo, módulos basados en el PL-2303 como el USB-STC-ISP de Prolific).
- [https://www.amazon.es/s/ref=nb_sb_noss?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&url=search-alias%3Daps&field-keywords=ttl-usb+converter+pl2303&rh=i%3Aaps%2Ck%3Attl-usb+converter+pl2303 \)](https://www.amazon.es/s/ref=nb_sb_noss?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&url=search-alias%3Daps&field-keywords=ttl-usb+converter+pl2303&rh=i%3Aaps%2Ck%3Attl-usb+converter+pl2303)

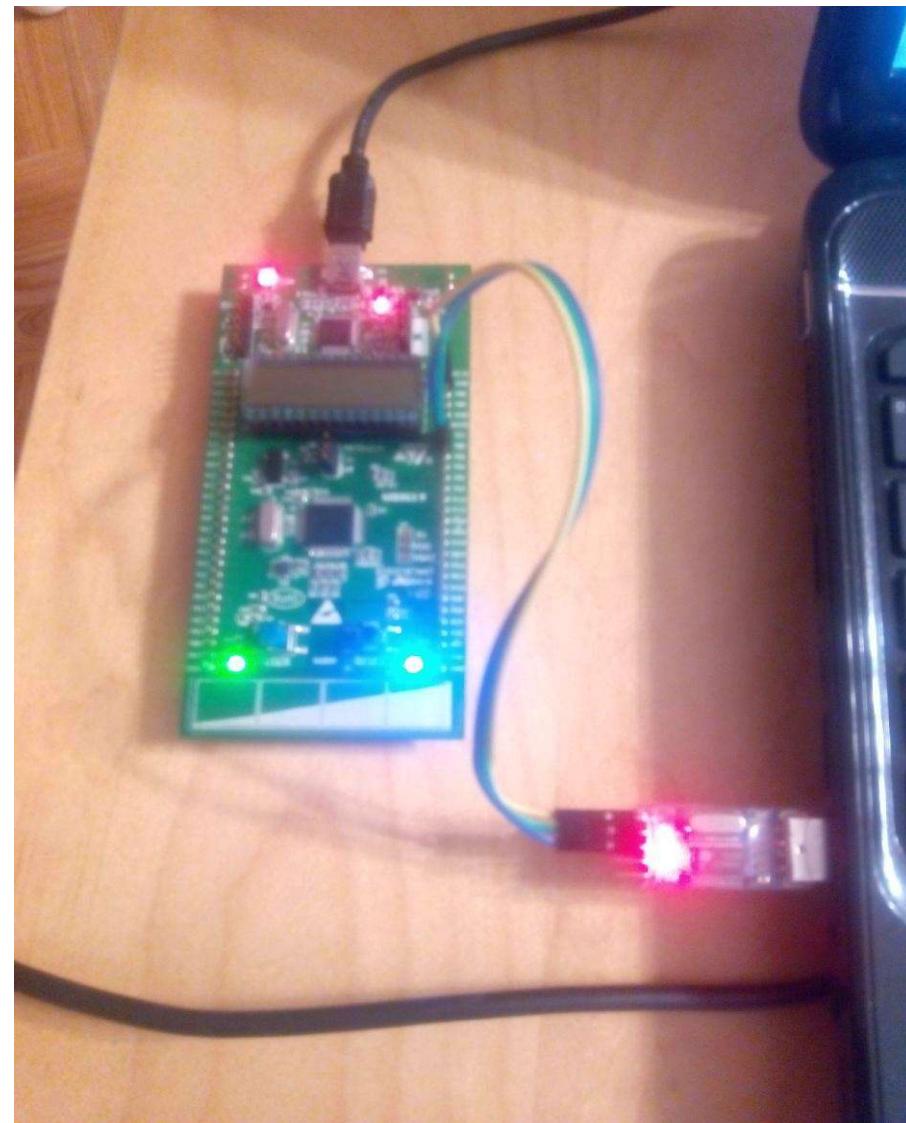
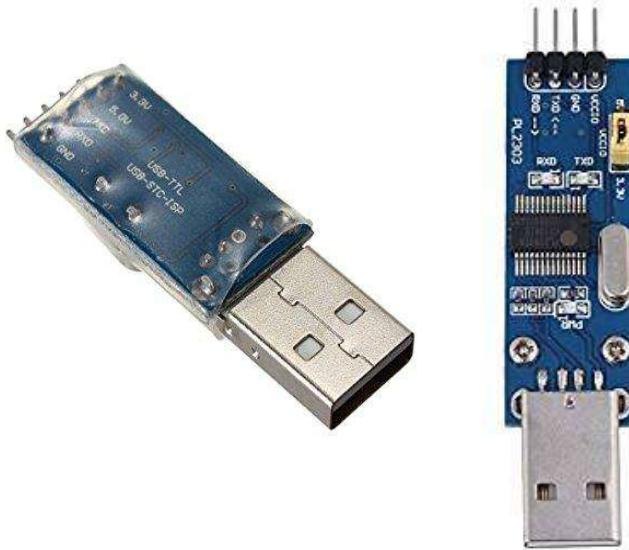


¿Cómo utilizar y probar la USART?

- En este caso, se conectan los pines de la USART al transductor TTL-USB
 - Como la placa STM32L-Discovery tiene disponibles los pines PB6 y PB7, se hace la conexión como se indica en la figura



¿Cómo utilizar y probar la USART?

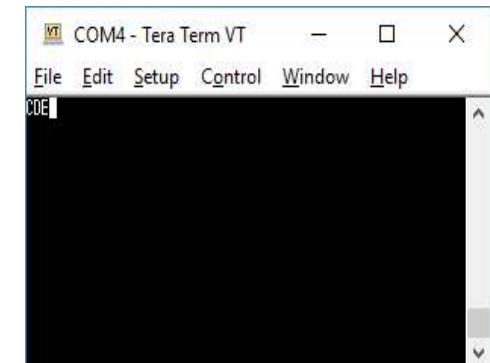
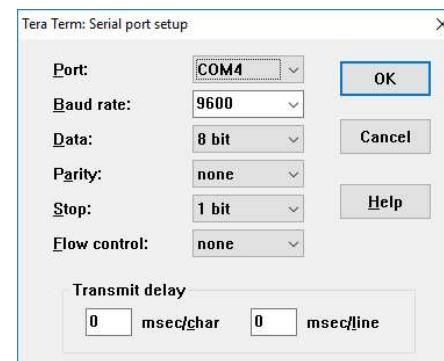
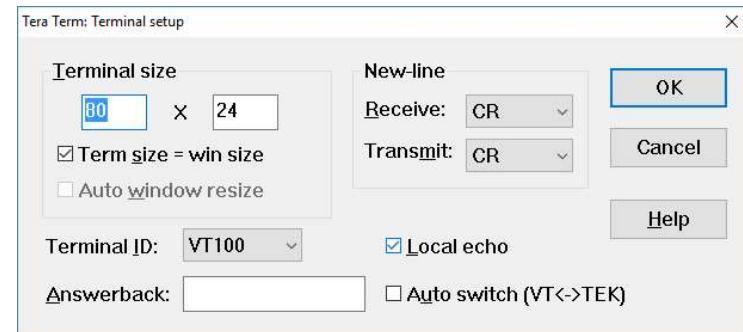
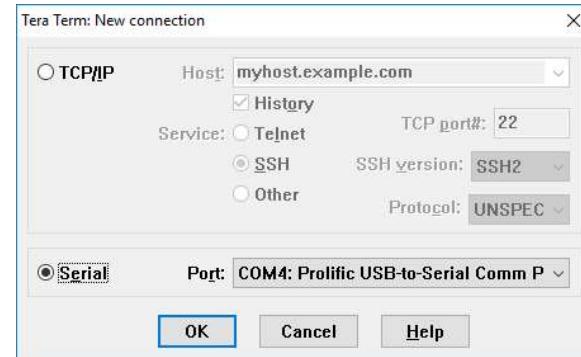


¿Cómo utilizar y probar la USART?

- Además casos será necesario que se instalen unos drivers del conversor en el sistema operativo.
 - En algunos casos dichos drivers se instalan automáticamente, en otros casos hay que buscar el driver en la red e instalarlo a mano.
 - En Aula Global están los drivers para Windows, tanto en su versión de 32-bits como de 64-bits.
- El driver convierte la conexión USB en un puerto serie, y entonces puede ser utilizado por cualquier programa de conexión serie, como por ejemplo el Hyperterminal de Windows (que desde Windows 7 NO se proporciona directamente con el sistema operativo) o el Tera Term, que tambien está en Aula global o la siguiente dirección: <https://osdn.net/projects/ttssh2/releases/>

¿Cómo utilizar y probar la USART?: Tera Term

- Para el caso del Tera Term, estos son los pasos para configurarlo una vez instalado y arrancado:
 - Arranca el programa y en la ventana que aparece “New Connection”, elige el puerto serie asociado al conversor
 - Elije ahora “Setup -> Terminal” y activa la opción “Local Echo” para ver lo que escribes en el programa
 - Elije ahora la opción “Setup -> Serial Port” y ajusta lo mostrado en la imagen
- A partir de ahora, lo que envía la placa lo verás en la pantalla del Tera Term
- Lo que escribas en la pantalla del Tera Term se enviará a la placa
- Y recuerda que lo que se ve y se escribe en la pantalla son caracteres ASCII



4 - Ejemplos

Ejemplo 1: Tx por Espera Activa

- Se transmite un carácter cada vez, tras la pulsación del botón USER, empezando por la ‘A’ e incrementando en una unidad.
- La comunicación es a 9600,8,N,1
 - Inicialización:

```
/* USER CODE BEGIN 1 */

    uint8_t valor = 'A';    / Variable global necesaria

/* USER CODE END 1 */
```

```
/* USER CODE BEGIN 2 */

// Inicialización del botón USER
BSP_PB_Init(BUTTON_USER, BUTTON_MODE_GPIO);

// Funciones de inicialización del LCD
BSP_LCD_GLASS_Init();
BSP_LCD_GLASS_BarLevelConfig(0);
BSP_LCD_GLASS_Clear();

// Nada más arrancar sale PULSA en el display
BSP_LCD_GLASS_DisplayString((uint8_t *)"PULSA");

/* USER CODE END 2 */
```

Ejemplo 1: Tx por Espera Activa

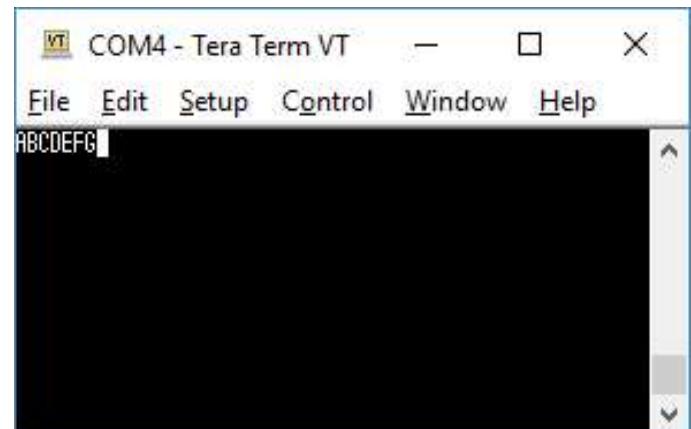
- Funcionamiento continuo:

```
/* USER CODE BEGIN WHILE */
while (1)
{
    if (BSP_PB_GetState(BUTTON_USER) == 1) {
        // Si pulso el botón

        // Transmiso el valor con estos parámetros:
        // * handler: &huart1
        // * Variable de donde saco lo que transmiso: valor
        // * Tamaño del bufer: 1:
        // * Timeout:10000
        HAL_UART_Transmit(&huart1, valor, 1, 10000);

        valor++;          // Aumento el valor y espero un poco
        espera(10000);

        // Espero a soltar el botón
        while (BSP_PB_GetState(BUTTON_USER)==1);
    }
/* USER CODE END WHILE */
```



Ejemplo 2: Tx y Rx por Espera Activa

- Se espera la recepción de un carácter enviado desde Teraterm, se incrementa en una unidad y se transmite ese nuevo valor al TeraTerm para sacarlo por su consola.
- En el ejemplo se envían los caracteres “a”, “b” y “c” varias veces y se recibe “b”, “c” y “d”
 - Inicialización

```
/* USER CODE BEGIN 1 */

    uint8_t texto[6]; // Variable global necesaria

/* USER CODE END 1 */
```

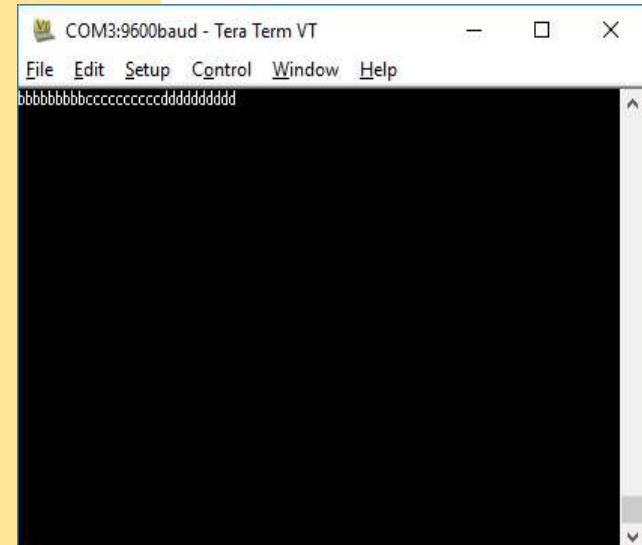
Ejemplo 2: Tx y Rx por Espera Activa

- Funcionamiento continuo:

```
/* USER CODE BEGIN WHILE */
while (1)
{
    // Si he enviado algo, recibo el valor con estos parámetros:
    // * handler: &huart1
    // * Variable donde guardo lo recibido: texto[6]
    // * Tamaño del bufer: 1:
    // * Timeout:10000
    HAL_UART_Receive(&huart1, texto, 1, 10000);

    // Aumento en uno la variable
    texto[0]++;

    // Y lo vuelvo a enviar con estos parámetros:
    // * handler: &huart1
    // * Variable a transmitir: texto[6]
    // * Tamaño del bufer: 1:
    // * Timeout:10000
    HAL_UART_Transmit(&huart1, texto, 1, 10000);
}
/* USER CODE END WHILE */
```



UART_HandleTypeDef huart1;

Esta variable la crea CubeMX al inicializar
la UART

Ejemplo 3: Rx por IRQ y Tx por Espera Activa

- El mismo ejemplo pero con IRQs en la Recepción

- Inicialización:

```
/* USER CODE BEGIN PV */  
  
uint8_t texto[6] = "        "; // Variable global necesaria  
  
/* USER CODE END PV */
```

```
/* USER CODE BEGIN 2 */  
  
//Inicialización de RECV para la INT  
HAL_UART_Receive_IT(&huart1, texto, 1);  
  
/* USER CODE END 2 */
```

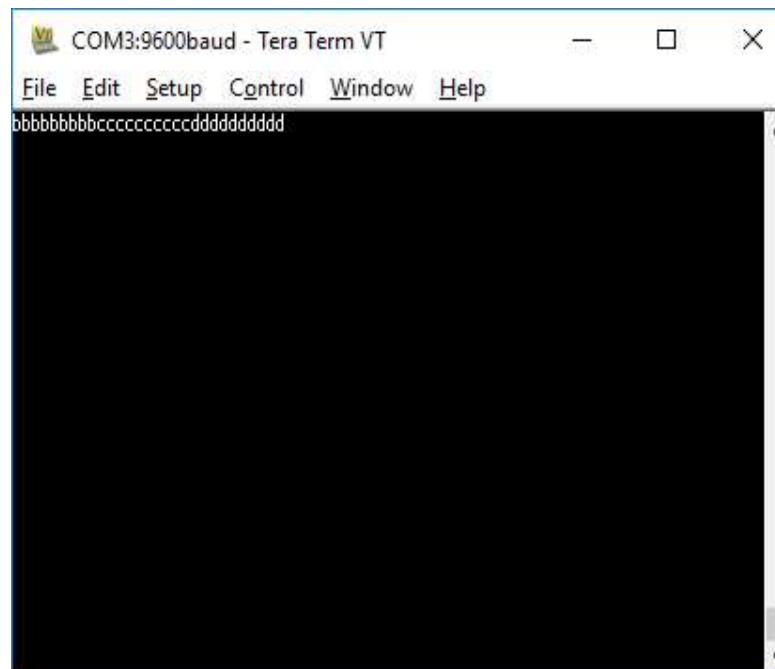
- Funcionamiento continuo:

```
/* USER CODE BEGIN WHILE */  
while (1)  
{  
    while (texto[0]==0);           // Si no he recibido nada, espero  
    texto[0] = texto[0]++;         // Si he recibido algo, aumento la variable  
    HAL_UART_Transmit(&huart1, texto, 1, 10000); // Envío el valor con los parámetros  
    texto[0]=0;                  // Borro la variable para la siguiente vez  
}  
/* USER CODE END WHILE */
```

Ejemplo 3: Rx por IRQ y Tx por Espera Activa

- El mismo ejemplo pero con IRQs en la Recepción
 - Interrupción:

```
/* USER CODE BEGIN 4 */  
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)  
{  
    HAL_UART_Receive_IT(huart, texto, 1); // Vuelve a iniciar Rx por haber acabado el buffer  
}  
/* USER CODE END 4 */
```



STM32 CubeMX: IRQ en USART

The screenshot shows the STM32 CubeMX interface with two callout boxes:

- A green callout box on the right points to the "NVIC Settings" tab in the top navigation bar of the "Configuration" window. It contains the text: "Activa la IRQ en la pestaña NVIC Settings".
- A green callout box on the left points to the "NVIC" category in the "System Core" section of the "Pinout & Configuration" window. It contains the text: "Se puede comprobar la activación en la tabla de NVIC en System Core".

Configuration Window (Top Left):

Enabled	Preemption Priority	Sub Priority
USART1 global interrupt	0	0

Pinout & Configuration Window (Bottom Left):

Categories: A-Z

- System Core
 - DMA
 - GPIO
 - IWDG
 - NVIC**
 - RCC
 - SYS
 - TS
 - WWDG
- Analog
- Timers
- Connectivity
 - I2C1
 - I2C2

NVIC Interrupt Table (Bottom Right):

NVIC Interrupt Table	Enabled	Preemption Priority
Non maskable interrupt	<input checked="" type="checkbox"/>	0
Hard fault interrupt	<input checked="" type="checkbox"/>	0
Memory management fault	<input checked="" type="checkbox"/>	0
Pre-fetch fault, memory access fault	<input checked="" type="checkbox"/>	0
Undefined instruction or illegal state	<input checked="" type="checkbox"/>	0
System service call via SWI instruction	<input checked="" type="checkbox"/>	0
Debug monitor	<input checked="" type="checkbox"/>	0
Pendable request for system service	<input checked="" type="checkbox"/>	0
Time base: System tick timer	<input checked="" type="checkbox"/>	0
Flash global interrupt	<input type="checkbox"/>	0
RCC global interrupt	<input type="checkbox"/>	0
ADC global interrupt	<input type="checkbox"/>	0
LCD interrupt	<input type="checkbox"/>	0
USART1 global interrupt	<input checked="" type="checkbox"/>	0

Notas sobre el uso de Tx por IRQ

- Las IRQs saltan cuando el flag correspondiente se activa
- El flag de Tx de la USART siempre está activo, por lo que saltaría constantemente la IRQ
- Ese flag sólo se desactiva cuando está transmitiendo, y se vuelve a poner cuando está listo para transmitir más
- Por tanto, las IRQs en la Tx sólo se utilizan para enviar grandes tramas de datos (bastantes bytes), y en ese caso el procedimiento sería:
 - Definir de forma global una variable que sea capaz de albergar la mayor trama de datos a definir
 - En el programa principal, llenar dicha variable con los datos a transmitir
 - Puede también tenerse que compartir una variable que indique la longitud de dichos datos
 - Activar las interrupciones por Tx
 - Por su parte la RAI debe:
 - Coger el byte que toca de dicho mensaje y transmitirlo
 - Incrementar el contador de bytes
 - Y si el contador de bytes llega al valor de la longitud de los datos a transmitir, inhibir la IRQ.
- Al gestionar las interrupciones por Tx con las HAL, estas precauciones pueden no tener aplicación, ya que la HAL puede resolver todos estos conflictos

5 - Ejercicios Propuestos

Ejercicios Propuestos

1. **Análisis de los Ejemplos:** Realice el diagrama de flujo de los ejemplos, cree los proyectos y al escribir el código comente lo que hace cada línea (a nivel funcional). Ejecútelos y depúrelos.
2. Cree un programa que mande una cuenta ascendente (1, 2, 3....) por el puerto serie y que cada número se envíe cada (timer con interrupciones + UART en Tx en espera activa).
3. Cree un programa que cuente los segundos que pasan entre recepción y recepción. El programa Teraterm manda caracteres, el microcontrolador los recibe y cuenta los segundos hasta el siguiente carácter recibido y lo muestra por el LCD (UART en Rx con interrupciones + Timer con interrupciones).
4. Cree un programa completo que genere señales cuadradas de duty cycle del 50%, en el que la frecuencia venga dada por puerto serie, y el periodo medido se envíe por puerto serie cada dos segundos.
 - Considere que la frecuencia la da en Hz con 3 caracteres (siempre tres), seguido de un retorno de carro.
 - Envíe el periodo en milisegundos.
5. Cree un programa que, a través del puerto serie, se pida comprobar cada una de las funcionalidades de la placa de la que dispone.
 - Para eso, considere juntar y adaptar todos los ejercicios realizados anteriormente durante el curso.

6 - Uso de la USART a nivel de Registros

Transmisión

- Para poder transmitir, el bit TE en $\text{USARTx} \rightarrow \text{CR1}$ debe estar a 1.
 - En ese caso, cada vez que se envíe una palabra, se irán poniendo los valores correspondientes de cada bit en el pin Tx, atendiendo a la configuración de reloj realizada.
- Procedimiento:
 1. Escribe el valor del bit **M** y del bit **PCE** para definir el tamaño de palabra y el uso de paridad, en $\text{USARTx} \rightarrow \text{CR1}$
 2. Pon a 1 el bit **TE** en $\text{USARTx} \rightarrow \text{CR1}$
 3. Decide el número de bits de parada en $\text{USARTx} \rightarrow \text{CR2}$
 4. Selecciona el baudrate en $\text{USARTx} \rightarrow \text{BRR}$
 5. Habilita la USART poniendo **UE=1** en $\text{USARTx} \rightarrow \text{CR1}$
 6. Espera a que el buffer no esté lleno (**TXE=1** en $\text{USARTx} \rightarrow \text{SR}$)
 7. Escribe el dato a enviar en $\text{USARTx} \rightarrow \text{DR}$
 8. Repite los pasos 6 y 7 hasta haber terminado de enviar todos los caracteres.
 9. Espera hasta que **TC=1** en $\text{USARTx} \rightarrow \text{SR}$ para asegurarse que toda la transmisión ha finalizado.
- Si se habilitan interrupciones por Tx, la RAI saltará siempre que **TXE=1** en $\text{USARTx} \rightarrow \text{SR}$

Recepción

- La USART, una vez habilitada la recepción, detecta la llegada de palabras de 8 o 9 bits (dependiendo del valor de M), con o sin paridad (bit PCE) de forma totalmente automática.
- Procedimiento:
 1. Programa el valor deseado de **M** y **PCE**, para definir el tamaño de palabra y el uso de paridad en **USARTx→CR1**
 2. Habilita la recepción poniendo **RE=1** en **USARTx→CR1**
 3. Programa el número de bits de parada deseado en **USARTx→CR2**
 4. Selecciona el baudrate deseado en **USARTx→BRR**
 5. Habilita la USART poniendo **UE=1** en **USARTx→CR1**
 6. Cuando se recibe un carácter:
 1. La USART pone **RXNE = 1** en **USARTx→SR** indicando que ha llegado algo
 - Si las IRQs están habilitadas, saltará la RAI
 - Si se ha detectado algún tipo de error, se activarán los flags correspondientes
 2. Lee el dato recibido en **USARTx→DR**
 - Esto pone **RXNE=0** en **USARTx→SR** de forma automática
 - Si no se lee el dato antes de que llegue el siguiente carácter, se detectará un error de overrun

Generación de Baudrates

- El baudrate es común para la recepción que para la transmisión
- La ecuación que relaciona el baudrate es:

$$\text{Tx/Rx baud} = \frac{f_{\text{CK}}}{8 \times (2 - \text{OVER8}) \times \text{USARTDIV}}$$

- Si OVER8=0, la parte fraccional se codifica con 4 bits y programado por DIV_fraction[3:0] en el USARTx→BRR
- Si OVER8=1, entonces se hace con 3 bits: DIV_fraction[2:0] en USARTx→BRR
- Ejemplo:
 - Con OVER8=0 y USARTDIV = 25.62
 - Parte fraccionaria:
 - $0.62 * 16 = 9.92$
 - El entero más cercano = 10
 - DIV_Fraction = 0x0A
 - Parte entera (mantisa):
 - DIV_Mantissa = 25 = 0x19
 - USARTx→BRR = 0x19A
 - Con OVER8=1 es igual pero multiplicando por 8

Generación de Baudrates

Table 90. Error calculation for programmed baud rates at $f_{PCLK} = 16\text{ MHz}$ or $f_{PCLK} = 32\text{ MHz}$), oversampling by 16⁽¹⁾

Oversampling by 16 (OVER8 = 0)							
Baud rate		$f_{PCLK} = 16\text{ MHz}$			$f_{PCLK} = 32\text{ MHz}$		
S.No	Desired	Actual	Value programmed in the baud rate register	% Error = (Calculated - Desired)B.Rate / Desired B.Rate	Actual	Value programmed in the baud rate register	% Error
1	1.2 KBps	1.2 KBps	833.3125	0	1.2 KBps	1666.6875	0
2	2.4 KBps	2.4 KBps	416.6875	0	2.4 KBps	833.3125	0
3	9.6 KBps	9.598 KBps	104.1875	0.02	9.601 KBps	208.3125	0.01
4	19.2 KBps	19.208 KBps	52.0625	0.04	19.196 KBps	104.1875	0.02
5	38.4 KBps	38.369 KBps	26.0625	0.08	38.415 KBps	52.0625	0.04
6	57.6 KBps	57.554 KBps	17.375	0.08	57.554 KBps	34.75	0.08
7	115.2 KBps	115.108 KBps	8.6875	0.08	115.108 KBps	17.375	0.08
8	230.4 KBps	231.884 KBps	4.3125	0.64	230.216 KBps	8.6875	0.08
9	460.8 KBps	457.143 KBps	2.1875	0.79	463.768 KBps	4.3125	0.64
10	921.6 KBps	941.176 KBps	1.0625	2.12	914.286 KBps	2.1875	0.79
11	2 MBps	NA	NA	NA	2000 KBps	1	0
12	4 MBps	NA	NA	NA	NA	NA	NA

- The lower the CPU clock the lower the accuracy for a particular baud rate. The upper limit of the achievable baud rate can be fixed with these data.

Generación de Baudrates

Table 91. Error calculation for programmed baud rates at $f_{PCLK} = 16\text{ MHz}$ or $f_{PCLK} = 32\text{ MHz}$), oversampling by 8⁽¹⁾

Oversampling by 8 (OVER8 = 1)							
Baud rate		$f_{PCLK} = 16\text{ MHz}$			$f_{PCLK} = 32\text{ MHz}$		
S.No	Desired	Actual	Value programmed in the baud rate register	% Error = (Calculated - Desired)B.Rate / Desired B.Rate	Actual	Value programmed in the baud rate register	% Error
1	1.2 KBps	1.2 KBps	1666.625	0	1.2 KBps	3333.375	0
2	2.4 KBps	2.4 KBps	833.375	0	2.4 KBps	1666.625	0
3	9.6 KBps	9.598 KBps	208.375	0.02	9.601 KBps	416.625	0.01
4	19.2 KBps	19.208 KBps	104.125	0.04	19.196 KBps	208.375	0.02
5	38.4 KBps	38.369 KBps	52.125	0.08	38.415 KBps	104.125	0.04
6	57.6 KBps	57.554 KBps	34.75	0.08	57.554 KBps	69.5	0.08
7	115.2 KBps	115.108 KBps	17.375	0.08	115.108 KBps	34.75	0.08
8	230.4 KBps	231.884 KBps	8.625	0.64	230.216 KBps	17.375	0.08
9	460.8 KBps	457.143 KBps	4.375	0.79	463.768 KBps	8.625	0.64
10	921.6 KBps	941.176 KBps	2.125	2.12	914.286 KBps	4.375	0.79
11	2 MBps	2000 KBps	1	0	2000 KBps	2	0
12	4 MBps	NA	NA	NA	4000 KBps	1	0

- The lower the CPU clock the lower the accuracy for a particular baud rate. The upper limit of the achievable baud rate can be fixed with these data.

USART: Registros de Control

- **USARTx→CR1** – Control Register 1:

- **OVER8** – Modo de sobremuestreo:
 - 0 – Oversampling por 16; 1 – Oversampling por 8
- **UE** – Habilitación de la USART
- **M** – Tamaño de Palabra:
 - 0 – 8 bits; 1 – 9 bits
- **WAKE**: *no se utiliza*
- **PCE**: Control de paridad
 - 0 – deshabilitado; 1 – habilitado
- **PS**: Selección de paridad
 - 0 – paridad par; 1 – paridad impar
- **PEIE**: *no se utiliza*
- **TXEIE**: Habilitación de IRQ por buffer de transmisión vacío
- **TCIE**: Habilitación de IRQ por transmisión completada
- **RXNEIE**: Habilitación de IRQ por carácter recibido
- **IDLIE** : *no se utiliza*
- **TE**: Habilitación de transmisión
- **RE**: Habilitación de recepción
- **RWU, SBK**: *no se utiliza*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OVER8	Reserved	UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
rw	Res.	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

USART: Registros de Control

- **USARTx→CR2** – Control Register 2:

- LINEN: no se usa
 - STOP – números de bit de parada:
 - 00 – 1 bit; 01 – 0.5 bits; 10 – 2 bits; 11 – 1,5 bits
 - CLKEN, CPOL, CPHA, LBCL, LBDIE, LBDL, ADD: no se usan

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	LINEN	STOP[1:0]		CLKEN	CPOL	CPHA	LBCL	Res.	LBDIE	LBDL	Res.	ADD[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

- **USARTx→CR3** – Control Register 3:

- Pensado para controlar las funciones adicionales de la USART (DMA, tarjetas inteligentes, IrDA, etc.)
 - *No se usa*

- **USARTx→GTPR** – Guard time and prescaler Register:

- No se usa

USART: Registros de Control

- **USARTx→BRR** – Baud Rate Register:

- Registro de 32 bits, donde sólo se usan los 16 menos significativos:
 - USARTx→BRR[15:4] – DIV_Mantissa
 - USARTx→BRR[3:0] – DIV_Fraction

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

USART: Registros de Datos

- **USARTx→DR** – Data Register

- Es un registro de 32 bits, pero del cual sólo se usan los 8 bits menos significativos
- Internamente son dos registros, uno de recepción y otro de transmisión, pero a nivel de software es uno sólo, en el que si se escribe, se transmite, y si se lee, se lee el último byte recibido.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								DR[8:0]							
								rw	rw	rw	rw	rw	rw	rw	rw

USART: Registros de Estado

- **USARTx→SR – Status Register:**

- Registro de 32 bits, que contiene los 10 flags de eventos:
 - *CTS y LBD: no se van a utilizar*
 - **TXE: Transmit Data Register Empty.**
 - Se limpia al escribir en USARTx→ DR
 - **TC: Transmission Complete**
 - Se limpia con la secuencia: 1.- Lee el USARTx→ SR, 2.- Escribe en USARTx→ DR
 - **RXNE: Read data register Not Empty**
 - Se limpia leyendo el USARTx→ DR
 - *IDLE: no se va a utilizar*
 - **ORE: Overrun error.**
 - Se limpia por secuencia.
 - *NF, FE, PE: no se van a utilizar.*

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE		
				rc_w0	rc_w0	r	rc_w0	rc_w0	r	r	r	r	r		

7 - Ejemplos a nivel de Registros

Ejemplos a nivel de registros

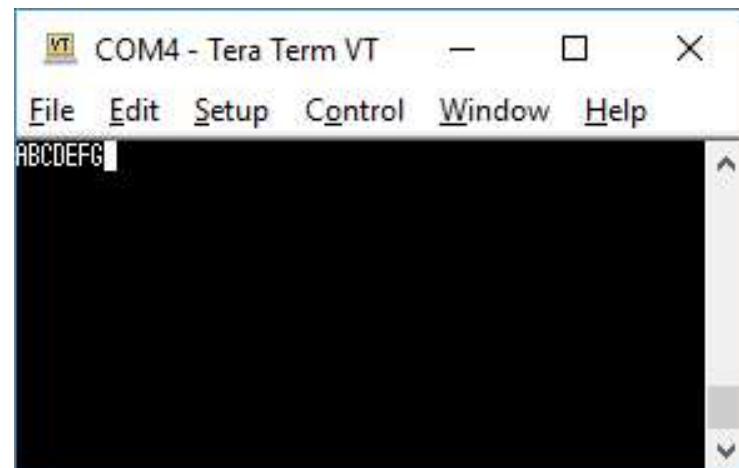
- A continuación se muestran los mismos ejemplos que los mostrados con HAL, pero ahora configurados con registros.
- Las configuraciones en CUBE siguen siendo válidas para los ejemplos 1 y 2, pero no para el ejemplo 3, en el que hay que hacer el cambio indicado en la transparencia 47.
- Los programa para µVision son los mostrados a continuación, teniendo en cuenta colocar cada parte del código en su sitio, es decir:
 - Variables globales entre /* USER CODE BEGIN PV */ y /* USER CODE END PV */
 - Inicialización entre /* USER CODE BEGIN 2 */ y /* USER CODE END 2 */
 - Funcionamiento cíclico entre /* USER CODE BEGIN WHILE */ y /* USER CODE END WHILE */
 - Interrupción entre /* USER CODE BEGIN 4 */ y /* USER CODE END 4 */

Ejemplo 1: Tx por Espera Activa

- Se transmite un carácter cada vez, tras la pulsación del botón USER, empezando por la 'A' e incrementando en una unidad.
- La comunicación es a 9600,8,N,1

```
int main(void){  
    unsigned char valor = 'A';  
  
    // Funciones de inicialización del LCD  
    BSP_LCD_GLASS_Init();  
    BSP_LCD_GLASS_BarLevelConfig(0);  
    BSP_LCD_GLASS_Clear();  
  
    // PA0 como entrada (00)  
    GPIOA->MODER &= ~(1 << (0*2 +1));  
    GPIOA->MODER &= ~(1 << (0*2));  
  
    // PB6 para la USART  
    GPIOB->MODER |= (0x01 << (2*6+1)); // AF => (10)  
    GPIOB->MODER &= ~(0x01 << (2*6));  
    GPIOB->AFR[0] &= 0xF0FFFFFF;// AF para UART => 7  
    GPIOB->AFR[0] |= 0x07000000;  
  
    // Configuración de la USART  
    USART1->CR1 = 0x00000008; // UE = 0 , 8 bits, sin  
                                // paridad, OVR8 = 16,  
                                // sin INT, TE = 1  
    USART1->CR2 = 0x00000000; // 1 bit de parada (00)  
    USART1->BRR = 0x00000D05; // 208 = D0h  
                            // 0,31*16 = 4,96 = 5h  
    USART1->CR1 |= 0x01 << 13; // UE = 1  
  
    // Saco el mensaje inicial  
    BSP_LCD_GLASS_Clear();  
    BSP_LCD_GLASS_DisplayString((uint8_t *)"PULSA");
```

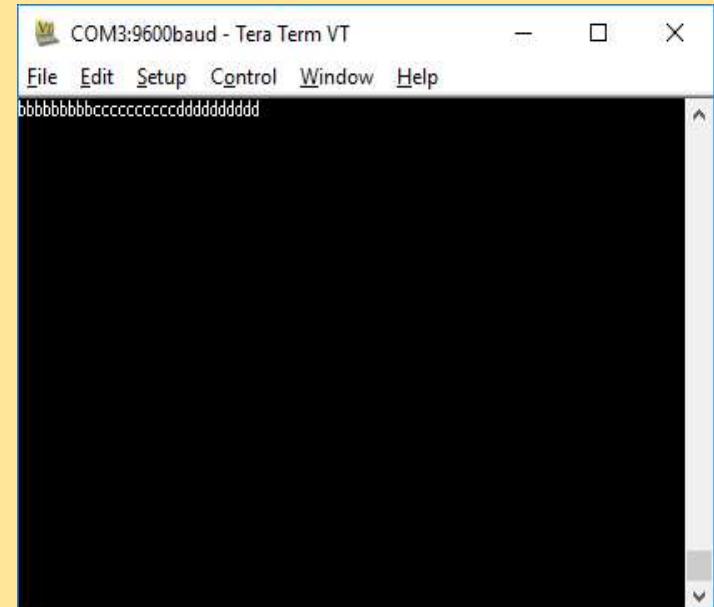
```
while (1) {  
  
    if ((GPIOA->IDR&0x00000001)!=0) {  
        // Si se pulsa el botón  
  
        while ((GPIOA->IDR&0x00000001)!=0){  
            espera(70000); // Antirebotes  
        }  
        // Si no puedo enviar, espero  
        // Espero hasta que el bit TXE = 1  
        while ((USART1->SR & 0x0080)== 0);  
  
        // Si ya puedo enviar, pongo el valor en DR  
        USART1->DR = valor;  
        valor++; // Y aumento "valor" en uno  
    }  
}
```



Ejemplo 2: Tx y Rx por Espera Activa

- Se espera la recepción de un carácter enviado desde Teraterm, se incrementa en una unidad y se transmite ese nuevo valor al TeraTerm para sacarlo por su consola
- En el ejemplo se envían los caracteres “a”, “b” y “c” varias veces y se recibe “b”, “c” y “d”

```
int main(void){  
  
    unsigned char valor = 'A';  
  
    // PB6 y PB7 para la USART  
    GPIOB->MODER |= (0x01 << (2*7+1)); // AF => (10)  
    GPIOB->MODER &= ~(0x01 << (2*7));  
    GPIOB->MODER |= (0x01 << (2*6+1)); // AF => (10)  
    GPIOB->MODER &= ~(0x01 << (2*6));  
    GPIOB->AFR[0] &= 0x00FFFFFF; // AF para UART => 7  
    GPIOB->AFR[0] |= 0x77000000;  
  
    // Configuración de la USART  
    USART1->CR1 = 0x0000000C; // UE = 0 , 8 bits, sin  
                            // paridad, OVR8 = 16,  
                            // sin INT, TE = 1, RE = 1  
    USART1->CR2 = 0x00000000; // 1 bit de parada (00)  
    USART1->BRR = 0x00000D05; // 208 = D0h  
                            // 0,31*16 = 4,96 = 5h  
    USART1->CR1 |= 0x01 << 13; // UE = 1  
  
    while (1) {  
        while ((USART1->SR & 0x0020)==0); // Espero a recibir algo, o sea espero a que el RXNE = 1  
        valor = USART1->DR;                // Cojo el valor y lo guardo en la variable "valor"  
        valor++;                          // Aumento "valor" en uno  
        while ((USART1->SR & 0x0080)== 0); // Espero hasta que el bit TXE = 1 y pueda enviar  
        USART1->DR = valor;              // Y lo envío  
    }  
}
```



Ejemplo 3: Rx por IRQ y Tx por Espera Activa

- El mismo ejemplo pero con IRQs en la Recepción

```
unsigned char valor = 'A';

void USART1_IRQHandler(void) {
    // Si recibo algo, lo guardo en
    // "valor" y aumento su valor
    if ((USART1->SR & 0x0020) !=0) {
        valor = USART1->DR;
        valor++;
    }
}

int main(void){
    // PB6 y PB7 para la USART
    GPIOB->MODER |= (0x01 << (2*7+1)); // AF => (10)
    GPIOB->MODER &= ~(0x01 << (2*7));
    GPIOB->MODER |= (0x01 << (2*6+1)); // AF => (10)
    GPIOB->MODER &= ~(0x01 << (2*6));
    GPIOB->AFR[0] &= 0x00FFFFFF;           // AF para UART => 7
    GPIOB->AFR[0] |= 0x77000000;

    // Configuración de la USART
    USART1->CR1 = 0x0000000C; // UE = 0 , 8 bits, sin
                                // paridad, OVR8 = 16,
                                // sin INT, TE = 1, RE = 1
    USART1->CR2 = 0x00000000; // 1 bit de parada (00)
    USART1->BRR = 0x00000D05; // 208 = D0h
                                // 0,31*16 = 4,96 = 5h
    USART1->CR1 |= 0x01 << 13; // UE = 1
    USART1->CR1 |= 0x01 << 5; // Habilitamos la interrupción para Rx (RXNEIE= 1)

    NVIC->ISER[1] |= (1 << (37-32)); // Habilito la INT de la USART en el NVIC
```

Ejemplo 3: Rx por IRQ y Tx por Espera Activa

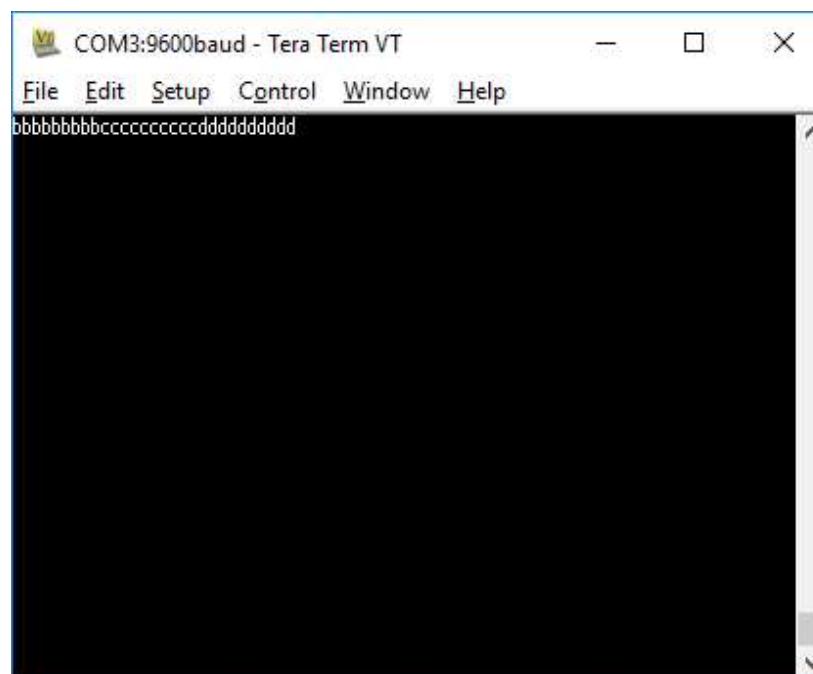
- El mismo ejemplo pero con IRQs en la Recepción

```
while (1) {
    while (valor==0);                                // Si no he recibido nada, espero

    while ((USART1->SR & 0x0080)== 0); // Espero hasta que el bit TXE = 1 y pueda enviar

    USART1->DR = valor;                            // Si puedo, envío el valor
    valor=0;                                         // Y limpio el valor para la siguiente vez

}
}
```



STM32 CubeMX: IRQ en USART con registros

- CUIDADO: En este caso, a diferencia del ejemplo mostrado con HAL, hay que desactivar la IRQ en CUBE. Si no, aparece un fallo de compilación por duplicación de ajuste

The screenshot shows the STM32 CubeMX interface with several windows open:

- Configuration** window (top left): Shows options like DMA Settings, GPIO Settings, User Constants, and NVIC Settings.
- Pinout & Configuration** window (center top): Shows the System Core section with categories: DMA, GPIO, IWDG, **NVIC**, RCC, SYS, TS, and WWDG.
- Clock Configuration** window (center middle): Shows the Clock Configuration tab.
- Project Manager** window (center right): Shows the Project Manager tab.
- NVIC Interrupt Table** window (bottom right): Shows the NVIC Mode and Configuration tab with a table of interrupts and their settings.

Annotations in green boxes:

- A green box labeled **Desactiva la IRQ en la pestaña NVIC Settings** has an arrow pointing to the **NVIC Settings** option in the Configuration window.
- A green box labeled **Se puede comprobar la desactivación en la tabla de NVIC en System Core** has an arrow pointing to the **NVIC** category in the Pinout & Configuration window.
- A green arrow points from the **Enabled** column in the NVIC Interrupt Table to the **Enabled** checkbox in the NVIC Settings window.

	Enabled	Preemption Priority
Non maskable interrupt	<input checked="" type="checkbox"/>	0
Hard fault interrupt	<input checked="" type="checkbox"/>	0
Memory management fault	<input checked="" type="checkbox"/>	0
Pre-fetch fault, memory access fault	<input checked="" type="checkbox"/>	0
Undefined instruction or illegal state	<input checked="" type="checkbox"/>	0
System service call via SWI instruction	<input checked="" type="checkbox"/>	0
Debug monitor	<input checked="" type="checkbox"/>	0
Pendable request for system service	<input checked="" type="checkbox"/>	0
Time base: System tick timer	<input checked="" type="checkbox"/>	0
Flash global interrupt	<input type="checkbox"/>	0
RCC global interrupt	<input type="checkbox"/>	0
ADC global interrupt	<input type="checkbox"/>	0
LCD interrupt	<input type="checkbox"/>	0
USART1 global interrupt	<input type="checkbox"/>	0