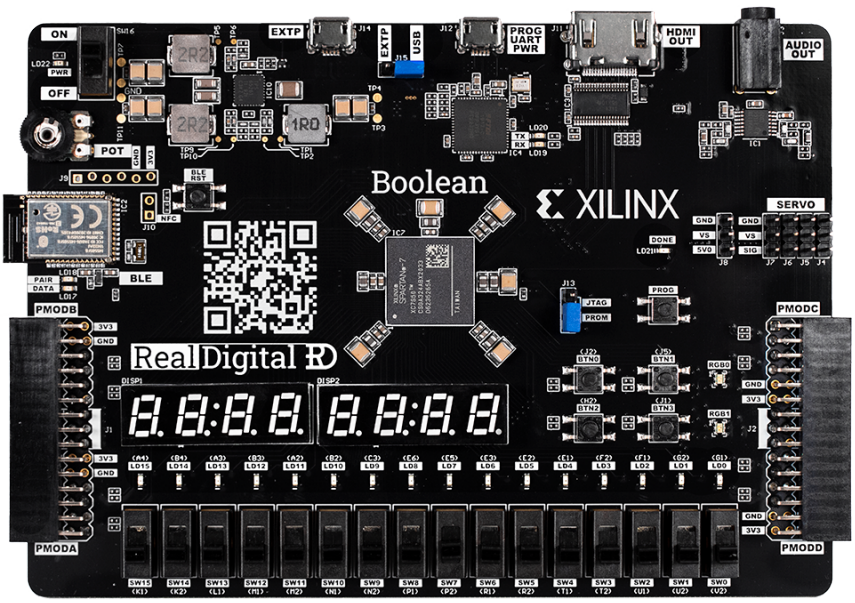
Digital Electronics – Lab 3

LED blink

Kent Abrahamsson, Andreas Lönn

2026-01-18



Revision history

Date	Author	Description
2023-03-28	Kent Abrahamsson	First edition
2023-04-01	Kent Abrahamsson	Minor corrections
2023-05-01	Kent Abrahamsson	Corrected process name in code template.
2025-02-20	Kent Abrahamsson	Clarified force command examples.
2026-01-18	Andreas Lönn	From Altera to Xilinx

Contents

1 General

1.1 Background

This document is intended to be used as Lab instructions for the LED blink lab in the VHDL course.

1.2 Intended outcome

In this assignment you shall write logic synchronous to the 100 MHz clock signal located on the development board. The idea is to give basic knowledge on how to create a precise delay that may be required in different applications. In this application the only functionality that is implemented is to blink a LED output in 1 Hz.

1.3 Checkpoints

In order to Pass the lab assignment the student shall be able to show that the following tasks have been fulfilled.

- Properly written VHDL implementation. Use proper indentation, spaces are cheap! Add comments when you feel it's suitable.
- A simulation shall be performed.
- The RTL Analyzer shall be viewed and analyzed.

Tip: Check out the simulator tips in the end of this document before simulating.

2 Lab instructions

2.1 General

In many designs it is critical to be able to generate precisely controlled delays. This becomes fundamental when using different data communication methods, such as SPI, I2C, RS-232, RS-485 etc. Generating correct delays are also important when interfacing with external components that often works slower than FPGA internal clock frequency, e.g. an alphanumeric LCD.

The only way to generate a precise delay with a CPLD/FPGA is to count clock cycles from a clock oscillator. This is a very good example of a synchronous design, where we store the counted number of clock cycles in a number of D-registers (using a signal).

In this assignment you will design a counter that counts a number of clock pulses from the 100 MHz clock source on the Boolean-board. The counter process shall then generate a short pulse (1 clock cycle long) on an internal signal. This internal signal is then checked by another process that toggles the state of an indicator LED. In other words, the overall goal of the assignment is to blink with a LED.

On the Boolean board there is a 100MHz clock oscillator available (the small golden component), see Figure 1.

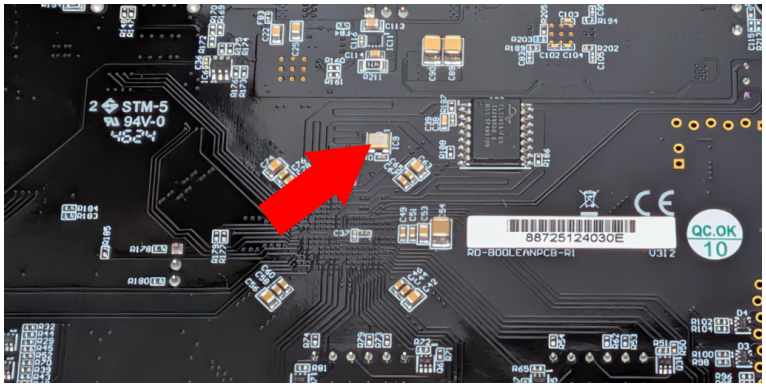


Figure 1: The 100Mhz clock

The clock source generates a "perfect" square wave signal with a 50% duty cycle with the given frequency. The clock oscillator is connected to a clock input on the FPGA.

2.2 Lab Task 1

The goal with the exercise is to blink with an LED with frequency 1Hz. This means that the LED shall be ON for 500ms and OFF for 500ms. You will need to calculate the correct amount of 100 MHz clock cycles in order to generate the required delay.

When using an FPGA it is very simple to be 100% sure that the timing of delays matches the exact correct number of clock cycles required.

Your design MUST be organized using two different processes, as per Figure 2.

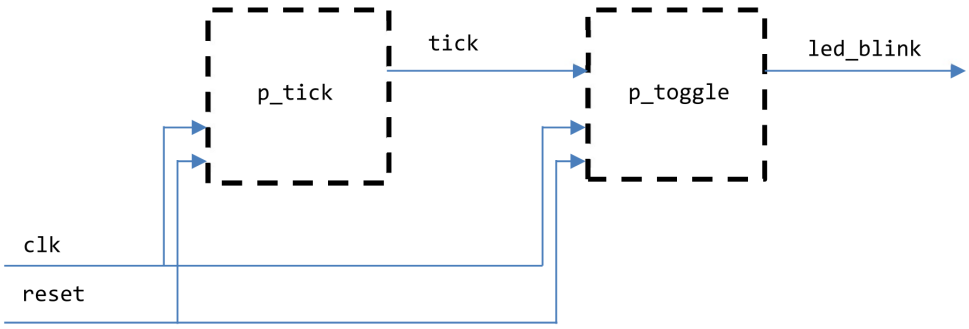


Figure 2: Diagram of the two processes

`p_tick` is a process which controls a counter which is incremented every cycle (when reset is inactive), after 500 ms the tick signal is set high one clock cycle and the counter is restarted again.

`p_toggle` is a process which toggles an internal `led_blink` signal. This `led_blink` signal is then routed to one of the LED outputs on the development board.

To get you started the code template below is given.

It's always a good idea to keep entity names connected to the circuit board signal names, e.g. `btn`. You can connect internal signals to the entity signals to make it easier to understand the functionality of the different entity signals as shown in the template that you received with the lab.

3 Simulation tips

3.1 Tip 1

When you run your design in the simulator, it's very inconvenient to simulate for several seconds. To ease the simulation, temporarily change the number of clock pulses you count, to a MUCH lower value, e.g. 25. This will still test the functionality of your code but keep the simulation time much shorter.

When you are done with the simulation, DO NOT forget to change back before synthesis.

3.2 Tip 1b (advanced)

One way to avoid needing to change the constant between simulation and synthesis is to use `generic`. We might not have covered it yet in the lectures. Below is a simple example of how it can be used. The `G_COUNT_MAX` can be used in your VHDL code.

```
entity top is
  generic (
    G_COUNT_MAX : integer := 50_000_000  -- default for synthesis
  );
  port (...);
end top;
```

In the testbench the generic value can be overridden:

```
...
  uut : entity work.top
    generic map (
      G_COUNT_MAX => 50    -- << simulation-friendly value
    )
    port map (...);
...
```

3.3 Tip 2

In your testbench, you can create multiple processes for doing different things. For example, you could create a process that automatically generates a clock signal, or use non-synthesizable VHDL to directly generate the clock signal.

```
...
architecture sim of tb_top is
  signal clk : std_logic := '0';
  ...
begin
  -- 100 MHz clock
  clk <= not clk after 5 ns;

  uut: entity work.top
    port map (
      clk => clk,
      ...
    );
...
```