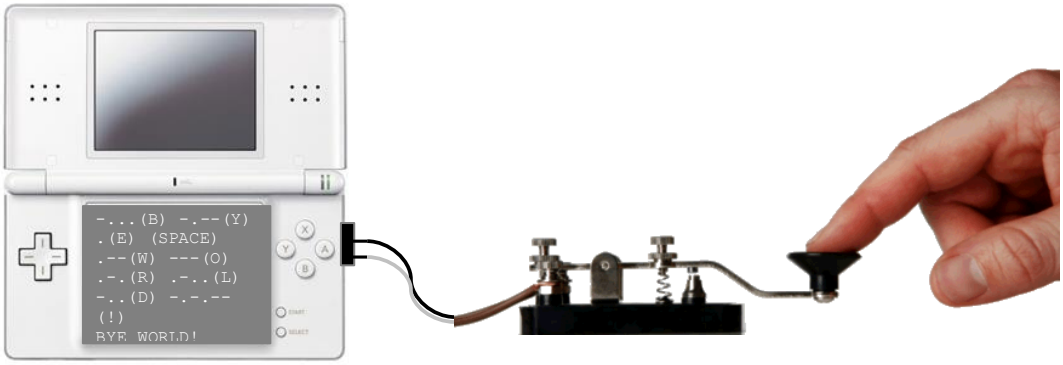
Problema 26: Lectura morse (Ex. 1ª Conv. 2018-19)

Se propone conectar un pulsador a la NDS para generar código Morse:



La interfaz que vamos a utilizar consiste en un único registro de Entrada/Salida de 32 bits, de nombre simbólico `REG_DATA`. El pulsador estará conectado al bit 15 del registro, aunque pueden haber otros sensores o actuadores conectados al resto de bits.

El usuario utilizará el pulsador para codificar un mensaje en forma de puntos y rallas. Para distinguir los elementos del código Morse se utiliza la duración de las pulsaciones. Para detectar dicha duración, vamos a usar un tiempo de referencia que denominaremos *tic*, cuyo periodo se fijará al inicio del programa entre 50 y 150 milisegundos.

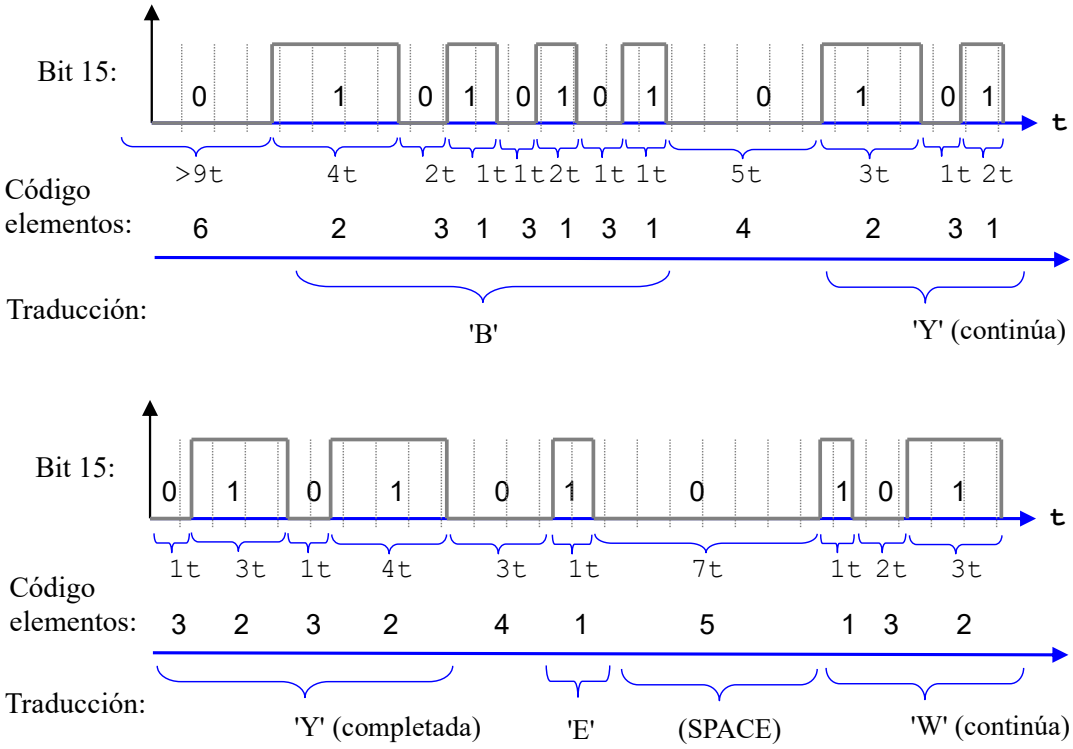
La siguiente tabla muestra el tiempo mínimo y máximo, en número de tics, que se deberá considerar para distinguir entre los símbolos '.' y '-', que son los elementos del código Morse generados con el pulsador a 1 (pulsado), pero también se presentan los tiempos mínimo y máximo de los espacios separadores entre símbolos, entre letras, entre palabras y entre mensajes, que son los elementos generados con el pulsador a 0 (soltado):

Código	Elemento	t_min	t_max	bit 15
1	'.' (símbolo punto)	1	2	1
2	'-' (símbolo ralla)	3	9	1
3	espacio entre símbolos	1	2	0
4	espacio entre letras	3	5	0
5	espacio entre palabras	6	9	0
6	espacio entre mensajes	10	∞	0

La variabilidad del tiempo de cada símbolo es debida a que el operador no puede generar dichos tiempos exactamente, de modo que será necesario establecer unos rangos que permitan cierta flexibilidad a la hora de introducir los pulsos. Para simplificar un poco el problema, se supondrá que el operador jamás mantendrá el pulsador a 1 más de 9 tics. Sin embargo, el

tiempo máximo del espacio entre mensajes no está limitado, aunque se podrá considerar que después de 10 tics ya se habrá terminado el último mensaje introducido.

El siguiente cronograma (partido en dos trozos consecutivos) muestra una introducción de pulsos para la frase “BYE WORLD!”, aunque solo llega hasta el inicio de la letra 'W'; en este cronograma se muestran, en línea discontinua, los puntos de muestreo periódico de la señal de pulsación, así como el número de tics obtenido para cada estado del pulsador y sus correspondientes códigos de elemento y traducción para el mensaje:



El programa a implementar deberá realizar un seguimiento de las pulsaciones del usuario. Cuando se detecte el final de una letra, se deberán escribir por pantalla todos los puntos y rallas introducidos hasta el momento, además de su letra correspondiente, a continuación (sin espacios en blanco) y entre paréntesis. Cuando se detecte el final de una palabra, se deberá escribir “(SPACE)” (sin las comillas). Hay que dejar un espacio en blanco después de cada bloque de puntos y rallas/letra, así como al final del bloque “(SPACE)”. Cuando se detecte el final del mensaje, se deberá insertar un salto de línea y volver a escribir todo el mensaje como un *string* convencional, es decir, solo las letras y los espacios como un carácter de espacio normal (no hay que escribir “(SPACE)”). A continuación se muestra la visualización en pantalla correspondiente a la introducción de la frase de ejemplo:

```
-...(B) -.--(Y) .(E) (SPACE) .--(W) ---(O) .-.(R) .-..(L) -.-.--(!)
BYE WORLD!
```

Además, al finalizar la introducción de un mensaje, hay que enviarlo por la Wifi a otra NDS. Concurrentemente, el programa también deberá recibir mensajes por la Wifi de la otra NDS y escribirlos en la otra pantalla, como *strings* convencionales. Estas tareas de comunicación a través de la Wifi corresponden a las "tareas independientes" del proceso de captación, traducción y visualización de mensajes en código Morse descrito en el párrafo anterior, aunque el envío de un nuevo mensaje está supeditado al final de la captación de dicho mensaje, obviamente.

Para realizar este programa se dispone de las siguientes rutinas ya implementadas:

Rutina	*Descripción*
`int inicializaciones()`	Inicializa el *hardware* (pantalla, interrupciones, etc.); devuelve el valor del tiempo de referencia, en ms (no inicializa el *timer* 0)
`inicializar_timer0(unsigned short freq)`	Inicializa el *timer* 0 para que genere interrupciones a la frecuencia especificada por parámetro, en hercios.
`recibir_mensaje()`	Gestiona la recepción de nuevos mensajes por la Wifi, escribiéndolos en la pantalla superior de la NDS (tiempo de ejecución máximo = 150 ms)
`enviar_mensaje(char *msg)`	Gestiona el envío de un mensaje por la Wifi, que se pasa por parámetro como un *string* (formato C), pero sin escribirlo por pantalla (t. ejecución máx = 100 ms)
`char traducir_morse(unsigned char *simbolos, int lon)`	Traduce un vector de símbolos Morse (códigos 1 y 2) con el número de elementos indicado por el parámetro `lon`, y devuelve como resultado el código ASCII de la letra correspondiente (t. ejecución máx. = 30 μs)
`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 y definiciones:

```
#define MAX_LETRAS  100             // máximo de letras por mensaje

unsigned char i_simb = 0;           // índice del vector simb[]
unsigned char simb[8];              // vector de símbolos Morse
                                        // 1 → '.' (punto)
                                        // 2 → '-' (ralla)
unsigned char i_mens = 0;           // índice del vector mensaje[]
char mensaje[MAX_LETRAS+1];         // vector de letras (string)
```

El `#define` permitirá configurar el número máximo de caracteres que se admiten por mensaje. La variable `i_simb` permitirá indexar el vector `simb`, cuyo propósito es registrar los símbolos Morse de una letra, con los códigos 1 y 2 para los puntos y las rallas. Todos los

códigos Morse tienen como máximo 8 símbolos. La variable `i_mens` permitirá indexar el vector `mensaje`, cuyo propósito es registrar los códigos ASCII de todas las letras del mensaje, en forma de *string* de lenguaje C (con centinela final).

Para conseguir una sincronización con las pulsaciones suficientemente rápida, se deberá utilizar la interrupción del *timer* 0 para muestrear periódicamente el valor del pulsador.

Se pide:

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