Laboratory 3

Adder

Digital Systems Design with VHDL
IAS0600

rev. 2

Marcel Cases Freixenet

Tallinn,
October 3, 2018

Tallinna Tehnikaülikool
TalTech

Contents

1 Introduction

1.1 Aim

This project consists in the design and implementation of a digital two bit adder using **structural design**. To do this, two methods are used. The first is using VHDL code only and a full adder and a half adder as components. The other is using a Xininx's Vivado feature named Block Design.

1.2 Background

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

Input				**Output**		
a1	**a0**	**b1**	**b0**	**carry**	**sum1**	**sum0**
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

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

There are several ways to develop a 2-bit adder. One of them is using a half-adder together with a full-adder, which can be made from other half-adders.

A half adder is a digital circuit that has the purpose to sum two bits [1]. It can be described by the following boolean equations:

$$s = a\ XOR\ b$$
$$c = a\ AND\ b$$

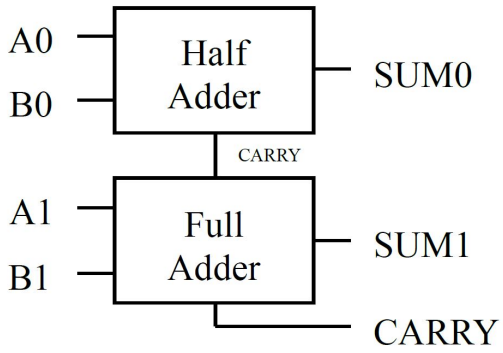

Figure 1.1 *Block diagram of a two-bit adder*

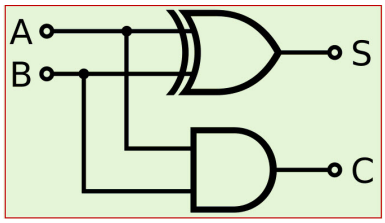

Figure 1.2 *Circuit of a half-adder*

A full-adder also sums two bits but has a carry as an input and uses it for calculations. A full-adder can be done using other half-adders as components:

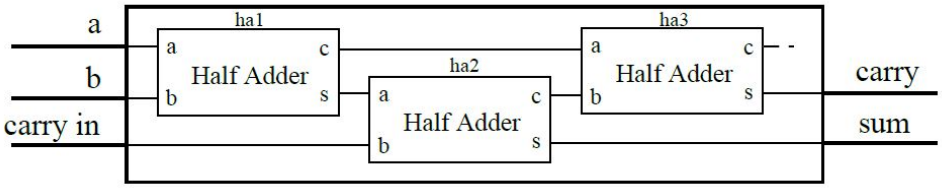

Figure 1.3 *Full-adder from three half-adders*

Although we still work with schematic designs, the input to the synthesis tool must be a VHDL description of the structure of the design (i.e. what blocks are present and how they are interconnected). This is termed a **netlist**, and is a feature of VHDL **structural design** [2].

2 Workflow

The hardware description of the two bit adder has been developed using two methods. The first is using only VHDL code and two components, a half-adder and a full-adder, that are instantiated in the top level. The other method is using a Block Design as the top level file.

2.1 VHDL-based 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 adder as well as its entity, with the 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 adder requires to use four single bit inputs in order to indicate the values of the bits to add (by groups of two). Physically, these inputs are routed to four switches of the Nexys 4 board.

Three output ports are used to show the result of the sum of these input bits. They are single signals physically routed to three LEDs above the switches of the Nexys 4 board. One of them is the output carry signal.
The definition of the ports is described below.

```
1 entity adder is
2         port (  a1, a0, b1, b0 : in std_logic ;
3                 sum1, sum0, carry : out std_logic
4                 );
5 end adder;
```

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.

This architecture has two components, as it has been described above, that have to be included in the hardware description. They are the *half-adder* and the *full-adder*.

An internal signal, *carry_internal*, is used to internally wire the output of the half-adder block to the input of the full-adder, according to the block diagram in ➪ Figure 1.1.

Once the two components have been described and called, we only have to make an instance of them once and indicate the map of ports-signals.

```
1  architecture Behavioral of adder is
2
3          component half_adder
4                  port (  a : in std_logic;
5                          b : in std_logic;
6                          sum : out std_logic;
7                          cout : out std_logic
8                          );
9          end component;
10
11         component full_adder is
12                 port (  a : in std_logic;
13                         b : in std_logic;
14                         cin : in std_logic ;
15                         sum : out std_logic;
16                         cout : out std_logic
17                         );
18         end component;
19
20         signal carry_internal : std_logic;
21
22 begin
23
24 ha : half_adder
25         port map (  a => a0,
26                 b => b0,
27                 sum  => sum0,
28                 cout  => carry_internal
29                 );
30
31 fa : full_adder
32         port map (  a => a1,
33                 b => b1,
34                 cin => carry_internal,
35                 sum  => sum1,
36                 cout  => carry
37                 );
38
39 end Behavioral;
```

2.1.4 Components

For this two-bit adder, two components have been used. The first is a half adder and it consists in the hardware description of the circuit shown in ➪ Figure 1.2.

```
entity half_adder is
        port (  a : in std_logic;
                b : in std_logic;
                sum : out std_logic;
                cout : out std_logic
                );
end half_adder;

architecture Behavioral of half_adder is

begin

sum  <=  a xor b ;
cout  <=  a and b ;

end Behavioral;
```

The second component is a full-adder. There are many ways to describe a full adder. One of them is using three half-adders as shown in ⇨ Figure 1.3, which has been implemented in this projects. For this purpose, the component has to perform three concurrent instances of the half-adder, as shown below:

```
entity full_adder is
        port (  a : in std_logic;
                b : in std_logic;
                cin : in std_logic ;
                sum : out std_logic;
                cout : out std_logic
                );
end full_adder;

architecture Behavioral of full_adder is

component half_adder
        port (  a : in std_logic;
                b : in std_logic;
                sum : out std_logic;
                cout : out std_logic
                );
end component;

signal sum1, cout1, cout2 : std_logic;

begin

ha1 : half_adder
        port map (  a => a,
                b => b,
                sum => sum1,
                cout => cout1
                );

ha2 : half_adder
        port map (  a => sum1,
                b => cin,
                sum => sum,
                cout => cout2
                );
```

```
37
38 ha3 : half_adder
39         port map (  a => cout1,
40                     b => cout2,
41                     sum => cout
42                     );
43
44 end Behavioral;
```

Then these two components are called in the top level file (*adder.vhd*) and used as many times as needed (only one is necessary).

2.2 Block Design-based top level

Block Design is a feature available for Xilinx Vivado environment that allows hardware designers to graphically route components each other using a user interface (UI) [3].

For this part of the project, the same components have been used and wired according to the block diagram. This Block Design is equivalent to the top level of the project. Given that a Block Design can not be a top level description, a *wrapper* has been generated according to class instructions.

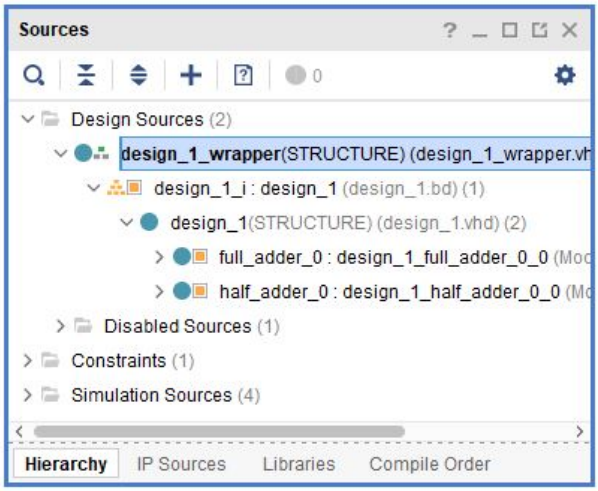

Figure 2.1 *Wrapper as top level file*

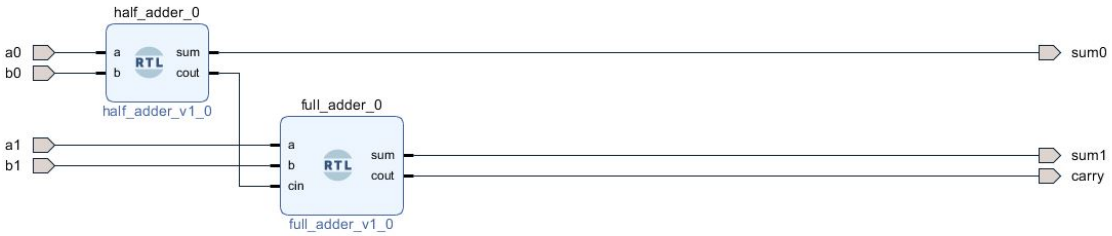

Figure 2.2 *Top-level block design routing two components*

2.3 Testbench

In general terms, a testbench is a useful tool for testing the output bits for every possible combination of input bits. For the current project, the testbench is the same for each one of the methods described above.

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

This is a reduced version of the file *adder_tb.vhd* containing the main lines of code.

```
stimulus: process begin
        input_tb <= "0000" ;    wait for 10 ns;
        input_tb <= "0001" ;    wait for 10 ns;
        input_tb <= "0010" ;    wait for 10 ns;
        input_tb <= "0011" ;    wait for 10 ns;
        input_tb <= "0100" ;    wait for 10 ns;
        input_tb <= "0101" ;    wait for 10 ns;
        input_tb <= "0110" ;    wait for 10 ns;
        input_tb <= "0111" ;    wait for 10 ns;
        input_tb <= "1000" ;    wait for 10 ns;
        input_tb <= "1001" ;    wait for 10 ns;
        input_tb <= "1010" ;    wait for 10 ns;
        input_tb <= "1011" ;    wait for 10 ns;
        input_tb <= "1100" ;    wait for 10 ns;
        input_tb <= "1101" ;    wait for 10 ns;
        input_tb <= "1110" ;    wait for 10 ns;
        input_tb <= "1111" ;    wait for 10 ns;

wait;

end process;

-- Concurrent signals
a1_tb <= input_tb(3);
a0_tb <= input_tb(2);
b1_tb <= input_tb(1);
b0_tb <= input_tb(0);
```

A single vector *input_tb* has been used for all the four inputs in order to have an easier visual definition of the inputs for the testbench.

3 Results and discussion

3.1 Simulation

Once the two methods have been described in hardware and the testbench is defined (see ➪ Section 2.3), running the simulation shows us that the results are as expected and that they are exactly the same for each one of the two methods according to the reference in ➪ Table 1.1.

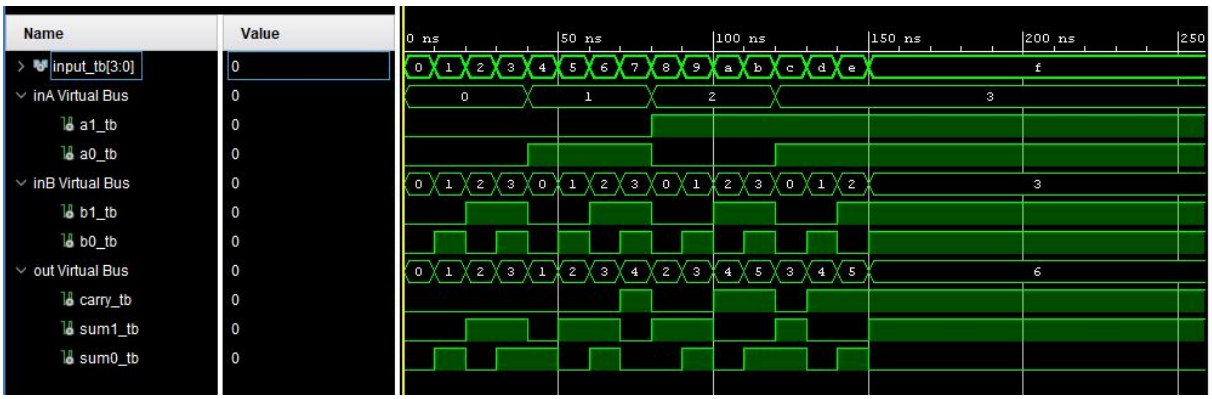

Figure 3.1 *Simulation using the testbench*

The input and output signals are groped in Virtual Buses for an easier visual check.

3.2 RTL analysis

RTL analysis [4] of the two-bit adder using all the two described methods is found below.

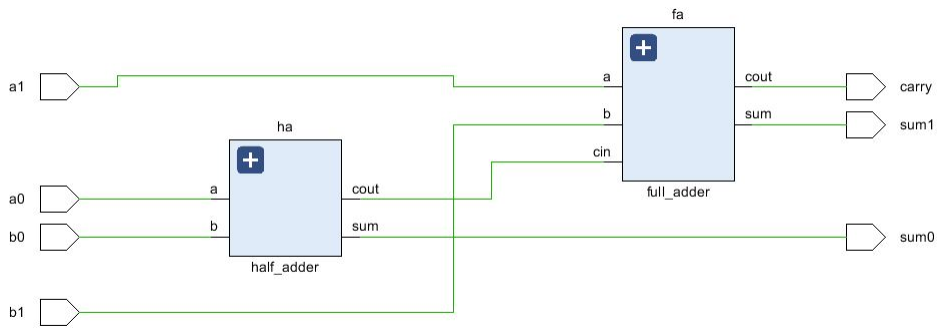

Figure 3.2 *RTL Analysis of VHDL-based method (blocks diagram)*

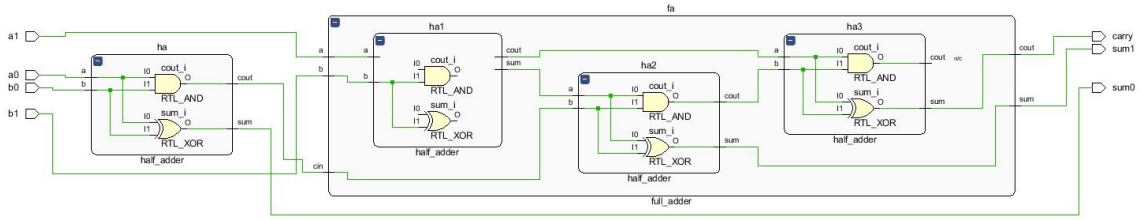

Figure 3.3 *RTL Analysis of VHDL-based method (components)*

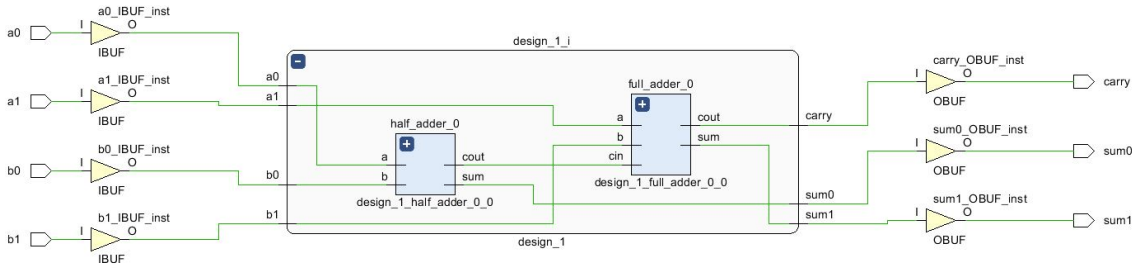

Figure 3.4 *RTL Analysis of Block Design-based method*

After the RTL schemes have been generated for both methods, it is possible to see that a certain similarity exist one another. The main difference is that the Block Design-based method uses buffers in its input and output ports.

3.3 Synthesis schematic

The synthesis, which is the process that starts from a high level of logic abstraction and automatically creates a lower level of logic abstraction using a library containing primitives [5], is shown below.

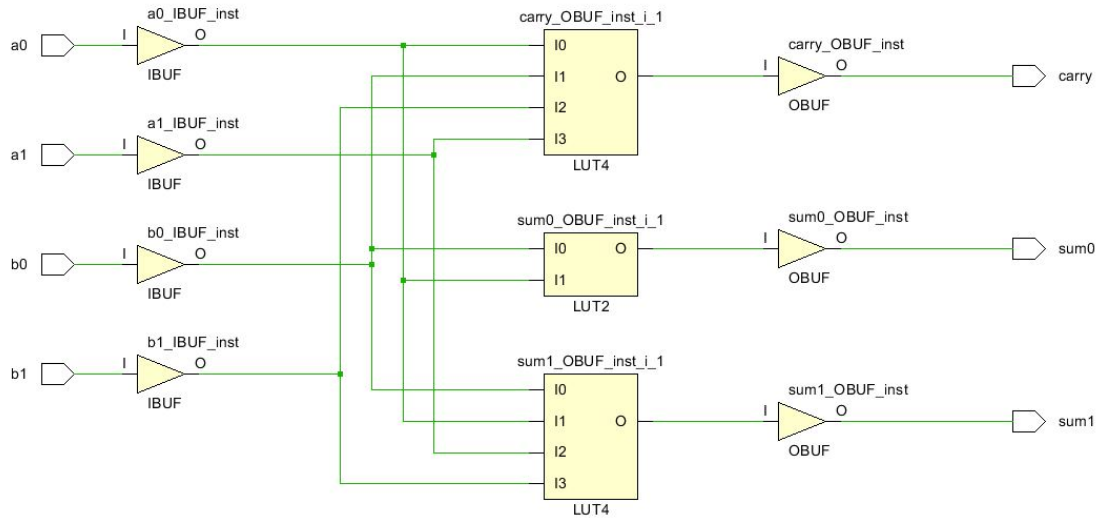

Figure 3.5 *Synthesis schematic of VHDL-based method*

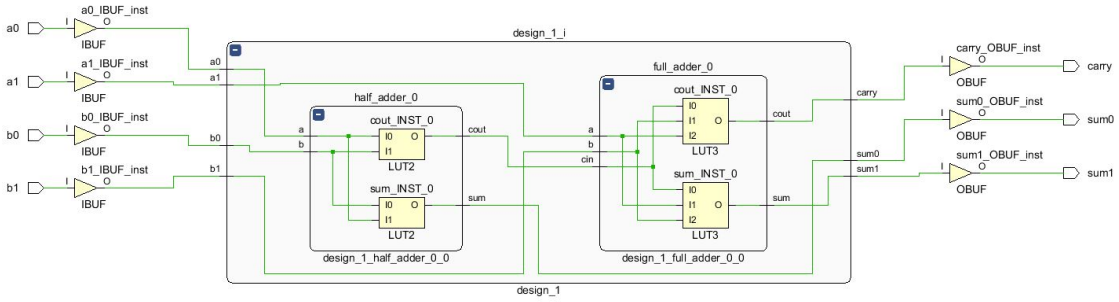

Figure 3.6 *Synthesis schematic of Block Design-based method*

Synthesis schematic shows major differences depending on the method that has been used. For the VHDL-based method, three look-up tables have been synthesized, one for each output. For the Block Design-based method, each component has synthesized their own internal look-up tables of reduced size.

4 Conclusion

A two-bit adder can be in fact designed at a hardware level using different methods or techniques. Two methods have been used for this laboratory.

A new concept from the course has been introduced: Xilinx's Vivado Block Design utility. This user interface method has given us the ability to graphically route signals among components and use it as a top level file.

The background of the report shows both the block diagram and the digital circuits of the half-adder and the full-adder, as well as the truth table.

The simulation shows how the sum of any combination of input bits is successfully displayed as the output.

Given that the purpose is to get the same output regardless of the method, we have seen that using one or another does not have a big impact in the RTL analysis, though the synthesis differs a little.

I consider that this laboratory has been successfully developed.

References

[1] Techopedia. Half adder. https://www.techopedia.com/definition/7509/half-adder, 2018. 1

[2] gla.ac.uk. Structural vhdl. http://userweb.eng.gla.ac.uk/scott.roy/DCD3/Structural_VHDL.pdf, 2017. 2

[3] Xilinx. Designing ip subsystems using ip integrator. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_3/ug994-vivado-ip-subsystems.pdf, 2013. 6

[4] Wikipedia. Rtl analysis. https://en.wikipedia.org/wiki/Register-transfer_level, 2018. 8

[5] Xilinx. Synthesis. http://www.xilinx.com/company/terms.htm, 2018. 9