How to Implement Your First VHDL Design on FPGA

The 10 steps to implement a good VHDL design

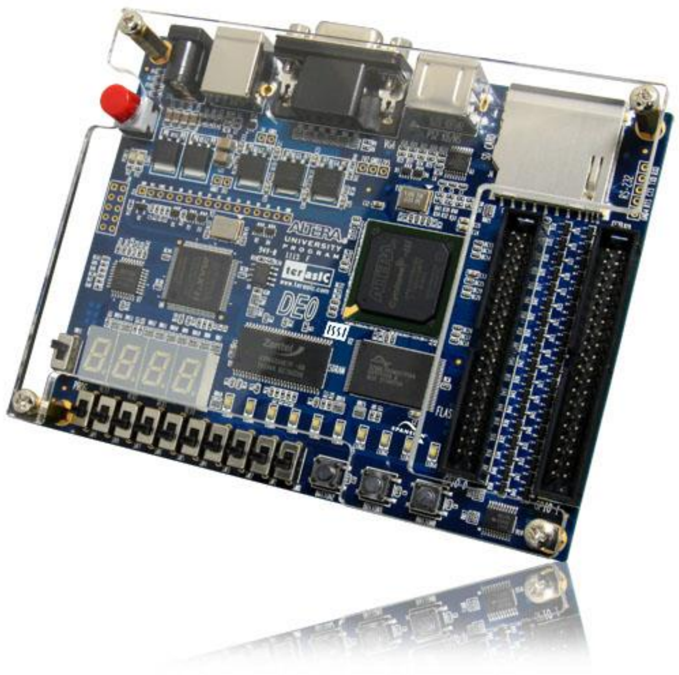


TABLE OF CONTENTS

INTRODUCTION

Hi,

I’m Francesco Richichi and I want to thank you or downloading this eBook.

Just few words to introduce myself.

I’m an Electronic Engineer and I worked in Digital design since 1997.

I developed a lot of HW design both in ASIC and FPGA:

High Speed MODEM, Turbo Encoder/Decoder, LDPC Encoder/Decoder, Radar processing.

In the middle of 2015 I decided to start the website “**http://surf-vhdl.com/**” because I think is the moment to give back some of my knowledge to whom need to know about VHDL.

When I started, I had to study from books and no tutorial were available on the internet. Fortunately, now is completely different. You can find all the info you need for any topic and you can find lot of information for free!

But there is the other side of the coin: you can find a lot of useless stuff!

Even if you don’t think about if, useless stuff is a big loss of money.

Yes, MONEY.

Every time you have to understand if the info you found is useful or not you are using your time.

TIME = MONEY, more TIME > MONEY!

Using your time in a proper way you can get money, the vice versa is not always true.

What do you think about it?

I hope I’ll provide you useful information and let’s start learning how to implement your first VHDL design in FPGA

WHAT DO YOU NEED?

The exercises are implemented using VHDL so you should have a

- VHDL editor
- VHDL simulator
- Board equipped with an FPGA to test the design (optional but recommended)

As VHDL editor on the internet you can find a lot of text editor with syntax highlight. You can use Textpad (http://www.textpad.com), Notepad++ (https://notepad-plus-plus.org) for windows user, TextWrangler for Mac user. These text editors are free.

As VHDL simulator, in the course we will use ModelSim that is one of the most powerful VHDL/Verilog simulator.

ModelSim license is very expensive but you can get a free license legally, next I’ll show how...

As layout tool, we will use Altera Quartus II. The test of the VHDL code on FPGA will be performed on an Altera DE0 board.

The Quartus II license are very expensive too…

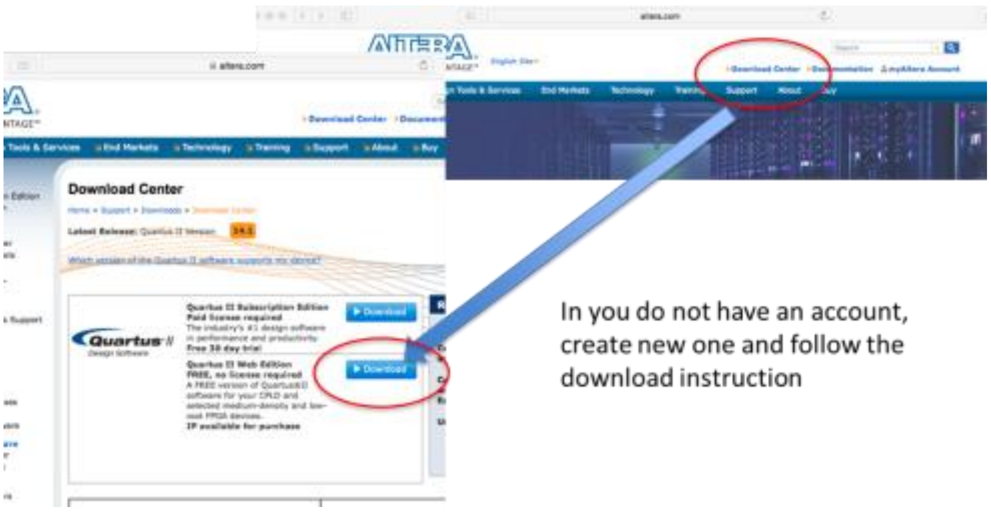
Maybe you are thinking… ok the eBook is free but I should spend a lot of money buying license!

Don’t worry, I’m here to help you.

Using the Altera Quartus II edition provided with the DE0 board you can get a free “lite” license both for ModelSim and Quartus II. “Lite” means that you can simulate and layout only a subset of Altera devices. This is enough because the you can simulate and layout all the “non-professional” (very expensive and very huge) FPGA devices, i.e. you can handle all the FPGA you can afford to buy.

OK… at this point you are asking: if I don’t have the DE0 board?

I'm still here to resolve this problem: go on Altera website, register and download the **Quartus II web edition**. You can get the full Altera suite with ModelSim simulator that you can use to implement your VHDL design.



In you do not have an account, create new one and follow the download instruction

Figure 1 Altera website where download Quartus II

INTRODUCTION TO VHDL DESIGN FLOW: LEARN THE RULES

A typical design flow for a VHDL design implementation is reported in Figure 2.

I want to start from the first two points, even if they seem to be redundant to address in detail.

Many of you send me email asking for help in VHDL implementation of their design/exercise. Generally, the problem isn't the VHDL implementation: the problem is that they do not understand or do not really know what they have to do.

They do not understand the requirements!

Sound strange? It is really so. For instance, time ago I received a mail where a guy asked me how he could use a particular device. It was an accelerometer with a I2C interface.

He asked me how to interface an accelerometer with FPGA. So I asked him:

"do you know what is "I2C" serial interface protocol?"

He answered "NO".

I asked also if he searched for info on I2C protocol, and the answer was NO.

As you can understand, the problem is not the device, it is that he had no idea where to start and which was the requirement.

The second issue is that when you understand the requirements, the first thing you do is start coding.

This is the worst thing you can do!

Before starting you always need to clarify what you have to do using a high level description of your design even on a piece of paper.

These two steps are general consideration valid for all your design HW or SW but are often ignored.

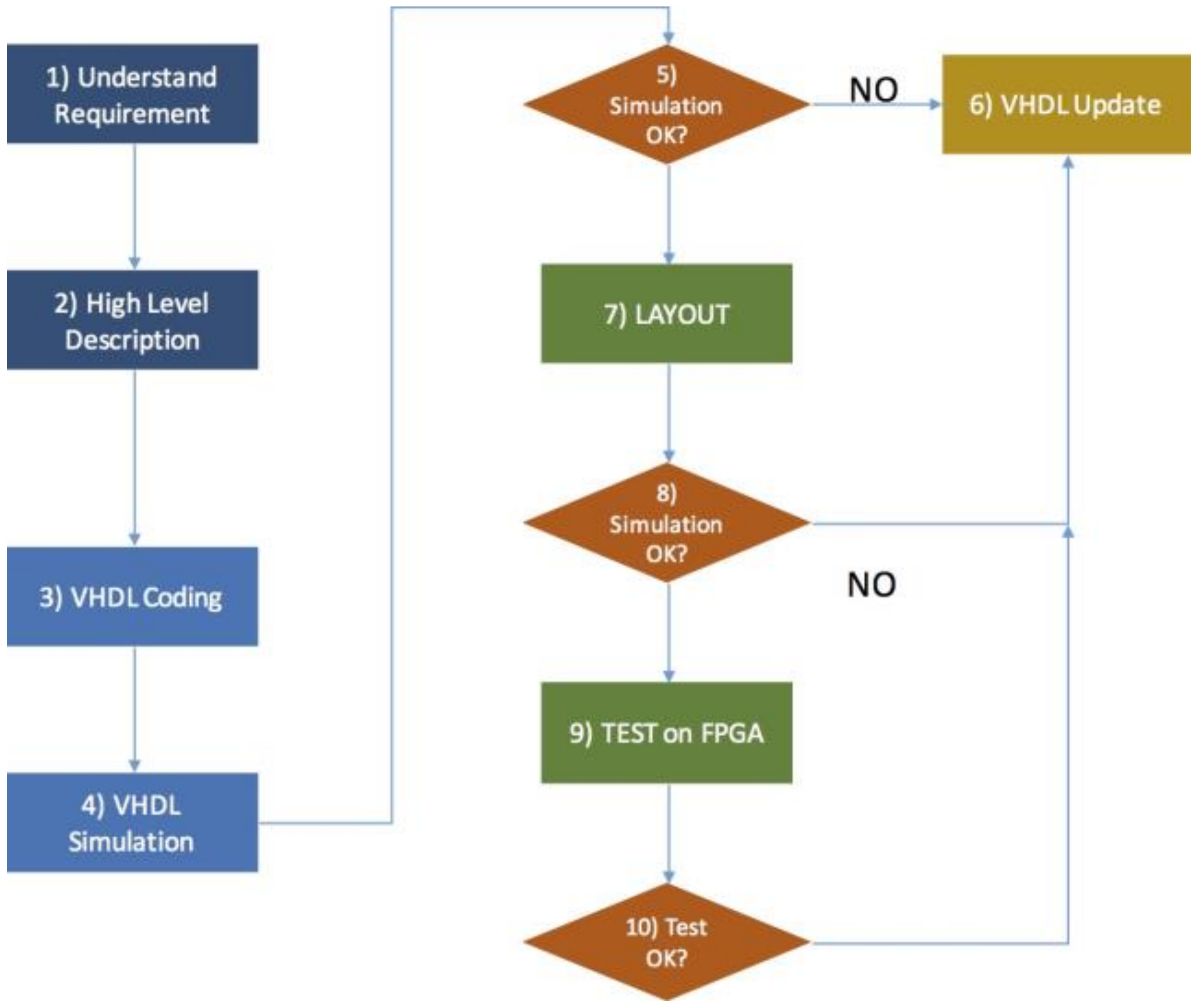


Figure 2 VHDL design flow

Once you

1. Understand what you have to do (**Requirement**)
2. Make your High-level description of the design (write a sketch on a piece of paper)

Only after these two point you can start coding VHDL and go ahead on the design flow of Figure 2:

VHDL Coding-> VHDL simulation-> FPGA Layout->FPGA test.

When the VHDL code simulation results are OK, you can layout your FPGA. If you find a bug or out-of-spec you shall review your VHDL code or your design and start again the flow sometimes, if the out-of-spec is important, going back to the point 1)

After VHDL code layout on FPGA, you can perform a post layout simulation using the technology library and introducing the physical delay. In this eBook I don’t’ want to address this issue, it is out of the scope.

After layouting the VHDL code you can test your design on FPGA.

DESIGN DESCRIPTION

In this eBook we will learn how to begin to design on FPGA using VHDL.

You will:

- Design VHDL for a debouncer
- Design VHDL 4-bit counter
- Design VHDL a 7-segment hardware driver
- Control the 4-bit counter using external push-button or switch using a Board
- Implement debouncer VHDL code for external switch

In order to complete the exercise and test the VHDL code we will refer to the ALTERA DE0 Board below. Don't worry if you have another board, the exercises are implemented in a pure VHDL/RTL, I mean you can port this design on different technology without any problem. Of course, you must re-map the pinout according to the new board, but it is very easy if you understand the complete design flow.

May be is the time to start with the exercise. The Figure 3 shows an overview of our exercise.

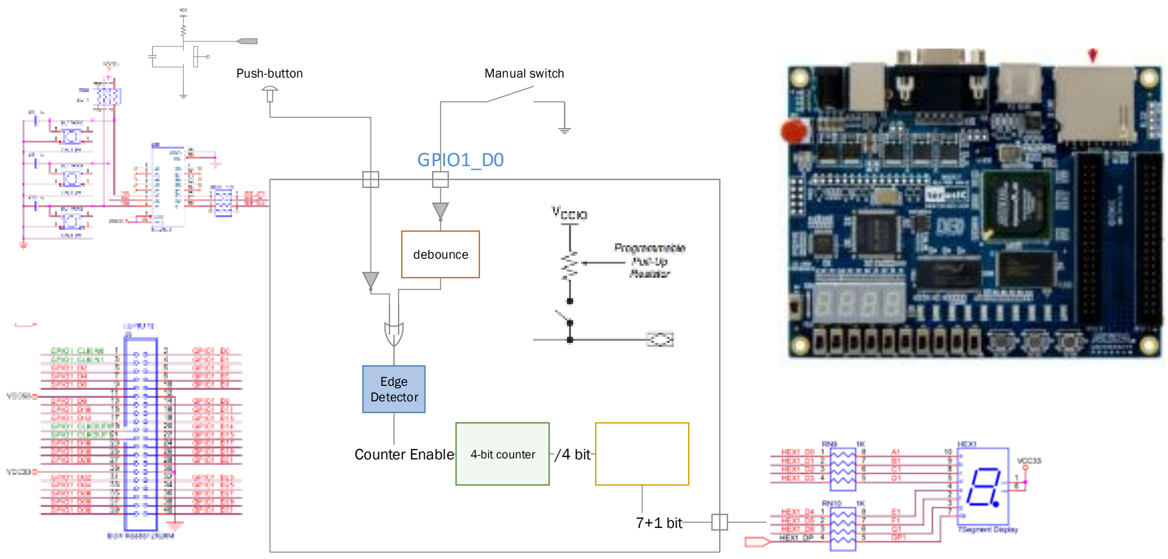


Figure 3 design overview

The VHDL code will implement the section of Figure 8:

The inputs from pin and push button that are active low as clear in Figure 4.

What is the meaning of "active low"?

This means that when the button is released (the circuit is open) the voltage value at the FPGA pin is 3.3V that represents a logical HIGH value at input pin. When you push the button the pin is connected to ground (the circuit is closed), so it is at a LOW voltage level as in Figure 4.

In this case we are assuming that a voltage level 3V3 is relative to a logic level HIGH, and a voltage level 0V is equal to a logic level "LOW".

In Figure 4 the component 74HC245 acts as de-bouncer. The component is a Schmitt trigger whose behavior is reported in Figure 5.

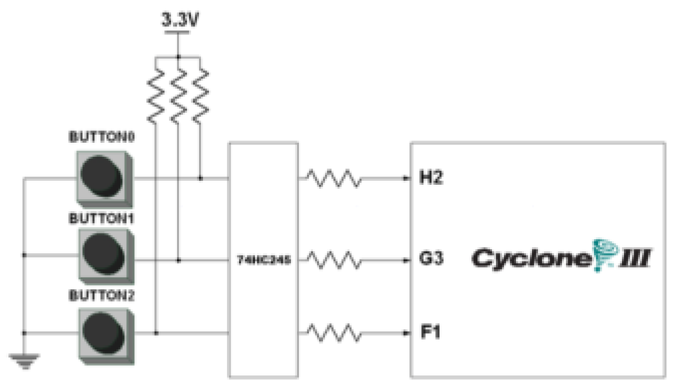


Figure 4 push button connection on DE0 Altera board

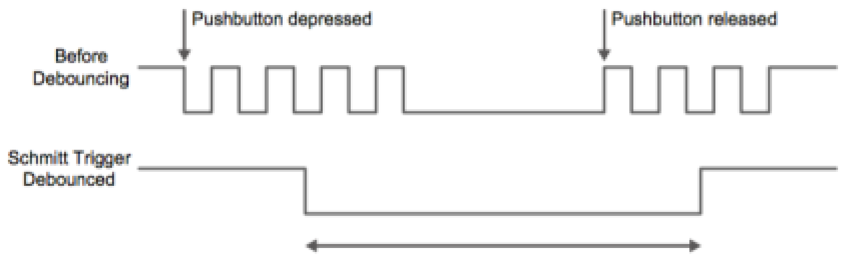


Figure 5 74HC245 behavior

In order to detect HIGH state when we push the button we must introduce an inverter on the line connected to the push button.

The module “Edge Detector” senses the edge of the signal and generate a pulse used to increment a 4-bit counter. The counter output drives the input of a 7-segment display and will be visualized on the 7-segment of the board.

In Figure 3 the inputs to the edge-detector are the OR of the input coming from the push button and a secondary input implemented using a piece of wire that connect the FPGA pin to ground as represented in Figure 6. In this situation the external debouncer is not present so we need to introduce a debounce VHDL module able to filter all the bounce caused by the manual switch as in Figure 7.

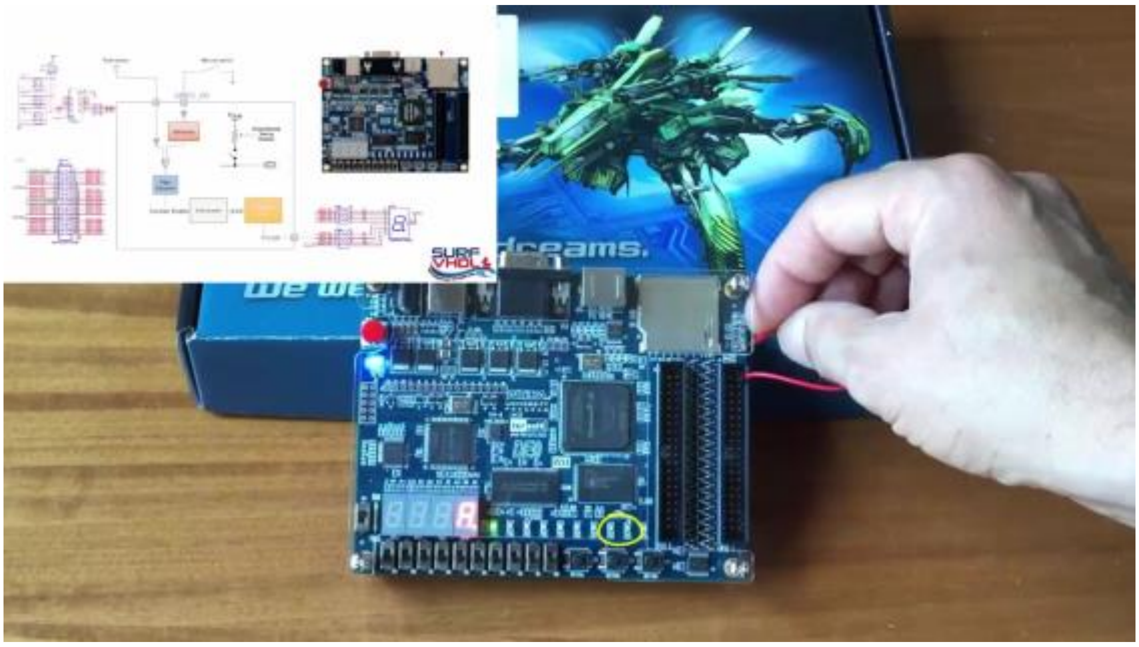


Figure 6 picture of input from fpga pin using a wire

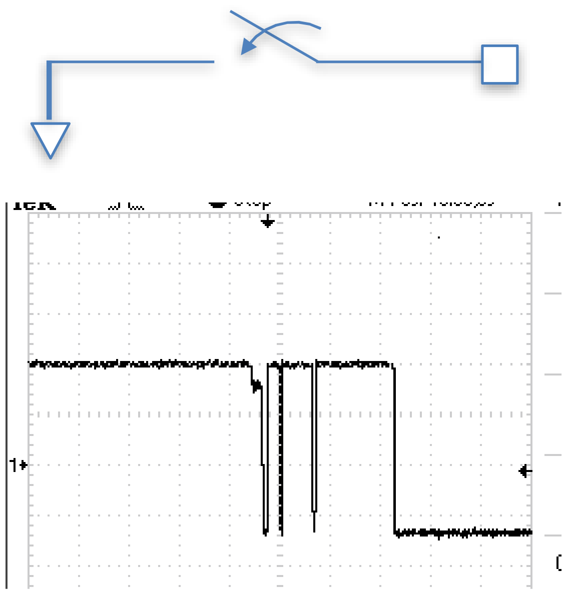


Figure 7 bouncing due to a manual switch

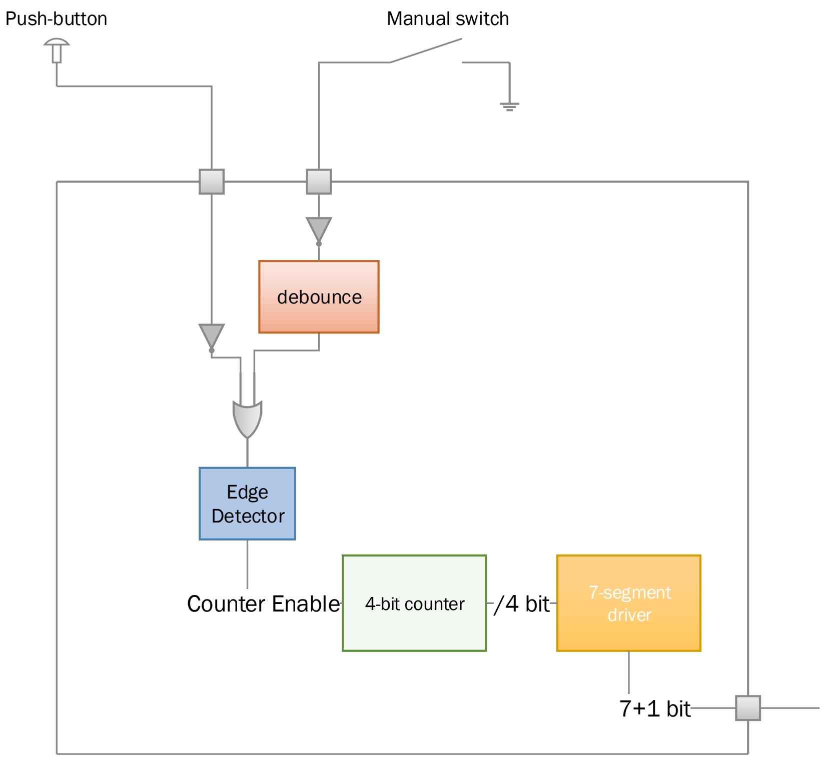


Figure 8 VHDL design

In the following paragraph we will detail each macro block of the design:

- Debouncer
- Edge Detector

- 4-bit counter
- 7-segment driver

VHDL IMPLEMENTATION

In this section we detail the architecture of the single modules proposing a possible VHDL implementation.

INPUT CONTROL

The input from FPGA pin is inverted and put in OR configuration in order to drive the edge-detector as in the piece of VHDL code below:

```
-- BUTTON Assignment
w_button2                    <= not pad_i_button(2);
w_gpio1_d0                   <= not pad_b_gpio1_d(0);

w_button_external            <= w_button2 or w_gpio1_d0_debounce;
```

DEBOUNCER

A debouncer is a circuit that allows filtering glitches form an input asynchronous source. A possible implementation can be the following (see Figure 10):

1. count the number of clock cycles the input signal is stable high;
2. every time the signal goes low during counting the counter is restated;
3. when the counter reaches the desired number of clock cycles, pulse is generated.

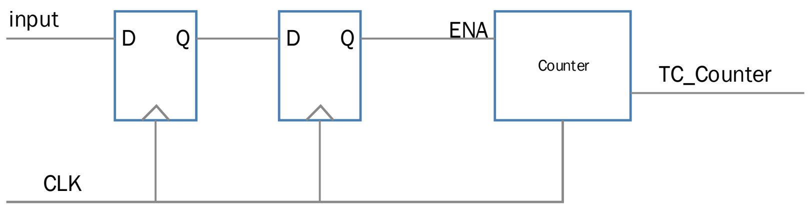


Figure 9 debouncer architecture

With such architecture if:

- Fc = clock frequency (Hz)

- Plen = pulse length to filter (sec)

The number of clock cycle need to count in order to filter pulses lesser than Plen is:

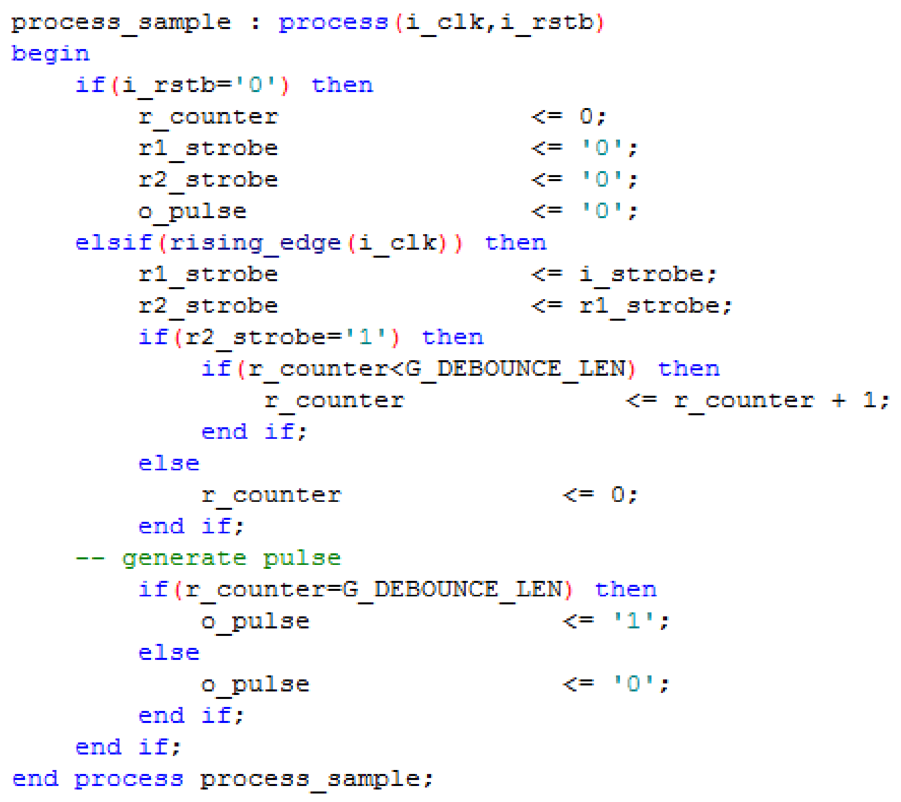
*Count Length = Fc * Plen*

Es:

Fc=50 MHz; Plen = 40 ms

Count Length = 50e6 * 40e-3 = 2.000.000 clock cycles

A possible VHDL implementation is reported in Figure 10



```vhdl
process_sample : process(i_clk,i_rstb)
begin
    if(i_rstb='0') then
        r_counter                <= 0;
        r1_strobe                <= '0';
        r2_strobe                <= '0';
        o_pulse                  <= '0';
    elsif(rising_edge(i_clk)) then
        r1_strobe                <= i_strobe;
        r2_strobe                <= r1_strobe;
        if(r2_strobe='1') then
            if(r_counter<G_DEBOUNCE_LEN) then
                r_counter              <= r_counter + 1;
            end if;
        else
            r_counter              <= 0;
        end if;
    -- generate pulse
        if(r_counter=G_DEBOUNCE_LEN) then
            o_pulse                <= '1';
        else
            o_pulse                <= '0';
        end if;
    end if;
end process process_sample;
```

Figure 10 possible VHDL implementation for a debouncer

The signals ***r1_strobe*** and ***r2_strobe*** implement the two input flip-flop. The ***r_counter*** counts only when the ***r2_strobe='1'*** and stop counting when the number of clock cycle is equal to ***G_DEBOUNCE_LEN.*** When the counter reaches the number of programmed clock cycle the circuit generate a pulse on ***o_pulse*** signal.

EDGE DETECTOR

An Edge-Detector implements a circuit that is able to detect the transition of an input signal:

- Rising edge detector: identify the transition ‘0’->’1’
- Falling edge detector: identify the transition ‘1’->’0’

In Figure 11 is represented the typical architecture of a rising edge detector. In the time representation is clear that no matter how long the input signal rest high, the output is a pulse of a single clock cycle.

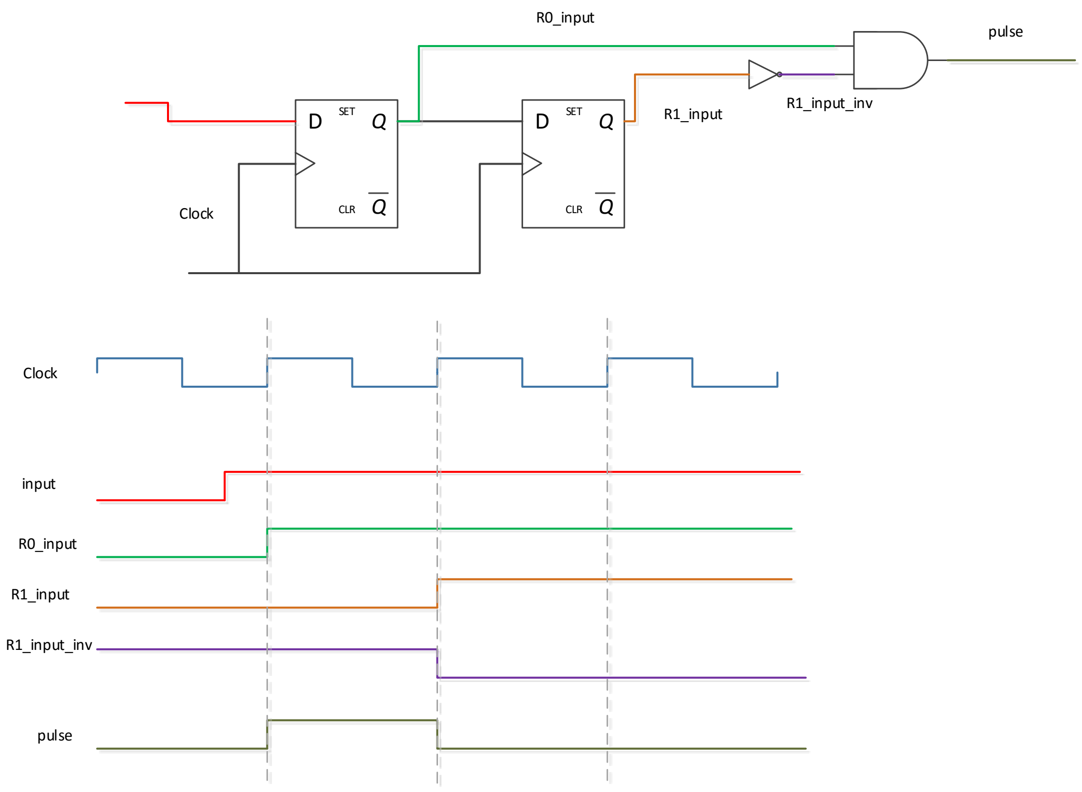
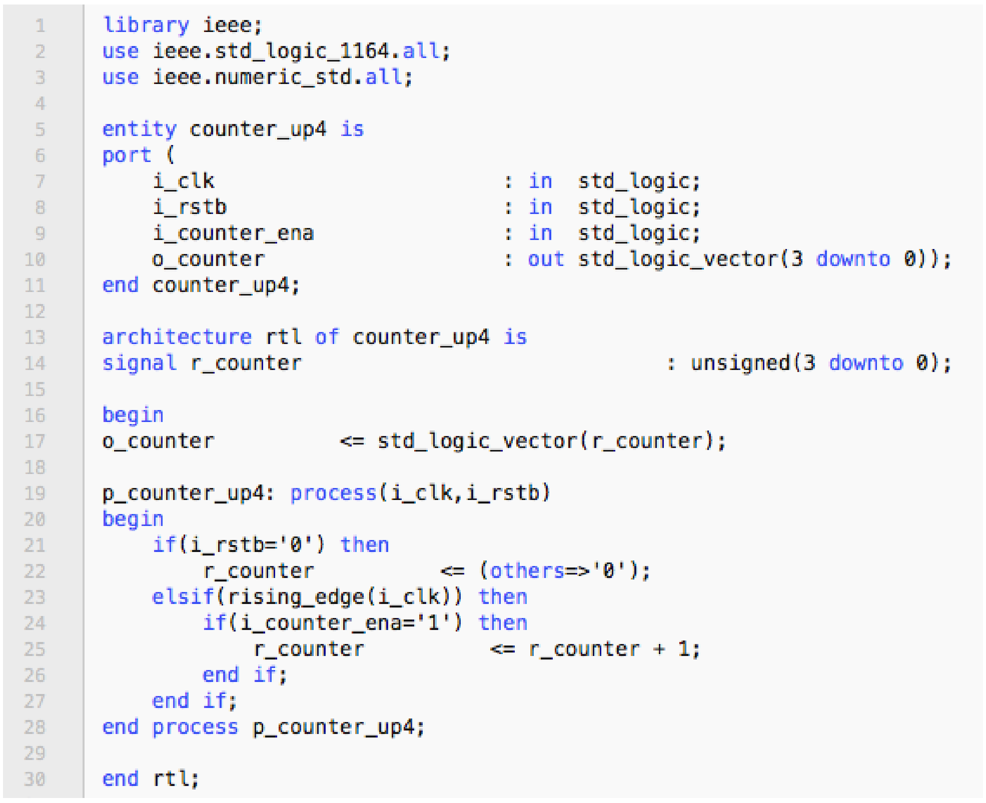


Figure 11 edge detector architecture

4-BIT COUNTER

A possible VHDL code for a 4-bit counter with external enable is reported in Figure 12. The counter is declared as unsigned (see line 14). The input signal ***i_counter_ena*** is used to enable counting. When ***i_counter_ena***=’1’ the counter is incremented by 1 at each clock cycle.



```
 1  library ieee;
 2  use ieee.std_logic_1164.all;
 3  use ieee.numeric_std.all;
 4
 5  entity counter_up4 is
 6  port (
 7      i_clk                           : in  std_logic;
 8      i_rstb                          : in  std_logic;
 9      i_counter_ena                   : in  std_logic;
10      o_counter                       : out std_logic_vector(3 downto 0));
11  end counter_up4;
12
13  architecture rtl of counter_up4 is
14  signal r_counter                                : unsigned(3 downto 0);
15
16  begin
17  o_counter          <= std_logic_vector(r_counter);
18
19  p_counter_up4: process(i_clk,i_rstb)
20  begin
21      if(i_rstb='0') then
22          r_counter          <= (others=>'0');
23      elsif(rising_edge(i_clk)) then
24          if(i_counter_ena='1') then
25              r_counter          <= r_counter + 1;
26          end if;
27      end if;
28  end process p_counter_up4;
29
30  end rtl;
```

Figure 12 4-bit counter VHDL code example

7-SEGMENT DRIVER

The 7-segment driver shall be able to turn ON and OFF the LED of the 7-segment on the board. In Figure 13 is reported the part of board schematic relative to the 7-segment display connection. As clear the LED of the display turn ON when the FPGA pin HEX0_Dx is driven low. In this configuration the current sink by the FPGA pin from 3V3 power supply.

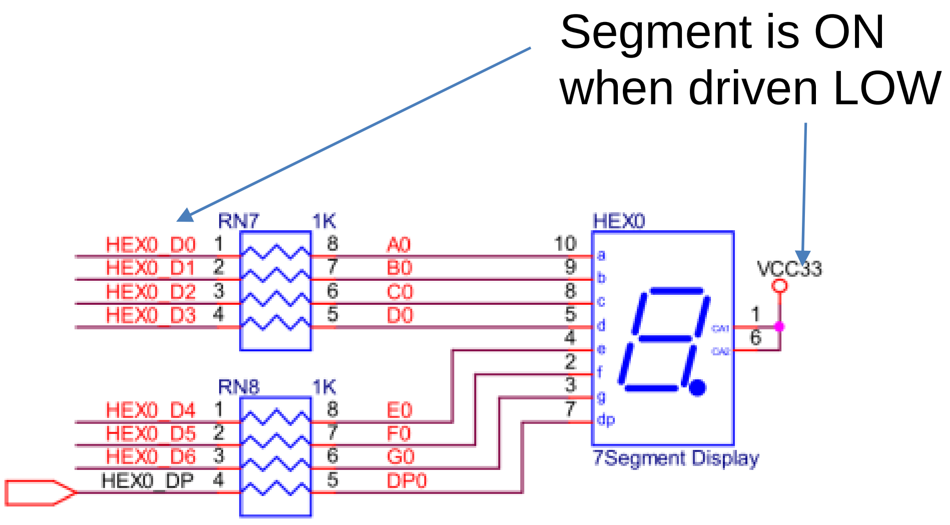
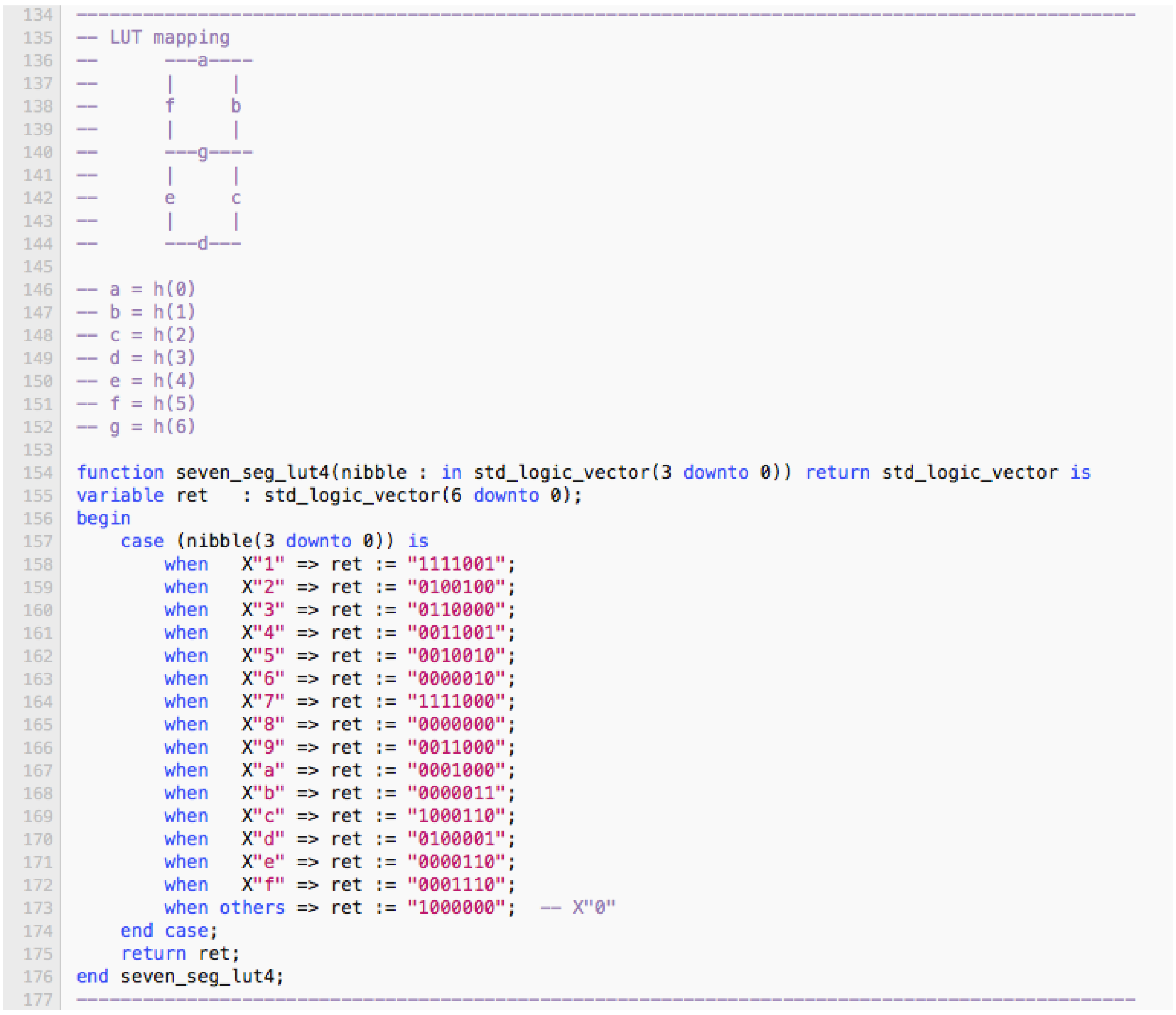


Figure 13 7-segment display connection

The 7-segment driver is used to output the 4-bit counter value. The possible counter values are 0,1,…15 = (2^4)-1. In this case we can represent the counter value as hexadecimal value using a LUT (Look Up Table) that convert the 4-bit value to its hexadecimal representation. Figure 14 reports a possible VHDL implementation of a 7-segment driver in VHDL using a function.



```vhdl
------------------------------------------------------------------------------------------------
-- LUT mapping
--      ----a-----
--      |     |
--      f     b
--      |     |
--      ----g-----
--      |     |
--      e     c
--      |     |
--      ----d----

-- a = h(0)
-- b = h(1)
-- c = h(2)
-- d = h(3)
-- e = h(4)
-- f = h(5)
-- g = h(6)

function seven_seg_lut4(nibble : in std_logic_vector(3 downto 0)) return std_logic_vector is
variable ret   : std_logic_vector(6 downto 0);
begin
    case (nibble(3 downto 0)) is
        when   X"1" => ret := "1111001";
        when   X"2" => ret := "0100100";
        when   X"3" => ret := "0110000";
        when   X"4" => ret := "0011001";
        when   X"5" => ret := "0010010";
        when   X"6" => ret := "0000010";
        when   X"7" => ret := "1111000";
        when   X"8" => ret := "0000000";
        when   X"9" => ret := "0011000";
        when   X"a" => ret := "0001000";
        when   X"b" => ret := "0000011";
        when   X"c" => ret := "1000110";
        when   X"d" => ret := "0100001";
        when   X"e" => ret := "0000110";
        when   X"f" => ret := "0001110";
        when others => ret := "1000000";  -- X"0"
    end case;
    return ret;
end seven_seg_lut4;
------------------------------------------------------------------------------------------------
```

Figure 14 example of 7-segment driver implementation in VHDL

VHDL CODE SIMULATION AND FPGA IMPLEMENTATION

If you want to see the results of the VHDL simulation and implementation on FPGA you can join the free course

Implement Your First VHDL Design On FPGA

Where you can download the VHDL source code of the design and try by yourself.

In the video you can review the concept we treated in this pages aided with a live explanation. Moreover, the design is played on an Altera DE0 board so you can verify the results on a live demo

Don’t forget to join SURF-VHDL on the social pages:

for any question write to:

surf.vhdl@gmail.com

LICENSE