CpE 272 Lab

Lab 7: Modular Design and Library Components

Prepared By:

Olujide Jacobs

Prepared For:

Matt Grubb

October 14, 2013.

**Introduction**

The aim of this lab was to teach students how to simplify the process of designing complex circuits using modular design, and also how make use of the contents in the Quartus II library.

**Experiment I**

The objective of this part of the lab was to design, program and verify the operations of a 4-bit full adder by the way of the modular design technique.

**Methodology**

To achieve this, the group wrote the VHDL code for a 1-bit full adder with inputs X1, Y1 and C0, and outputs Z1 and Z2 where Z1 represented the sum output and Z2 the carryout, finally the file was saved as “light”. Next a new VHDL file named ‘light2’ was created where the inputs, outputs and signals were defined using vectors instead of listing each one of them individually. So inputs eight inputs X1 and Y1 from 3 down to 0, five outputs from Z1 from 4 down to 0 and three signals Z2 from 2 down to 0 were defined. Next the group designed the 4-bit full adder by instantiating the component light, which is the 1-bit full adder design entity previously defined. The component light was instantiated four times, and the connections between all four full adders were made by port mapping the inputs and outputs of each full adder to their correct positions. The schematics for the 4-bit full adder served as a guide while port mapping, and it can be seen below in figure 1, followed by the VHDL code of the 1 and 4 bit adder respectively.

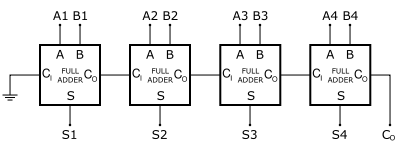


Figure 4-bit full adder

**Code for 1-bit full adder**

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

ENTITY light IS

PORT (

X1 : IN STD\_LOGIC;

Y1 : IN STD\_LOGIC;

C0 : IN STD\_LOGIC;

Z1 : OUT STD\_LOGIC;

Z2 : OUT STD\_LOGIC

);

END light;

ARCHITECTURE Behavior OF light IS

BEGIN

Z1 <= C0 xor (X1 xor Y1);

Z2 <= ((X1 xor Y1) and C0) or (X1 and Y1);

END Behavior;

**Code for 4-bit full adder**

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

ENTITY light2 IS

PORT (

X1 : IN STD\_LOGIC\_VECTOR(3 downto 0);

Y1 : IN STD\_LOGIC\_VECTOR(3 downto 0);

Z1 : OUT STD\_LOGIC\_VECTOR(4 downto 0)

);

END light2;

ARCHITECTURE Behavior OF light2 IS

signal Z2 : STD\_LOGIC\_VECTOR (2 downto 0);

component light

PORT (

X1 : IN STD\_LOGIC;

Y1 : IN STD\_LOGIC;

C0 : IN STD\_LOGIC;

Z1 : OUT STD\_LOGIC;

Z2 : OUT STD\_LOGIC

);

end component;

BEGIN

adder1 : light port map(X1(0), Y1(0), '0', Z1(0), Z2(0));

adder2 : light port map(X1(1), Y1(1), Z2(0), Z1(1), Z2(1));

adder3 : light port map(X1(2), Y1(2), Z2(1), Z1(2), Z2(2));

adder4 : light port map(X1(3), Y1(3), Z2(2), Z1(3), Z1(4));

END Behavior;

After port mapping, the code was compiled and switches and LEDs were assigned to the inputs and outputs respectively. Finally the code was compiled again and then programmed on the FGPA and tested.

**Result**

The circuit worked as it should, the right output was gotten for the different input combinations tested. (1010 + 1010 = 10110), (1010 + 1011 = 10101), (0111 + 1001 = 10000)

**Experiment II**

The objective here is to download a comparator, which compares the magnitude of two n-bit numbers, from the Quartus II library unto the Altera DE2 board and verify its operation.

**Methodology**

Following the instructions provided in the lab handout, the group opened the “mega wizard manager” and “LPM\_COMPARE” from the list of available LPMs. A VHDL output file was chosen and named “cmp.vhd”. Next the group chose three output types to be used namely equal, greater-than and less-than outputs. Afterwards files “cmp.cmp” and “cmp\_inst.vhd”, the two files that contained the declaration and instantiation VHDL codes which were required to be generated by the wizard were chosen. The inputs of the comparator were defined using vectors and are data from 3 down to 0 and datab also from 3 down to 0, with the three outputs AeB, AgB and AlB representing A=B, A>B and A<B.The VHDL code can be seen below

**VHDL Code**

LIBRARY ieee;

USE ieee.std\_logic\_1164.all;

LIBRARY lpm;

USE lpm.all;

ENTITY cmp IS

PORT

(

dataa : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

datab : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

AeB : OUT STD\_LOGIC ;

AgB : OUT STD\_LOGIC ;

AlB : OUT STD\_LOGIC

);

END cmp;

ARCHITECTURE SYN OF cmp IS

SIGNAL sub\_wire0 : STD\_LOGIC ;

SIGNAL sub\_wire1 : STD\_LOGIC ;

SIGNAL sub\_wire2 : STD\_LOGIC ;

COMPONENT lpm\_compare

GENERIC (

lpm\_representation : STRING;

lpm\_type : STRING;

lpm\_width : NATURAL

);

PORT (

dataa : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

datab : IN STD\_LOGIC\_VECTOR (3 DOWNTO 0);

AlB : OUT STD\_LOGIC ;

AgB : OUT STD\_LOGIC ;

AeB : OUT STD\_LOGIC

);

END COMPONENT;

BEGIN

AlB <= sub\_wire0;

AgB <= sub\_wire1;

AeB <= sub\_wire2;

lpm\_compare\_component : lpm\_compare

GENERIC MAP (

lpm\_representation => "UNSIGNED",

lpm\_type => "LPM\_COMPARE",

lpm\_width => 4

)

PORT MAP (

dataa => dataa,

datab => datab,

AlB => sub\_wire0,

AgB => sub\_wire1,

AeB => sub\_wire2

);

END SYN;

Finally switches and LEDs were assigned to the inputs and outputs respectively, the code was compiled, downloaded unto the FGPA and tested.

**Result**

Different values for A and B were used in the test, and on each occasion the comparator correctly signaled whether A was equal to, less than or greater than B through the means of the respective LED lighting up.

**Conclusion**

This experiments performed in the lab were all successful, and I learned how to use modular design to simplify building complex circuits, and also how to make use of the comparator from the Quartus IIs’ library.