Laboratory Session 2

Introduction to VHDL: Structural VHDL

Objective

- Understanding the basics of VHDL descriptions
- Use Quartus IDE
- Use ModelSim Simulator

Software Packages

- Altera Quartus II
- ModelSim Software

1. Introduction

In the previous laboratory session, the basics of behavioral VHDL were revised. The current lab session revises the basics of structural VHDL to build a hardware system in bottom-up fashion.

2. Samples

2.1. Half Adder

For this exercise we will implement a Half Adder using *behavioral* VHDL. A Half Adder takes two 1-bit inputs and outputs the sum and carry of the addition (See Figure 1). This is a repeated exercise from lab 1, however, the Half_Adder code will be used to implement the structural VHDL of the full_Adder. If you saved the file from last lab, you may skip the next steps and use the previously created Half_Adder.

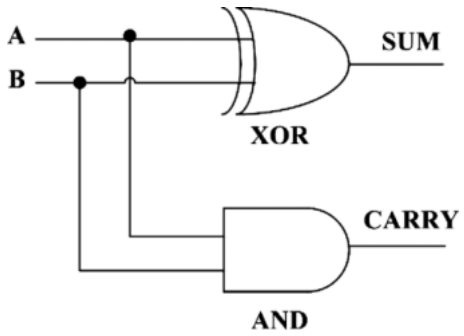

Figure 1. Half Adder Diagram

First start by creating the project in Quartus II using the following steps:

1. Start by creating a folder (eg. Half_Adder) to save your project.
2. Run Quartus II.
3. Click on Create a New Project and then click on Next
4. On the right of the first Textbox click on, to select the location of the folder created previously.

5. In the second Textbox, write a name for the project name (eg. Half_Adder) and then click on Finish.
Next create a new VHDL file
1. File → New → VHDL File → OK

2. Or click on → VHDL File → OK

Paste the following code into the new VHDL file created:

```
library ieee;
use ieee.std_logic_1164.all;
entity Half_Adder is port(
        A, B: in std_logic;
        SUM, CARRY: out std_logic
);
end entity;

architecture beh of Half_Adder is
begin

SUM <= A xor B;
CARRY <= A and B;

end architecture;
```

3. Save the file using the entity name (Half_Adder)

4. Press the start compilation button and wait until it finishes.

5. If the compilation is successful a popup window will appear stating that "Full compilation was successful"

6. The flow summary tab will appear showing the total logic elements used.

7. Generate the RTL diagram: Tools → Netlist Viewer → RTL viewer

8. Now, to check if the schematic of the Half Adder is the same as the previous figure we need to get the RTL view.

9. Verify both designs

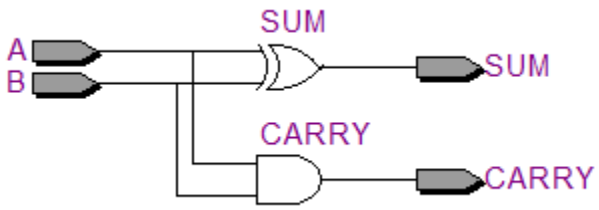

Figure 2. RTL View of Half_Adder

2.2. Full Adder

In this exercise, we will implement a Full Adder using Half Adders implemented in the previous exercise (See Figure 2). A Full Adder takes two 1-bit inputs and a carry in and outputs the sum and carry of the addition.

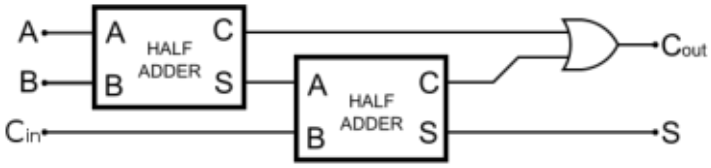

Figure 3. Full Adder using Half Adder diagram

First, start by creating the project in Quartus II using the following steps:
1. Start by creating a folder (eg. Full_Adder2) to save your project.
2. Run Quartus II.
3. Click on Create a New Project and then click on Next
4. On the right of the first Textbox click on, to select the location of the folder created previously.
5. In the second Textbox, write a name for the project name (eg. Full_Adder) and then click on Next.
6. The Add Files page will appear, on the right of the Textbox click on , go to the location of the Half_Adder2 folder and select {Half_Adder.vhd} and click Open then Finish.
Next create a new VHDL file and paste the following code:

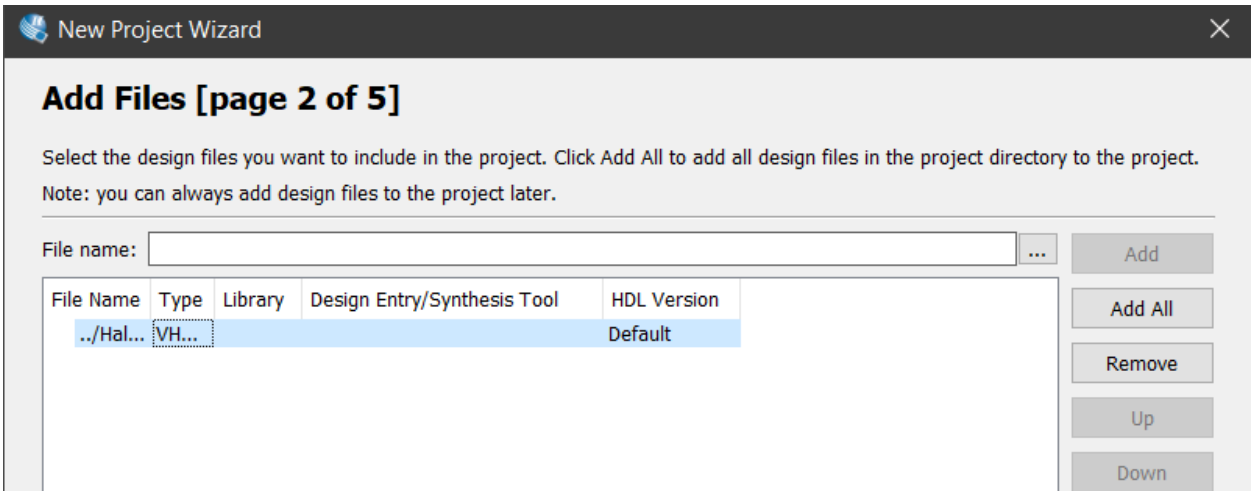

Figure 4. Add Files section of New Project Wizard

```
library ieee;
use ieee.std_logic_1164.all;
entity Full_Adder is
port (
a,b,cin : in STD_LOGIC;
s,cout : out STD_LOGIC
);
end entity;

architecture struct of Full_Adder is

component Half_Adder is
port(
A, B: in std_logic;
SUM, CARRY: out std_logic
);
end component;

signal c1, c2, s1 : std_logic;
begin
HA1: Half_Adder port map (a, b, s1, c1);
HA2: Half_Adder port map (s1, cin, s, c2);
cout <= c1 or c2;
end architecture;
```

Verify the RTL for the Full Adder.

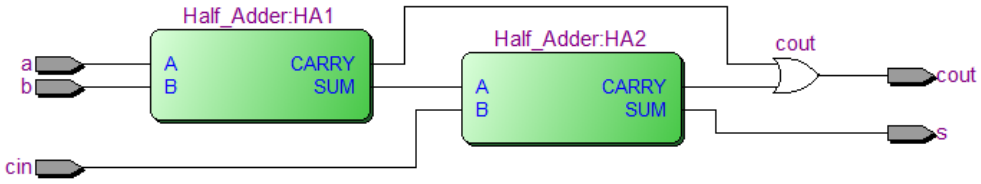

Figure 5. RTL View of Full_Adder

8. Check the power consumption.

Verify the functionality of the Full Adder using ModelSim. Note that when adding the signals, c1, c2 and s1 can be ignored as they are internal signals and are not needed to verify the functionality, but they can be used for debugging.

3. Practice Exercises:

3.1. 4-Bit Adder using Full Adder

Implement a 4-Bit Adder using Full Adders (See Figure 3). Use structural VHDL. Use separate files for the reusable entities.

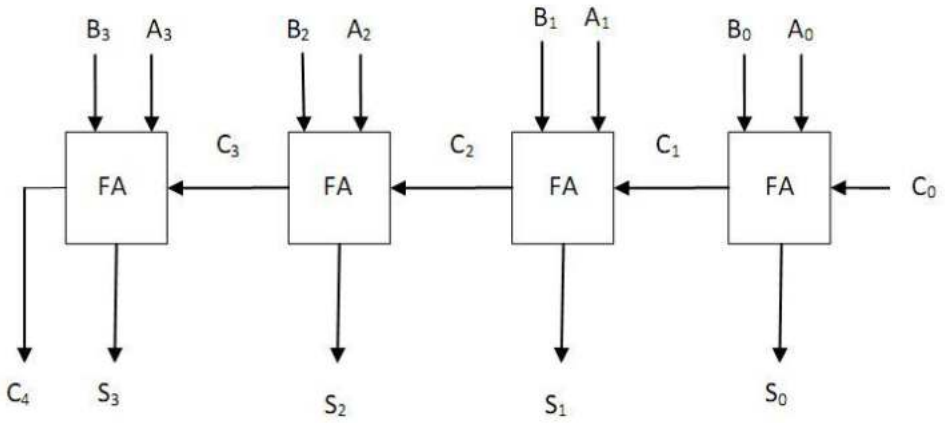

Figure 6. 4-Bit Adder diagram

3.2. BCD Adder

Design, implement and verify a BCD Adder (See Figure 4). The BCD Adder takes two BCD values and outputs the addition as a BCD value and an output carry. Use structural VHDL. Use separate files for the reusable entities.

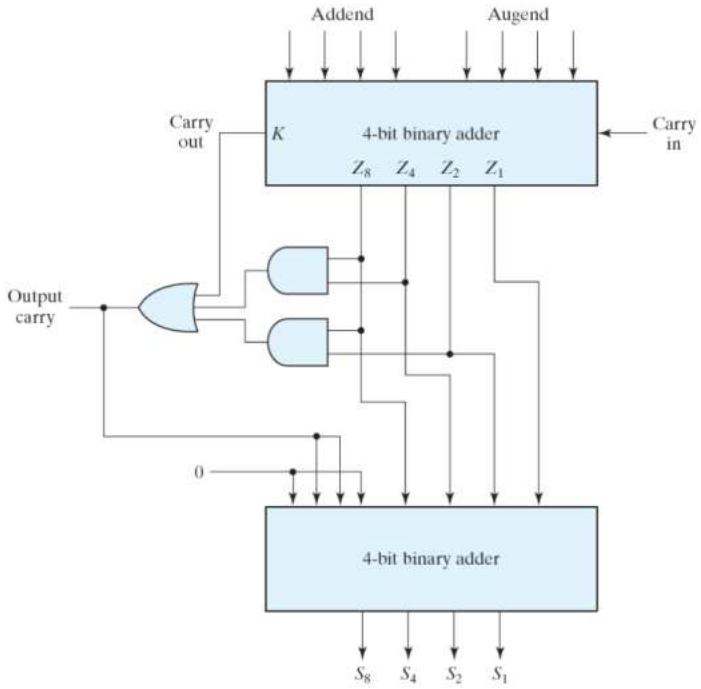

Figure 7. BCD Adder diagram

4. Deliverables

Practical demonstration of the sample exercises will be requested during the lab. Keep in mind that a combined report that presents the analysis and testing of the sample exercises as well as the practice exercises of lab 1 and 2 will be requested, so maintain your project file for the snapshots of the work done.