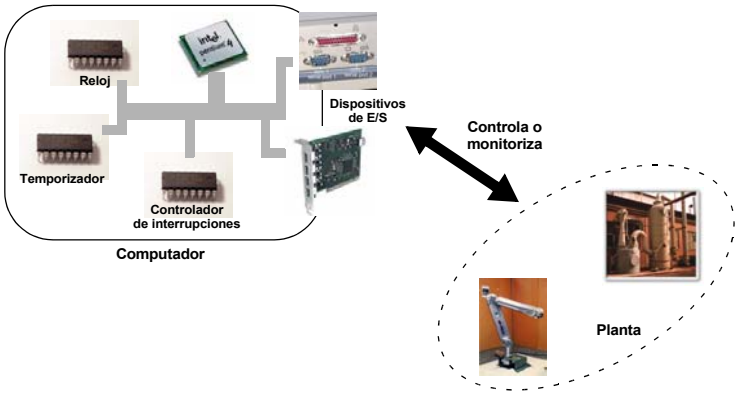
# Prácticas de la asignatura: Instrumentación de Tiempo Real (Curso 2005/2006)

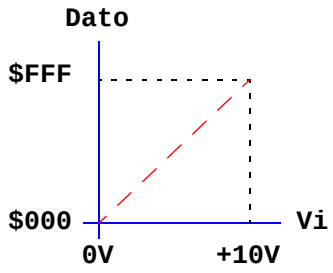
- Práctica 2.1: Generación de Formas de Onda
- Práctica 2.2: Captura y Reproducción de Señales
- Práctica 3.1: Control de un sistema de semáforos utilizando el puerto paralelo
- Práctica 4.1: Muestreo y reconstrucción de señales
- Práctica 5.1: Música utilizando el altavoz del PC
- Práctica 6.1: Control del nivel de agua de un depósito
- Práctica 6.2: Control de la posición de una plataforma móvil

# Práctica 2.1
# Generación de Formas de Onda

En la tarjeta AX5411 que se encuentra instalada en los computadores del laboratorio se dispone de un convertidor D/A con 12 bits de resolución. Para generar un valor en la salida del conversor se debe escribir sucesivamente en los dos registros de datos los bits que corresponden al valor. Primero debe ser escrito el byte menos significativo en el registro LoDA0 ($304) y luego debe escribirse el byte más significativo en el registro HiDA0 ($305). Sólo cuando ambos valores han sido escritos, el valor se hace efectivo en la salida (diferencia de voltaje entre los pines 23 y 25).

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| LoDato ($304) | D3 | D2 | D1 | D0 | X | X | X | X |
| HiDato ($305) | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 |




Realizar un programa que permita generar una forma de onda triangular o cuadrada con el periodo indicado por el usuario. Las formas de onda se almacenan como arrays:

```
type Forma_De_Onda is array (1 .. N) of Float;
Onda_Cuadrada : Forma_De_Onda;
Onda_Triangular: Forma_De_Onda;
```

La forma de espaciar en el tiempo los sucesivos valores de la forma de onda elegida es utilizando la sentencia delay según lo expuesto en el siguiente pseudocódigo:

```
  loop
    Pone nuevo valor en el convertidor D/A
    delay Duration (Periodo) / N;
  end loop;
```

A la vez que el programa va escribiendo los sucesivos valores en el convertidor D/A, deberá de ir mostrando el código hexadecimal escrito en los registros LoDato y HiDato.

Para mostrar un número en base hexadecimal se debe utilizar:

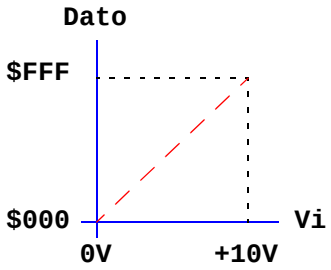
```
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
...
  Put ("Muestra el valor de N en hexadecimal:");
  Put (N, Base => 16);
```

# Práctica 2.2
# Captura y Reproducción de Señales

Objetivo: utilizando el convertidor A/D de la tarjeta AX5411, capturar sucesivos valores de una señal de entrada y generar la señal amplificada utilizando el convertidor D/A proporcionado por esta misma trajeta.

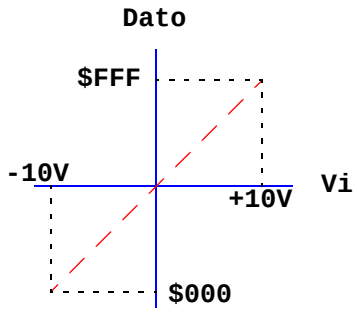

Convertidor Digital/Analógico (salida entre los pines 23 y 25):

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| LoDA ($304) | D3 | D2 | D1 | D0 | X | X | X | X |
| HiDA ($305) | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 |




Convertidor Analógico/Digital (pines 1 y 25):

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| LoAD ($300) | D3 | D2 | D1 | D0 | X | X | X | X |
| HiAD ($301) | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 |
| ASTA ($300) | X | X | X | X | X | X | X | X |
| STAT ($308) | B | X | X | X | X | X | X | X |




Para iniciar una conversión del A/D es necesario escribir cualquier valor en el registro **ASTA**. Un cambio de 1 a 0 en el bit 7 del registro **STAT** indica que la conversión ha finalizado y el valor resultante puede leerse de los registros **LoAD** y **HiAD**.

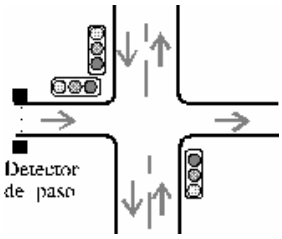
El pseudocódigo del programa es el mostrado a continuación:

```
with MaRTE_OS;
with IO_Interface; use IO_Interface;
with Basic_Integer_Types; use Basic_Integer_Types;
procedure Amplifica_Senal is
   --  Registros de E/S de la tarjeta AX5411
   Base_AX5411 : constant IO_Port := 16#300#;
   ASTA : constant IO_Port := 0 + Base_AX5411;
   CGA  : constant IO_Port := 1 + Base_AX5411;
```

```
   LoAD : constant IO_Port := 0 + Base_AX5411;
   HiAD : constant IO_Port := 1 + Base_AX5411;
   MUC  : constant IO_Port := 2 + Base_AX5411;
   STAT : constant IO_Port := 8 + Base_AX5411;
   CNTR : constant IO_Port := 9 + Base_AX5411;
   LoDA : constant IO_Port := 4 + Base_AX5411;
   HiDA : constant IO_Port := 5 + Base_AX5411;

   --  Lee_Entrada_AD
   --  Lee los registros correspondientes al convertidor A/D y
   --  convierte a voltios el código leido
   function Lee_Entrada_AD return Float is
   begin
      --  Comienza conversión
      Outb(..., ...);
      --  Espera a que la conversión haya finalizado
      loop
         exit when ...;
      end loop;
      --  Lee código de los registros del A/D
      ...
      --  Convierte a voltios y retorna el valor convertido
      ...
      return ...;
   end Lee_Entrada_AD;

   --  Configura_DA
   procedure Configura_DA is
   begin
      Outb (CNTR, 2#0_000_0_0_00#); --  disparo software
      Outb (MUC, 16#00#);           -- Convierte canal 1
      Outb (CGA, 0);                --  Ganancia 1
   end Configura_DA;

   --  Escribe_Salida_DA
   --  Toma un valor en voltios le convierte a su código binario
   --  correspondiente y le escribe en los registros del convertidor
   --  D/A
   procedure Escribe_Salida_DA (V : Float) is
   begin
      ...
   end Escribe_Salida_DA;

   V : Float;
begin
   Configura_DA;
   loop
      V := Lee_Entrada_AD;
      Escribe_Salida_DA (V*2.0);
   end loop;
end Amplifica_Senal;
```

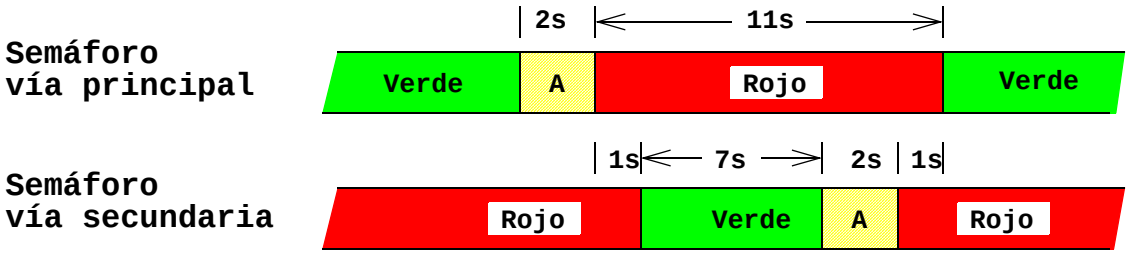
# Práctica 3.1
# Control de un sistema de semáforos utilizando el puerto paralelo

Realizar un programa que permita controlar un sistema de dos semáforos complementarios que regulan el tráfico de un cruce de una vía principal de tráfico intenso y una vía secundaria de tráfico ligero.




Los dos tipos de semáforos son complementarios, lo cual significa que mientras que el que controla la vía principal está verde o ámbar, el otro debe estar rojo, y viceversa.

Normalmente los semáforos están habilitando el paso por la vía principal, y permanece en ese estado si por la vía secundaria no hay coche esperando. En la vía secundaria hay un detector de coche en espera, y cuando en el mismo se detecta la presencia de un coche, debe comenzarse a iniciar el cambio de los semáforos, siempre que no se hubiese habilitado el paso de la vía secundaria en los 120 segundos anteriores. Cuando se produce la transición debe llevarse a cabo la siguiente secuencia de colores:




Emular este sistema, utilizando dos ternas de LEDs (Rojo, Amarillo y Verde) con los que se emulan los dos semáforos, y un pulsador con el que se emula el detector de coche en la vía auxiliar.

El pulsador se encuentra conectado a la entrada Acknowledge del puerto paralelo. Los LEDs se encuentran conectados a las seis líneas de datos menos significativos.

Estructura del programa:

```
with ...
procedure Practica_31 is

  function Manejador_Puerto_Paralelo (...) is
  begin
    ...
  end Manejador_Puerto_Paralelo;

begin
  loop
    espera que llegue coche
    pone primer estado semáforos
    delay ...
    pone segundo estado semáforos
    delay ...
    ...
    espera el tiempo que falta para los 120 segundos
  end loop;
end Practica_31;
```

# Práctica 4.1
# Muestreo y reconstrucción de señales

Utilizando los convertidores A/D y D/A de la tarjeta AX5411, realizar un programa que permita capturar y almacenar en un array 6000 muestras de una señal. Posteriormente el programa deberá reproducir repetidamente, a la frecuencia deseada por el usuario, los valores obtenidos.

La captura de datos se realizará por el canal de entrada analógica 0 (pines 1 y 25), programando la tarjeta para que genere una interrupción por cada fin de conversión.

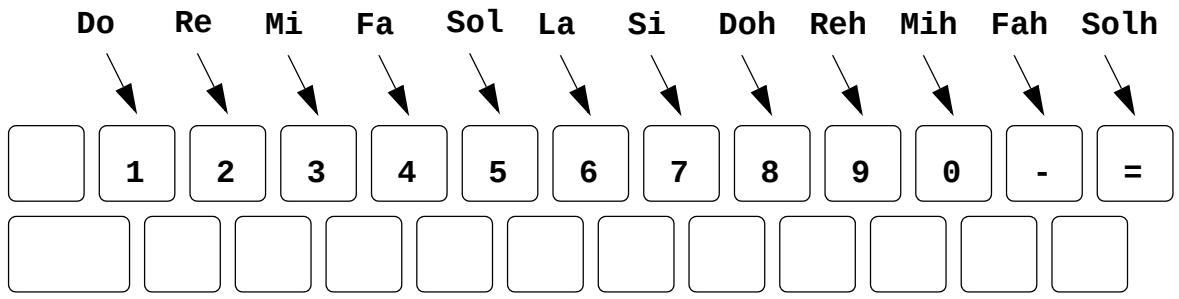
La reproducción de la señal se realizará utilizando el convertidor D/A 0 (pines 23 y 25).

El esquema del programa a realizar es el mostrado en el siguiente pseudocódigo:

```
with ...

procedure Muestreo_Y_Reconstruccion is

   --  Direcciones de los registros de E/S del convertidor
   ...

   --  Variables globales utilizadas por el manejador y el
   --  programa principal
   ...
   Datos : array (...) of Unsigned_16;
   ...


   --  Manejador de la interrupción
   function Manejador_AD (Area : in System.Address;
                          Intr : in Hardware_Interrupt)
                         return Handler_Return_Code is
      ...

begin
   instala manejador, configura tarjeta A5411, ...

   loop
      muestra por pantalla el número de datos que van capturados
      exit when se han capturado todos los datos
   end loop;

   pide periodo por pantalla
   loop
      for i in 1..6000 loop
         escribe dato en el convertidor D/A
         delay periodo/6000
      end loop;
   end loop;

end Muestreo_Y_Reconstruccion;
```

# Práctica 5.1
# Música utilizando el altavoz del PC

Realizar un programa que simule el comportamiento de un "piano" utilizando el altavoz del PC para generar las notas.




El periodo de cada una de las notas es el mostrado a continuación (puede copiarse de /home1/comun/ITR/practica51.adb):

```
type Nota is (Solh, Fah, Mih, Reh, Doh, Si, La, Sol, Fa, Mi,
              Re, Doo);
Periodo_Nota : constant array (Nota) of Float :=
               (Solh => 1.0/779.0,
                Fah  => 1.0/705.0,
                Mih  => 1.0/639.0,
                Reh  => 1.0/579.0,
                Doh  => 1.0/524.0,
                Si   => 1.0/475.0,
                La   => 1.0/430.0,
                Sol  => 1.0/389.0,
                Fa   => 1.0/352.0,
                Mi   => 1.0/319.0,
                Re   => 1.0/289.0,
                Doo  => 1.0/262.0);
```

# Práctica 6.1
# Control del nivel de agua de un depósito

Realizar un programa que permita controlar el nivel del agua de un depósito simulado por software. El nivel de agua puede ser modificado por el usuario utilizando el teclado.

El depósito se controla mediante una tarjeta AX5411 situada en la posición de E/S $300 (al igual que el depósito, la tarjeta también está simulada por software). La conexión al deposito es:

- Salida D/A 0: control de la apertura del grifo de entrada de agua en el depósito. Un valor $0 corresponde al grifo cerrado y un valor $FFF corresponde al grifo abierto a tope.
- Salida D/A 1: control de la apertura de la vávula de salida de agua. Un valor $0 corresponde a la válvula cerrada y un valor $FFF corresponde a la vávula abierta a tope.
- Entrada A/D: medida del nivel actual de agua en el depósito. Un valor $0 corresponde al depósito vacío y un valor $FFF corresponde al depósito lleno.
- Salida digital **DO0** (línea menos significativa del conector principal): cuando está a 1 enciende la luz roja de alarma, cuando vale 0 la luz es verde.

El programa de control constará de 3 tareas

- Tarea **Lee_Teclado**: se encarga de leer las teclas pulsadas por el usuario. Deberá responder ante las teclas:
  - "flecha arriba" (genera un carácter '8'): aumenta en un pequeño incremento el valor del nivel de agua deseado.
  - "flecha abajo" (carácter '2'): disminuye en un pequeño decremento el valor del nivel de agua deseado.
- Tarea **Controla_Nivel**: es la encargada de realizar el control del nivel de agua del depósito abriendo y cerrando el grifo y la válvula de salida. Esta tarea deberá tener un periodo de 1.0 seg. Realiza un simple control proporcional:

```
--  Si el nivel está por debajo del objetivo: abre el grifo
Consigna_Grifo := (Nivel_Actual - Nivel_Objetivo) * Kp;
--  Si el nivel está por encima del objetivo: abre el vávula
Consigna_Valvula := (Nivel_Objetivo - Nivel_Actual) * Kp;
```

- Tarea **Alarma_Nivel**: debe comprobar que el nivel no supere unos límites inferior y superior. Para ello deberá obtener el nivel actual con un periodo de 0.1 seg. Si detecta que se ha superado el límite superior deberá abrir totalmente la vávula de salida y cerrar el grifo (lo opuesto si el nivel baja por debajo del límite inferior). Además de las acciones anteriormente citadas, deberá mantener la luz de alarma en color rojo mientras se produce la alarma y cambiarla a verde cuando deja de producirse.

El pseudocódigo de las tareas es el mostrado a continuación:

```
with MaRTE_OS;
with Simulated_IO_Interface; use Simulated_IO_Interface;
with Basic_Integer_Types; use Basic_Integer_Types;
with Text_IO; use Text_IO;
with Ada.Calendar; use Ada.Calendar;
with Deposito;
with Ada.Exceptions;

procedure Control_Deposito is

   --  Direcciones de los registros de E/S de la tarjeta AX5411
   BASE_AD    : constant IO_Port := 16#300#;
```

```
Reg_LoDato : constant IO_Port := BASE_AD + 0;
Reg_HiDato : constant IO_Port := BASE_AD + 1;
Reg_CGA    : constant IO_Port := BASE_AD + 1;
Reg_MUC    : constant IO_Port := BASE_AD + 2;
Reg_ASTA   : constant IO_Port := BASE_AD + 0;
Reg_CLI    : constant IO_Port := BASE_AD + 8;
Reg_STATUS : constant IO_Port := BASE_AD + 8;
Reg_CNTR   : constant IO_Port := BASE_AD + 9;
Reg_ADL0   : constant IO_Port := BASE_AD + 4;
Reg_ADH0   : constant IO_Port := BASE_AD + 5;
Reg_ADL1   : constant IO_Port := BASE_AD + 6;
Reg_ADH1   : constant IO_Port := BASE_AD + 7;
Reg_DOUT   : constant IO_Port := BASE_AD + 3;

--  Datos compartidos entre las tareas
Nivel_Objetivo : Float := Deposito.Mx_Capacidad_Deposito / 2.0;
pragma Volatile (Nivel_Objetivo);
pragma Atomic (Nivel_Objetivo);
Nivel_Actual : Float;
pragma Volatile (Nivel_Actual);
pragma Atomic (Nivel_Actual);

Incremento_Nivel : constant Float :=
                          Deposito.Mx_Capacidad_Deposito / 200.0;

--  Tarea Lee teclado
task Lee_Teclado is
   pragma Priority (1);
end Lee_Teclado;

task body Lee_Teclado is
   Key : Character;
begin
   loop
      --  Espera una tecla
      Get_Immediate (Key);

      --  En función de la tecla leída:
      --   '8' => incrementa el nivel objetivo
      --   '2' => decrementa el nivel objetivo

      --  Muestra el valor del nivel objetivo actual, llamando
      --  para ello al procedimiento proporcionado por el paquete
      --  'Deposito'
   end loop;
end Lee_Teclado;

--  Tarea Controla Nivel
task Controla_Nivel is
   pragma Priority (4);
end Controla_Nivel;

task body Controla_Nivel is
   Proxima_Activacion : Ada.Calendar.Time := Ada.Calendar.Clock;
```

```
      Periodo : constant Duration := 1.0;
      Kp : constant Float := 200.0; -- Constante proporcional para
                                    -- el algoritmo de control
   begin
      --  Configura AX5411: ganancia 1, rango de canales a
      --  convertir 0 .. 0, disparo software, no usa interrupciones.

      --  Lazo de control
      loop
         Proxima_Activacion := Proxima_Activacion + Periodo;
         delay until Proxima_Activacion;

         --  Calcula apertura de grifo y válvula
         if Nivel_Actual < Nivel_Objetivo then
            --  Aplica el algoritmo de control para el grifo
            --  Abre grifo
            --  Cierra vávula salida

         else
            --  Aplica el algoritmo de control para la válvula
            --  Abre válvula
            --  Cierra grifo
         end if;
      end loop;

   exception
      when E : others =>
         Put ("Exception in Controla_Nivel: ");
         Put (Ada.Exceptions.Exception_Name (E) & " ");
         Put (Ada.Exceptions.Exception_Message (E));
   end Controla_Nivel;

   --  Alarma Nivel
   task Alarma_Nivel is
      pragma Priority (6);
   end Alarma_Nivel;

   task body Alarma_Nivel is
      Proxima_Activacion : Ada.Calendar.Time := Ada.Calendar.Clock;
      Periodo : constant Duration := 0.1;
      Nivel_Mn_Alarma : constant Float :=
        0.25 * Deposito.Mx_Capacidad_Deposito;
      Nivel_Mx_Alarma : constant  Float :=
        0.75 * Deposito.Mx_Capacidad_Deposito;
   begin
      loop
         --  Comienza conversión del convertidor A/D del AX5411

         Proxima_Activacion := Proxima_Activacion + Periodo;
         delay until Proxima_Activacion;

         --  Lee nivel actual del agua (seguro que ya ha acabado la
         --  conversión ya que el periodo es muy superior al tiempo
         --  de conversión de la tarjeta)
```

```
         --  comprueba alarmas
         if Nivel_Actual > Nivel_Mx_Alarma then
            --  Enciende luz alarma
            --  Abre salida y cierra grifo

         elsif Nivel_Actual < Nivel_Mn_Alarma then
            --  Enciende luz alarma
            --  Abre grifo y cierra salida

         else
            -- Anula alarma
         end if;
      end loop;
   exception
      when E : others =>
         Put ("Exception in Alarma_Nivel: ");
         Put (Ada.Exceptions.Exception_Name (E) & " ");
         Put (Ada.Exceptions.Exception_Message (E));
   end Alarma_Nivel;

begin
   null;
end Control_Deposito;
```

El paquete **Deposito** proporciona un conjunto de constantes y procedimientos que pueden ser utilizadas desde el programa de control:

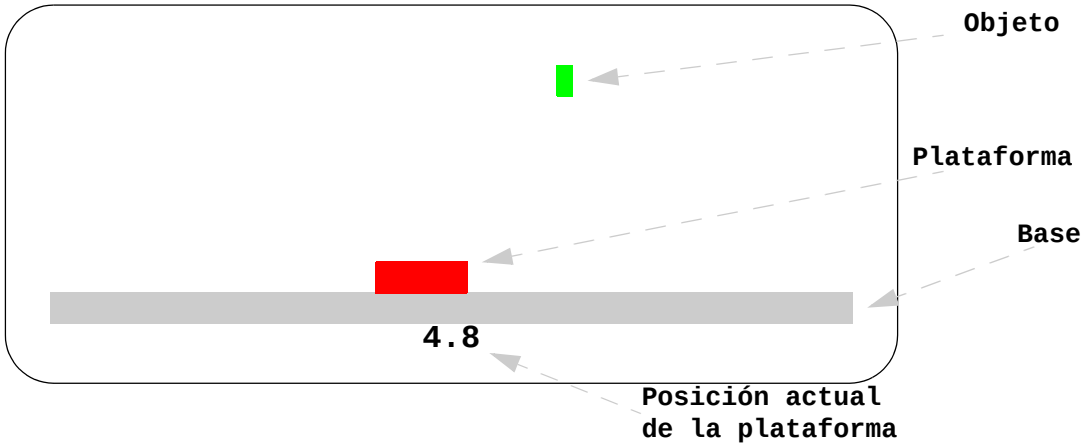
```
package Deposito is
   --  Constantes depósito
   Mx_Caudal_Grifo  : constant Float := 5.0; -- Litros/seg
   Mx_Caudal_Salida : constant Float := 5.0; -- Litros/seg
   Mx_Capacidad_Deposito : constant Float := 200.0; -- Litros
   Mx_Voltaje : constant Float := 10.0; -- Rango A/D: -10V .. 10V
                                        -- Rango D/A:   0V .. 10V

   procedure Muestra_Nivel_Objetivo (Obj : in Float);
end Deposito;
```

El pseudocódigo del programa está en el fichero **/home1/comun/ITR/practica61/control_deposito.adb**. En ese mismo directorio se encuentran los paquetes que permiten simular por software el funcionamiento del depósito y de la tarjeta AX5411. Crear el directorio **practica61** y copiar en él todos los ficheros que hay en **/home1/comun/ITR/practica61**

```
$ mkdir practica61
$ cd practica61
$ cp /home1/comun/ITR/practica61/* .
```

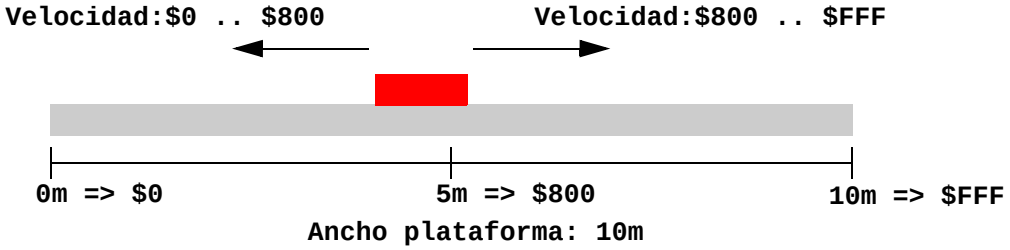
# Práctica 6.2
# Control de la posición de una plataforma móvil

Realizar un programa que permita controlar la posición de una plataforma móvil de manera que recoja los objetos que caen.

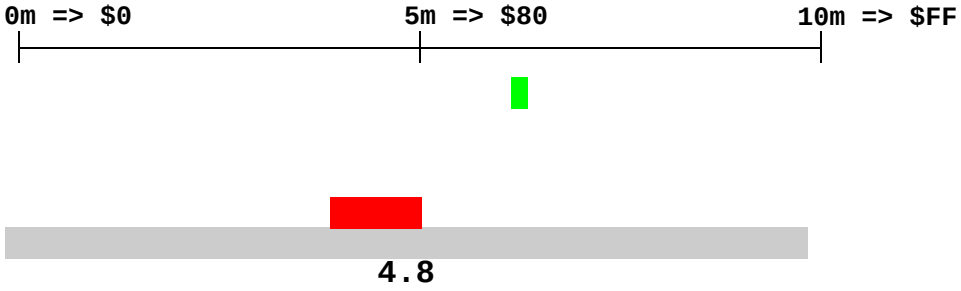



La plataforma se controla mediante una tarjeta AX5411 situada en la posición de E/S $300:

- Salida D/A 0: conectada al motor encargado de mover la plataforma. Un valor $0 corresponde a la máxima velocidad hacia la izquierda y un valor de $FFF a la máxima velocidad hacia la derecha.
- Entrada A/D: permite leer el potenciómetro que indica la posición actual de la plataforma. Un valor $0 corresponde a la plataforma situada en el extremo izquierdo de la base y un valor de $FFF a la plataforma situada en el extremo derecho..




Un detector conectado al puerto paralelo permite detectar el comienzo de la caída de un nuevo objeto. Cuando un nuevo objeto comienza su caída se produce un flanco de bajada en la línea **ACKNOWLEDGE** del puerto paralelo. La posición desde la que cae el objeto puede conocerse leyendo las líneas de datos del puerto durante los 0.03 seg siguientes a la generación del flanco. Una vez pasado este tiempo, el valor de las líneas de datos del puerto paralelo pasa a ser 0. Un código $0 corresponde a un objeto cayendo sobre el extremo izquierdo de la plataforma y un código $FF a un objeto cayendo sobre el extremo derecho.




Realizar un diseño del programa (tareas a utilizar, manejadores de interrupción, etc.) y mostrar al profesor antes de comenzar a escribir el código.