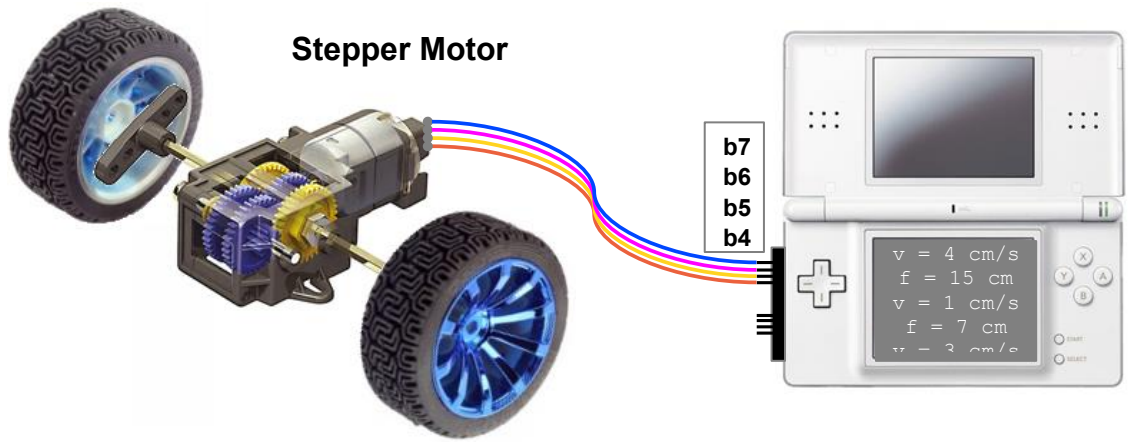
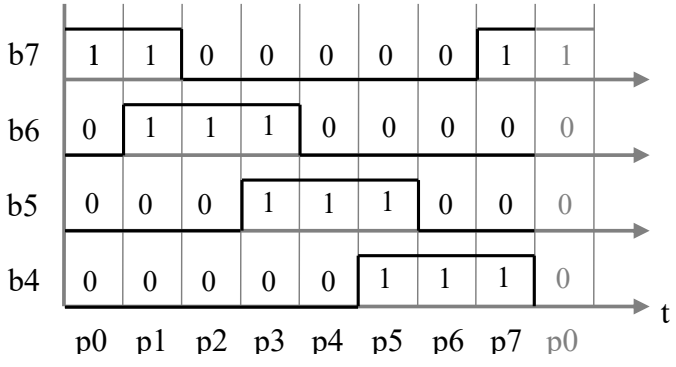
Problema 20: Motor de tracción (Ex. 1ª Conv. 2016-17)

Se propone controlar un robot móvil tipo coche con la NDS, utilizando un motor para accionar las dos ruedas traseras de tracción y otro motor para fijar la orientación de las dos ruedas delanteras de dirección. En este problema, sin embargo, solo se pide el código del control del motor de las ruedas de tracción:

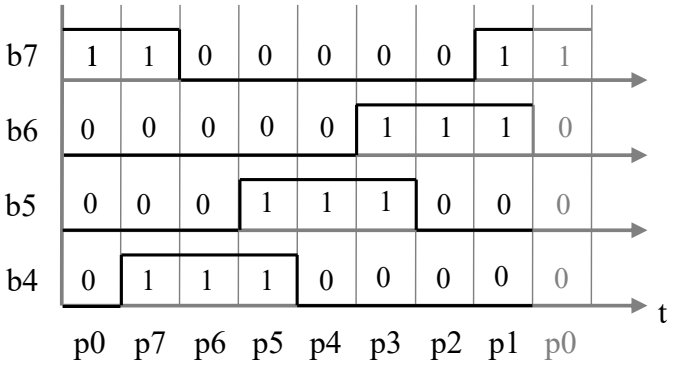


En el esquema anterior se ha representado la conexión de los 4 cables de control del motor paso a paso (*Stepper motor*) de tracción, que están asociados a los bits b7-b4 de un registro de E/S de 32 bits, al cual se accederá a través de la dirección de memoria `0x0A000000`. El resto de bits del mismo registro se utilizarán para otras tareas de control del robot móvil, como la orientación de las ruedas de dirección. Por lo tanto, será necesario no modificar el estado del resto de bits cuando se actualicen los bits b7-b4. Para ello debemos suponer que se podrá leer, directamente del registro de E/S, el estado de los bits de salida fijado en su última escritura.

El motor paso a paso se controla mediante un determinado patrón en sus 4 bits de control. Existen 8 patrones posibles, que denominaremos **fases**, de la `p0` a la `p7`. Al incrementar la fase, el motor avanza un paso, es decir, gira un cierto ángulo en un sentido (horario, por ejemplo). Al decrementar la fase, el motor retrocede un paso, es decir, gira el mismo ángulo en sentido contrario. Los siguientes esquemas muestran los patrones de bits para las 8 fases, además de la evolución de fases correspondiente a los dos sentidos de movimiento del robot móvil, hacia adelante (*Forward stepping*) y hacia atrás (*Backward stepping*):



Forward stepping



Backward stepping

La evolución de las fases es circular, es decir, la fase superior a la `p7` es la `p0` y la fase inferior a la `p0` es la `p7`. La frecuencia a la que se cambie de fase determinará la velocidad de rotación del motor. Según el fabricante, el motor utilizado admite hasta 1.000 cambios de fase (pasos) por segundo. Además, se nos indica que el motor requiere 64 pasos para dar una vuelta completa (5,625º/paso). Sin embargo, el eje del motor está conectado a una serie de engranajes que reducen la rotación de las ruedas respecto a la del motor, de manera que se requieren 4.076 pasos para conseguir una rotación completa de las ruedas. Como las ruedas son de 5,34 cm de diámetro (16,78 cm de perímetro), serán necesarios 243 pasos para avanzar o retroceder 1 cm (± 0,5% error).

Se utilizará la RSI del *timer* 0 para generar los cambios de fase, a la frecuencia necesaria para conseguir una velocidad del vehículo determinada. Como la frecuencia de cambios de fase no puede superar 1 KHz, la velocidad del vehículo, en valor entero, podrá ser de 4 cm/s (972 pasos/s), como máximo. Aunque es una velocidad relativamente lenta, respecto a un coche teledirigido, por ejemplo, hay que tener en cuenta que la ventaja de utilizar un motor paso a paso es que podremos determinar la posición del vehículo con mucha precisión (0,005 cm de error por cm avanzado), lo cual permitirá realizar aplicaciones de conducción automática y exploración del entorno.

El programa a implementar deberá ir ejecutando una serie de tareas (semi-)independientes al control del motor de tracción, como recibir órdenes, calcular la trayectoria, detectar obstáculos, etc. Concurrentemente, el código a implementar deberá obtener y ejecutar las consignas de movimiento del vehículo, generando los cambios de fase correspondientes. Cada consigna se obtendrá con una llamada a la rutina `siguiente_movimiento()`, que devolverá (por referencia) dos valores:

`vel` : valor entero entre -4 y 4, donde el valor absoluto indica la velocidad del robot en centímetros por segundo y el signo indica si hay que avanzar (+) o retroceder (-); si vale cero significa que no hay nueva consigna de movimiento, o sea, que el robot debe estar parado,

`avn`: valor natural entre 1 y 100, que indica los centímetros que tiene que avanzar o retroceder el robot, según el signo de la velocidad.

Es importante no llamar a la rutina `siguiente_movimiento()` hasta que no se haya terminado el movimiento anterior.

Además, en la pantalla inferior de la NDS se debe ir escribiendo cada nueva consigna recibida, en una línea la velocidad absoluta, por ejemplo, "v = 3 cm/s" y en la siguiente línea el avance, indicando 'f' si es hacia delante o 'b' si es hacia atrás, por ejemplo, "b = 10 cm" (ver pantalla inferior del gráfico inicial).

Se dispone de las siguientes rutinas, ya implementadas:

Rutina	*Descripción*
`inicializaciones()`	Inicializa *hardware* (pantalla, interrupciones, *timers*, etc.)
`tareas_independientes()`	Tareas que no dependen directamente del control del motor de tracción (ej. control de otro motor para fijar el ángulo de las ruedas de dirección); tiempo de ejecución entre 0,1 μs y 1 segundo
`activar_timer0()`	Activa el funcionamiento del *timer* 0
`desactivar_timer0()`	Desactiva el funcionamiento del *timer* 0
`fijar_frecuencia(int frec)`	Calcula y fija el divisor de frecuencia del *timer* 0 para que genere interrupciones a la frecuencia de salida especificada por parámetro (en Hz)
`siguiente_movimiento(char *vel, unsigned char *avn)`	Devuelve por referencia los valores de una nueva consigna (ver descripción anterior), o velocidad igual a cero si no hay nueva consigna de movimiento; puede tardar entre 0,1 μs y 1 ms en ejecutarse
`swiWaitForVBlank()`	Espera hasta el próximo retroceso vertical
`printf(char *format,...)`	Escribe un mensaje en la pantalla inferior de la NDS

Además del programa principal y la RSI del *timer* 0, se pide el código de la rutina principal de gestión de interrupciones (RPSI):

```
void intr_main();
```

la cual debe detectar si se ha activado el bit IRQ_TIMER0 en el registro REG_IF del controlador de interrupciones y, en caso afirmativo, debe invocar la RSI del *timer* 0. Además, debe notificar la resolución de cualquier IRQ que se haya producido sobre el propio registro REG_IF, así como sobre la posición de memoria INTR_WAIT_FLAGS, para conseguir desbloquear posibles llamadas a funciones de la BIOS que esperan la generación de interrupciones, como la `swiWaitForVBlank()`.

La rutina `inicializaciones()` instalará la dirección de la rutina `intr_main()` en la posición de memoria `0x0B003FFC`, destinada a almacenar la dirección de la RPSI de la NDS (bajo compilación con *devkitPro* + *libnds*).

Se pide:

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