6. Very Simple Delays

The microcontrollers in our boards are clocked at quite high frequencies. These frequencies are only small fractions of what you could expect in a desktop or server computer, but for the tasks that microcontrollers are usually used for this is not a bad number. In fact, this frequency is high enough that a typical microcontroller spends most of its time waiting for something to happen and in many systems it is a common design assumption that individual actions are *immediate* and most of the time is spent doing nothing.

The "waiting" part of embedded system design is actually quite a complex problem and even though microcontrollers offer extensive functionality that can be used for different kinds of waiting, we will start with something extremely simple - as simple as twiddling thumbs, which is actually a very good analogy of what we are going to do.

In short - let's waste some time.

Burning processor cycles is a possible way to waste time so we can extend our `Utils` package with another procedure, appropriately called `Waste_Some_Time`. The new package specification is:

```
package Utils is

   procedure Spin_Indefinitely;

   procedure Waste_Some_Time;

end Utils;
```

and the package body is:

```
package body Utils is

   procedure Spin_Indefinitely is
   begin
      loop
         null;
      end loop;
   end Spin_Indefinitely;

   procedure Waste_Some_Time is
      Iterations : constant := 100_000;
   begin
      for I in 1 .. Iterations loop
         null;
      end loop;
   end Waste_Some_Time;

end Utils;
```

The `Waste_Some_Time` procedure contains a simple loop that does "nothing", but repeats that "nothing" hundred thousand times, thus implementing an observable delay - note however, that the time will be different for each board, as their microcontrollers have different processing speeds. It is similar and equivalent in effect to the following loop in C:

```
   const int iterations = 100000;
   int i;
   for (i = 1; i <= iterations; ++i)
   {
   }
```

We can hope that the compiler will not be smart enough to remove this trivial loop during code optimization and indeed, without any optimization options it accepts our intent and leaves the loop as is.

Now we can reuse the other packages and the linker script from the previous chapter and write the program that constantly changes state of pin 12 with observable delays between changes:

```
with Pins;
with Utils;
```

```
package body Program is

   procedure Run is
   begin
      Pins.Enable_Output (Pins.Pin_11);
      Pins.Enable_Output (Pins.Pin_12);

      loop
         Pins.Write (Pins.Pin_12, True);
         Utils.Waste_Some_Time;

         Pins.Write (Pins.Pin_12, False);
         Utils.Waste_Some_Time;
      end loop;
   end Run;

end Program;
```

Feel free to experiment with the "right" value for the `Iterations` constant to get the required delay and don't hesitate to play with some LEDs during this process (hint: pin 13 has an associated LED already installed on the board).

Note that the above method for wasting time is only our first attempt at introducing some notion of "waiting" to our embedded projects. In later chapters we will come up with much better methods that rely on hardware features to introduce not only delays, but also time-awareness to our programs.

Previous: Digital Output, next: Random Numbers.
See also Table of Contents.

Did you find this article interesting? Share it!

SHARE

Copyright © 2007-2017, Inspirel