VHDL – Lab 3

LED blink

Address:
Mor Kerstins väg 27
511 53 Kinna
Sweden

Tel:
+46-703-748429

Registration no:
559037-6603

E-mail:
kent@elekteknik.se

Revision history

Date	Author	Description
2023-03-28	Kent Abrahamsson	First edition
2023-04-01	Kent Abrahamsson	Minor corrections

Address:
Mor Kerstins väg 27
511 53 Kinna
Sweden

Tel:
+46-703-748429

Registration no:
559037-6603

E-mail:
kent@elekteknik.se

1 Table of Contents

Address:
Mor Kerstins väg 27
511 53 Kinna
Sweden

Tel:
+46-703-748429

Registration no:
559037-6603

E-mail:
kent@elekteknik.se

2 General

2.1 Background

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

2.2 Intended outcome

In this assignment you shall write logic synchronous to the 50 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.

2.3 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 ModelSim simulation shall be performed.
- The RTL (Netlist) viewer shall be viewed and analyzed.

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

Address:
Mor Kerstins väg 27
511 53 Kinna
Sweden

Tel:
+46-703-748429

Registration no:
559037-6603

E-mail:
kent@elekteknik.se

3 Lab instructions

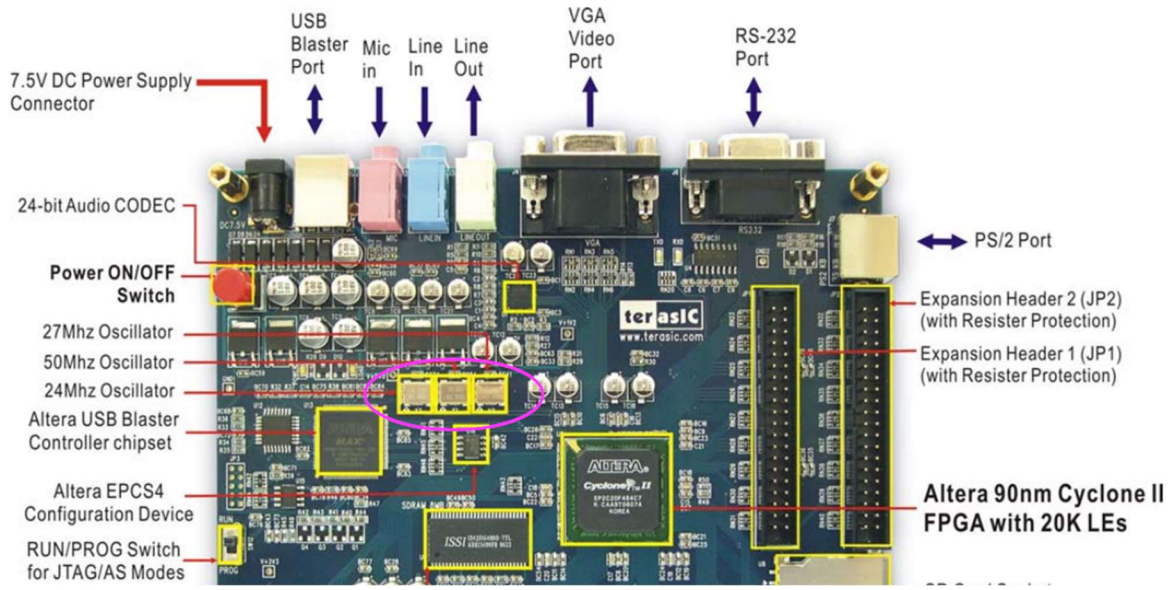
3.1 General

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 50 MHz clock source on the DE1-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 DE1 board there are three different clock oscillators available (the small golden components):



Different clock oscillators may be needed for different applications, in our application (for this course) we will try to keep everything synchronous with the 50 MHz signal, and only use that one. Note that there are also a 27 MHz, and a 24 MHz oscillator mounted on the DE1 board, this differs between the different development boards available, but 50 MHz is available on all development boards.

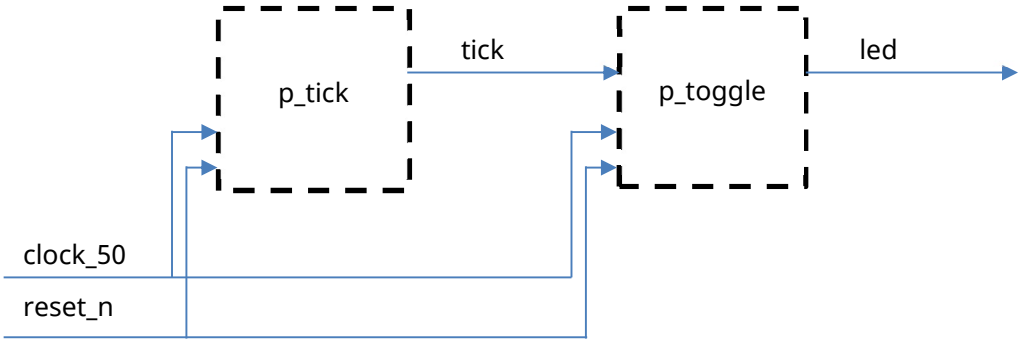
The clock sources generate a “perfect” square wave signal with a 50% duty cycle with the given frequency. Each clock oscillator is connected to a dedicated clock input on the FPGA, see the Development board User Manual to figure on what the name of these pins are.

3.2 Lab Task 1

The goal with the exercise is to blink with a 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 50 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.



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 signal, this led signal is then routed out to one of the LED outputs on the development board.

To get you started the code template below is given.

```
library ieee;
   use ieee.std_logic_1164.all;
   use ieee.numeric_std.all;

entity top_level is
port(
   clock_50 : in  std_logic;  -- 50 MHz clock
   sw       : in  std_logic_vector(9 downto 0);
   key_n    : in  std_logic_vector(3 downto 0);
   ledr     : out std_logic_vector(9 downto 0);
   ledg     : out std_logic_vector(7 downto 0)
   );
end entity top_level;

architecture rtl of top_level is

   -- Constant and type declarations
   constant c_500ms_count_max    : integer := <value here>;

   -- Internal signal declarations
   signal reset_n                : std_logic;
   signal counter_500ms          : integer range 0 to c_500ms_count_max;
   signal tick                   : std_logic;
   signal led                    : std_logic;
begin
   -- Assign internal signals from entity inputs
   -- Active low reset reset from KEY0
   reset_n  <= key_n(0);

   -- turn off unused LED outputs
   ledr        <= (others => '0');    -- USE ONE OF THESE FOR THE TOGGLE LED
   ledg        <= (others => '0');

   p_tick : process(clock_50, reset_n)
   begin
      -- fill in tick process functionality here
   end process p_tick;
   p_toggle : process(clock_50, reset_n)
   begin
      -- fill in toggle process functionality here
   end process p_tick;
end architecture rtl;
```

4 Simulation tips

4.1 Tip 1

When you run your design in Modelsim, 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 and Place and Route.

4.2 Tip 2

In larger simulations that uses a clock (but still forcing signals in the transcript window) it is not very smooth to force the clock signal up and down every clock cycle. The force command have support for periodic signals which help a lot in this case.

By adding information on when a signal is set high and the period time of the signal it is easy to create a clock signal in ModelSim. Study the following command:

```
force clk 0 0, 1 10 ns -r 20 ns
```

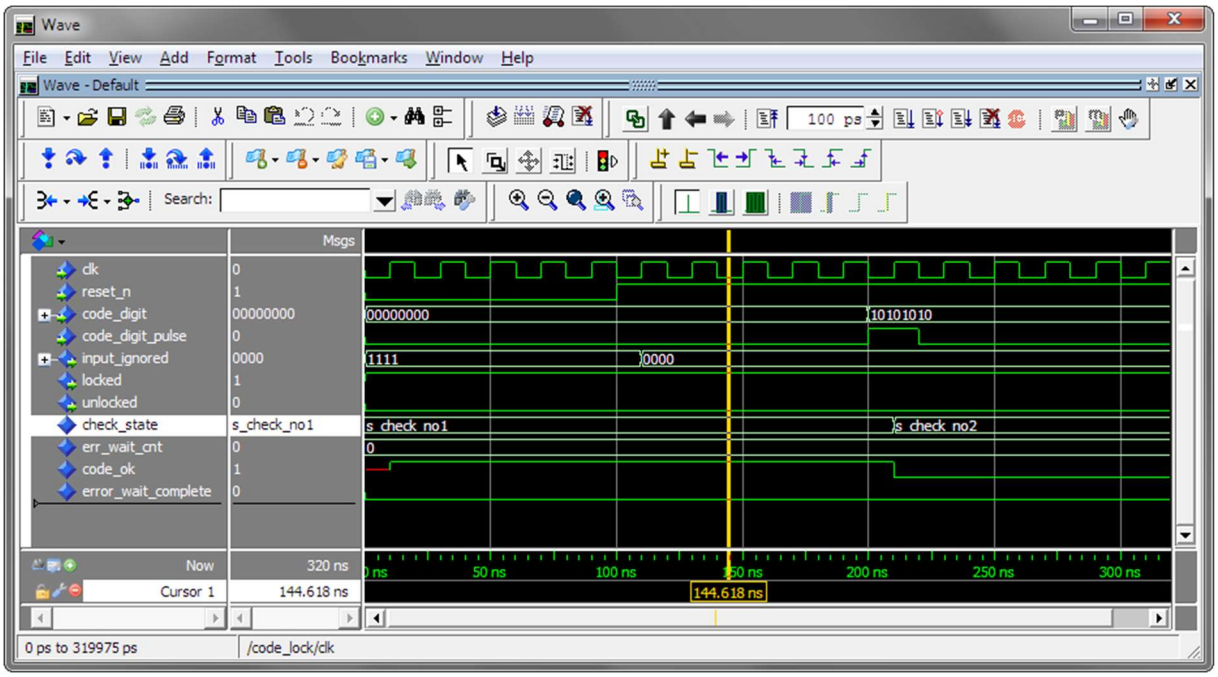
It is read as "force the signal clk 0 at time 0, then force it high at time 10 ns repeat with a period time of 20 ns"

Another tip for .do files is to add a `restart -f` command at the first row of the .do file to do an automatic restart of the simulation. A .do-file example could in this case look something like this:

```
#Restart simulation
restart -f

#define all input signals, reset active
force clk 0 0, 1 10 ns -r 20 ns
force reset_n 0
force code_digit 00000000
force code_digit_pulse 0
run 100 ns

#release reset
force reset_n 1
run 100 ns

#Set code_digit to 0xA5 and pulse signal high for one clock cycle
force code_digit 10101010
force code_digit_pulse 1
run 20 ns

# reset pulse signal
force code_digit_pulse 0
run 100 ns
```

Result would be something like shown in the simulation window below (note that the do script only controls the input signals):

Address: Mor Kerstins väg 27, 511 53 Kinna, Sweden
Tel: +46-703-748429
Registration no: 559037-6603
E-mail: kent@elekteknik.se



Address:
Mor Kerstins väg 27
511 53 Kinna
Sweden

Tel:
+46-703-748429

Registration no:
559037-6603

E-mail:
kent@elekteknik.se