LAB2 GRP7 SESS202 REPORT

ECE-124 Lab-2 Submission Form – Winter 2018		
GROUP NUMBER: 7		
SESSION NUMBER: 202		

NAME: (Print)	UW User ID (not Student ID)	Signature
Partner A: Yifan Yu	yf3yu	
Partner B: Daguang Ruan	dzruan	

LAB2 DESIGN DEMO	Marks Allotted	A	B
Seven Segment Display bugs (quantity 3) corrected ?	1	1	1
Operands appear on Digit1 & Digit2 when PB's are OFF ?	1	1	1
Logical Results shown correctly on LEDs[3..0] when PB[2..0] ON ?	1	1	1
Arithmetic results shown on Digits and LED's when PB(3) ON ?	2	2	2
LEDs[7..4] OFF when Arithmetic result Less than or Equal to 1111	2	2	2
DISCUSSION: Describe how you implemented the VHDL coding.	3	3	3
LAB2 DEMO MARK			
LAB2 DESIGN REPORT (see rubric on LEARN for details)	**Marks Allotted**		
Structural VHDL Used in top level VHDL design	2		
Sub-block VHDL files with good Coding Style	2		
Simulation of Logic functions showing the AND,OR,XOR modes	2		
Simulation of Arithmetic functions showing the ADD mode	2		
Total Design Logic Elements Used from Compilation Report	2		
Delay in Report Submission (-1 per day) x number of days:			
LAB2 Report MARK	**Out of 10**		

1) Top level VHDL FILE:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity LogicalStep_Lab2_top is port (
   clkin_50                  : in    std_logic;
       pb                                      : in    std_logic_vector(3 downto 0);
       sw                              : in  std_logic_vector(7 downto 0); -- The switch
inputs
   leds                      : out std_logic_vector(7 downto 0); -- for displaying the
switch content
   seg7_data          : out std_logic_vector(6 downto 0); -- 7-bit outputs to a 7-segment
       seg7_char1     : out   std_logic;                                          -- seg7
digit1 selector
       seg7_char2     : out   std_logic                                           -- seg7
digit2 selector

);
end LogicalStep_Lab2_top;

architecture SimpleCircuit of LogicalStep_Lab2_top is
--
-- Components Used ---
------------------------------------------------------------------
  component SevenSegment port ( -- converts a 4-bit number to 7 bit-number to be use for
the 7-segment display
   hex                :  in  std_logic_vector(3 downto 0);   -- The 4 bit data to be
displayed
   sevenseg    :  out std_logic_vector(6 downto 0)    -- 7-bit outputs to a 7-segment
   );
   end component;

       component concatenate port ( -- concatenate 2 signals(4-bit numbers)
              hexA      :  in  std_logic_vector(3 downto 0);
              hexB           :  in std_logic_vector(3 downto 0);
              output  :      out std_logic_vector(7 downto 0)
       );
       end component;

       component mux port ( --selector to let pass hexin1 or hexin2
              hex_in1, hex_in2       : in std_logic_vector(7 downto 0);
              mux_select : in std_logic;
              hex_out :      out std_logic_vector(7 downto 0)
              );
       end component;

       component segment7_mux port (
              clk    : in  std_logic :='0';
              DIN2   : in std_logic_vector(6 downto 0);
              DIN1   : in std_logic_vector(6 downto 0);
              DOUT   : out std_logic_vector(6 downto 0);
              DIG2   : out std_logic;
              DIG1   : out std_logic
              );
       end component;

       component add port( --function for adding two 4-bit numbers and outputs a 8-bit
number
          hexA    :  in  std_logic_vector(3 downto 0);
       hexB           :  in std_logic_vector(3 downto 0);
              output  :      out std_logic_vector(7 downto 0)
              );
       end component;

       component  Logic_Processor is port ( --function seletor between add, xor, or
```

```
   hexA    :  in  std_logic_vector(3 downto 0);
        hexB            :  in std_logic_vector(3 downto 0);
        pressButton : in std_logic_vector ( 2 downto 0);
        output  :       out std_logic_vector(7 downto 0)

        );
        end component;


-- Create any signals, or temporary variables to be used
--
--  std_logic_vector is a signal which can be used for logic operations such as OR, AND,
NOT, XOR
--
        signal seg7_A          : std_logic_vector(6 downto 0); -- final output for hexA to
seg7
        signal hex_A           : std_logic_vector(3 downto 0); -- input number 1

        signal seg7_B          : std_logic_vector(6 downto 0); --final output for hexB to
seg7
        signal hex_B           : std_logic_vector(3 downto 0); --input number 2

        signal concatenationResult : std_logic_vector(7 downto 0); --result after
concatenation of both inputs
        signal sumResult: std_logic_vector(7 downto 0); --result after adition of the two
inputs

        signal arithmetic_result      :       std_logic_vector(7 downto 0); -- = sum if
pb[3] is pressed  ELSE = concatenation result

        signal logicOutput: std_logic_vector (7 downto 0); ----result obtain after the
logical processor

-- Here the circuit begins

begin

        hex_A <= sw(3 downto 0); --assign input 1 to switches 3 to 0
        hex_B <= sw(7 downto 4); --assign input 2 to switches 7 to 4


        INST1: SevenSegment port map(arithmetic_Result(7 downto 4), seg7_A); --ports input
2 to 7-segment display on the fpga
        INST2: SevenSegment port map(arithmetic_Result(3 downto 0), seg7_B); --ports input
1 to 7-segment display on the fpga
        INST3: segment7_mux port map(clkin_50, seg7_B, seg7_A,  seg7_data, seg7_char2,
seg7_char1); --
        INST4: concatenate port map( hex_B, hex_A, concatenationResult); -- ports
concatenated inputs
        INST5: add port map( hex_B, hex_A, sumResult); --ports added result
        INST6: mux port map(concatenationResult, sumResult, not pb(3), arithmetic_Result);
--ports concatenationResult

        INST7: Logic_Processor port map(hex_A, hex_B, pb (2 downto 0), logicOutput); --
reverted at the lower level
        INST8: mux port map(sumResult, logicOutput, pb(3), leds);  --ports mux result
depending on selected buttons

end SimpleCircuit;
```

2) Subordinate VHDL files

- Mux.vhdl

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity mux is port (

   hex_in1, hex_in2       : in std_logic_vector(7 downto 0); -- input 1 and 2
    mux_select :in std_logic; -- selector
    hex_out :      out std_logic_vector(7 downto 0)


);
end mux;

architecture mux_logic of mux is

begin

    hex_out <= hex_in1 when (mux_select = '0') else hex_in2;
    -- output gets input 1 if selector is 0 (button is pushed) else get input2


end mux_logic;
```

- Add.vhdl

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity add is port (

   hexA       :  in  std_logic_vector(3 downto 0);   --input 1

    hexB           :  in std_logic_vector(3 downto 0); --input 2

    output  :      out std_logic_vector(7 downto 0) --output for added result


);
end add;

architecture Behavioral of add is


--
begin

    --trying to add hexA to hexB

    output(7 downto 0) <=std_logic_vector(unsigned("0000" & hexA) + unsigned("0000" &
hexB)); --function to add both inputs(concatenated)



end architecture Behavioral;
----------------------------------------------------------------------
```

- Concatenate.vhdl

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity concatenate is port (

   hexA       :  in  std_logic_vector(3 downto 0);   --input 1

    hexB           :  in std_logic_vector(3 downto 0); --input 2

    output  :       out std_logic_vector(7 downto 0) --concatenate result


);
end concatenate;

architecture Behavioral of concatenate is

--
begin

    output <= hexA & hexB; --function for concatenation


end architecture Behavioral;
-----------------------------------------------------------------------------------------
---
```

- SevenSegment.vhdl

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

-----------------------------------------------------------------------
-- 7-segment display driver. It displays a 4-bit number on a 7-segment
-- This is created as an entity so that it can be reused many times easily
--

entity SevenSegment is port (

   hex      :  in  std_logic_vector(3 downto 0);   -- The 4 bit data to be displayed

   sevenseg :  out std_logic_vector(6 downto 0)    -- 7-bit outputs to a 7-segment
);
end SevenSegment;

architecture Behavioral of SevenSegment is

--
-- The following statements convert a 4-bit input, called dataIn to a pattern of 7 bits
-- The segment turns on when it is '1' otherwise '0'
--
begin
   with hex select                                                    --GFEDCBA        3210      -- data in
         sevenseg                                                           <= "0111111" when "0000",    --
[0]
                                                                                   "0000110" when "0001",
-- [1]
                                                                                   "1011011" when "0010",
-- [2]      +---- a -----+
                                                                                   "1001111" when "0011",
-- [3]      |            |
                                                                                   "1100110" when "0100",
-- [4]      |            |
                                                                                   "1101101" when "0101",
-- [5]      f            b
                                                                                   "1111101" when "0110",
-- [6]      |            |
                                                                                   "0000111" when "0111",
-- [7]      |            |
                                                                                   "1111111" when "1000",
-- [8]      +---- g -----+
                                                                                   "1101111" when "1001",
-- [9]      |            |
                                                                                   "1110111" when "1010",
-- [A]      |            |
                                                                                   "1111100" when "1011",
-- [b]      e            c
                                                                                   "1011000" when "1100",
-- [c]      |            |
                                                                                   "1011110" when "1101",
-- [d]      |            |
                                                                                   "1111001" when "1110",
-- [E]      +---- d -----+
                                                                                   "1110001" when "1111",
-- [F]
                                                                                   "0000000" when others;
-- [ ]
end architecture Behavioral;
-----------------------------------------------------------------------
```

- Logic_Processor.vhdl

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity Logic_Processor is port (

   hexA   :  in  std_logic_vector(3 downto 0);   --input 1

        hexB           :  in std_logic_vector(3 downto 0); --input 2

        pressButton : in std_logic_vector ( 2 downto 0); --buttons for operator selection

        output  :       out std_logic_vector(7 downto 0) --output of the operation


);
end Logic_Processor;

architecture Behavioral of Logic_Processor is

signal operator : std_logic_vector(2 downto 0);


begin

operator <= not pressButton(2) & not pressButton(1) & not pressButton(0);


        with operator select
        output<= "0000" & (hexA and hexB) when "001", --define the operator depending on button
inputs
                                "0000" & (hexA or hexB) when "010",
                                "0000" & (hexA xor hexB) when "100",
                                "00000000" when others;



end architecture Behavioral;
----------------------------------------------------------------------
```

3) Supporting Images

Image1: graphical simulation of the add functionality

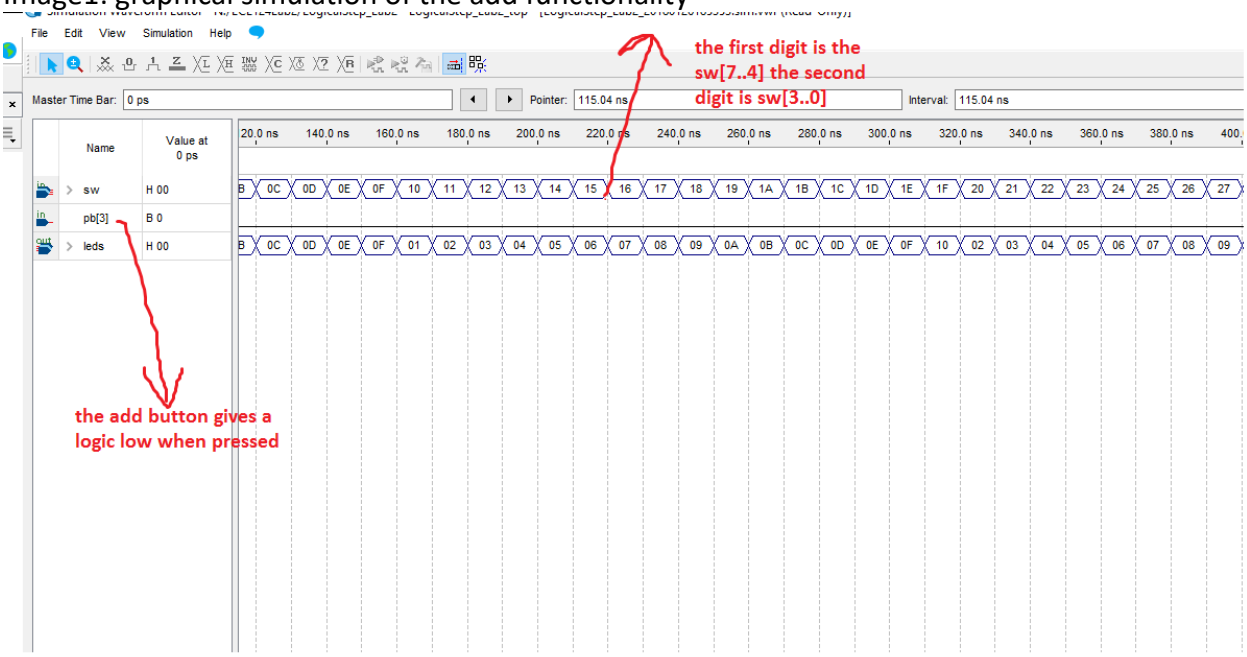

Image2: graphical simulation of the logic gates (AND, XOR, OR) functionalities

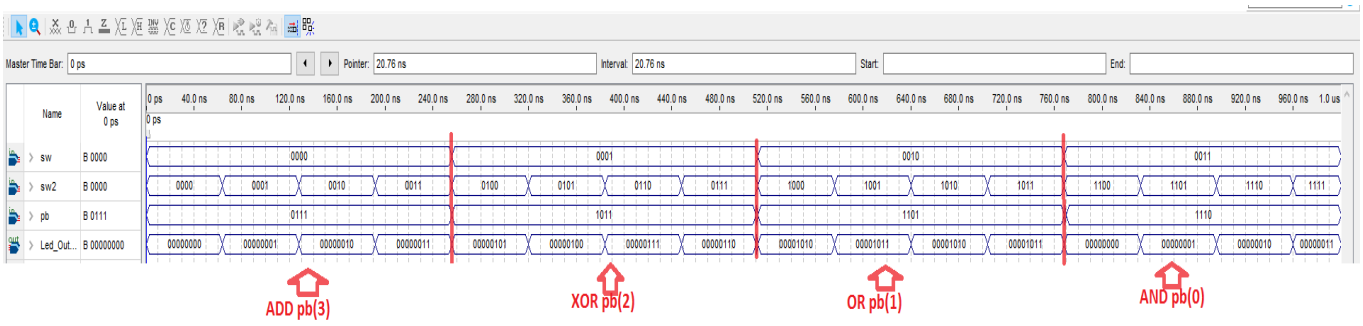

Image3: RTL diagram of all the gates representing a crude ALU

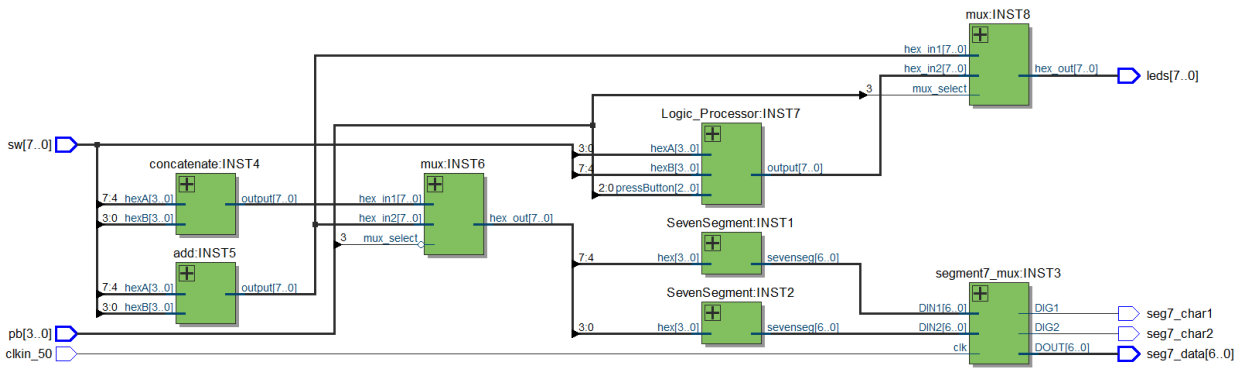

Image 4: Compilation log on Quartus Prime and total number of logic elements