Rótulo oscilante

Palero Molina, Fernando

ÍNDICE

1.- Introducción

El objetivo de este documento es el de explicar detalladamente el código implementado para permitir que el **display oscilante XP3** pueda escribir mensajes, el pendulo utiliza la persistencia de la visión *para pintar* una imagen de luz en el aire.

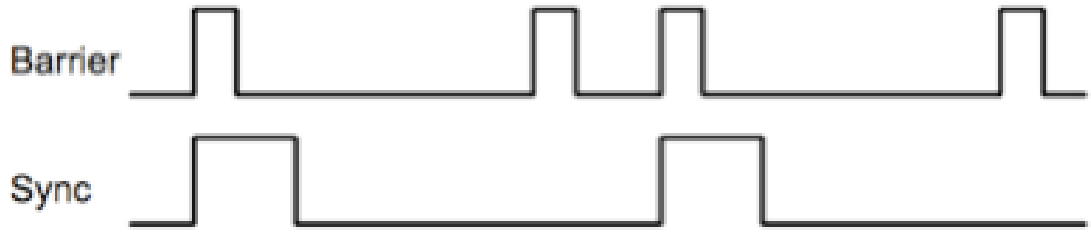
2.- Objetivos del trabajo

Para conseguir escribir los mensajes debemos primero seguir unas pautas:

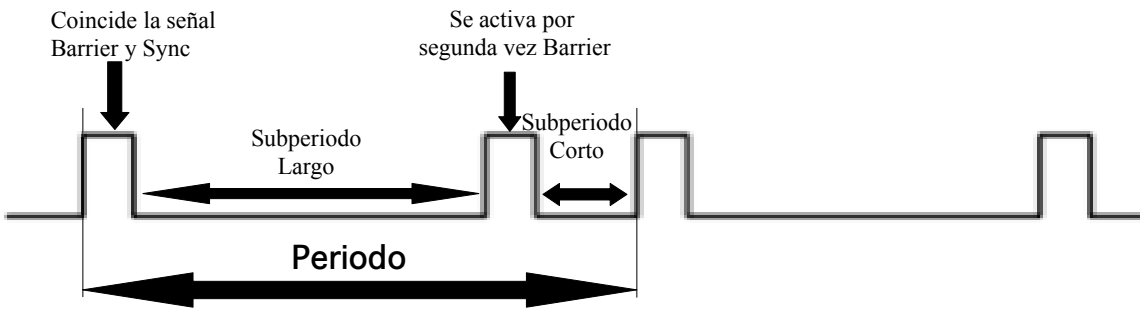
Es preciso sincronizar los instantes en que los LEDs se enciende y se apagan. Para ello se utilizan 2 señales del XP3 que son accesibles por el programa Barrier y Sync.

La señal Barrier, detecta el paso de la barra por un determinado punto de su recorrido, la proporciona un sensor que detecta ópticamente el paso de la barra. Dicha señal, da dos pulsos por periodo del péndulo: un pulso a la ida y otro a la vuelta del péndulo

La señal Sync, la cual es pseudo-periódica, se extrae del propio circuito de control del XP3. Esta señal da un pulso en cada periodo, utilizandose para sincronizar el péndulo, coincide aproximadamente con el pulso de vuelta de la señal Barrier.



Una vez detectadas las señales Barrier y Sync, debemos obtener los diferentes tramos del periodo:



Finalmente, una vez detectados los tramos, debemos obtener el inicio de ciclo, que es justo a la mitad de los dos pulsos de Barrier del tramo más corto.

3. – Descripción del diseño de la solución

La solución está compuesta por los ficheros:

- ***low_level_types:*** Contiene la definición de tipos como por ejemplo ***Byte***
- ***pendulum_io_sim:*** Este paquete simula el funcionamiento del péndulo. Su interfaz en muy similar a la del péndulo real.
- ***chars_8x5:*** Define la tabla de traducción de cada carácter en las cinco líneas de puntos con que se escriben.
- ***pendulo:*** Contiene todos los elementos para la solución al problema del péndulo. De los cuales, los importantes son 2 tareas:
  - ***calcular_timepo:*** Calcula el periodo y los diferentes tramos del este.
  - ***Pendul:*** Esta tarea se encarga de dibujar en el rótulo oscilante.

Examinando más a fondo el contenido de pendulo.adb podemos ver su funcionamiento más detalladamente, podemos ver:

Para las prioridades se utiliza una política de prioridad de techo y una gestión de prioridades de tipo pila con prioridades.

```
pragma Task_Dispatching_Policy (Fifo_Within_Priorities);
pragma Locking_Policy (Ceiling_Locking);
```

Procedure Barrier: se encarga de devolver el tiempo cuando hay un cambio de flanco en la señal de Barrier.

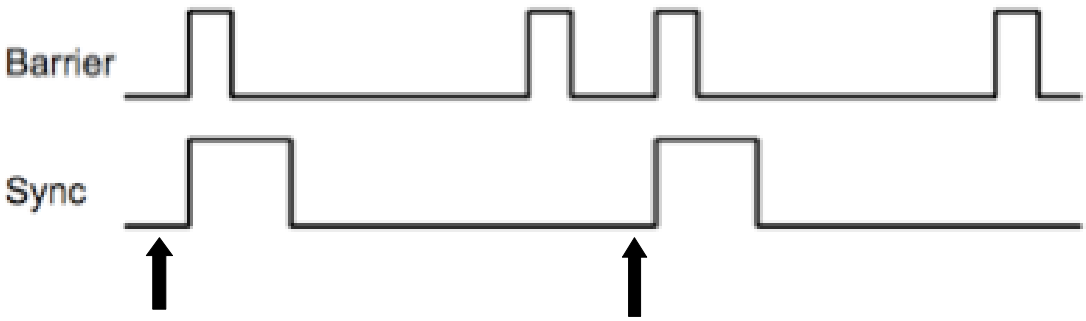
```
procedure Barrier (tiempo : out Ada.Real_Time.Time) is
      b : Boolean;
   begin

      -- obtnemos el valor booleano actual de la señal Barrier
      b := Get_Barrier;

      -- esperamos a que cambie de valor la señal Barrier
      while Get_Barrier = b loop
         delay 0.001;
      end loop;

      -- devolvemos el tiempo cuando cambia la señal
      tiempo := Ada.Real_Time.Clock;
      end Barrier;
```

Procedure Sync: la señal Sync se utiliza para sincronizar el rótulo oscilante, entonces, para conseguir empezar a calcular el periodo en el momento que se activa Sync, utilizando este procedimiento nos hace esperar hasta que la señal cambie a nivel alto.

```
procedure Sync is
   begin
       -- esperamos a que la señal Sync este a nivel bajo
       while Get_Sync = true loop
         delay 0.001;
       end loop;

      -- esperamos a que la señal Sync cambie a nivel alto
      while Get_Sync /= true loop
         delay 0.001;
      end loop;

   end Sync;
```

Task calcular_tiempo: como hemos comentado antes, esta tarea se encarga de calcula el periodo y los diferentes tramos del este.

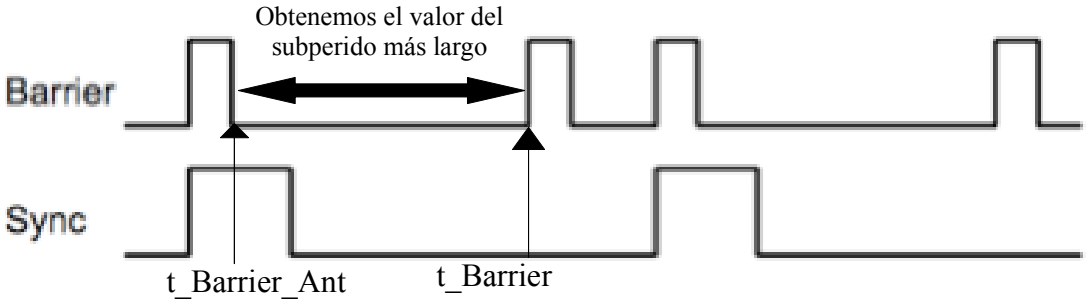
```
task calcular_tiempo is

      -- le asignamos la prioridad que debe ser menor a la utilizada
      --  por el pendulo simulado
      pragma Priority (System.Priority'Last-1);
   end calcular_tiempo;

   task body calcular_tiempo is
   begin

      loop

      -- Utilizamos el procedimiento Sync para empezar a calcular el
      -- periodo siempre desde el mismo sitio, en este caso lo
      -- empezamos a calcular siempre en el subperiodo más largo
         Sync;
```



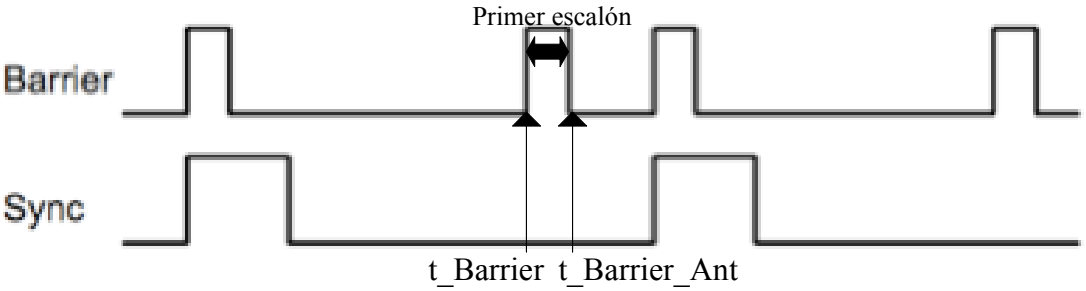
Esperamos a llegar al siguiente escalón de Sync para empezar a calcular los diferentes tramos del periodo

```
      -- Como nos hemos esperado a que la señal Sync este a nivel
      -- alto, también los estará la señal Barrier. Esperamos a que
      -- la señal Barrier este a nivel bajo y devolvemos el tiempo
      -- en ese instante
      Barrier(t_Barrier_Ant);
```

```
-- Como antes, volvemos a esperar que la señal Barrier cambie
-- de valor esta vez esperaremos que cambie de de nivel bajo a
-- alto y nos devolverá el tiempo de ese instante.
Barrier(t_Barrier);


--Con los instantes calculados y almacenados en la variables
--t_Barrier y t_Barrier_Ant, obtenemos el tiempo del subperiodo
--más largo
P1 := t_Barrier - t_Barrier_Ant;

Put_Line("1 : " & Duration'Image(To_Duration(P1)));
```

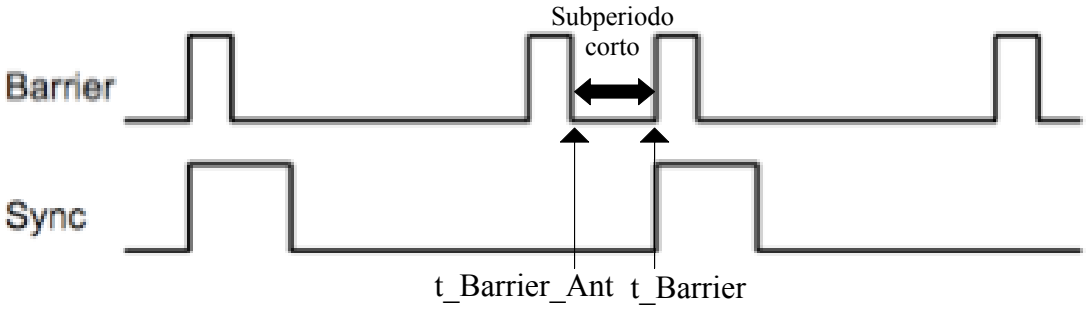



```
-- Después de obtener el subperiodo largo, viene la primera
-- activación de la señal de Barrier y haremos como antes, con
-- el valor obtenido en t_Barrier y t_Barrier_Ant, obtenemos el
-- valor del escalón

Barrier(t_Barrier_Ant);
B1 := t_Barrier_Ant - t_Barrier;
Put_Line("2 : " & Duration'Image(To_Duration(B1)));

-- Este instante, es cuando se ha obtenido el escalón, el cual
-- que coincide con el movimiento de vuelta del péndulo. Es
-- justamente el momento en que la señal Sync esta a nivel bajo
-- y Barrier activa.
-- Lo que vamos ha hacer es guardar cuando será la próxima vez
-- que suceda esto con la variable inicio y sumándole el
-- periodo total que es la suma de todos los tramos del
-- periodo, inicialmente cuando todavía no se han calculado los
-- tramos vale 3 segundos
```

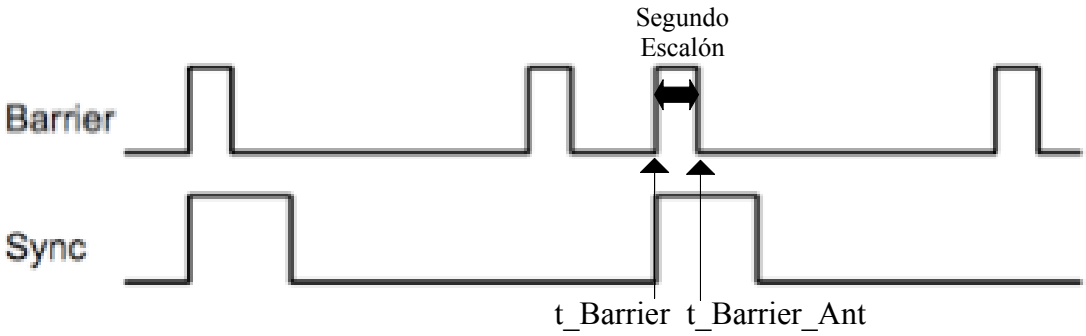



```
   Inicio := Ada.Real_Time.Clock;
   Inicio := Inicio + Periodo_Total;
   n := 1;
```

```
      -- Después de obtener el primer escalón, viene el subperiodo
      -- corto y volvemos ha hacer como antes, con
      -- el valor obtenido en t_Barrier y t_Barrier_Ant, obtenemos el
      -- valor del subperiodo más corto

      Barrier(t_Barrier);
      P2 := t_Barrier - t_Barrier_Ant;
      Put_Line("3 : " & Duration'Image(To_Duration(P2)));
```




```
      -- Finalmente obtenemos el segundo escalón, que va después el
      -- subperiodo corto y así completamos el periodo.
      Barrier(t_Barrier_Ant);

      B2 := t_Barrier_Ant - t_Barrier;
      Put_Line("4 : " &  Duration'Image(To_Duration(B2)));
```




```
-- Obtenemos el periodo total que es la suma de los 4 tramos
      Periodo_Total := P1 + P2 + B1 + B2;
-- Calculamos la fracción de tiempo necesaria, para saber en que
-- momento debemos encender los LEDs para formar las letras.

-- Para ello, primero obtenemos las lineas necesarias en cada
-- periodo.

-- Como el péndulo no va siempre a la misma velocidad,  debemos dejar
-- 10 lineas a cada lado antes de encender los LEDs y como máximo se
-- pueden visualizar hasta 12 letras y cada letra se representa con 5
-- lineas, teniendo en cuenta las 10 líneas y que cada letra debe
-- tener una línea de separación, sabemos que necesitamos solo para
-- la ida del péndulo 10+10+12*5+12 = 92. Como hace un movimiento de
-- ida y vuelta finalmente tenemos 92 * 2 = 184 lineas;

      Total := Periodo_Total/((10+10+72)*2);

      end loop;

   end calcular_tiempo;
```

Task pendul: esta tarea también la hemos explicado anteriormente, la cual se encarga de dibujar en el rótulo oscilante.

```
task pendul is
      -- le asignamos la prioridad que debe ser menor a la utilizada
      --  por el pendulo simulado
      pragma Priority (System.Priority'Last-1);
   end pendul;

   task body pendul is

      Periodo, Sig : Ada.Real_Time.Time_Span;
      Ini, Siguiente, Mitad : Ada.Real_Time.Time;

   begin

      loop

         if n > 0 then
          -- La variable Inicio la hemos calculado previamente en la
          -- tarea calcular_tiempo, la cual nos indica que el péndulo
          -- esta de  vuelta y el periodo se repite.
          -- Sabemos que se empieza a escribir en el extremo de
          -- la izquierda del rótulo oscilante, por lo tanto una vez
          -- pasado el tiempo de la variable inicio debemos volver a
          -- esperar para que el péndulo este en la posición de
          -- inicio, cuya espera correspondo a la mitad del
          -- subperiodo corto. La cual se calcula con el
          -- procedimiento “Esperar”.
            Ini := Inicio;
            delay until Ini;
            Esperar;

           Periodo := Periodo_Total;
           Sig := Total;

-- Una vez ya estamos en e inicio empezamos ha encender los LEDs

            for I in  0 .. 71 loop

               Siguiente := Ada.Real_Time.Clock;
               Siguiente := Siguiente + Sig;
               delay until Siguiente;
               Set_Leds(cadena(I+1));

            end loop;

            Reset_Leds;

-- Una vez terminamos de encender las 72 lineas de LEDs, debemos
-- esperar a para dibujar los LEDs a la vuelta, esto la hacemos con
-- la variable Mitad. El tiempo de espera corresponde a las 20 lineas
-- en blanco.

             Mitad := Ada.Real_Time.Clock;
             Mitad := Mitad + 20*Sig;
            delay until Mitad;
```

```
            for I in  0 .. 71 loop

               Set_Leds(cadena(72-(I)));
               Siguiente := Ada.Real_Time.Clock;
               Siguiente := Siguiente +Sig;

               delay until Siguiente;

            end loop;

            Reset_Leds;

         end if;

      end loop;

   end pendul;
```

Procedure Esperar: Este procedimiento se encarga de hacer esperar el tiempo necesario para que el rótulo oscilante esté en la posición inicial para dibujar.

```
   procedure Esperar is
      Next : Ada.Real_Time.Time;
      aux : Ada.Real_Time.Time_Span;
   begin
      aux:= Total;
      Next := Ada.Real_Time.Clock;

-- Esperamoas hasta llegar al inicio más las 10 lineas en blanco.

      Next := Next + ((P2)/2) + 10*aux;
      delay until Next;

   end Esperar;
```

4. – Principales dificultades encontradas y solución que se les ha dado.

Una dificultad encontrada es que el péndulo no hace un recorrido lineal, con la consecuencia que la velocidad no es lineal, en realidad el rótulo oscilante se detiene y se acelera en los extremos. Esto produce un defecto de linealidad, esto se apreciará porque los caracteres del centro quedarán más separados en el centro del recorrido que en los extremos.

Una solución posible es, una vez calculado el tiempo entre líneas explicado anteriormente, debemos de hacer que espere el tiempo correspondiente a 10 lineas a ambos lados del recorrido.