Laboratory 2

Comparator

Digital Systems Design with VHDL
IAS0600

rev. 3

Marcel Cases Freixenet

Tallinn,
September 19, 2018

Tallinna Tehnikaülikool
TalTech

Contents

1 Introduction

1.1 Aim

This project is an introductory approach to FPGA's hardware description language VHDL. It consists in the development of a two-bit comparator in VHDL that generates an output depending on whether the comparation is greater, less or equal each other. In this report we will analyse how different VHDL statements influence the outcome. This is done using three different methods and showing its RTL and post-synthesis schemes.

1.2 Background

A 2-bit magnitude comparator can be designed using the the following table as a reference:

Input				**Output**		
A1	**A0**	**B1**	**B0**	**A>B**	**A=B**	**A<B**
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

Table 1.1 *Truth table of a two-bit comparator*

There are several ways to describe the hardware capable of giving an output of this type. In this laboratory, three methods have been implemented.

The first of them is using the boolean equations of each output. To calculate these equations, it is necessary to solve their equivalent Karnaugh's maps [1]. The other two methods consist in "when .. else" and "with .. select" VHDL statements.

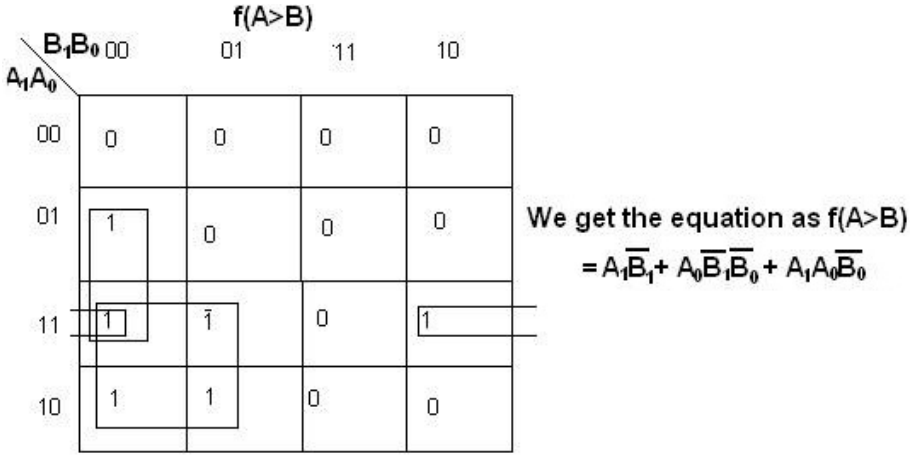

Figure 1.1 *Karnaugh map for A>B*

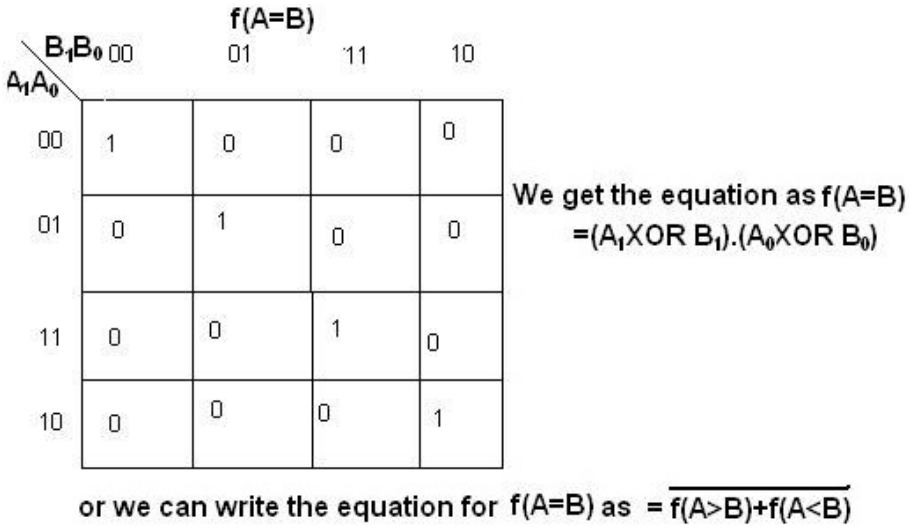

Figure 1.2 *Karnaugh map for A=B*

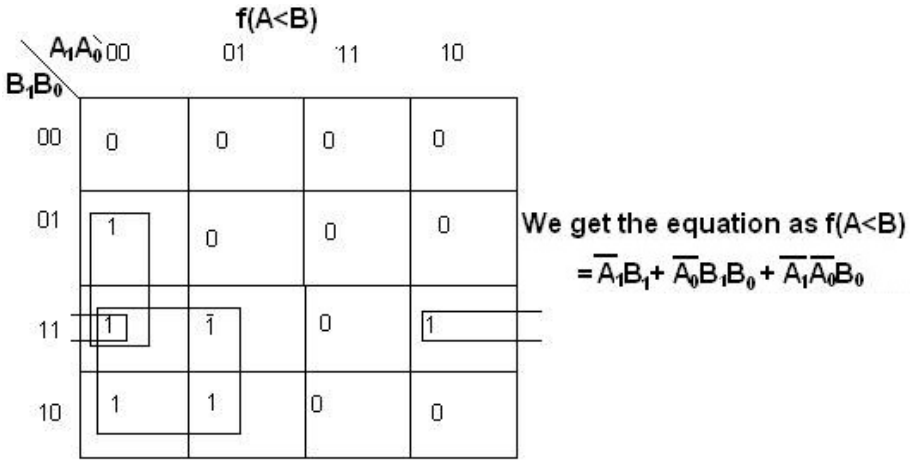

Figure 1.3 *Karnaugh map for A<B*

2 Workflow

The hardware description of the two bit comparator has been developed in a single file of VHDL source code. No components have been used.

2.1 Top level

The following code is the hardware description of the main file in the hierarchy that contains all of the libraries needed to run the two bit comparator as well as the entity, with its physical input and output ports, and the architecture of each one of the three methods.

2.1.1 Libraries

The first lines of the code define the libraries needed for the two bit comparator. In this project, only one library is needed, which is the standard logic revision 1164 defined by IEEE.

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
```

2.1.2 Entity

The entity of the two bit comparator requires to use two inputs which are vectors of two bits in order to indicate the values of the bits to compare. Physically, these two inputs are routed to four switches of the Nexys 4 board.

Three output ports are used to show the result of the comparison of the bits. They are single signals physically routed to three LEDs above the switches of the Nexys 4 board. *eq_o* is for equal than; *gr_o* is for greater than; and *ls_o* is for less than.

The definition of the ports is defined below.

```
1 entity comparator is
2         Port (  in1 : in std_logic_vector (1 downto 0);
3                 in2 : in std_logic_vector (1 downto 0);
4                 eq_o : out std_logic;
5                 gr_o : out std_logic;
6                 ls_o : out std_logic
7                 );
8 end comparator;
```

2.1.3 Architecture

The architecture of the project has the main purpose to define the behavior of the signals that go in and out of the physical ports of the FPGA.

An internal signal is defined to be used with the third method (with .. select statement), and is described below.

```
architecture Behavioral of comparator is
        -- Internal signals
        signal tmp : std_logic_vector (3 downto 0); -- Used only in the 3rd
        ↪  method

begin

        -- Method 1 code [...]
        -- Method 2 code [...]
        -- Method 3 code [...]

end Behavioral;
```

2.1.3.1 Method 1: boolean equations

Task ➪ *Complete the truth table for a 2-bit comparator and write out the corresponding Boolean equations. Use these equations to describe the comparator in VHDL*

The first method to compare two bits consists in its boolean equations. They are calculated from Karnaugh's maps described in ➪ Chapter 1. It consists in three concurrent assignations that contain the result of Karnaugh's maps.

```
eq_o <= not((in1(1) and not(in2(1))) or (in1(0) and not(in2(1)) and
↪  not(in2(0))) or (in1(1) and in1(0) and not(in2(0))) or (not(in1(1)) and
↪  in2(1)) or (not(in1(0)) and in2(1) and in2(0)) or (not(in1(1)) and
↪  not(in1(0)) and in2(0)));
gr_o <= (in1(1) and not(in2(1))) or (in1(0) and not(in2(1)) and not(in2(0)))
↪  or (in1(1) and in1(0) and not(in2(0)));
ls_o <= (not(in1(1)) and in2(1)) or (not(in1(0)) and in2(1) and in2(0)) or
↪  (not(in1(1)) and not(in1(0)) and in2(0));
```

2.1.3.2 Method 2: "when .. else" statement

Task ➪ *Use "when .. else" VHDL statement to describe a 2-bit comparator*

The second method consists in concurrent assignments too, but it does not need any boolean equation. It uses a VHDL statement instead, "when .. else". By default, all of the signals are set to '0' unless the condition after "when" is true; then the signal is set to '1'.

```
eq_o <= '1' when in1 = in2 else '0';
gr_o <= '1' when in1 > in2 else '0';
ls_o <= '1' when in1 < in2 else '0';
```

2.1.3.3 Method 3: "with .. select" statement

Task ➪ *Use "with .. select" VHDL statement to describe a 2-bit comparator*

Similar to "when .. else" statement, "with .. select" is used to define the output of a signal when a certain combination of bits is given. All of the cases must be covered when using "with .. select" using "else" if needed.

To sort the bits from most significant to least significant, an internal signal named "tmp" is defined at the beginning of the architecture and is assigned concurrently with the corresponding input bits (line number 1). Then this signal is used as the condition inside the statement.

```
1  tmp <= (in1(1), in1(0), in2(1), in2(0));
2  with tmp select
3          eq_o <= '1' when "0000",
4                  '1' when "0101",
5                  '1' when "1010",
6                  '1' when "1111",
7                  '0' when others;    -- default
8  with tmp select
9          gr_o <= '1' when "0100",
10                 '1' when "1000",
11                 '1' when "1001",
12                 '1' when "1100",
13                 '1' when "1101",
14                 '1' when "1110",
15                 '0' when others;
16 with tmp select
17         ls_o <= '1' when "0001",
18                 '1' when "0010",
19                 '1' when "0011",
20                 '1' when "0110",
21                 '1' when "0111",
22                 '1' when "1011",
23                 '0' when others;
```

2.2 Testbench

For this project, a testbench is a useful tool to test the output bits for every combination of input bits. The testbench is the same for each one of the methods described above.

The results of the behavior of the hardware description using the following testbench are commented in ➪ Chapter 3.

The *stimuli* process has the purpose to test every possible combination of inputs, as it is described below. This is a reduced version of the file *comparator_tb.vhd* containing the main lines of code.

```
1  stimulus: process begin        -- Beginning of the stimulus process
2          input_tb <= "0000" ;    wait for 10 ns; -- Every 10ns move to the
           ↪  following input_tb configuration
3          input_tb <= "0001" ;    wait for 10 ns;
```

```
4           input_tb <= "0010" ;    wait for 10 ns;
5           input_tb <= "0011" ;    wait for 10 ns;
6           input_tb <= "0100" ;    wait for 10 ns;
7           input_tb <= "0101" ;    wait for 10 ns;
8           input_tb <= "0110" ;    wait for 10 ns;
9           input_tb <= "0111" ;    wait for 10 ns;
10          input_tb <= "1000" ;    wait for 10 ns;
11          input_tb <= "1001" ;    wait for 10 ns;
12          input_tb <= "1010" ;    wait for 10 ns;
13          input_tb <= "1011" ;    wait for 10 ns;
14          input_tb <= "1100" ;    wait for 10 ns;
15          input_tb <= "1101" ;    wait for 10 ns;
16          input_tb <= "1110" ;    wait for 10 ns;
17          input_tb <= "1111" ;    wait for 10 ns;
18
19          wait; -- Wait indefinitely
20
21  end process;
22
23  end bench;
```

3 Results and discussion

3.1 Simulation

Once the three methods have been described in hardware and the testbench is defined, running the simulation shows us that the results are as expected and that they are exactly the same for each one of the three methods.

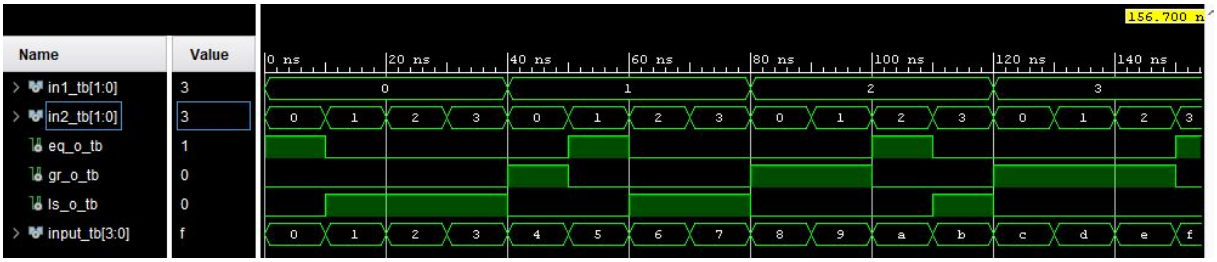

Figure 3.1 *Simulation using the testbench*

Comparing the output results of the simulation with the ➪ Table 1.1, we can assure that the behavior of the hardware description is as it was expected for every combination of input bits.

3.2 RTL analysis

According to Wikipedia [2], *register-transfer-level abstraction is used in hardware description languages (HDLs) like Verilog and VHDL to create high-level representations of a circuit, from which lower-level representations and ultimately actual wiring can be derived. Design at the RTL level is typical practice in modern digital design.*

RTL analysis of the two-bit comparator using all the three methods can be found below.

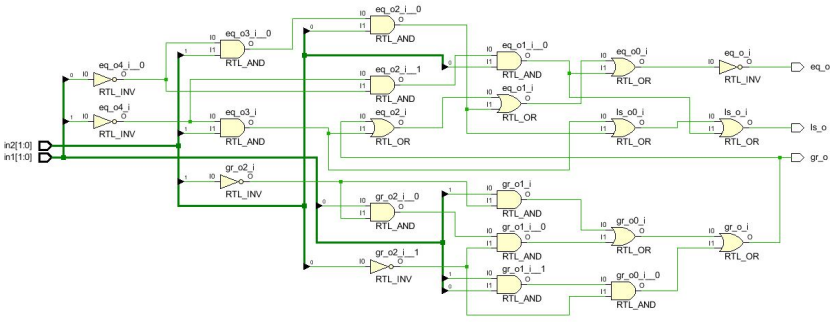

Figure 3.2 *RTL Analysis of Method 1 (boolean equations)*

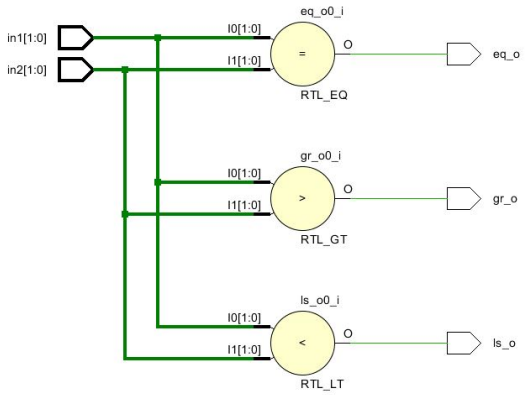

Figure 3.3 *RTL Analysis of Method 2 ("when .. else" statement)*

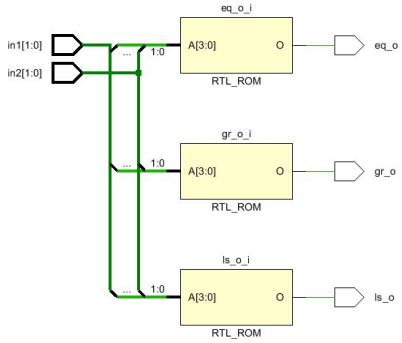

Figure 3.4 *RTL Analysis of Method 3 ("with .. select" statement)*

As we can see in the figures, RTL schematics is different for each method, being the first method the most complex one and the second and third the least complex.

3.3 Synthesis schematic

According to Xilinx Glossary [3], "synthesis" is defined as *a process that starts from a high level of logic abstraction (typically Verilog or VHDL) and automatically creates a lower level of logic abstraction using a library containing primitives.*

After the synthesis has been performed for each method, the obtained schemes show a certain similarity one another. More specifically, the second and the third method have exactly the same configuration; only the boolean equations have produced an extra look-up table connected to the equal output signal.

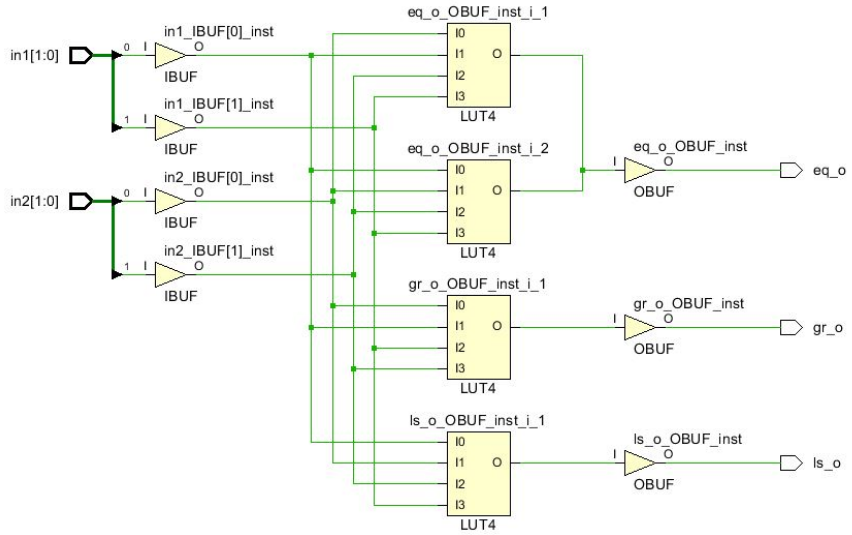

Figure 3.5 *Synthesis schematic of Method 1 (boolean equations)*

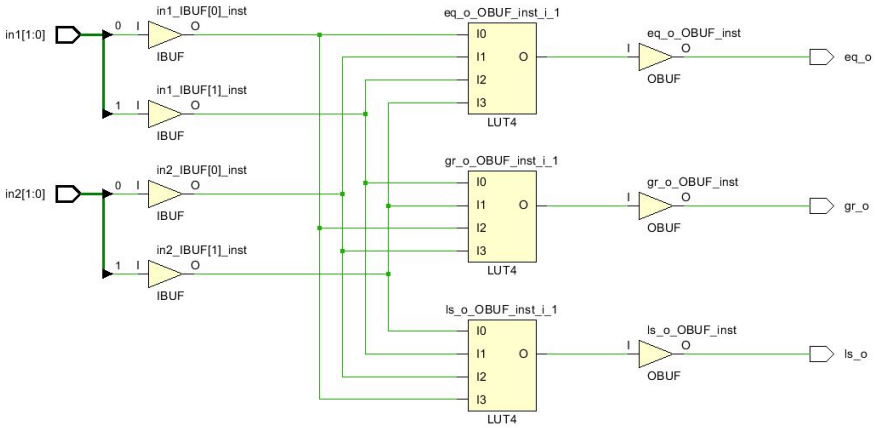

Figure 3.6 *Synthesis schematic of Method 2 ("when .. else" statement)*

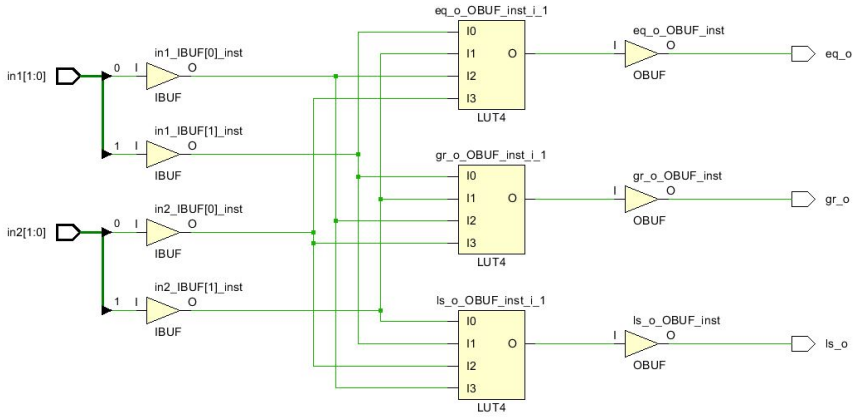

Figure 3.7 *Synthesis schematic of Method 3 ("with .. select" statement)*

4 Conclusion

We have exposed a theoretical approach to obtaining the boolean equations for the first method, as well as the VHDL syntax for the other two.

All lines of code are exposed in the next section, describing which part they belong to (import, entity or architecture). A testbench for the 16 different input combinations has been exposed too.

Then the results are discussed. They show a simulation of the 2-bit comparator, with the expected results. Then a RTL scheme and a synthesis scheme are shown for each one of the three methods, and their differences are commented.

I consider that the laboratory has been developed successfully.

References

[1] exploreroots.com. 2-bit comparator. http://www.exploreroots.com/dc18.html, 2018. 1

[2] Wikipedia. Rtl analysis. https://en.wikipedia.org/wiki/Register-transfer_level, 2018. 7

[3] Xilinx. Synthesis. http://www.xilinx.com/company/terms.htm, 2018. 8