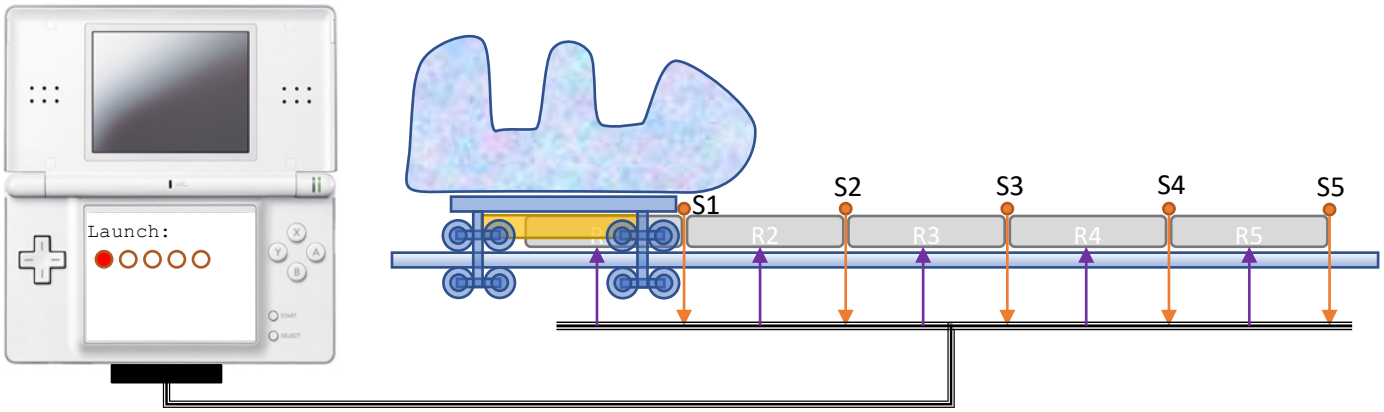
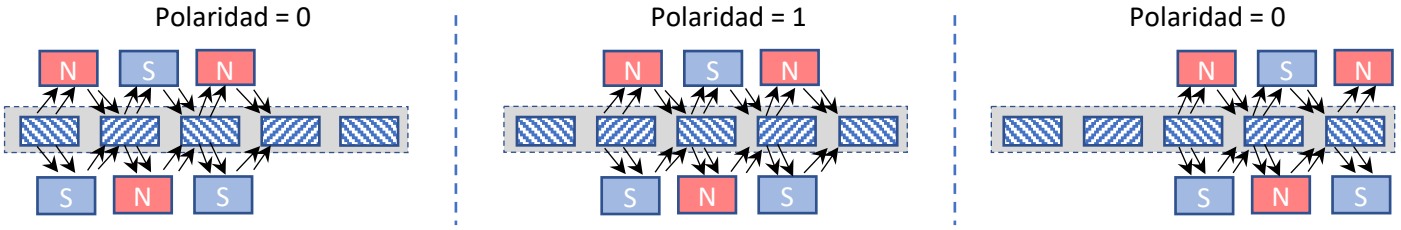
Lanzamiento con electroimanes

Se propone controlar con la NDS el lanzamiento de un coche de montaña rusa propulsado por un motor electromagnético lineal. La siguiente figura muestra un esquema del sistema:



En el esquema anterior se muestra una serie de cinco bloques etiquetados del R1 al R5. Cada uno de estos bloques contiene seis electroimanes. Estos bloques están situados en medio del raíl y son estrechos, de modo que forman una especie de mampara. El esquema muestra la mampara de lado (alto x ancho); con una vista superior del raíl, observaríamos una línea.

Los electroimanes son bobinas (rodillos) de cable conductor que generan un campo magnético cuando circula corriente eléctrica. Cambiando el sentido de circulación de la corriente se consigue cambiar la polaridad magnética (Norte/Sur). A su vez, el coche de la montaña rusa lleva instalados doce imanes permanentes, seis a cada lado de la mampara de bloques.



Para ilustrar cómo actúa el motor, el esquema anterior muestra una vista superior de una versión reducida de los componentes descritos inicialmente: un bloque de cinco electroimanes (rectángulos con un patrón de rayas oblicuas), y seis imanes laterales (tres + tres) que van unidos a la parte inferior del coche. Con la circulación de corriente por todos los electroimanes del bloque se producen unas fuerzas de atracción y repulsión (representadas con flechas) con los imanes permanentes que hacen que el coche avance. Cuando el coche ha avanzado hasta la siguiente posición de electroimanes, se debe cambiar la polaridad para seguir moviendo el coche en el mismo sentido de la marcha. La polaridad se ha indicado como 0 y 1, que en realidad se corresponde con los dos sentidos de circulación de la corriente eléctrica. Evidentemente, el cambio de polaridad debe ir sincronizado con el movimiento del coche.

Además, para conseguir resincronizar el cambio de polaridad con el movimiento del coche, se ha instalado un sensor al final de cada bloque de electroimanes. Cada sensor está etiquetado desde S1 hasta S5. El control de los bloques de electroimanes y la detección del estado de los sensores se realizará mediante dos registros de 16 bits:

REG_EM:

9	8	7	6	5	4	3	2	1	0
x	x	x	x	x	x	0	0	0	0	1	0	1	0	0	0
						R5		R4		R3		R2		R1	

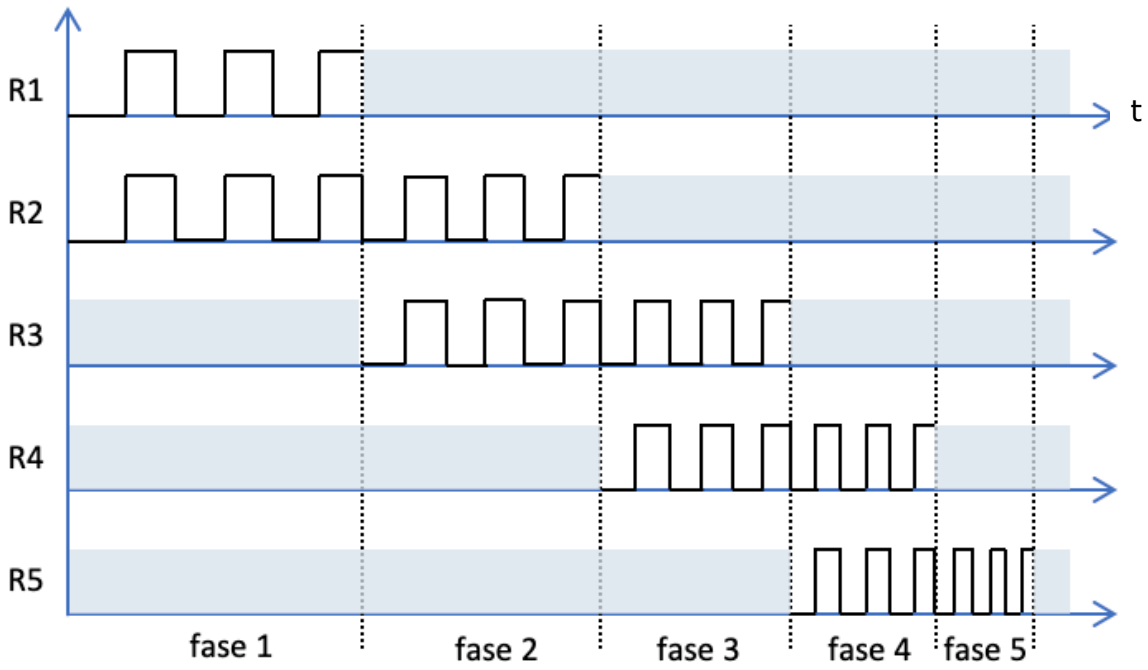
REG_SENS:

5	4	3	2	1	0
x	x	x	x	x	x	x	x	x	x	0	0	0	1	0	x
						S5	S4	S3	S2	S1	

Los bits del 1 al 5 del registro REG_SENS indican si el coche está pasando por encima del sensor correspondiente (1) o no (0). El resto de bits no se utilizan.

Los bits del 0 al 9 del registro REG_EM permiten controlar la activación de cada bloque de electroimanes y su polaridad. Concretamente, para cada bloque se reservan un par de bits, tal como se indica en el diagrama del registro. El bit de más peso del par sirve para activar (1) o desactivar (0) la transmisión de corriente eléctrica a través de los electroimanes del bloque, mientras que el bit de menos peso del par sirve para modificar el sentido de la circulación de la corriente. El resto de bits no se utilizan.

Cada bloque de electroimanes consume una cantidad muy elevada de potencia eléctrica (centenares de kilovatios). Por lo tanto, se requiere que no se activen todos los bloques a la vez, sino solo dos bloques como máximo, cuando el coche esté entre esos dos bloques. El siguiente cronograma representa la evolución de la activación y cambio de polaridad de los cinco bloques de electroimanes, durante cinco fases.



Cuando el bloque está desactivado se muestra una banda sombreada. Cuando el bloque está activado, se muestran los pulsos que representan la polaridad de los electroimanes. A medida que el coche avanza, se incrementa la frecuencia de los pulsos para acelerarlo.

Se dispone de las siguientes rutinas ya implementadas:

Rutina	*Descripción*
`inicializaciones()`	Inicializa el *hardware* (pantalla, interrupciones, etc.)
`tareas_independientes()`	Tareas que no dependen del lanzamiento del coche, por ejemplo, control del acceso de pasajeros, preparación de los sistemas de frenado, iluminación, etc. (tiempo de ejecución < 0,1 s)
`scanKeys() / keysDown()`	Funciones de lectura de botones de la librería *libNDS*
`swiWaitForVBlank()`	Espera hasta el próximo retroceso vertical
`show_sensor(unsigned char id_sens, unsigned char status)`	Visualiza por pantalla el estado del sensor indicado por parámetro (`id_sens`: [1..5]), mostrando un círculo lleno o vacío según si el segundo parámetro es cierto o falso
`prog_timer(unsigned char id_tim, short div_freq);`	Programa el *timer* indicado con el primer parámetro (`id_tim`: [0..3]), con las interrupciones habilitadas y el divisor de frecuencia pasado como segundo parámetro, usando la frecuencia de entrada mínima. Si se pasa un cero como divisor de frecuencia, detiene el *timer*.

El sistema electrónico de control generará una petición de interrupción (IRQ_CART) cada vez que se active un nuevo sensor. A esta petición se vinculará una RSI que denominaremos `rsi_sensor()`. Esta RSI se debe encargar del control de la fase actual del lanzamiento, modificando convenientemente los bits de activación de los bloques de electroimanes.

Para controlar el cambio de polaridad de los electroimanes se utilizará la RSI del *timer* 0: cada vez que se active la `rsi_timer0()`, se debe cambiar el estado de los bits de polaridad de al menos los bloques que estén activos en ese momento. Sin embargo, para simplificar el código se pueden cambiar los bits de polaridad de todos los bloques a la vez, ya que estos cambios solo afectaran a los electroimanes de los bloques activos.

Para determinar el divisor de frecuencia adecuado a cada cambio de polaridad se dispone de un vector global denominado `DivFreq[]`. Este vector contiene 30 valores negativos, que están precalculados para obtener la velocidad óptima de todos los cambios de polaridad. En un sistema real, los tiempos entre cambios de polaridad se deben ir reajustando dinámicamente según evoluciona el coche, puesto que este movimiento depende de muchos factores físicos (peso de los pasajeros, rozamiento, viento, vibraciones, etc.), pero para simplificar el problema vamos a suponer que este reajuste no será necesario.

Por su parte, el programa principal deberá atender a múltiples tareas de control de la montaña rusa. La tarea de preparación del lanzamiento se supone que ya está implementada dentro de

las tareas independientes. Esta tarea es compleja, puesto que hay que asegurar que el coche está en la posición inicial correcta, es decir, justo encima del primer bloque de electroimanes, con velocidad inicial cero. Para simplificar el problema, vamos a suponer que hay una variable global denominada `launch_ready`, que será cierta si ya se puede realizar el lanzamiento, en cuyo caso el operario podrá pulsar el botón START. Mientras se realiza el lanzamiento, hay que ir mostrando por pantalla el estado de los sensores. Se propone el siguiente código:

```
#define MAXDIVFREQ 30
const short DivFreq[MAXDIVFREQ]; // vector con los divisores de frecuencia
unsigned char ind_polx = 0;      // índice del cambio de polaridad actual
unsigned char phase = 0;         // fase actual del lanzamiento
unsigned char launch_ready = 0;  // indica si el lanzamiento está preparado

void main(void)
{
  unsigned char i;
  inicializaciones();
  do
  {
     tareas_independientes();
     if (launch_ready)           // si está preparado para lanzamiento
     {
       scanKeys();
       if (keysDown() & KEY_START)    // detectar pulsación START
       {
          phase = 1; ind_polx = 0;
          REG_EM = ¿_a_?;             // primera activación electroimanes
          prog_timer(0, DivFreq[0]);
          launch_ready = 0;
       }
     }
     else if (¿_b_?)             // si está realizando el lanzamiento
     {
       swiWaitForVBlank();
       for (i = 5; i > 0; i--)
       {
          show_sensor(i, ¿_c_?);
       }
       if (ind_polx == MAXDIVFREQ)    // si ya se han realizado todos
       {                              // los cambios de polaridad,
          phase = 0;             // volver al estado inicial
          REG_EM = 0;            // desactivar el último bloque de em
       }
     }
  } while (1);
}
```

Se pide:

`rsi_sensor()` y `rsi_timer0()` en lenguaje ensamblador, partes del programa principal marcadas con `¿_X_?` en lenguaje C.