SISTEMAS DE TIEMPO REAL

FACULTAD DE INFORMÁTICA
UNIVERSIDAD POLITÉCNICA DE VALENCIA

Description of the oscillating display

Jorge Real, 2011

Hardware description

The oscillating display is built from the XP3 programmable message display. Figure 1 shows various pomotional images of this device.

Figura 1. The programmable message display XP3

The display consists of an oscillating bar that swings at an approximate rate of 8 times a second. The free end of the bar contains an array of 8 Light Emitting Diodes (LEDs) that may be switched on and off in order to display (portions of) characters. Due to the human eye's visual persistence, the effect of quickly switching the LEDs on and off is a *floating display* as shown in Figure 1.

The following image has been obtained from a photograph of the display showing the message "Ya funciona!".

In order to print readable messages in the oscillating display it is necessary to carefully synchronise the switching of the LEDs. Two signals of the pendulum are available for reading from a program. First, the Barrier signal allows detecting the pendulum bar crossing a particular point in its run. Second, the XP3 keeps the bar oscillating by means of an electromagnet to which a pseudo-periodic signal is applied, the so called Sync signal. Figure 2 shows a simple diagram of the pendulum parts.

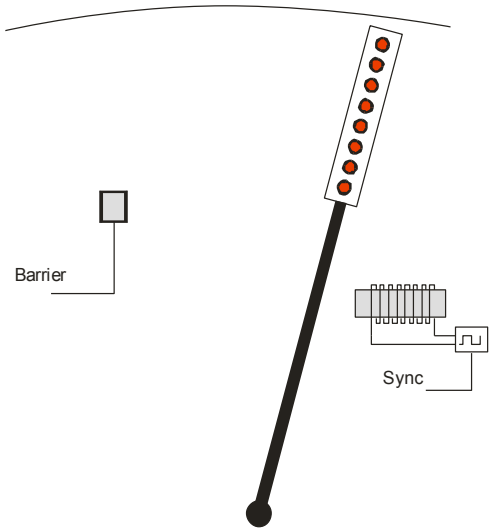


Figura 2. Simplified diagram of the XP3

The Barrier signal is obtained from a sensor that optically detects when the pendulum crosses in front of it. This sensor is located in the left part of the pendulum run, when you are facing the LEDs. Barrier therefore causes two pulses at each pendulum period. On the other hand, the Sync signal is produced by the control circuit of the pendulum. This signal gives one pulse per period to keep the pendulum oscillating. The **approximate** relation between the Sync and Barrier signals is shown in Figure 3. As you can see, there is a rising edge of Sync when the pendulum crosses the barrier from left to right.

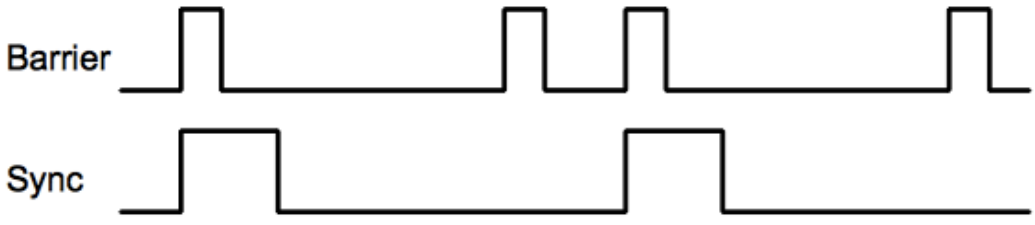


Figura 3. Signals obtained from the pendulum

Accessing the pendulum from software

Access to the pendulum is provided by the package `Pendulum_IO`. This package provides the following interface:

```
with Low_Level_Types; use Low_Level_Types;

package Pendulum_IO is

   function Get_Sync return Boolean;    -- True if Sync=1; False otherwise
   function Get_Barrier return Boolean; -- True if Barrier=0; False otherwise

   procedure Reset_Leds;                -- Blanks LEDs column
   procedure Set_Leds (B : in Byte);    -- Lights LEDs whose position=1 in B

end Pendulum_IO;
```

The package `Low_Level_Types` contains the definition of types such as `Byte`.

Simulator

The package Pendulum_IO_Sim simulates the signals generated by the real pendulum and displays the value of the LEDs at regular intervals of time, trying to emulate the movement of the pendulum. The interface to this package is almost identical to the interface of Pendulum_IO (see above.) so that moving from the Simulator to the real pendulum requires a minimum number of changes. The specification for Pendulum_IO_Sim is as follows:

```
with Low_Level_Types; use Low_Level_Types;
with Ada.Real_Time;   use Ada.Real_Time;

package Pendulum_Io_Sim is

   -- This constant defines the simulated pendulum period.
   -- The actual XP3 period is 114 ms
   -- Keep this constant above 2 seconds
   Oscillation_Period : constant Time_Span := Milliseconds(3000);

   function Get_Sync return Boolean;    -- Returns (Sync=1)
   function Get_Barrier return Boolean; -- Returns (Barrier=1)

   procedure Set_Leds (B : in Byte);   -- Lights LEDs whose position is 1 in B
   procedure Reset_Leds;               -- Blanks LEDs column.
                                       -- Equivalent to Set_Leds(0)
end Pendulum_Io_Sim;
```

The constant Oscillation_Period determines the oscillation period of the simulated pendulum. A value of 3 seconds is adequate for the computers in the laboratory.

The Simulator imitates the signals generated by the pendulum and uses a graphical window to display the binary value of a variable representing the state of the pendulum's LEDs. That variable is accessible through the subprograms Set_Leds and Reset_Leds in the same way as when you use the real pendulum through the package Pendulum_IO. Figure 4 shows the aspect of the Simulator window for an application displaying the message "It's working".

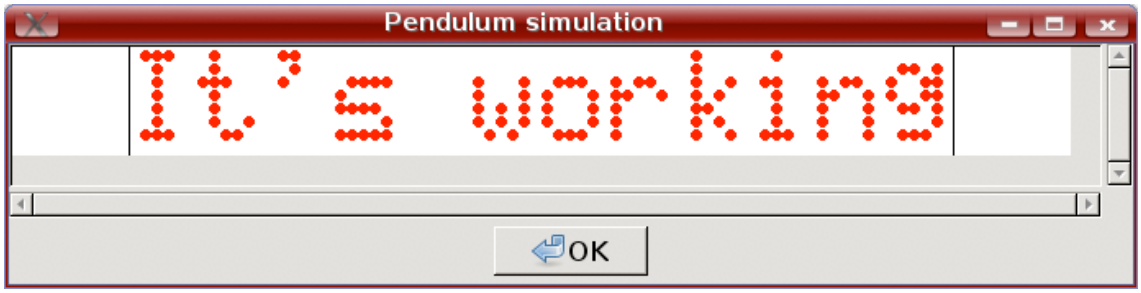


Figure 4: Graphical window of the simulated pendulum

The white background represents the whole pendulum run. The two black lines bound the usable area of the diplay. Given the varying speed of the pendulum, it is not convenient to print any text beyond those limits: the speed at both ends is minimal and any text displayed in that area would look just too narrow.

Considerations when using the simulator

Since the interface of the simulator is the same as for the real pendulum, changing from a simulated to a real version of our program is almost as simple as changing the with and use clauses to import one package or the other. Nevertheless, the simulated pendulum is justa an approximation, hence you'll have to take the following considerations into account:

- The period of the simulated pendulum is much larger than the period of the real pendulum. Instead of around 115 ms, the simulated period defaults to 3 seconds. The simulator period is defined by the variable Oscillation_Period, declared in the specification of Pendulum_IO_Sim (file pendulum_io_sim.ads.)
- The simulator itself uses the priority level Priority'Last for its own tasks and protected objects. You must take this into account when assigning priorities to your application's tasks and protected objects so that they do not interfere the simulator.
- Since the Sync and Barrier signals are produced by simulator tasks, you must give these tasks the chance to actually execute. In other words, you must not implement busy waits on Barrier or Sync. Instead, let some time elapse between consecutive samples of these signals as in the following example:

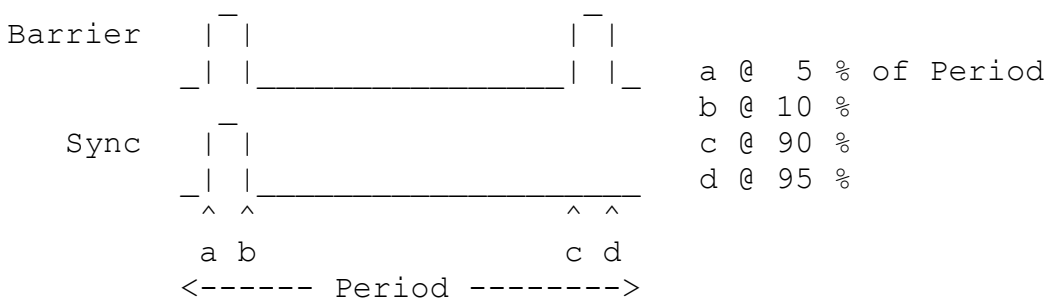
```
while Get_Sync loop -- Wait for Sync to fall
   delay 0.001; -- delay 1 ms
end loop;
```

Simulator modelling of the signals Barrier and Sync

Here we explain how the Simulator generates the signals Barrier and Sync. This is just an approximation, not to be taken as the exact behaviour of the pendulum. Don't take this timing as the real pendulum's timing: the real signals may last longer, although the approximation is good enough.

Barrier gives two pulses every oscillator period, corresponding to the LED bar crossing the barrier detector. Sync gives only one pulse per period, *more or less* coinciding with one of the two Barrier pulses, when the pendulum travels to the farthest end away from

the barrier detector. The following figure shows the model followed by the simulator, which approaches the timing relationship of the signals in the real pendulum:



Let me insist: this is just how the simulator emulates the signals produced by the real pendulum. Your program does not have to assume these particular instants of the period for the signals to change. Instea, you should pay attention at when the signals change.

The project to develop

The minimum requirement for this project is to implement an Ada program that makes the pendulum display a 12 character string.

The program must sample the Barrier signal to calculate when does the oscillation period start, and modify the LEDs' state at the right times for the string to be properly displayed. Once a value for the LEDs is output (by means of Set_Leds), you must switch the LEDs off again after 350 microseconds. (Nevertheless, **in the simulated pendulum** it is not necessary, nor convenient, to switch the LEDs on and off. Just call Set_Leds at the right times and deal with the flashing time when you work with the real pendulum.)

The package `Chars_8x5` is provided to facilitate the transformations between characters and their graphical representation. It defines the values for a matrix of 8*5 dots used to represent characters. You must add a blank line after each character to keep them separate from each other. This translation matrix is defined as follows:

```
Char_Map : array (Character range ' '..'~',0..4) of Byte;
```

For example, position ('A',0) contains the first dot line for the character 'A' (the leftmost line).

It is well known that the speed of the pendulum is not uniform accross its run. The effect of this is that, if we keep a regular interval between dot lines, the center characters (where the pendulum runs faster) will be wider than the characters at both ends of the run (where the linear speed decreases to zero and then changes direction). This effect is visible in the figure on page 1. The simulator however ignores the effect of the varying speed.

Some ideas to start...

The first thing you should do is to check whether you are detecting the Barrier and Sync signals correctly when the pendulum oscillates. From that moment on, measure the pendulum period, since you'll need that value to derive other delays such as the time

between two consecutive lines, or the time when the pendulum starts a new period. Check that the period you measure coincides with the value of the constant `Oscillation_Period`. We'll assume that the start of the period occurs when the pendulum is at the leftmost end, when you ara fecing the LEDs side of the pendulum.

The start of the pendulum cycle occurs just in the middle of the two closest Barrier pulses. This will be the time when we must start displaying the text. Leave the first 10 lines of text blank to avoid displaying too narrow characters.

It is convenient to have a task sample the Barrier signal at the proper frequency and determine at what point in time the different signal changes occur. Back to the previous figure, interesting points would be those marked as a, b, c and d. With those values updated, you can estimate when the next period will start. You'll need to have the period constantly measured so that you calculate the next start with respect to the immediately previous period.

Once you have solved this, you may want to try to display a simple pattern (e.g., a vertical line everytime the pendulum crosses the barrier) and then try to display a string. This makes a total of 2*(10+(12*6)+10) lines per pendulum period.