

4. Comunicación serial por Uart (Transmisor Receptor Asíncrono Universal).

Este es uno de los periféricos de mayor utilidad, pues permite la comunicación del microcontrolador con otro periférico o con un computador. La comunicación se llama serial porque la información va de bit a bit por un solo cable. Hay tres pines en una comunicación serial por Uart que son RX, TX y GND. El pin RX es para recibir datos, mientras que el pin TX es para transmitir. En algunas comunicaciones seriales usan otros pines para hacer la comunicación más eficiente, como el pin DTR (terminal de datos listo), o el pin CTS (listo para enviar). En una comunicación Uart no hay un pin de reloj, que va desde el aparato “maestro” hasta el aparato “esclavo”, como si lo hay en una comunicación síncrona, como el Usart. El reloj es necesario para transmisión y recepción porque se usa la transición de subida o bajada para enganchar el bit del dato. Cuando no hay un reloj común, cada parte debe tener su reloj local con una variación no mayor al 3% entre estos; porcentajes superiores hacen que la comunicación se pierda. El protocolo generalizado de una comunicación serial es como se muestra en la figura 39.

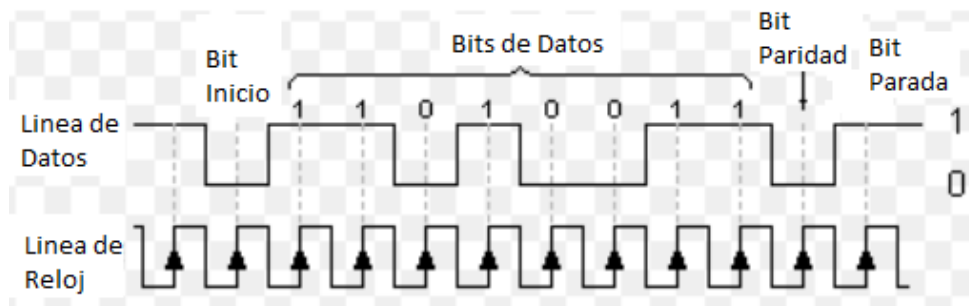


Figura 39. Protocolo de comunicaciones del Uart.

Cuando no hay datos que enviar, la línea de transmisión TX se mantiene en uno lógico. Un paquete de datos inicia con un bit en cero, luego vienen los bits de datos, que pueden ser 8 o 9, de acuerdo a la configuración, luego el bit de paridad, que puede ser par o impar o sin bit de paridad y que su valor depende de los bits del dato, y por último la indicación de parada, que puede ser un bit o dos bits y tiene el valor de uno lógico. La transición de la comunicación Uart se hace entre cero y la polaridad del microcontrolador, o sea, +3.3 voltios. Cuando se conecta con un módulo periférico cercano, el riesgo de perder información es mínimo si se hace a velocidades que no excedan los 115 mil bits por segundo. Cuando la comunicación se hace a distancias en el orden de varios metros, se usan cambios de nivel de acuerdo a los estándares RS232, RS422 o RS485. El estándar RS232 es también conocido como EIA/TIA RS-232C y lo que hace es variar los estados entre un voltaje positivo y uno negativo (NRZ- No retorno a cero), de esta forma se pueden evitar errores en la comunicación, más si se hiciera la transmisión entre un valor de voltaje y GND. La figura 40 muestra la salida del integrado MAX232 con el estándar RS232.

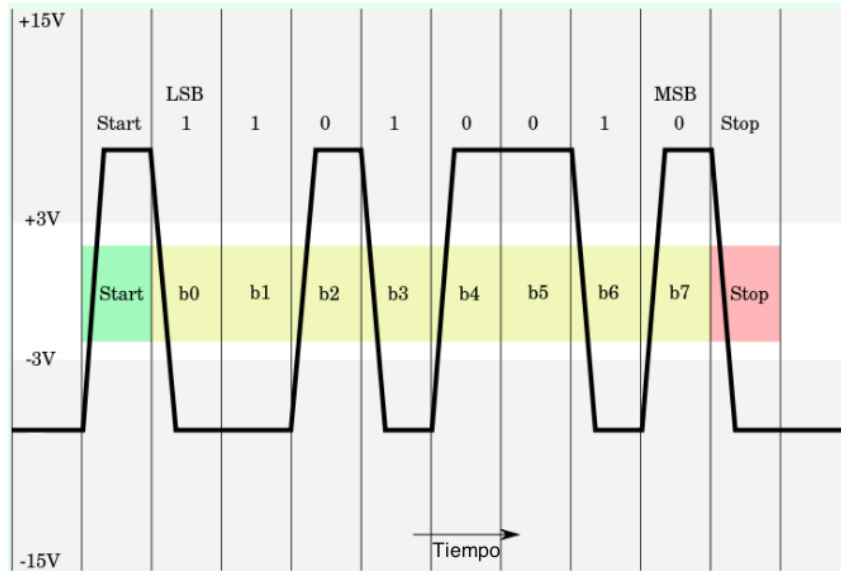


Figura 40. Salida de un convertidor Uart a RS232.

La diferencia entre los estándares está en la distancia y la velocidad de transmisión, además del tipo de cable. El estándar RS232 es para comunicaciones no mayores a 15 metros, velocidades no mayores a 20 Kilo bits por segundo (kbps) y usa cable normal. El estándar RS422 usa cable trenzado y se usa para distancias hasta de 12 metros con velocidad de transmisión entre 100 kbps hasta 10 Mbps y se puede conectar hasta con 10 equipos simultáneamente. El estándar RS485 usa cable par trenzado y terminales RJ11 que permite resistencia a interferencia electromagnética. Puede alcanzar transmisiones hasta de 1.200 metros y velocidades hasta de 10 Mbps cuando se hace con una distancia de 12 metros. Hay una relación velocidad- distancia donde a mayor distancia se debe transmitir a menor velocidad.

Cuando se hace la comunicación con el STM32F103C8T6 y un computador, se pueden monitorear las tareas que se están realizando en el programa, simplemente enviando esta información con una función del periférico serial. La forma que se comunica el microcontrolador es por Uart, pero la mayoría de los computadores de hoy no tienen ese puerto disponible si no USB. Por esto hay que usar un convertidor de Uart a USB. En la figura 41 se puede observar uno disponible comercialmente y como se conecta con la tarjeta azul. El pin A9 es TxD y se conecta al convertidor TTL a USB con RXD. El pin A10 es RxD y se conecta con el convertidor TTL a USB con TXD. Se debe conectar GND en la tarjeta del microcontrolador y el convertidor. Se debe conectar el pin 5V de la tarjeta azul con +5V del convertidor. Para la tarjeta negra, hacer la misma conexión de A9 y A10 a los mismos pines del convertidor, pero la tarjeta negra no dispone del pin 5V, entonces hay que conectarlo a +3V de la tarjeta y del convertidor. Solo un dispositivo provee de voltaje a la tarjeta, o sea, que si se va a programar con el STLink, ya sea este o el USB a TTL debe estar conectado a Vcc de la tarjeta, no dos al tiempo, pero si GND.

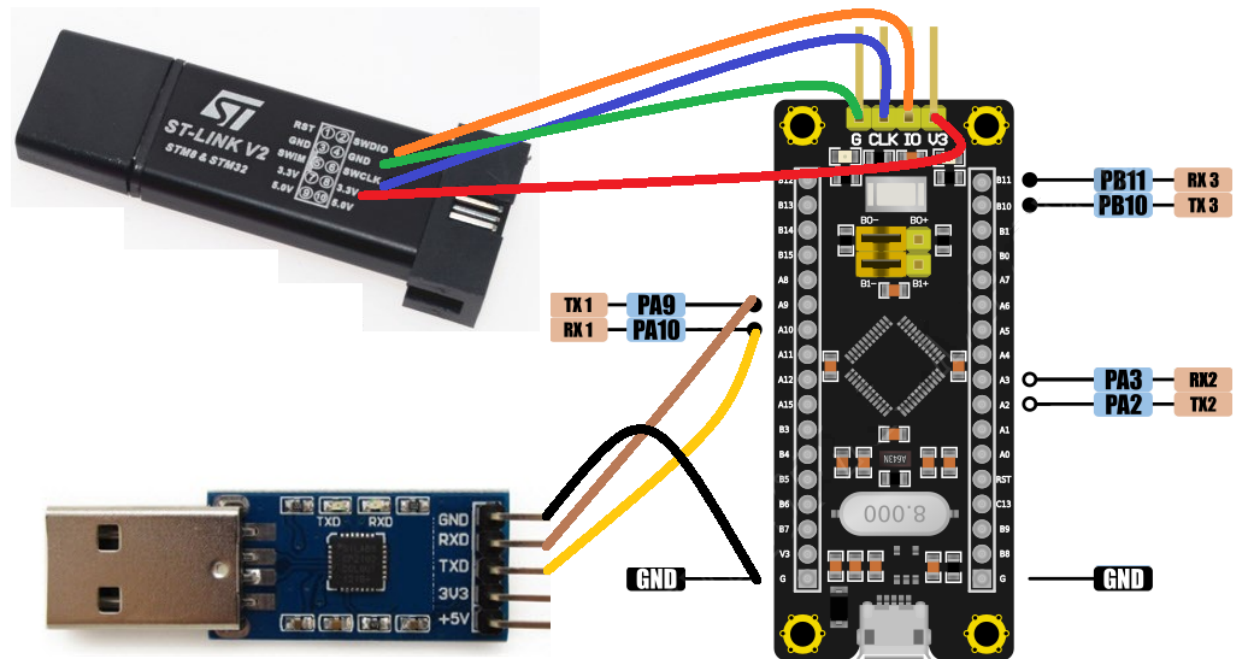


Figura 41. Conexión del convertidor TTL a USB y del Stlink V2 al STM32F103C8T6.

Los datos que llegan desde el microcontrolador hasta el computador deben ser recibidos e interpretados por un programa para PC. Hay varios programas para Windows que sirven para comunicarse con el microcontrolador. Entre estos está Hercules:

https://www.hw-group.com/products/hercules/index_es.html

Realterm:

<https://sourceforge.net/projects/realterm/>

Terminal:

<https://sites.google.com/site/terminalbpp/>

El microcontrolador STM32F103C8T6 tiene tres puertos seriales, con los primeros pines siendo TX:

- Serial1, conectado en los pines PA_9 y PA_10.
- Serial2, conectado en los pines PA_2 y PA_3.
- Serial3, conectado en los pines PB_10 y PB_11.

La configuración de un módulo serial comprende los siguientes pasos:

- Habilitar el reloj del puerto serial, del puerto Gpio donde está ubicado y de la función alterna.
- Configurar el modo de función alterna de cada pin del puerto serial.
- Habilitar el puerto serial.
- Configurar los parámetros del puerto serial.

Habilitación del reloj del puerto serial, del puerto Gpio donde está ubicado y de la función alterna.

Todo módulo tiene un reloj para su funcionamiento y esta es la forma de deshabilitarlo para reducir consumo si no se está usando. Esto se hace con el registro RCC_APB2ENR para el puerto serial Serial1 y RCC_APB1ENR para los puertos seriales Serial2 y Serial3. Por ejemplo, el puerto Serial1, usa los pines PA_9 y PA_10 (Tx y Rx, respectivamente), por lo que se debe también habilitar el reloj del puerto A y además, sus funciones alternas. La función disponible para esto es:

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1 | RCC_APB2Periph_AFIO |  
                        RCC_APB2Periph_GPIOA, ENABLE);
```

En el caso del puerto Serial2, en los pines PA_2 y PA_3:

```
RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);  
RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO | RCC_APB2Periph_GPIOA, ENABLE);
```

Y para el puerto Serial3, en los pines PB_10 y PB_11:

```
RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3, ENABLE);  
RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO | RCC_APB2Periph_GPIOB, ENABLE);
```

Configuración del modo de función alterna de cada pin del puerto serial.

Como algunos pines comparten sus salidas o entradas con varios módulos, es necesario indicarles cuál. Para el puerto Serial1 los pines de configuración son PA_9 para Tx y PA_10 para Rx. Para Tx, el pin debe ser de salida Push- Pull como función alterna:

```
GPIO_Struct.GPIO_Pin = GPIO_Pin_9;  
GPIO_Struct.GPIO_Speed = GPIO_Speed_50MHz;  
GPIO_Struct.GPIO_Mode = GPIO_Mode_AF_PP;  
GPIO_Init(GPIOA, &GPIO_Struct);
```

Para el pin Rx, que es de entrada, el pin se configura en entrada flotante:

```
GPIO_Struct.GPIO_Pin = GPIO_Pin_10;  
GPIO_Struct.GPIO_Speed = GPIO_Speed_50MHz;  
GPIO_Struct.GPIO_Mode = GPIO_Mode_IN_FLOATING;  
GPIO_Init(GPIOA, &GPIO_Struct);
```

Habilitación del puerto serial.

Ahora se habilita el puerto serial. Para los puertos seriales Serial2 y Serial3, se reemplaza USART1 por USART2 o USART3, respectivamente.

```
USART_Cmd(USART1, ENABLE);
```

Configuración de los parámetros del puerto serial.

Aquí se define la velocidad del puerto, los bits de parada, si tiene paridad y si es par o impar, los bits del dato, si es una comunicación full duplex y el tipo de control.

```
usart1_init_struct.USART_BaudRate = 9600;  
usart1_init_struct.USART_WordLength = USART_WordLength_8b;  
usart1_init_struct.USART_StopBits = USART_StopBits_1;  
usart1_init_struct.USART_Parity = USART_Parity_No ;  
usart1_init_struct.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;  
usart1_init_struct.USART_HardwareFlowControl = USART_HardwareFlowControl_None;  
USART_Init(USART1, &usart1_init_struct);
```

En resumen, se puede hacer una función de inicio con estos pasos:

```
/******  
 * Inicialización del puerto serial Serial1  
******/  
void UART1_Init(void)  
{  
    GPIO_InitTypeDef GPIO_Struct;  
    // Inicializa parámetros de Uart 9600 bps, 8 bits, 1 stop, no paridad  
    USART_InitTypeDef UART_Struct;  
    USART_StructInit(&UART_Struct);  
    UART_Struct.USART_BaudRate= 9600;  
  
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1 | RCC_APB2Periph_AFIO |  
                           RCC_APB2Periph_GPIOA, ENABLE);  
  
    // TX: GPIOA PIN9 funcion alterna  
    GPIO_Struct.GPIO_Pin = GPIO_Pin_9;  
    GPIO_Struct.GPIO_Speed = GPIO_Speed_50MHz;  
    GPIO_Struct.GPIO_Mode = GPIO_Mode_AF_PP;  
    GPIO_Init(GPIOA, &GPIO_Struct);  
    // RX: GPIOA PIN9 funcion alterna  
    GPIO_Struct.GPIO_Pin = GPIO_Pin_10;  
    GPIO_Struct.GPIO_Speed = GPIO_Speed_50MHz;  
    GPIO_Struct.GPIO_Mode = GPIO_Mode_IN_FLOATING;  
    GPIO_Init(GPIOA, &GPIO_Struct);  
  
    USART_Cmd(USART1, ENABLE);  
    USART_Init(USART1, &UART_Struct);  
}
```

Algunas funciones útiles del puerto serial.

```
void USART_SendData(USART_TypeDef* USARTx, uint16_t Data)
```

Esta función se usa para enviar un dato de 8 o 9 bits, de acuerdo a la configuración del puerto.
Se puede mirar el estado de la memoria de envío.

```
uint16_t USART_ReceiveData(USART_TypeDef* USARTx)
```

Esta función devuelve el dato recibido. Es necesario primero observar la bandera adecuada para saber si hay dato sin leer.

```
FlagStatus USART_GetFlagStatus(USART_TypeDef* USARTx, uint16_t USART_FLAG)
```


Esta función devuelve el estado de una bandera. Hay varias banderas que indican un estado en especial, por ejemplo, para saber si la memoria de transmisión está vacía y se puede enviar otro dato, se usa la bandera USART_FLAG_TXE. O si se quiere saber si ha llegado un dato, se usa la bandera USART_FLAG_RXNE. Estas banderas se ponen en uno cuando el evento ocurre. El parámetro USARTx es para definir el puerto que se está usando, o sea, USART1 para el puerto Serial1, etc.

Un ejemplo de aplicación donde un caracter 'A' hace que se apague el led D2 y 'B' que se encienda, se muestra a continuación.

```
int main(void)
{
    char dato;
    LED_Init();
    UART_Init();

    while(1)
    {
        if(USART_GetFlagStatus(USART1, USART_FLAG_RXNE) != RESET)
        {
            dato= (char) USART_ReceiveData(USART1);
            // si recibe 'A', apaga el LED y transmite 'a'
            if(dato == 'A')
            {
                GPIO_SetBits(GPIOB, GPIO_Pin_12);
                while(USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET);
                USART_SendData(USART1, 'a');
            }
            // si recibe 'B', enciende el LED y transmite 'b'
            if(dato == 'B')
            {
                GPIO_ResetBits(GPIOB, GPIO_Pin_12);
                while(USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET);
                USART_SendData(USART1, 'b');
            }
        }
    }
}
```

Las funciones disponibles en la librería SPL del programa Embitz son las siguientes:

- void USART_DeInit(USART_TypeDef* USARTx); 
- void USART_Init(USART_TypeDef* USARTx, USART_InitTypeDef* USART_InitStruct);

Estas dos funciones se usan para desconfigurar el puerto serial previamente configurado y la segunda, para configurarlo, como se indicó arriba.

- void USART_StructInit(USART_InitTypeDef* USART_InitStruct);

Esta función es para configurar los parámetros básicos del puerto serial. Los parámetros deben estar definidos dentro de la estructura:

```
typedef struct
{
    uint32_t USART_BaudRate;// velocidad de comunicación. Se usa la ecuación:
        // IntegerDivider = ((PCLKx) / (16 *
        // (USART_InitStruct->USART_BaudRate)))
        // FractionalDivider = ((IntegerDivider - ((u32) IntegerDivider)) *

```

```

        // 16) + 0.5
uint16_t USART_WordLength;// número de bits del dato. Puede ser USART_WordLength_8b o
        // USART_WordLength_9b
uint16_t USART_StopBits;// número de bits de parada. Puede ser USART_StopBits_1,
        // USART_StopBits_0_5, USART_StopBits_2 o USART_StopBits_1_5
uint16_t USART_Parity;// paridad. Puede ser USART_Parity_No, USART_Parity_Even,
        // USART_Parity_Odd
uint16_t USART_Mode;// habilita recepción y/ o transmisión. Puede ser: USART_Mode_Rx,
        // USART_Mode_Tx
uint16_t USART_HardwareFlowControl;// tipo de control. Puede ser:
        // USART_HardwareFlowControl_None,
        // USART_HardwareFlowControl_RTS,
        // USART_HardwareFlowControl_CTS o
        // USART_HardwareFlowControl_RTS_CTS
} USART_InitTypeDef;

• void USART_ClockStructInit(USART_ClockInitTypeDef* USART_ClockInitStruct);
• void USART_ClockInit(USART_TypeDef* USARTx, USART_ClockInitTypeDef*
    USART_ClockInitStruct);

```

Estas dos funciones se usan para configurar el reloj del puerto serial síncrono. La primera inicializa la estructura que tiene los parámetros del reloj, la segunda usa esta estructura como entrada para hacer la configuración. La estructura está definida de la siguiente manera:

```

typedef struct
{
    uint16_t USART_Clock; // Indica si el reloj está habilitado o no. Puede ser
        // USART_Clock_Disable o USART_Clock_Enable
    uint16_t USART_CPOL; // Indica el valor del reloj. Puede ser USART_CPOL_Low
        // o USART_CPOL_High
    uint16_t USART_CPHA; // Indica la transición del reloj para captura de bit. Puede ser
        // USART_CPHA_1Edge o USART_CPHA_2Edge
    uint16_t USART_LastBit; // Indica si el pulso del reloj del último bit transmitido
        // se debe poner en el pin SCLK en modo síncrono. Puede ser
        // USART_LastBit_Disable o USART_LastBit_Enable
} USART_ClockInitTypeDef;

```

- void USART_Cmd(USART_TypeDef* USARTx, FunctionalState NewState);

Esta función es para habilitar el puerto serial. El primer parámetro es USART1, USART2 o USART3. El segundo parámetro puede ser ENABLE o DISABLE.

- void USART_ITConfig(USART_TypeDef* USARTx, uint16_t USART_IT, FunctionalState NewState);

Esta función se usa para habilitar o no una interrupción por un evento cualquiera del puerto serial. La fuente de interrupción puede ser cualquiera de las siguientes fuentes:

USART_IT_CTS: Cambio en CTS

USART_IT_LBD: Detección de ruptura en LIN. LIN es una forma de comunicación alterna.

USART_IT_TXE: Registro de datos de transmisión vacío.

USART_IT_TC: Transmisión completa.

USART_IT_RXNE: Registro de datos de recepción no vacío.

USART_IT_IDLE: Detección de línea sin actividad.

USART_IT_PE: Error de paridad.

USART_IT_ERR: Cualquiera de los errores de paquete, ruido o escritura/ lectura con registro lleno/ vacío.

- void USART_DMACmd(USART_TypeDef* USARTx, uint16_t USART_DMAREq, FunctionalState NewState);

La función USART_DMACmd() se usa para hacer la configuración del DMA del Uart. Cuando se llama esta función hace que los datos se almacenen en posiciones de memoria sin necesidad de llamar las subrutinas de lectura o escritura en registros. El parámetro USART_DMAREq puede ser USART_DMAREq_Tx, para transmisión, y USART_DMAREq_Rx para recepción.

- void USART_SetAddress(USART_TypeDef* USARTx, uint8_t USART_Address);

Esta función se usa cuando la comunicación es punto a multipunto, es decir, hay varios nodos receptores unidos a un nodo transmisor. Como todos los nodos reciben la misma información, el dato se puede direccionar a un solo nodo usando una dirección para cada nodo.

- void USART_WakeUpConfig(USART_TypeDef* USARTx, uint16_t USART_WakeUp);

Selecciona el método de alerta para el nodo en una comunicación multinodo. El parámetro USART_WakeUp puede ser USART_WakeUp_IdleLine, cuando la línea no está en uso, o USART_WakeUp_AddressMark, cuando hay una detección de la dirección del nodo.

- void USART_ReceiverWakeUpCmd(USART_TypeDef* USARTx, FunctionalState NewState);

En una comunicación multinodo esta función sirve para deshabilitar o habilitar un nodo.

- void USART_LINBreakDetectLengthConfig(USART_TypeDef* USARTx, uint16_t USART_LINBreakDetectLength);
- void USART_LINCmd(USART_TypeDef* USARTx, FunctionalState NewState);

Estas dos funciones son para comunicación con LIN (Red Local Interconectada), un protocolo de comunicaciones dirigido a comunicar sistemas dentro de un automóvil, de una forma económica y efectiva.

- void USART_SendData(USART_TypeDef* USARTx, uint16_t Data);
- uint16_t USART_ReceiveData(USART_TypeDef* USARTx);

Estas dos funciones se usan para enviar un dato por el puerto serial y para leer el registro de datos de entrada cuando llega un dato, respectivamente. Antes de usar cualquiera de estas dos funciones se debería usar la función de lectura de estado de los registros de datos de entrada y salida.

- void USART_SendBreak(USART_TypeDef* USARTx);
- void USART_SetGuardTime(USART_TypeDef* USARTx, uint8_t USART_GuardTime);

La primera función se usa para enviar bits en ceros, mientras que la segunda se usa para enviar bits en unos.

- `void USART_SetPrescaler(USART_TypeDef* USARTx, uint8_t USART_Prescaler);`

Esta función se usa para fijar el divisor del reloj en el modo IrDA. La comunicación IrDA se refiere a protocolo de comunicación con leds infrarrojos.

- `void USART_SmartCardCmd(USART_TypeDef* USARTx, FunctionalState NewState);`
- `void USART_SmartCardNACKCmd(USART_TypeDef* USARTx, FunctionalState NewState);`

Estas dos funciones se usan para habilitar el modo de tarjetas inteligentes (Smart Cards) y para habilitar la transmisión de indicación de no reconocimiento (NACK). El modo Smart Cards se usa para comunicación con tarjetas inteligentes como las que tienen las tarjetas de crédito o los teléfonos celulares.

- `void USART_HalfDuplexCmd(USART_TypeDef* USARTx, FunctionalState NewState);`

Habilita o no el uso de comunicación Halfduplex, o sea transmite o recibe, pero no al tiempo.

- `void USART_OverSampling8Cmd(USART_TypeDef* USARTx, FunctionalState NewState);`
- `void USART_OneBitMethodCmd(USART_TypeDef* USARTx, FunctionalState NewState);`

La primera función se usa para habilitar el incremento de muestreo de bits. La lectura de datos de recepción se hace muestreando cada bit dentro del tiempo destinado a un bit. La mayoría de muestras en un valor lógico determina el valor del bit. Si esta función se desea usar, se debe llamar antes de la función de configuración, `USART_Init()`. La segunda función se usa para habilitar el método de muestreo de un bit.

- `void USART_IrDAConfig(USART_TypeDef* USARTx, uint16_t USART_IrDAMode);`
- `void USART_IrDACmd(USART_TypeDef* USARTx, FunctionalState NewState);`

Estas dos funciones son para configuración y habilitación de la comunicación IrDA.

- `FlagStatus USART_GetFlagStatus(USART_TypeDef* USARTx, uint16_t USART_FLAG);`
- `void USART_ClearFlag(USART_TypeDef* USARTx, uint16_t USART_FLAG);`

Estas dos funciones se usan para determinar el estado de la comunicación serial y para borrar las banderas de indicación. Las banderas del parámetro `USART_FLAG` de la primera función pueden ser:

`USART_FLAG_CTS`: Cambio en el estado de CTS (Clear to Send).

`USART_FLAG_LBD`: Detección de señal Break en comunicación LIN.

`USART_FLAG_TXE`: Indicación de registro de datos de transmisión vacío.

`USART_FLAG_TC`: Indicación de transmisión completada.

`USART_FLAG_RXNE`: Indicación de registro de datos de recepción no vacío.

`USART_FLAG_IDLE`: Detección de línea en estado flotante.

`USART_FLAG_ORE`: Indicación de error por recepción de dato sin haber leído uno anterior.

`USART_FLAG_NE`: Indicación de error por ruido.

`USART_FLAG_FE`: Indicación de error por pérdida de sincronismo o mucho ruido.

`USART_FLAG_PE`: Indicación de error por paridad.

Las banderas del parámetro `USART_FLAG` de la segunda función pueden ser: `USART_FLAG_CTS`, `USART_FLAG_LBD`, `USART_FLAG_TC` y/o `USART_FLAG_RXNE`.

- `ITStatus USART_GetITStatus(USART_TypeDef* USARTx, uint16_t USART_IT);`
- `void USART_ClearITPendingBit(USART_TypeDef* USARTx, uint16_t USART_IT);`

Estas dos funciones son para obtener el estado de las fuentes de interrupción y borrar las banderas indicadoras de interrupción, respectivamente. Las banderas USART_IT de la primera función pueden ser:

USART_IT_CTS: Interrupción por cambio de estado en la línea CTS.

USART_IT_LBD: Interrupción por detección de comando LIN Break.

USART_IT_TXE: Interrupción por registro de datos de transmisión vacío.

USART_IT_TC: Interrupción por transmisión completada.

USART_IT_RXNE: Interrupción por registro de datos de recepción no vacío.

USART_IT_IDLE: Interrupción por detección de línea flotante.

USART_IT_ORE: Interrupción por indicación de error por recepción de dato sin haber leído uno anterior.

USART_IT_NE: Interrupción por indicación de error por ruido.

USART_IT_FE: Interrupción por indicación de error por pérdida de sincronismo o mucho ruido.

USART_IT_PE: Interrupción por indicación de error por paridad.