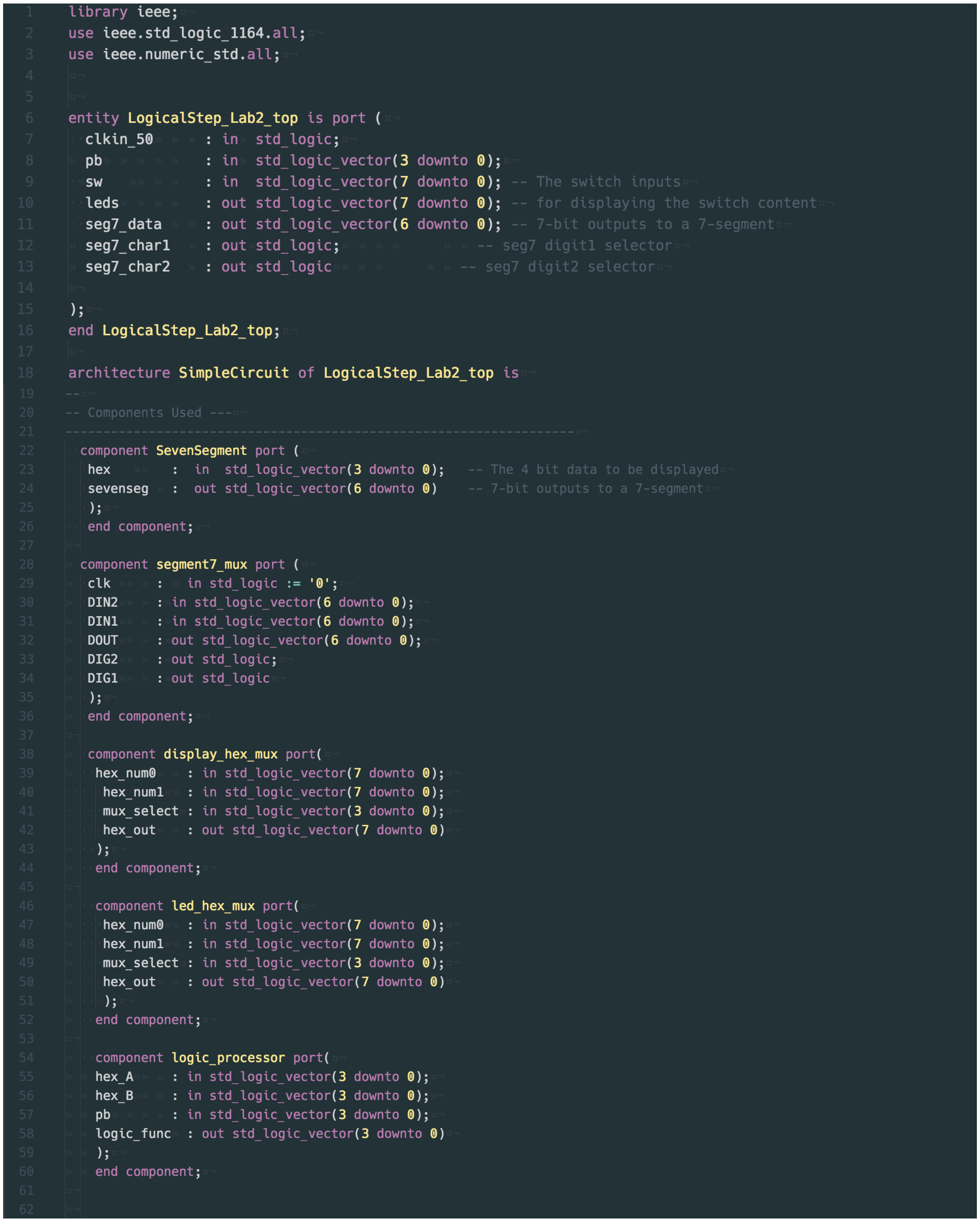
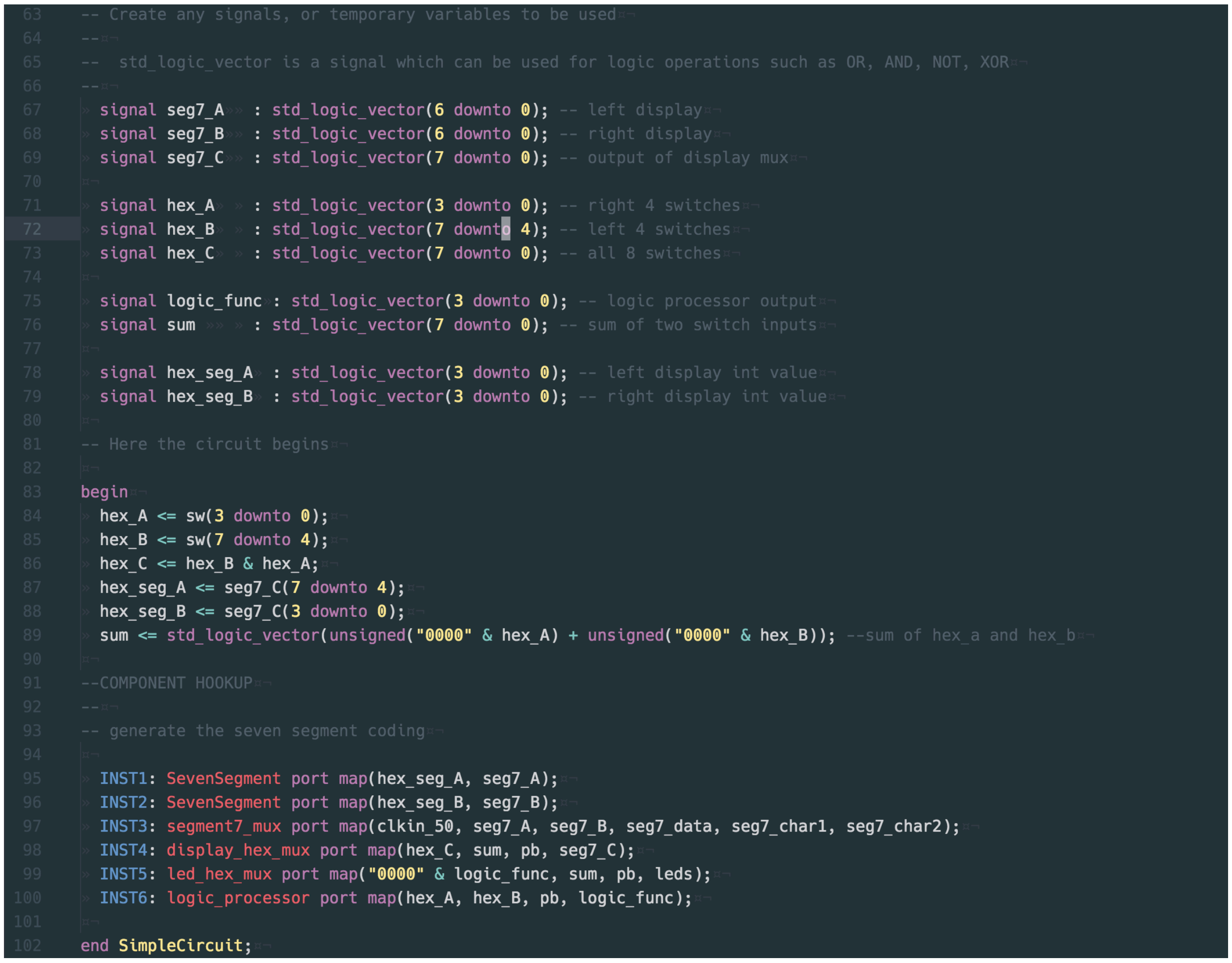
LogicalStep_Lab2_top.vhd

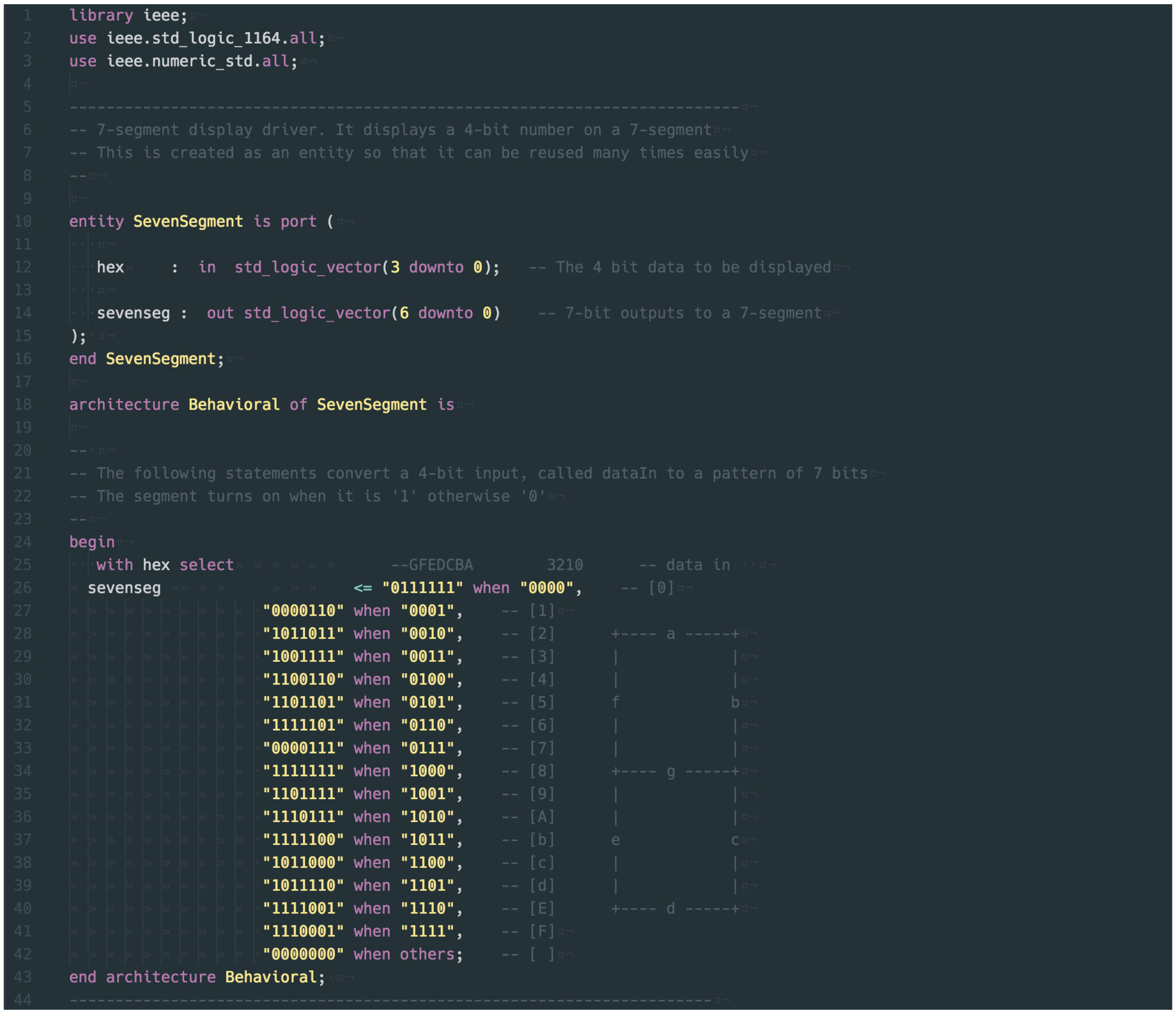


```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


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 (
   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 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 display_hex_mux port(
    hex_num0     : in std_logic_vector(7 downto 0);
     hex_num1    : in std_logic_vector(7 downto 0);
     mux_select : in std_logic_vector(3 downto 0);
     hex_out     : out std_logic_vector(7 downto 0)
    );
    end component;

    component led_hex_mux port(
     hex_num0    : in std_logic_vector(7 downto 0);
     hex_num1    : in std_logic_vector(7 downto 0);
     mux_select : in std_logic_vector(3 downto 0);
     hex_out     : out std_logic_vector(7 downto 0)
     );
    end component;

    component logic_processor port(
    hex_A       : in std_logic_vector(3 downto 0);
    hex_B       : in std_logic_vector(3 downto 0);
    pb          : in std_logic_vector(3 downto 0);
    logic_func  : out std_logic_vector(3 downto 0)
    );
    end component;


```



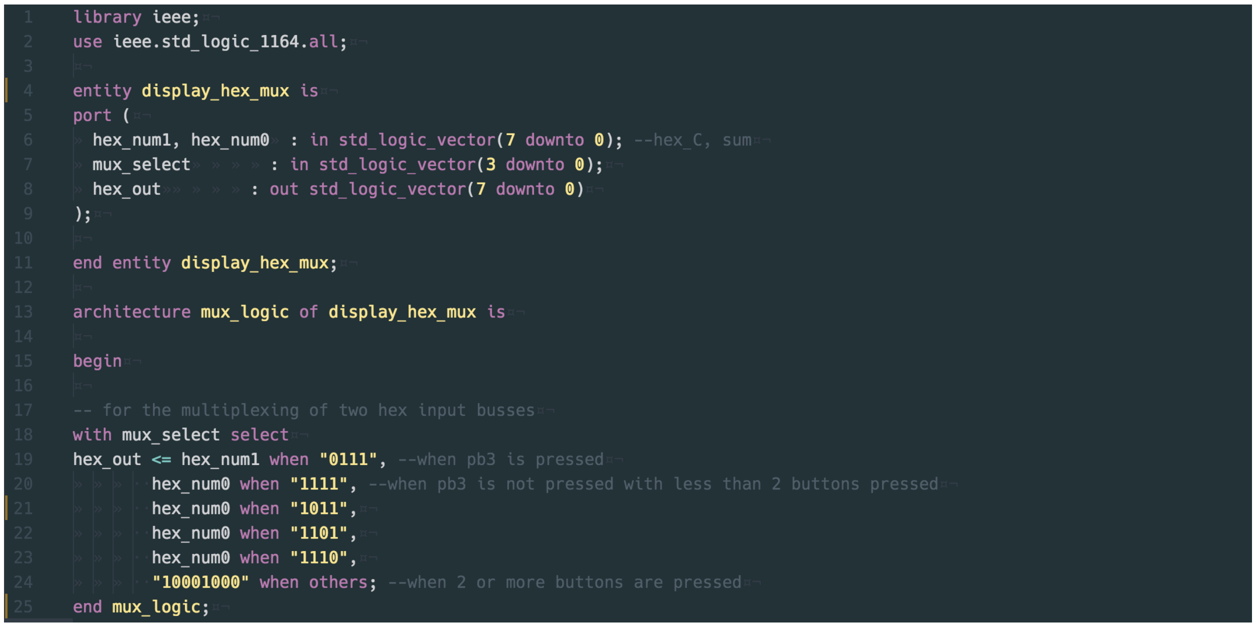
```vhdl
-- Create any signals, or temporary variables to be used
--
--  std_logic_vector is a signal which can be used for logic operations such as OR, AND, NOT, XOR
--
	signal seg7_A		: std_logic_vector(6 downto 0); -- left display
	signal seg7_B		: std_logic_vector(6 downto 0); -- right display
	signal seg7_C		: std_logic_vector(7 downto 0); -- output of display mux

	signal hex_A		: std_logic_vector(3 downto 0); -- right 4 switches
	signal hex_B		: std_logic_vector(7 downto 4); -- left 4 switches
	signal hex_C		: std_logic_vector(7 downto 0); -- all 8 switches

	signal logic_func : std_logic_vector(3 downto 0); -- logic processor output
	signal sum			: std_logic_vector(7 downto 0); -- sum of two switch inputs

	signal hex_seg_A	: std_logic_vector(3 downto 0); -- left display int value
	signal hex_seg_B	: std_logic_vector(3 downto 0); -- right display int value

-- Here the circuit begins

begin
	hex_A <= sw(3 downto 0);
	hex_B <= sw(7 downto 4);
	hex_C <= hex_B & hex_A;
	hex_seg_A <= seg7_C(7 downto 4);
	hex_seg_B <= seg7_C(3 downto 0);
	sum <= std_logic_vector(unsigned("0000" & hex_A) + unsigned("0000" & hex_B)); --sum of hex_a and hex_b

--COMPONENT HOOKUP
--
-- generate the seven segment coding

	INST1: SevenSegment port map(hex_seg_A, seg7_A);
	INST2: SevenSegment port map(hex_seg_B, seg7_B);
	INST3: segment7_mux port map(clkin_50, seg7_A, seg7_B, seg7_data, seg7_char1, seg7_char2);
	INST4: display_hex_mux port map(hex_C, sum, pb, seg7_C);
	INST5: led_hex_mux port map("0000" & logic_func, sum, pb, leds);
	INST6: logic_processor port map(hex_A, hex_B, pb, logic_func);

end SimpleCircuit;
```

SevenSegment.vhd



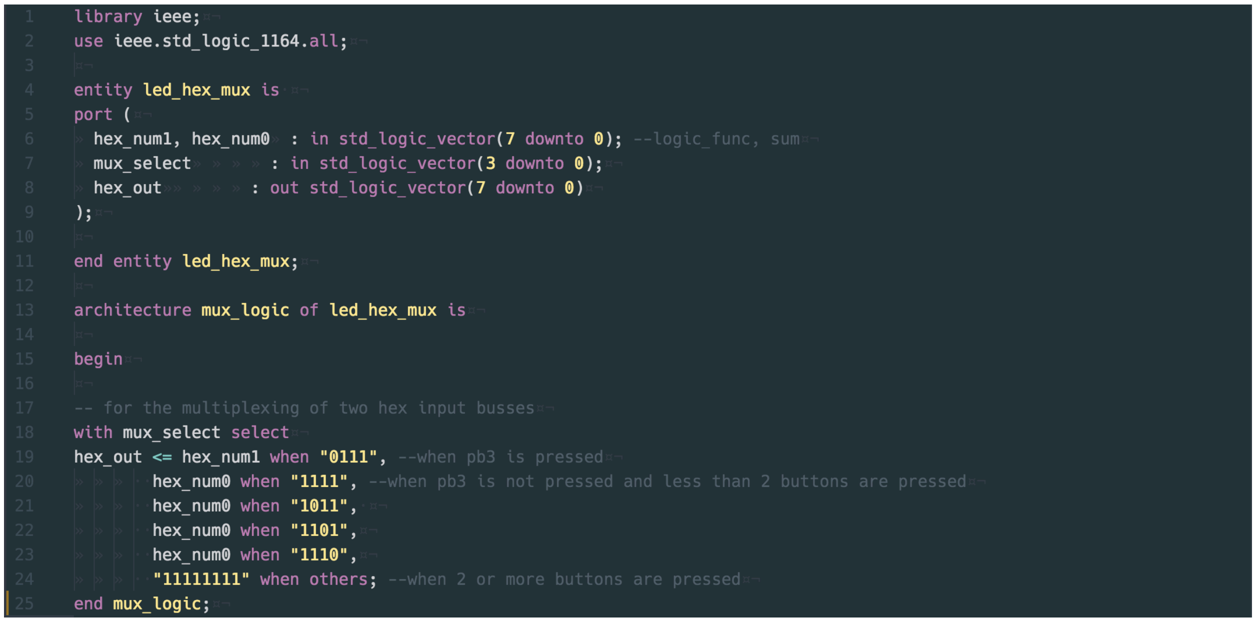
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

-------------------------------------------------------------------------
-- 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;
----------------------------------------------------------------------
```

display_hex_mux.vhd



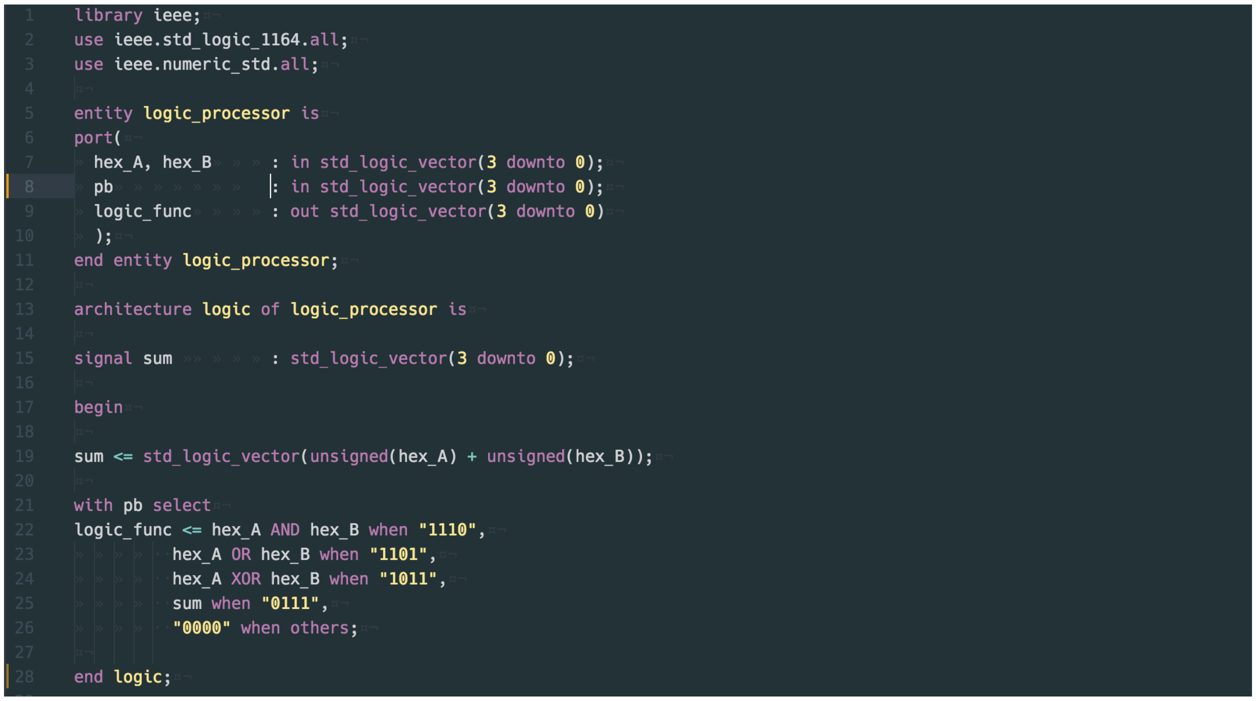
```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity display_hex_mux is
port (
  hex_num1, hex_num0   : in std_logic_vector(7 downto 0); --hex_C, sum
  mux_select         : in std_logic_vector(3 downto 0);
  hex_out          : out std_logic_vector(7 downto 0)
);

end entity display_hex_mux;

architecture mux_logic of display_hex_mux is

begin

-- for the multiplexing of two hex input busses
with mux_select select
hex_out <= hex_num1 when "0111", --when pb3 is pressed
        hex_num0 when "1111", --when pb3 is not pressed with less than 2 buttons pressed
        hex_num0 when "1011",
        hex_num0 when "1101",
        hex_num0 when "1110",
        "10001000" when others; --when 2 or more buttons are pressed
end mux_logic;
```

led_hex_mux.vhd

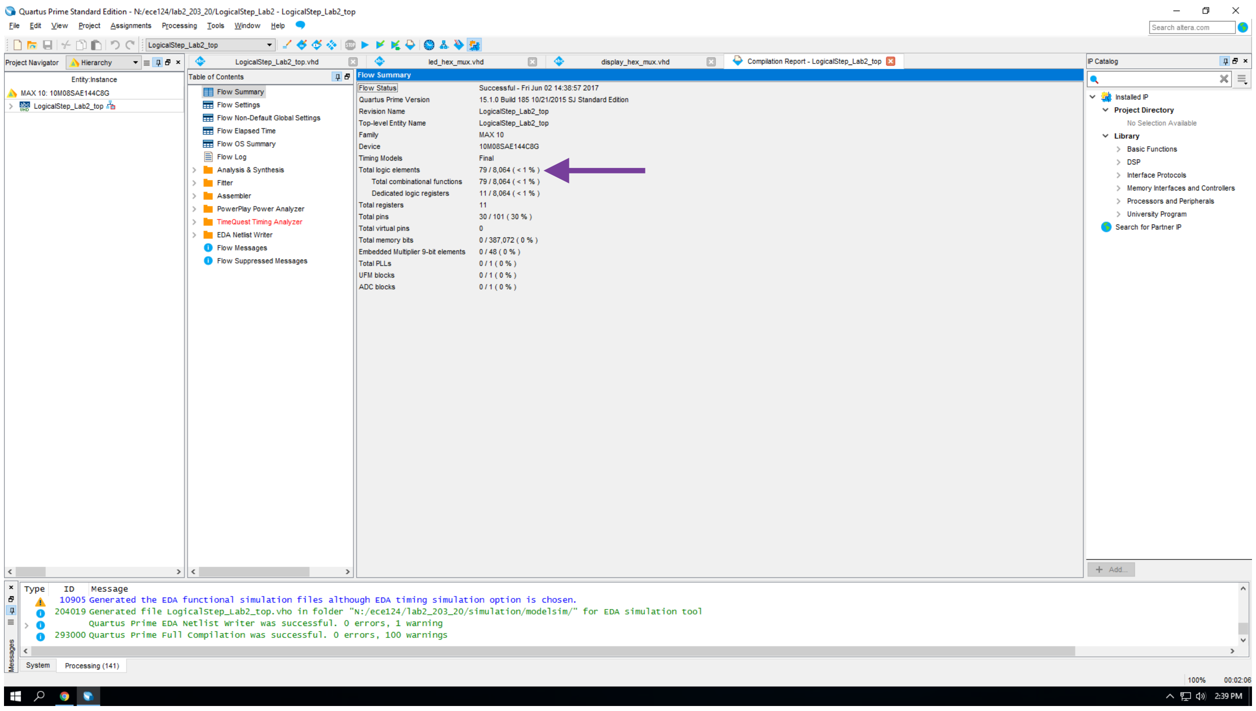


```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity led_hex_mux is
port (
	hex_num1, hex_num0	: in std_logic_vector(7 downto 0); --logic_func, sum
	mux_select		: in std_logic_vector(3 downto 0);
	hex_out			: out std_logic_vector(7 downto 0)
);

end entity led_hex_mux;

architecture mux_logic of led_hex_mux is

begin

-- for the multiplexing of two hex input busses
with mux_select select
hex_out <= hex_num1 when "0111", --when pb3 is pressed
			hex_num0 when "1111", --when pb3 is not pressed and less than 2 buttons are pressed
			hex_num0 when "1011",
			hex_num0 when "1101",
			hex_num0 when "1110",
			"11111111" when others; --when 2 or more buttons are pressed
end mux_logic;
```

logic_processor.vhd



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

entity logic_processor is
port(
	hex_A, hex_B		: in std_logic_vector(3 downto 0);
	pb					: in std_logic_vector(3 downto 0);
	logic_func			: out std_logic_vector(3 downto 0)
	);
end entity logic_processor;

architecture logic of logic_processor is

signal sum			: std_logic_vector(3 downto 0);

begin

sum <= std_logic_vector(unsigned(hex_A) + unsigned(hex_B));

with pb select
logic_func <= hex_A AND hex_B when "1110",
				hex_A OR hex_B when "1101",
				hex_A XOR hex_B when "1011",
				sum when "0111",
				"0000" when others;

end logic;
```



Flow Summary

Flow Status	Successful - Fri Jun 02 14:38:57 2017
Quartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Standard Edition
Revision Name	LogicalStep_Lab2_top
Top-level Entity Name	LogicalStep_Lab2_top
Family	MAX 10
Device	10M08SAE144C8G
Timing Models	Final
Total logic elements	79 / 8,064 (< 1 %)
Total combinational functions	79 / 8,064 (< 1 %)
Dedicated logic registers	11 / 8,064 (< 1 %)
Total registers	11
Total pins	30 / 101 (30 %)
Total virtual pins	0
Total memory bits	0 / 387,072 (0 %)
Embedded Multiplier 9-bit elements	0 / 48 (0 %)
Total PLLs	0 / 1 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 1 (0 %)

```
Type    ID   Message
      10905 Generated the EDA functional simulation files although EDA timing simulation option is chosen.
     204019 Generated file LogicalStep_Lab2_top.vho in folder "N:/ece124/lab2_203_20/simulation/modelsim/" for EDA simulation tool
            Quartus Prime EDA Netlist Writer was successful. 0 errors, 1 warning
     293000 Quartus Prime Full Compilation was successful. 0 errors, 100 warnings
```

Figure 0: compilation report

Simulations

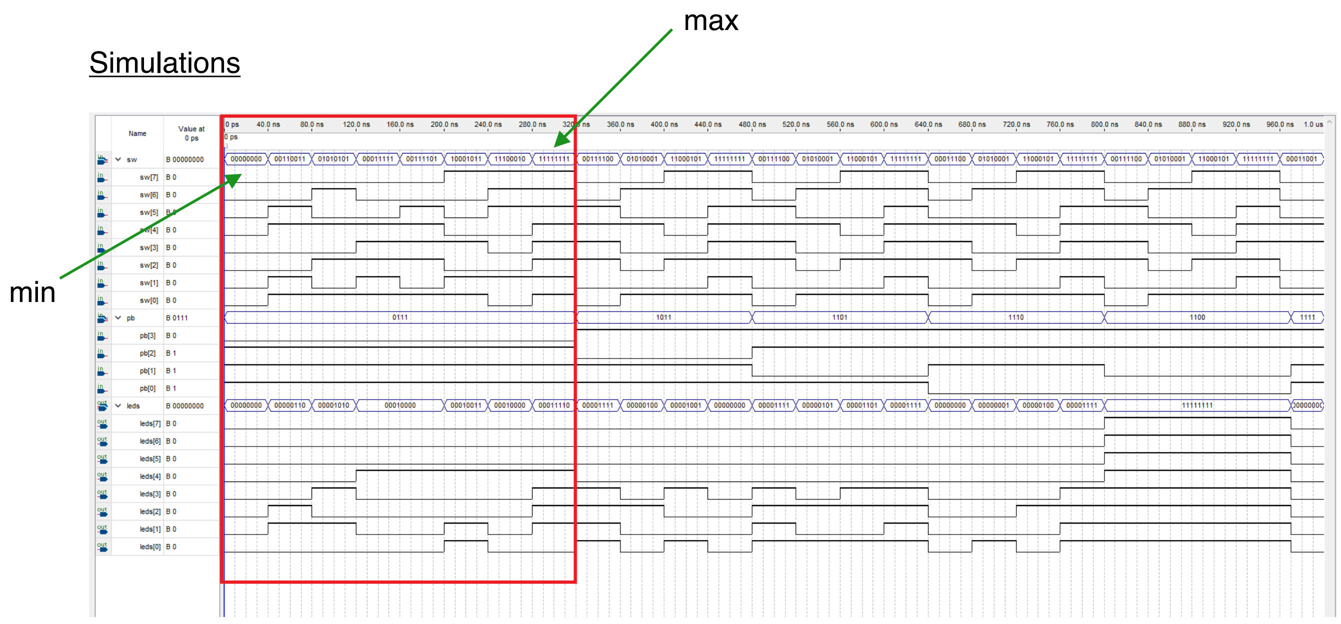


Figure 1: ADD operator
min value is 0 and max value is 30 (F + F = 15 + 15 = 30)
eg. 1000 ADD 1011 —> 00010011 (8 + D = 13)

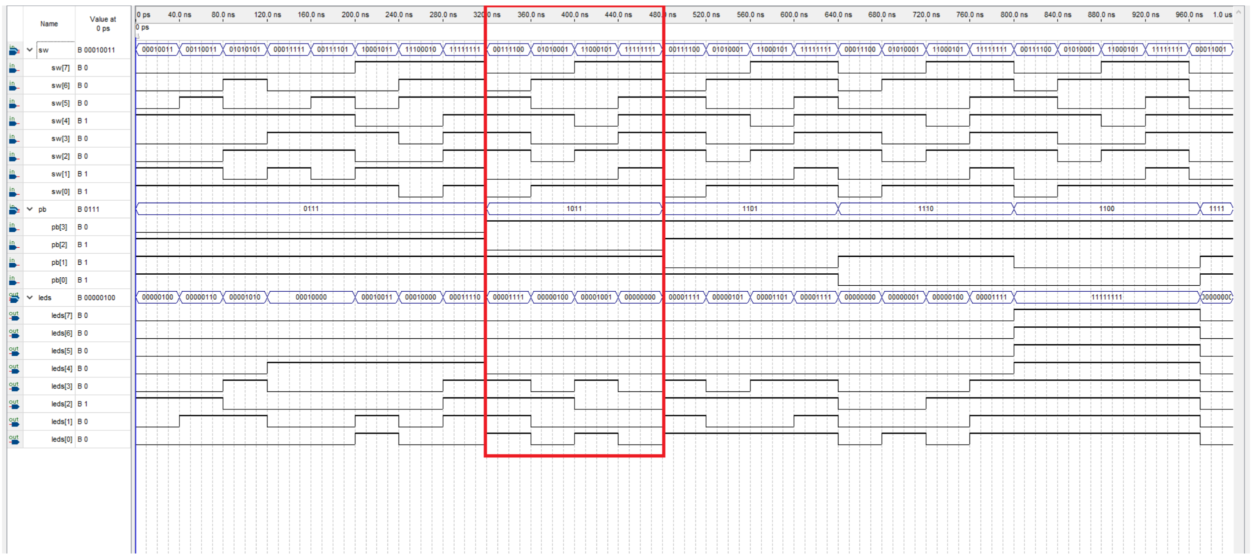


Figure 2: XOR operator
eg. 1000 XOR 1011 —> 00000011

Note: AND, OR and XOR operators will always have an output with “0000” as the first 4-bits because the output is only 4-bits which is shown on leds[3..0]

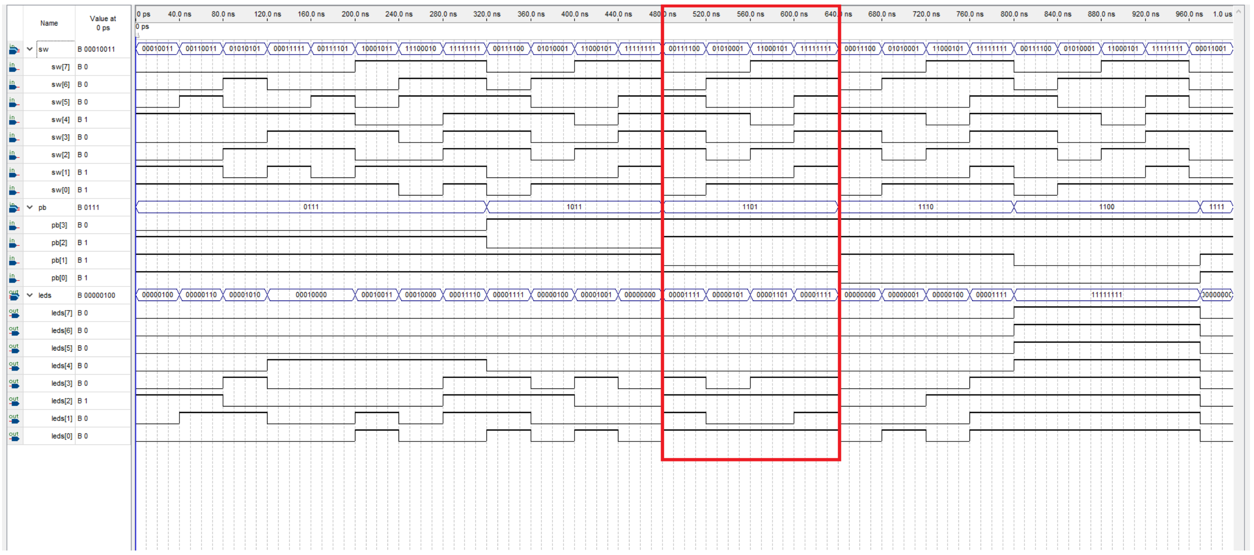


Figure 3: OR operator
eg. 1000 OR 1011 —> 00001011

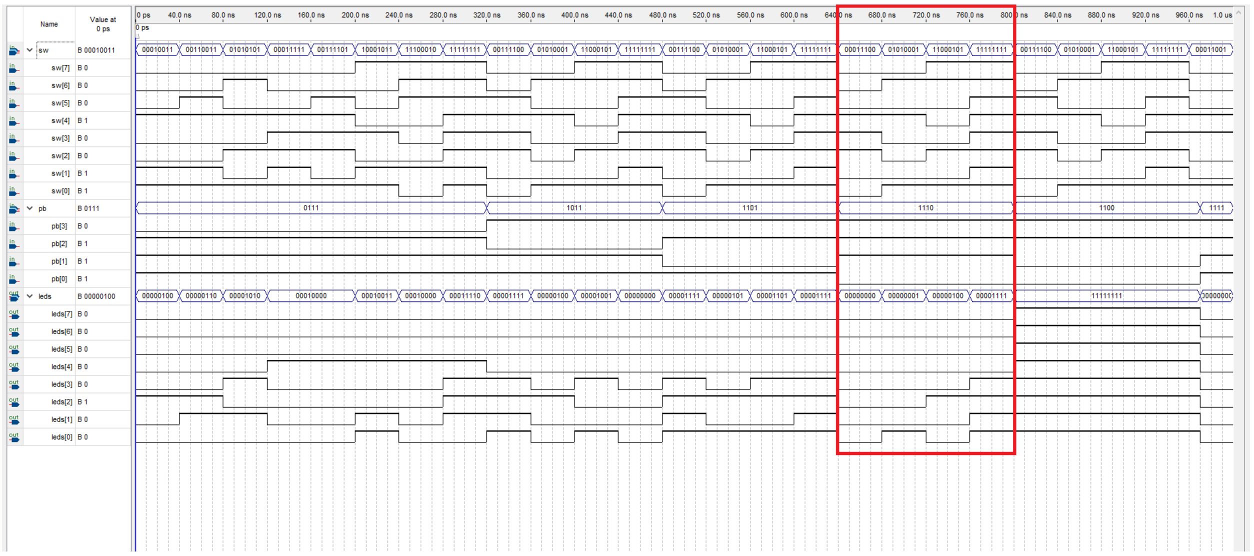


Figure 4: AND operator
eg. 1000 OR 1011 —> 00001000

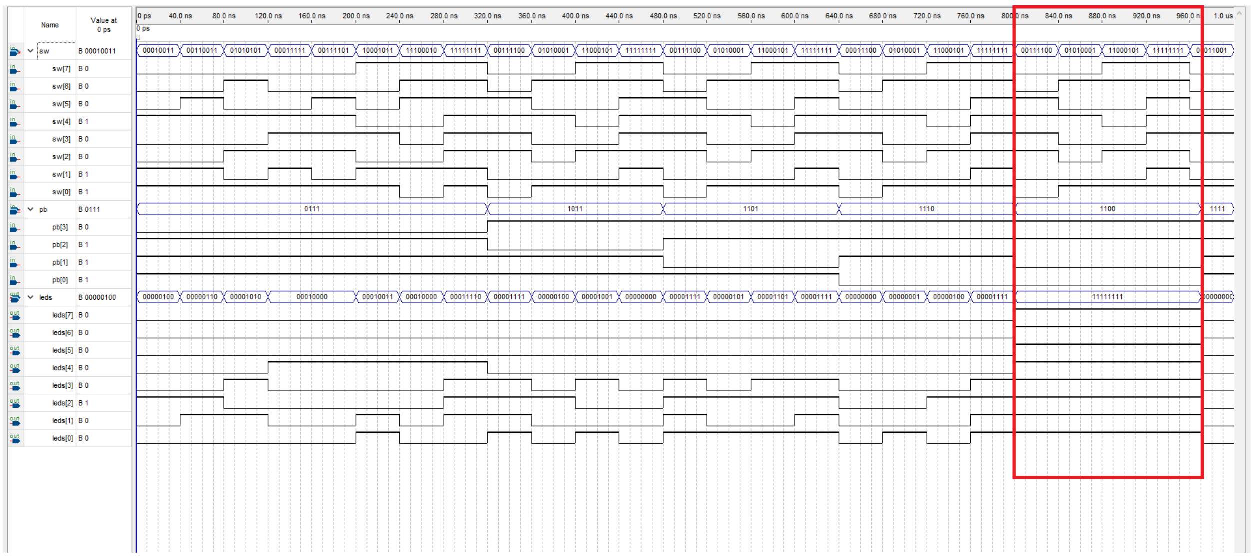


Figure 5: BONUS (when more than one button is pressed, all leds are on)

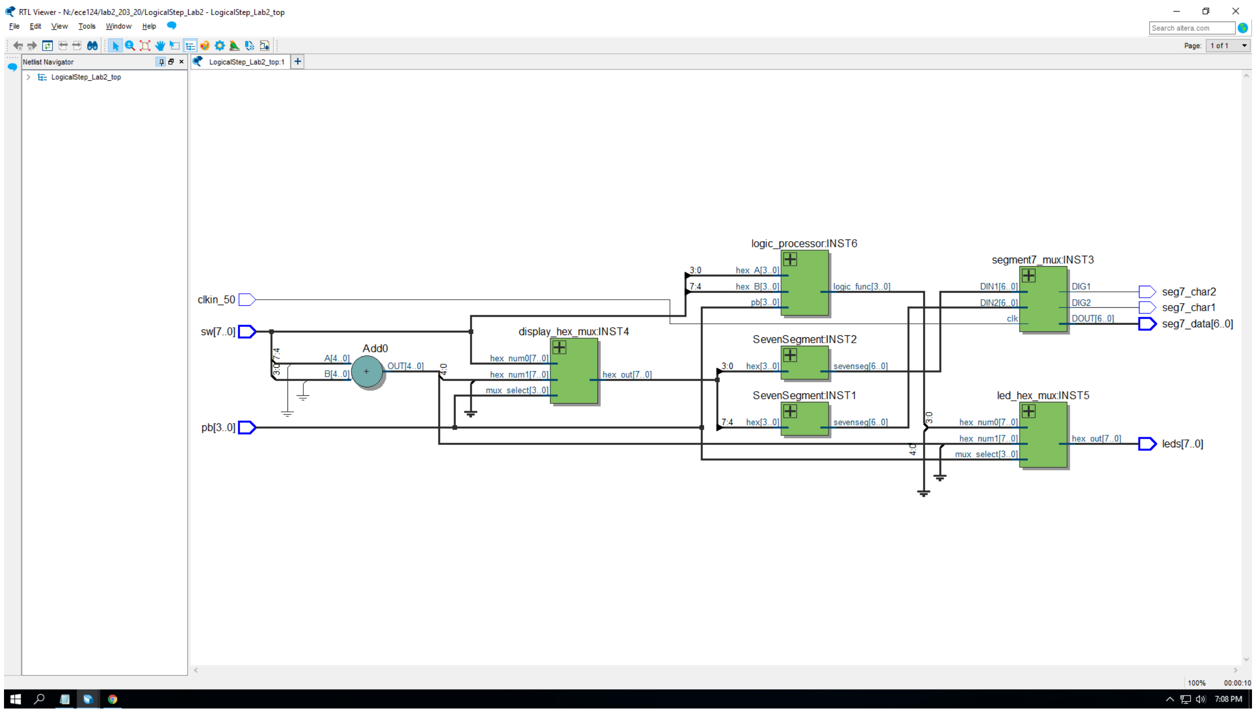


Figure 6: RTL viewer