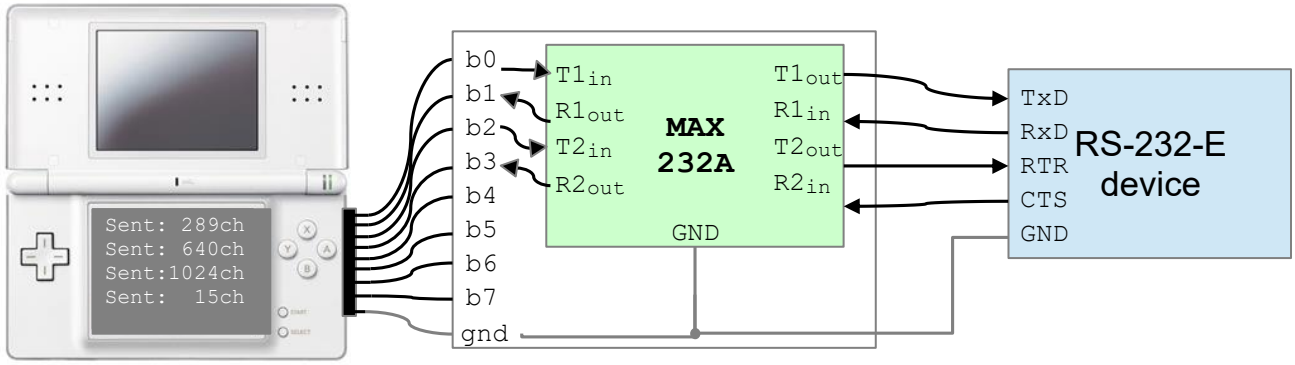
Problema 29: Envío de datos por RS-232-E (Ex. 2ª Conv. 2018-19)

Se propone controlar un dispositivo que recibe datos serializados (bit a bit) a través de un conector compatible con el estándar RS-232-E, por ejemplo, una impresora serie, una máquina industrial, un foco de luz programable o un sintetizador de sonido. Los datos se enviarán desde un ordenador central hacia una NDS a través de una conexión wifi, y la NDS deberá transmitirlos al dispositivo a través de una interfaz como la del siguiente esquema:



Dicha interfaz consiste en un único registro de Entrada/Salida de 8 bits, de nombre simbólico `REG_RS232`, conectado a un circuito adaptador de señal eléctrica MAX232A, cuyo propósito es transformar señales lógicas TTL (0 → 0V..+0,8V; 1 → +2V..+5V) a señales lógicas RS-232 (0 → +3V..+15V; 1 → -15V..-3V). A continuación se describe la función de cada señal (bit del reg. E/S ↔ línea) de la interfaz NDS/dispositivo:

Interfaz NDS	*Líneas RS-232*	*Descripción*
`Reg.E/S: bit 0`	`TxD (Transmitted Data)`	Línea de transmisión de datos (bits), desde el computador (NDS) hacia el dispositivo
`Reg.E/S: bit 1`	`RxD (Received Data)`	Línea de recepción de datos (bits), desde el dispositivo hacia el computador (NDS)
`Reg.E/S: bit 2`	`RTR (Ready To Receive)`	El computador (NDS) activará esta señal (1) cuando esté preparado para recibir datos por la línea RxD
`Reg.E/S: bit 3`	`CTS (Clear To Send)`	El dispositivo activará esta señal (1) cuando esté preparado para recibir datos por la línea de transmisión de la NDS (TxD)
`gnd`	`GND (GrouND)`	Nivel de voltaje de referencia del 0

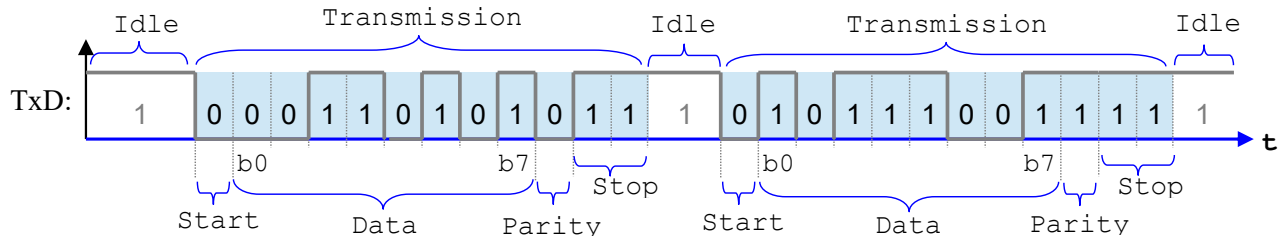
El funcionamiento del sistema para enviar datos al dispositivo será el siguiente:

- cuando se ponga en marcha la NDS, se establecerá la conexión wifi con el ordenador central y se determinará la velocidad de transferencia de datos (ver descripción de la rutina de inicializaciones),
- la NDS recibirá datos desde la wifi, en forma de paquetes de caracteres (vectores de

bytes); como máximo, se recibirán 1024 caracteres por paquete; se dispone de una rutina ya implementada para gestionar esta comunicación,

- cuando se disponga de un paquete, cada uno de sus bytes se deberá transformar en un bloque de 12 bits, según un formato concreto para su transmisión asíncrona en serie (1 bit de *start*, 8 bits de datos, 1 bit de paridad y 2 bits de *stop*); este aspecto se detalla más adelante, aunque se dispone de una rutina ya implementada para realizar esta conversión,
- a continuación, si el dispositivo está preparado para la recepción de datos (`CTS` = 1), la NDS pasará a transferir los bits de todos los bloques del paquete actual a través de la línea `TxD`, bit a bit, a la velocidad (bits/s) establecida en la inicialización,
- al final de la transferencia de cada cada bloque de 12 bits, la NDS debe consultar el estado de la señal `CTS`; en el caso de que esté a 0, la NDS debe detener temporalmente la transferencia del siguiente bloque,
- mientras el dispositivo no esté preparado (`CTS` = 0), la NDS tendrá que ir consultando periódicamente esta señal para detectar cuándo pasa a valer 1; en ese caso, la NDS deberá empezar la transmisión o reanudarla, si la había detenido previamente,
- cuando se hayan transferido todos los bloques del paquete actual, la NDS escribirá por su pantalla inferior el número de caracteres enviados al dispositivo (ver pantalla NDS en el esquema inicial) y pasará de nuevo a esperar la recepción del siguiente paquete que le pueda enviar el ordenador central,
- todo el proceso de recepción de paquetes, conversión a bloques y transmisión de los bits por la conexión RS-232 se debe realizar concurrentemente con ciertas tareas independientes; puede que dichas tareas accedan a los bits del registro `REG_RS232` no utilizados por este protocolo de envío de datos al dispositivo.

Con el fin de ejemplificar la transmisión en serie de los bloques, el siguiente cronograma muestra la evolución de la activación de la línea `TxD` para dos bloques consecutivos, correspondientes a los bytes 10101100 y 10011101:



Cada bloque empieza siempre con un bit de inicio (*Start*) que vale 0, seguido por los 8 bits del byte (*Data*), enviados de menor a mayor peso, seguidos por un bit de paridad (*Parity*), cuyo valor debe asegurar que el número de bits a 1 de todo el paquete sea siempre par, y termina con dos bits de final de bloque (*Stop*), que siempre valen 11.

El periodo de transmisión de cada bloque está indicado en la parte superior del cronograma como *Transmission*, mientras que los periodos de reposo están indicados como *Idle*, donde la

línea `TxD` debe permanecer a 1. Aunque el cronograma de ejemplo muestra cierto tiempo de reposo entre los dos bloques, en realidad este *Idle* no es estrictamente necesario, ya que los bits de *Start* y *Stop* son suficientes para que el receptor identifique el final de un bloque y el inicio del siguiente.

Para realizar el programa de la NDS se dispone de las siguientes rutinas ya implementadas:

Rutina	*Descripción*
`short inicializaciones()`	Inicializa el *hardware* (pantalla, interrupciones, etc.); no inicializa el *timer* 0; establece la conexión wifi con el ordenador central y devuelve el número de bits por segundo al que debe realizarse la transmisión serie; los valores típicos serán 75, 110, 300, 1200, 2400, 4800, 9600 y 19200 bps.
`tareas_independientes()`	Tareas que no dependen de la transmisión de datos al dispositivo en cuestión (t. máx. = 100 ms)
`inicializar_timer0(unsigned short freq)`	Inicializa el *timer* 0 para que genere interrupciones a la frecuencia especificada por parámetro, en hercios
`desactivar_timer0()`	Desactiva el *timer* 0
`unsigned short recibir_datos( unsigned char by[])`	Si el ordenador central está preparado para enviar un paquete de caracteres, esta rutina recibirá y copiará dichos caracteres dentro del vector que se pasa por referencia, devolviendo como resultado el total de caracteres copiados (máx. 1024); si no hay ningún paquete preparado, la rutina retornará directamente y devolverá cero como resultado; tiempo máximo de ejecución = 50 ms
`generar_bloques( unsigned short bl[], unsigned char by[], unsigned short nb)`	Rutina que convierte los bytes contenidos en el vector `by[]` en bloques de 12 bits, dentro del vector `bl[]`; el parámetro `nb` indica cuantos bytes se deben convertir (t. máx. = 0,1 ms)
`swiWaitForVBlank()`	Espera hasta el próximo retroceso vertical
`printf(char *format, ...)`	Escribe un mensaje en la pantalla inferior de la NDS

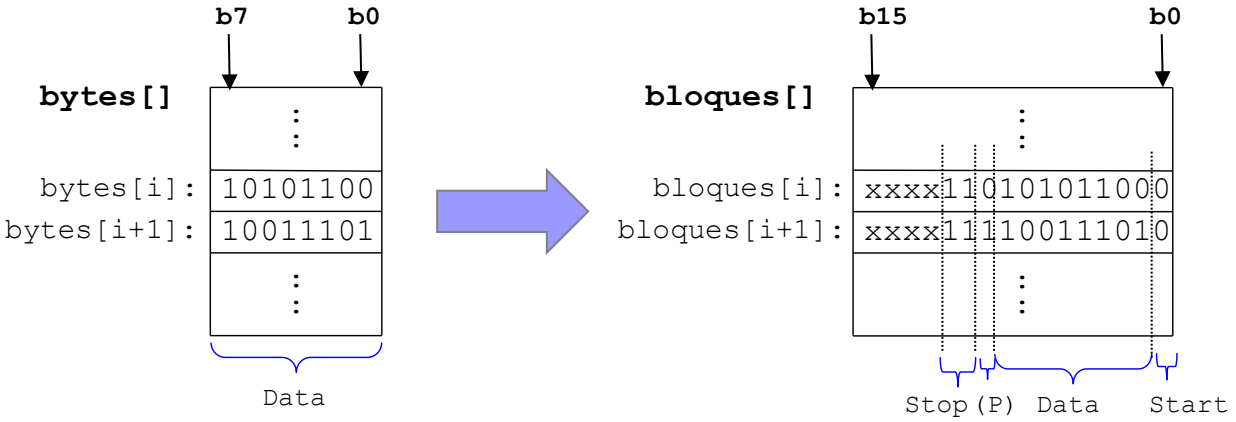
Se sugiere el uso de las siguientes variables globales:

```
#define MAX_DATOS     1024              // número máximo de datos a enviar

unsigned short num_datos = 0;           // número de datos actual
unsigned short bloques[MAX_DATOS];      // vector de bloques a enviar
unsigned char bytes[MAX_DATOS];         // vector para almacenar los bytes
                                        // que se recibirán por wifi
```

La variable `num_datos` permitirá registrar el número de datos que contiene el paquete actual de caracteres a transmitir. Dicho paquete se podrá almacenar dentro del vector `bytes[]`. Además, se define otro vector `bloques[]` para poder albergar los bloques de 12 bits correspondientes a cada carácter de 8 bits, según la conversión que realiza la rutina `generar_bloques()`.

Aunque dicha rutina ya está implementada, hay que saber cómo empaqueta los bits de cada bloque dentro del vector correspondiente. El siguiente gráfico muestra la codificación de dos caracteres consecutivos almacenados en el vector de bytes:



Como se puede observar, los 8 bits de datos quedan desplazados un bit a la izquierda, conservando su ordenación inicial, es decir, el bit de menos peso del dato (`b0`) se almacena en el bit 1 del bloque y el bit de más peso del dato (`b7`) se almacena en el bit 8 del bloque. El bit de menos peso del bloque se reserva para el *Start*, el bit de paridad (`P`) se almacena en `b9`, mientras que los bits de *Stop* se almacenan en `b10` y `b11`. Los cuatro bits de más peso en el vector de bloques quedan indeterminados (`xxxx`).

Para gestionar la transmisión en serie de los bits de los bloques del paquete actual se debe utilizar la RSI del *timer0*, programada a la frecuencia adecuada.

Se pide:

Programa principal y variables globales adicionales en C, RSI del *timer0* en ensamblador.