Wiki »

EXERCISE 4-BIT ADDER

Exercise Goals

- Design a half-/full-/4-bit adder using dataflow and structural styles.
- Use signals and the composite type std_logic_vector.
- Use and understand the signed and unsigned logic types.

Prerequisites

- Quartus Prime incl Modelsim + IntelFPGA University Program Software must be installed and working DO NOT INSTALL IN PATH CONTAINING SPACE CHARACTERS!!!
- See the Quartus Introduction for help on workflow
- See the DE1SoC User Manual for details about the development board
- Optional: Install Notepad++ and plugins
- You can pull template code from the repository (see tab) If you do so, please investigate all the files carefully (incl .qpf) and push to your own repo only

If you need to pull from the repo later on, you can do this by the following cmds:

```
git remote add upstream https://
git fetch upstream
git merge
#git branch -u upstream/master
```

Exercise Steps

1 Create a new project

Follow the Quartus Introduction Using VHDL Design . You must use Modelsim for simulation

2 Half Adder

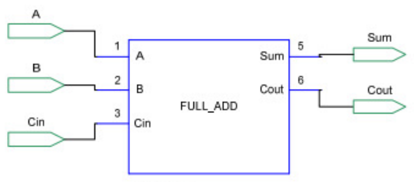
```
Important note for instantiation syntax
 Generally the author of "VHDL for Engineers" uses direct entity instantiation in st

    <label>: entity <component name> port map (a=>a,b=>b,c=>c) e.g.:
    U1: entity or_2 port map  (a=>s3, b=>s2, c=>or_out => carry_out);


 This syntax will not compile, because of namespace issues. The correct syntax is:

    <label>: entity work.<component name> port map (a=>a,b=>b,c=>c) e.g.:
    U1: entity work.or_2 port map (a=>s3, b=>s2, c=>or_out => carry_out);
```

A) Investigate and discuss the half adder listing: 2.6.1, 2.6.2 and 2.6.3 located on page 57-59 in VHDL for Engineers within the group – make sure to understand both the functionality and syntax. See page p. 288- p. 289 in digital fundamentals for further information about half/full adders.

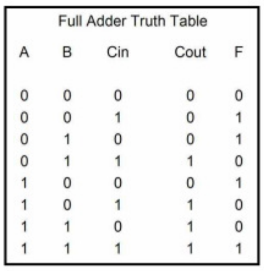
B) Implement the half adder in dataflow style in Quartus. Verify syntax and process the design in Quartus by running "Start Analysis & Synthesis" (Ctrl+k)

C) Verify that your design is synthesized as expected. View the design in Tools => Netlist Viewers => RTL Viewer and Tools => Netlist Viewers=>Technology Map Viewer(post mapping). Is the number of LE used and the truth table for the LE correct?

3 Full adder



Full adder

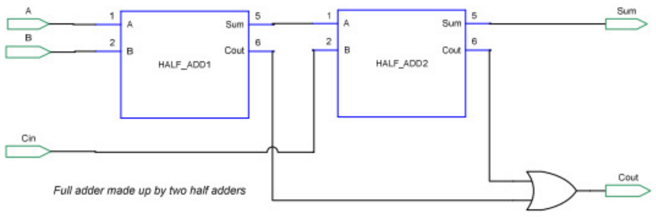


Full Adder Truth Table

A	B	Cin	Cout	F
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$S = (A \oplus B) \oplus C_{in}$$

$$C_{out} = (A \cdot B) + (C_{in} \cdot (A \oplus B)) = (A \cdot B) + (C_{in} \cdot B) + (C_{in} \cdot A)$$



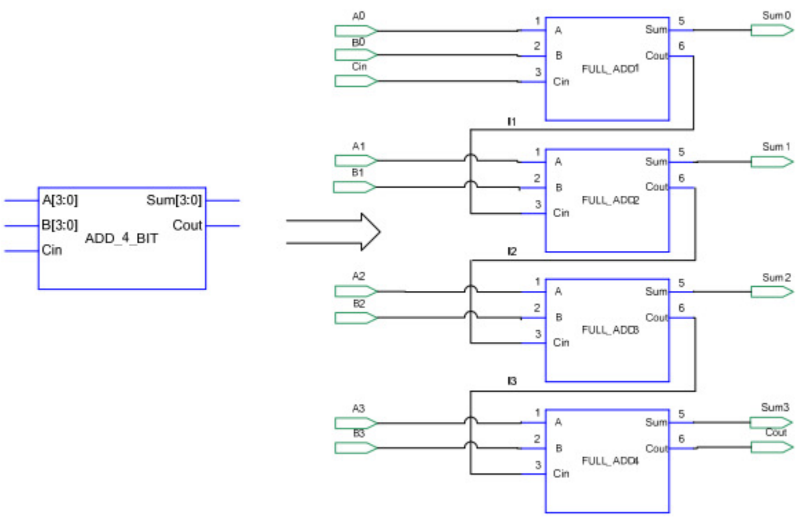
Full adder made up by two half adders

A) Design and implement a full adder as structural style so it consists of two half adders and one or gate.

```
HINTS:
 Look at page 288-289 in digital fundamentals.
 Structural style is not that bad. Full adders are made  up by two half adders, see
 above or see page 288 in digital fundamentals. Also see at the end of chapter 2 whe
 author illustrates a full adder made up with two dataflow style half adders and one
```

B) Verify you design is synthesized as you expect it in Tools => Netlist Viewers => RTL Viewer

4 4-Bit Adder



A) Design and implement a 4 bit parallel adder as a structural style design. Use the full adders from the prior exercise to build the 4 bit parallel adder. Place the full adder in a separate file in the project. A,B and SUM should be of std_logic_vector types. Define Sum0 to Sum3 as internal signals and use concatenation (&) to produce the Sum output.

B) Verify that the full adders are connected correctly in the RTL viewer.

C) Next you have to assign the ports of the FPGA to physical pins. To do so, import an assignments file for the de1soc board:

Download the file from here

Import the assignments from the Assignments->Import menu.

C) Now, create a new top-level design entity for testing you design on the board. This top-level entity should have ports corresponding to those of the FPGA mounted on the DE1SOC-board, that is it shoud use the names just copied into the .qsf file. The architecture of this design-entity should only contain an instantiation of your 4-bit adder that maps ex SW to input a(3 downto 0) and so forth. Note that the names used are the same as the ones printed in white on the DE1SOC board

```
-- Top level wrapper entity
library  ieee;
use ieee.std_logic_1164.all

entity four_bit_adder_de1soc is
port (SW   : in  std_logic_vector(7 downto 0);
      LEDR : out std_logic_vector(4 downto 0));
end four_bit_adder_de1soc ;

architecture structural of four_bit_adder_de1soc is
 begin
  u1: four_bit_adder
    port map(
    a => SW(3 downto 0),
    b => SW(7 downto 4),
    sum => LEDR(3 downto 0),
    carry_out => LEDR(4));
end structural;
```

D) Optional, in the project assignments enable the "SignalTap II Logic Analyzer". Add probes in your design and execute it on the target platform again. Now observe how the logic analyzer is able to monitor internal signals in the FPGA.
See this tutorial on how to use the analyzer There is also a brief demo video

5 Signed / unsigned operators

A) Modify your 4-bit adder to use the "+" operator instead of the full adders previously used. WITH SW[9] you should be able to SELECT between signed and unsigned versions of the adder.

```
Hint! You should look at the signed, unsigned and resize operators of VHDL in solvin
```

B) Download and verify the result

signal_tap_demo.mp4 (7,38 MB) Peter Høgh Mikkelsen, 2018-01-31 11:34