Problema 6: Micrófono

Se propone capturar audio con el micrófono con el que está equipada la plataforma NDS. El programa a realizar debe capturar sonido continuamente y mostrar por pantalla una visualización gráfica instantánea de la energía de cada rango de frecuencias.

Se dispone de las siguientes rutinas, ya implementadas:

Rutina	*Descripción*
`inicializaciones()`	Realiza inicializaciones del *hardware*
`tareas_independientes()`	Tareas que no dependen de la captación del audio (ej. reconocimiento de texto en el audio capturado anteriormente)
`swiWaitForVBlank()`	Espera retroceso vertical
`mostrar_frecuencias(char *audio)`	Dibuja en una pantalla de la NDS la distribución de energía en diferentes rangos de frecuencias

Además, hay que realizar la captura del audio utilizando las interrupciones del *timer* 0. Se dispone de una rutina ya implementada de nombre `inicializar_timer0()` que programa la interrupción IRQ_TIMER0 con una frecuencia de 11 KHz.

También disponemos de las siguientes rutinas para comunicarnos con el micrófono:

Rutina	*Descripción*
`iniciar_MIC()`	Activa el *hardware* del micrófono
`byte recibir_MIC()`	Recibe un byte de datos del micrófono, con el nivel de audio actual digitalizado sobre el rango 0..255
`parar_MIC()`	Desactiva el *hardware* del micrófono

El protocolo de comunicación consta de los siguientes comandos:

- iniciar el micrófono, una vez al principio del programa
- recibir y almacenar los bytes necesarios
- parar el micrófono, al final del programa

Cada comando de comunicación tarda menos de 40 microsegundos en ejecutarse.

La problemática principal del programa consiste en gestionar los "trozos" de audio que se tienen que pasar a la función `mostrar_frecuencias()`. Esta función recibe por parámetro un *buffer* de 1.100 bytes con el sonido muestreado a 11 KHz, es decir, todo el sonido capturado por el micrófono en una décima de segundo.

El cálculo del gráfico de frecuencias es relativamente lento, ya que puede tardar entre 20 y 80 milisegundos. Mientras se está realizando este cálculo, el *buffer* no se puede modificar. Por lo tanto, los datos capturados por el micrófono durante ese tiempo deben ser almacenados en otro *buffer*.

En consecuencia, para gestionar la información de audio se usarán dos *buffers*, más dos variables de soporte:

```
char buffer_mic[2][1100];
short mic_buffer_actual;
short mic_index;
short mic_buffer_disponible;
```

Los *buffers* se pueden referenciar como `buffer_mic[0]` y `buffer_mic[1]`. La variable `mic_buffer_actual` indicará sobre qué *buffer* (0 o 1) se está guardando la información que captura el micrófono. La variable `mic_index` se utilizará para saber en qué posición del *buffer* actual se tiene que almacenar la captura del micrófono. La variable `mic_buffer_disponible` indicará si ya existe un *buffer* de información de audio disponible para ser visualizada por la función `mostrar_frecuencias()`.

En definitiva, habrá que controlar el micrófono para ir capturando datos sobre un *buffer* mientras la función de `mostrar_frecuencias()` trabajará sobre los datos del otro *buffer*.

Habrá que crear una rutina de nombre `cambiar_buffers()`, que cambiará el buffer actual de captura.

Se pide:

Programa principal en C, RSI del *timer 0* y rutina `cambiar_buffers()` en ensamblador.