FYS4220 - Fall 2013 : Lab. Exercise 1

Kévin Zaporogezt and Nathalie Bonatout

September 19, 2013

Contents

List of Figures

List of Algorithms

Introduction

This lab exercise is an introduction to the FPGA programmation in the environment Quartus II. Thus, we discovered during this work some of the most useful features provided by Quartus, and the way to do a simulation thanks to the software ModelSim. The exercise too allowed us to learn how to use and connect the board DE2, how to manage its components such as the LEDs or the 7-segment decoder.

1 Getting Started

A - Switches and LEDs

Algorithm 1 Blinking LEDs

```
library IEEE;
use IEEE.std_logic_1164.all;

entity lab1_gx is
  port (
    LEDR : out std_logic_vector(17 downto 0);   -- Red LEDs
    SW   : in  std_logic_vector(17 downto 0));  -- Switches
end entity lab1_gx;


architecture top_level of lab1_gx is
begin

--Lab 1a: Assign all switches to LEDs
  LEDR <= SW;

end architecture top_level;
```

We checked that this algorithm produced exactly what we wanted on the board: it linked the switches to the LEDs, allowing us to make them blink by pushing the switches. This simple piece of code too allowed us to discover how to map the components of the DE2-70 board manually.

B - 7-segment decoder with functional and timing simulation

We want here to implement a 7-segment decoder displayed on the output HEX0 of the board.

7. Running the functional simulation:

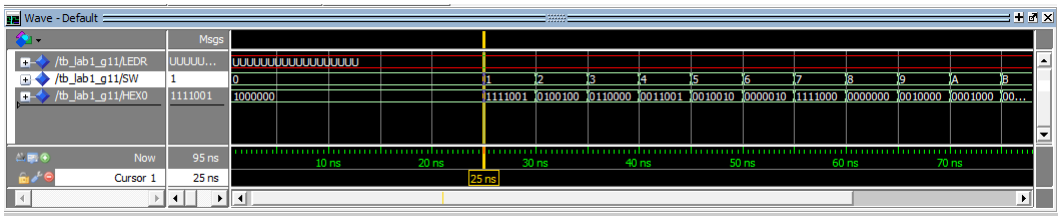

Figure 1: Functional simulation

10. The gate simulation gives us a timing result slightly different from the RTL simulation. It can be explained by the fact that the components, even though they are kind of physically close, still induce a delay. So when the RTL simulation only checks the functional result of the system (so there is no timing information here), the gate simulation takes delays into account. And we can notice that a transition of 5 ns is way too fast for the system to work the way we expect it to work. Indeed, the intern frequency of the DE2-70 is 50MHz. So it allows us to change state every 20 ns. Thus, the output HEX0 takes from times to times random values with the gate simulation.

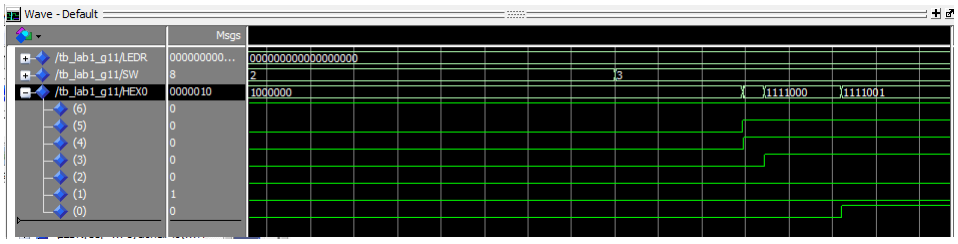

Figure 2: Real Time simulation with transitions of 5 ns

11. We increased the time between the transition on SW from 5 ns to 20 ns. And the outcome of the simulation obviously changed: the system became more stable, but still has some unpredictable behaviors, which can be explained by the "physical delay".

12. We can evaluate the time it takes before a transaction on the SW input results in a stable signal on HEX0 as 11,48ns.

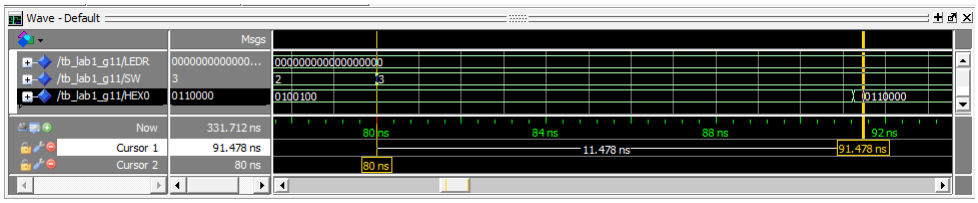

Figure 3: Real time simulation with transitions of 20 ns

2 4-bit Counter and push Button

11. After pressing KEY0, we can see that the value displayed seems to jump from a value to another, discontinously. We cannot see the counter incrementing smoothly, by step of 1, and this is rather expected. Indeed, the frequency of the clock is 50MHz. This means that the value of the counter changes every 20ns as long as the button KEY0 is pushed.

We could see the value of the counter be incremented by step of 1 by delaying the clock, or by implementing a boolean which would block block the value of the counter after once incrementation, when KEY0 is pressed.

Algorithm 2 Implementation of the blocking boolean

```
counter <= "0000" when reset_n = '1'
else unsigned(std_logic_vector(unsigned(counter) +1))
when rising_edge(clk50) and ext_ena_n = '0'
and (countenable = false);
with ext_ena_n select countenable <= false when '1',
true when others;
```

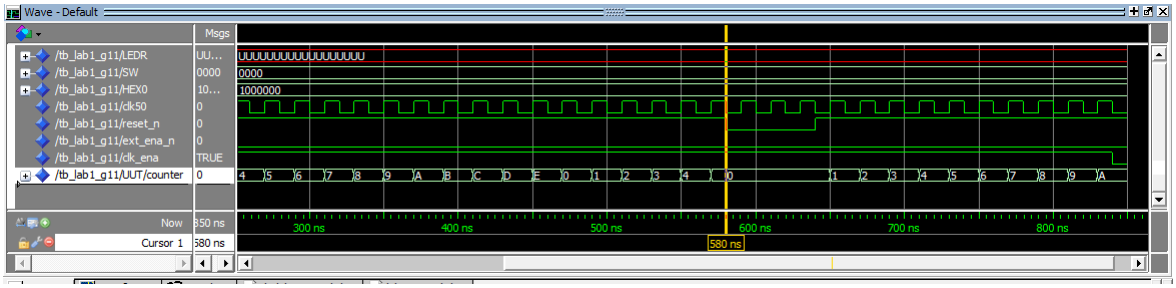

Figure 4: Functional simulation for the counter

3 Conclusion

Finally, we learned how to use Quartus (at least in its most basic features), and how to manage the tools provided with ModelSim. We encountered problems, some of them lead us to find a work around method (f.e., we had to deal with a system file already existing on the computer we were using, and preventing the compilation to go on. We still don't know why this problem occured, but achieved to continue anyway), and had to start from anew since we were used to Xilink, but this lab work really helped us to remember the main problems when programming a FPGA: the timing, and sequential, parallel programmation.