Getting Started with VHDL Coding
Lab Report 1

Lab Section: 003
Group 29

Zhiyi Sun 260768094
Nariman Zendehrooh 260700556
Mcgill University, Montreal, Quebec, Canada

zhiyi.sun@mail.mcgill.ca
Nariman.zendehroohKermani@mail.McGill.ca

1.0 Table of contents

2.0 7-segment Decoder circuit

2.1 A description of the 7-segment decoder circuit. Explain why you used the selected signal assignment instead of the conditional signal assignment.

The selected signal assignment allows us to implement the functionality of a multiplexer. In the 7-segment decoder circuit, we take 4 bits binary number as a control input and generates the appropriate 7-segment display associated with the input code. The select input chooses between 16 sources of data which are 7 bits numbers. These outputs numbers are made active-low means that 0 will turn the segment on. Each bit of the 7 bits numbers is used to activate a single segment of the 7-segments LED. Combination of these segments will represent a hexadecimal digit. Therefore, these 7 segments will display numbers from 0 to 9 and letters from A to F. Thus, there are 16 cases to be considered and the other cases are unwanted that will be considered as others clause in the code. The selected signal assignment is used because there are limited numbers of cases to be considered, and it will not allow for choices to be overlapped. In the selected signal assignment, all possible conditions must be specified, so it is useful for cases that we have a limited number of desirable options like our case. When the process is executed, signal assignment statements are performed sequentially. However, only the assignment with the given input will be executed. The given case for each input will prevent a priority to be applied. However, the conditional signal assignment always enforces a priority on the conditions which is implied by the ordering of the expressions. For our purpose, it is better to use the selected signal assignment because we have a limited number of desirable cases. Plus, we have no priority and order to choose between the cases. In other words, each case can be chosen at any moments.

2.2 A discussion of how the 7-segment decoder circuit was tested, showing representative simulation plots. How do you know that the circuit works correctly?

Before combining the 7-segments decoder with the adder to show the addition of two hexadecimal numbers on the hardware, we have to test that the logic behind the 7-segments decoder is correct to prevent any further problems in the next steps of the design and to make the debugging process easier. For our purpose, we perform a functional simulation that means we only test if the circuit performs the desired function so that we do not have to map our design to the target hardware, so we cannot test on timing constraints. Since we do not have any hardware to give inputs to the circuit manually, if we run the simulation, both inputs and outputs will be undefined. Thus, we used testbench to generate different inputs that will be applied to our circuit so that we can automate the simulation of the circuit and see how the outputs respond to different inputs. We have 16 cases to be tested so that inside the testbench code we create a loop that will generate all the 16 possible cases as inputs. After the compilation of the testbench code, we run the simulation. The simulation creates the waveforms platform. We add the signals to the waveform which are the generated inputs by the testbench code and the output signal will be displayed at the bottom of each input signal. If the displayed inputs and outputs are matched as we expected, it means that the circuit works functionally correct.

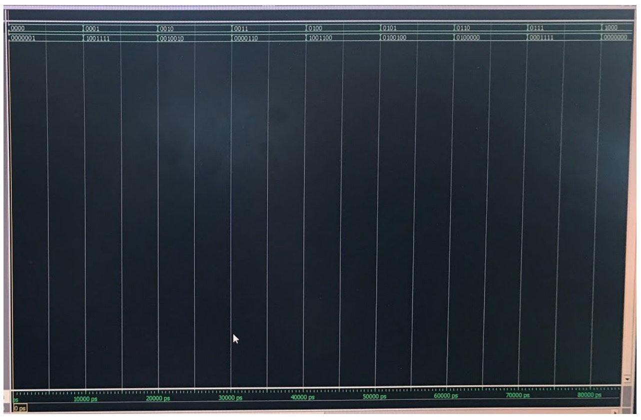


Fig.1.2.1 This figure gives the result of signal testing

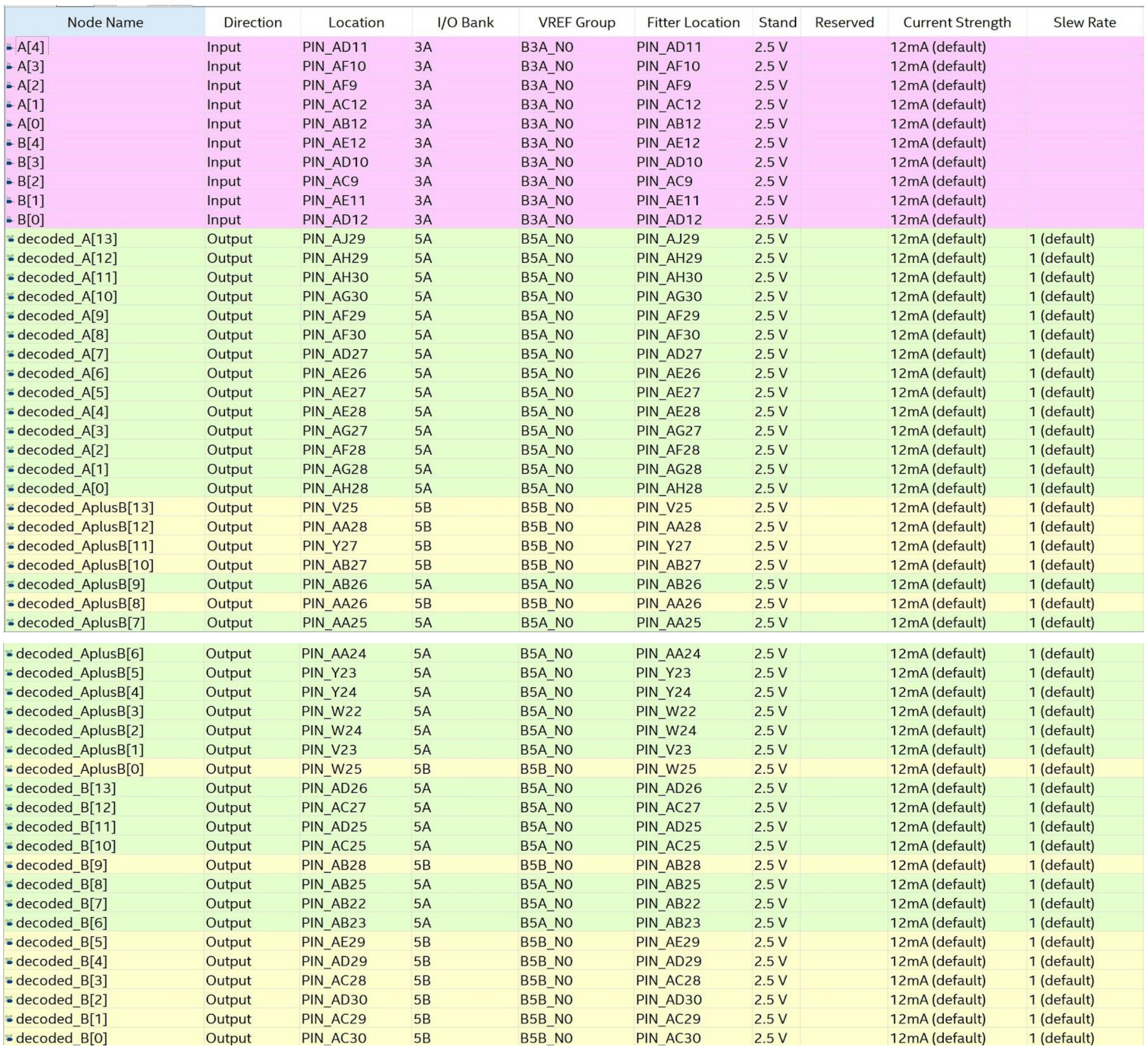
3.0 Adder circuit

3.1 A description of the adder circuit. How many 7-segment decoder instances did you use in your design and why?

Six 7-segment decoders are used in the adder circuit. As the two inputs are 5-bits binary numbers controlled by 10 switches (each switch controls a corresponding bit), these two numbers range from 0 to 31 in decimal and the output will range from 0 to 62, which all require 2 digits in hexadecimal to represent (range from 0 to 1F for the two inputs and from 0 to 3E for the output). Since one 7-segment decoder controls 7 corresponding LEDs on the board, which will represent a digit as a whole, six 7-segment decoders are needed to represent the two input values in hexadecimal (each contains 2 digits: 2+2=4 in total) and one output (which contains two digits). Basically, it's 2(input 1)+2(input 2)+2(output)=6.

3.2 A discussion of how the adder circuit was tested.

In this lab, the adder code is tested using the Altera DE1-SoCboard. The code for adder circuit is first compiled in the Quartus software; once it is done, we map the code to the Altera DE1-SoCboard by assigning the 7-segments declared in the code to their corresponding switches and output LEDs on the board. The locations for the inputs values and output values are given in the tables in the DE1-SoC Development and Education Board Users Manual. However, the order of these locations has to be adjusted to their corresponding variables declared in the adder code. We first fill in the locations to the pin planner following the exact same ascending order as in the given tables and find the values shown by the LEDs on the board left-to-right reversed. Due to this observation, we finally adjusted the pin locations as the following:



Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	Stand	Reserved	Current Strength	Slew Rate
A[4]	Input	PIN_AD11	3A	B3A_N0	PIN_AD11	2.5 V		12mA (default)	
A[3]	Input	PIN_AF10	3A	B3A_N0	PIN_AF10	2.5 V		12mA (default)	
A[2]	Input	PIN_AF9	3A	B3A_N0	PIN_AF9	2.5 V		12mA (default)	
A[1]	Input	PIN_AC12	3A	B3A_N0	PIN_AC12	2.5 V		12mA (default)	
A[0]	Input	PIN_AB12	3A	B3A_N0	PIN_AB12	2.5 V		12mA (default)	
B[4]	Input	PIN_AE12	3A	B3A_N0	PIN_AE12	2.5 V		12mA (default)	
B[3]	Input	PIN_AD10	3A	B3A_N0	PIN_AD10	2.5 V		12mA (default)	
B[2]	Input	PIN_AC9	3A	B3A_N0	PIN_AC9	2.5 V		12mA (default)	
B[1]	Input	PIN_AE11	3A	B3A_N0	PIN_AE11	2.5 V		12mA (default)	
B[0]	Input	PIN_AD12	3A	B3A_N0	PIN_AD12	2.5 V		12mA (default)	
decoded_A[13]	Output	PIN_AJ29	5A	B5A_N0	PIN_AJ29	2.5 V		12mA (default)	1 (default)
decoded_A[12]	Output	PIN_AH29	5A	B5A_N0	PIN_AH29	2.5 V		12mA (default)	1 (default)
decoded_A[11]	Output	PIN_AH30	5A	B5A_N0	PIN_AH30	2.5 V		12mA (default)	1 (default)
decoded_A[10]	Output	PIN_AG30	5A	B5A_N0	PIN_AG30	2.5 V		12mA (default)	1 (default)
decoded_A[9]	Output	PIN_AF29	5A	B5A_N0	PIN_AF29	2.5 V		12mA (default)	1 (default)
decoded_A[8]	Output	PIN_AF30	5A	B5A_N0	PIN_AF30	2.5 V		12mA (default)	1 (default)
decoded_A[7]	Output	PIN_AD27	5A	B5A_N0	PIN_AD27	2.5 V		12mA (default)	1 (default)
decoded_A[6]	Output	PIN_AE26	5A	B5A_N0	PIN_AE26	2.5 V		12mA (default)	1 (default)
decoded_A[5]	Output	PIN_AE27	5A	B5A_N0	PIN_AE27	2.5 V		12mA (default)	1 (default)
decoded_A[4]	Output	PIN_AE28	5A	B5A_N0	PIN_AE28	2.5 V		12mA (default)	1 (default)
decoded_A[3]	Output	PIN_AG27	5A	B5A_N0	PIN_AG27	2.5 V		12mA (default)	1 (default)
decoded_A[2]	Output	PIN_AF28	5A	B5A_N0	PIN_AF28	2.5 V		12mA (default)	1 (default)
decoded_A[1]	Output	PIN_AG28	5A	B5A_N0	PIN_AG28	2.5 V		12mA (default)	1 (default)
decoded_A[0]	Output	PIN_AH28	5A	B5A_N0	PIN_AH28	2.5 V		12mA (default)	1 (default)
decoded_AplusB[13]	Output	PIN_V25	5B	B5B_N0	PIN_V25	2.5 V		12mA (default)	1 (default)
decoded_AplusB[12]	Output	PIN_AA28	5B	B5B_N0	PIN_AA28	2.5 V		12mA (default)	1 (default)
decoded_AplusB[11]	Output	PIN_Y27	5B	B5B_N0	PIN_Y27	2.5 V		12mA (default)	1 (default)
decoded_AplusB[10]	Output	PIN_AB27	5B	B5B_N0	PIN_AB27	2.5 V		12mA (default)	1 (default)
decoded_AplusB[9]	Output	PIN_AB26	5A	B5A_N0	PIN_AB26	2.5 V		12mA (default)	1 (default)
decoded_AplusB[8]	Output	PIN_AA26	5B	B5B_N0	PIN_AA26	2.5 V		12mA (default)	1 (default)
decoded_AplusB[7]	Output	PIN_AA25	5A	B5A_N0	PIN_AA25	2.5 V		12mA (default)	1 (default)
decoded_AplusB[6]	Output	PIN_AA24	5A	B5A_N0	PIN_AA24	2.5 V		12mA (default)	1 (default)
decoded_AplusB[5]	Output	PIN_Y23	5A	B5A_N0	PIN_Y23	2.5 V		12mA (default)	1 (default)
decoded_AplusB[4]	Output	PIN_Y24	5A	B5A_N0	PIN_Y24	2.5 V		12mA (default)	1 (default)
decoded_AplusB[3]	Output	PIN_W22	5A	B5A_N0	PIN_W22	2.5 V		12mA (default)	1 (default)
decoded_AplusB[2]	Output	PIN_W24	5A	B5A_N0	PIN_W24	2.5 V		12mA (default)	1 (default)
decoded_AplusB[1]	Output	PIN_V23	5A	B5A_N0	PIN_V23	2.5 V		12mA (default)	1 (default)
decoded_AplusB[0]	Output	PIN_W25	5B	B5B_N0	PIN_W25	2.5 V		12mA (default)	1 (default)
decoded_B[13]	Output	PIN_AD26	5A	B5A_N0	PIN_AD26	2.5 V		12mA (default)	1 (default)
decoded_B[12]	Output	PIN_AC27	5A	B5A_N0	PIN_AC27	2.5 V		12mA (default)	1 (default)
decoded_B[11]	Output	PIN_AD25	5A	B5A_N0	PIN_AD25	2.5 V		12mA (default)	1 (default)
decoded_B[10]	Output	PIN_AC25	5A	B5A_N0	PIN_AC25	2.5 V		12mA (default)	1 (default)
decoded_B[9]	Output	PIN_AB28	5B	B5B_N0	PIN_AB28	2.5 V		12mA (default)	1 (default)
decoded_B[8]	Output	PIN_AB25	5A	B5A_N0	PIN_AB25	2.5 V		12mA (default)	1 (default)
decoded_B[7]	Output	PIN_AB22	5A	B5A_N0	PIN_AB22	2.5 V		12mA (default)	1 (default)
decoded_B[6]	Output	PIN_AB23	5A	B5A_N0	PIN_AB23	2.5 V		12mA (default)	1 (default)
decoded_B[5]	Output	PIN_AE29	5B	B5B_N0	PIN_AE29	2.5 V		12mA (default)	1 (default)
decoded_B[4]	Output	PIN_AD29	5B	B5B_N0	PIN_AD29	2.5 V		12mA (default)	1 (default)
decoded_B[3]	Output	PIN_AC28	5B	B5B_N0	PIN_AC28	2.5 V		12mA (default)	1 (default)
decoded_B[2]	Output	PIN_AD30	5B	B5B_N0	PIN_AD30	2.5 V		12mA (default)	1 (default)
decoded_B[1]	Output	PIN_AC29	5B	B5B_N0	PIN_AC29	2.5 V		12mA (default)	1 (default)
decoded_B[0]	Output	PIN_AC30	5B	B5B_N0	PIN_AC30	2.5 V		12mA (default)	1 (default)

Fig.2.2.1 These figures show the locations of the pins in the pin planner

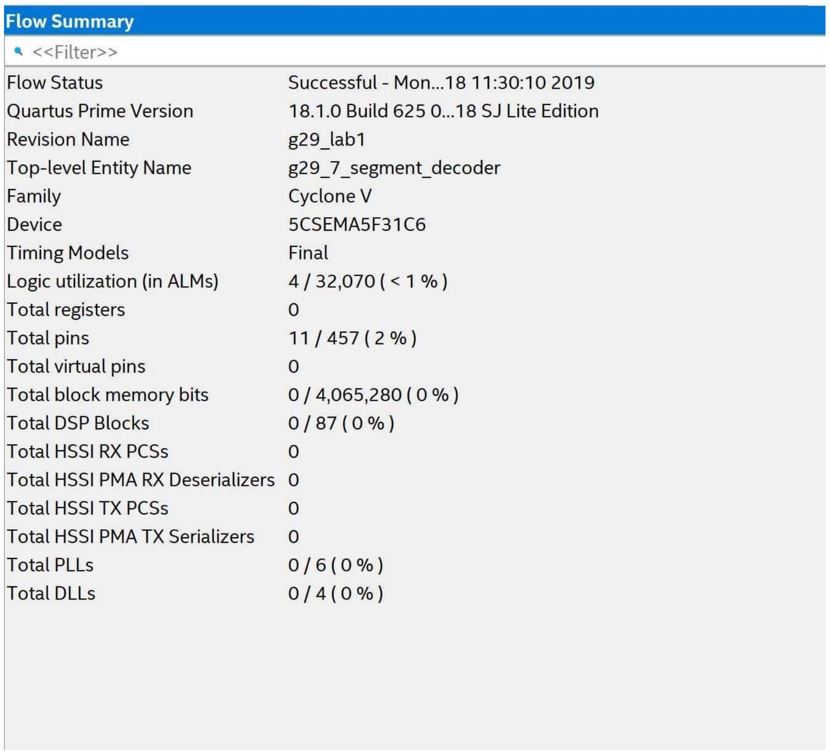
Tests are then run using the board and the records are shown in the following table. The two input values are controlled by ten switches on the board: switch on indicating a "1" and switch off indicating a "0". From these tests, we determine our adder circuit code is working properly.

Trials	input value 1 controlled by switches (5-bit unsigned binary	input value 2 controlled by switches (5-bit unsigned binary	input value 1 shown on the board (2-bit hexadecimal	input value 2 shown on the board (2-bit hexadecimal	Output value on the board (2-bit hexadecimal

	number)	number)	number)	number)	number)
1	00000	00000	00	00	00
2	00001	00001	01	01	02
3	00100	00010	04	02	06
4	01000	01100	08	0C	14
5	10000	10000	10	10	20
6	10100	00010	14	02	16
7	10001	11000	11	18	29
8	10011	00100	13	04	17
9	10100	11111	14	1F	33
10	11111	11111	1F	1F	3E

Table 2.2-1 This table gives the results of the test on adder circuit using the Altera DE1-SoCboard

4.0 Summary of FPGA resource utilization

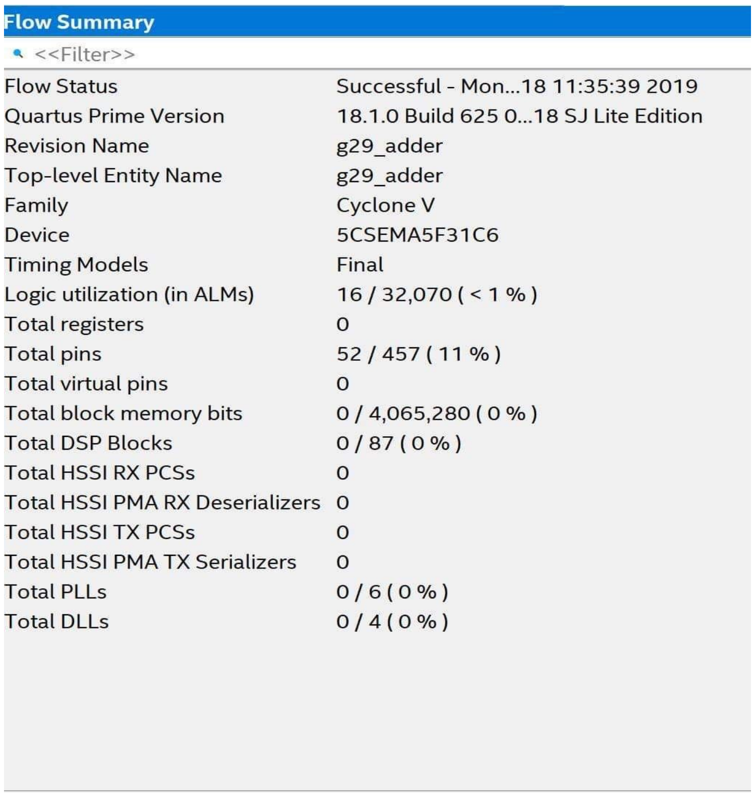


Flow Summary

<<Filter>>

Flow Status	Successful - Mon...18 11:30:10 2019
Quartus Prime Version	18.1.0 Build 625 0...18 SJ Lite Edition
Revision Name	g29_lab1
Top-level Entity Name	g29_7_segment_decoder
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	4 / 32,070 (< 1 %)
Total registers	0
Total pins	11 / 457 (2 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Fig.4.1 This figure shows the compilation report's flow summary of the 7-segments decoder circuit

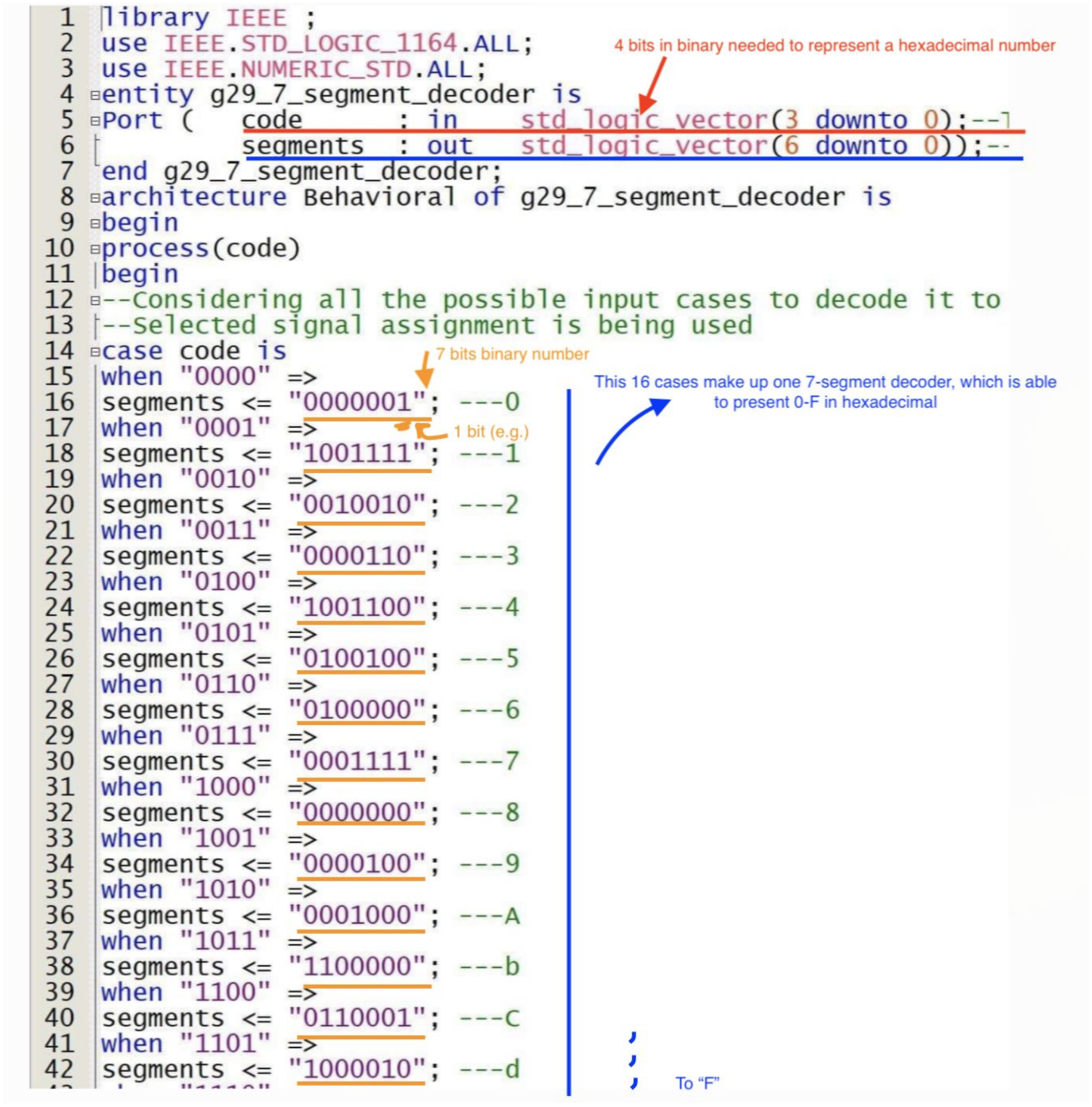


Flow Summary

<<Filter>>

Flow Status	Successful - Mon...18 11:35:39 2019
Quartus Prime Version	18.1.0 Build 625 0...18 SJ Lite Edition
Revision Name	g29_adder
Top-level Entity Name	g29_adder
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	16 / 32,070 (< 1 %)
Total registers	0
Total pins	52 / 457 (11 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Fig.4.2 This figure shows the compilation report's flow summary of the adder circuit



```vhdl
library IEEE ;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity g29_7_segment_decoder is
Port (    code        : in     std_logic_vector(3 downto 0);--1
          segments   : out    std_logic_vector(6 downto 0));--
end g29_7_segment_decoder;
architecture Behavioral of g29_7_segment_decoder is
begin
process(code)
begin
--Considering all the possible input cases to decode it to
--Selected signal assignment is being used
case code is
when "0000" =>
segments <= "0000001"; ---0
when "0001" =>
segments <= "1001111"; ---1
when "0010" =>
segments <= "0010010"; ---2
when "0011" =>
segments <= "0000110"; ---3
when "0100" =>
segments <= "1001100"; ---4
when "0101" =>
segments <= "0100100"; ---5
when "0110" =>
segments <= "0100000"; ---6
when "0111" =>
segments <= "0001111"; ---7
when "1000" =>
segments <= "0000000"; ---8
when "1001" =>
segments <= "0000100"; ---9
when "1010" =>
segments <= "0001000"; ---A
when "1011" =>
segments <= "1100000"; ---b
when "1100" =>
segments <= "0110001"; ---C
when "1101" =>
segments <= "1000010"; ---d
```

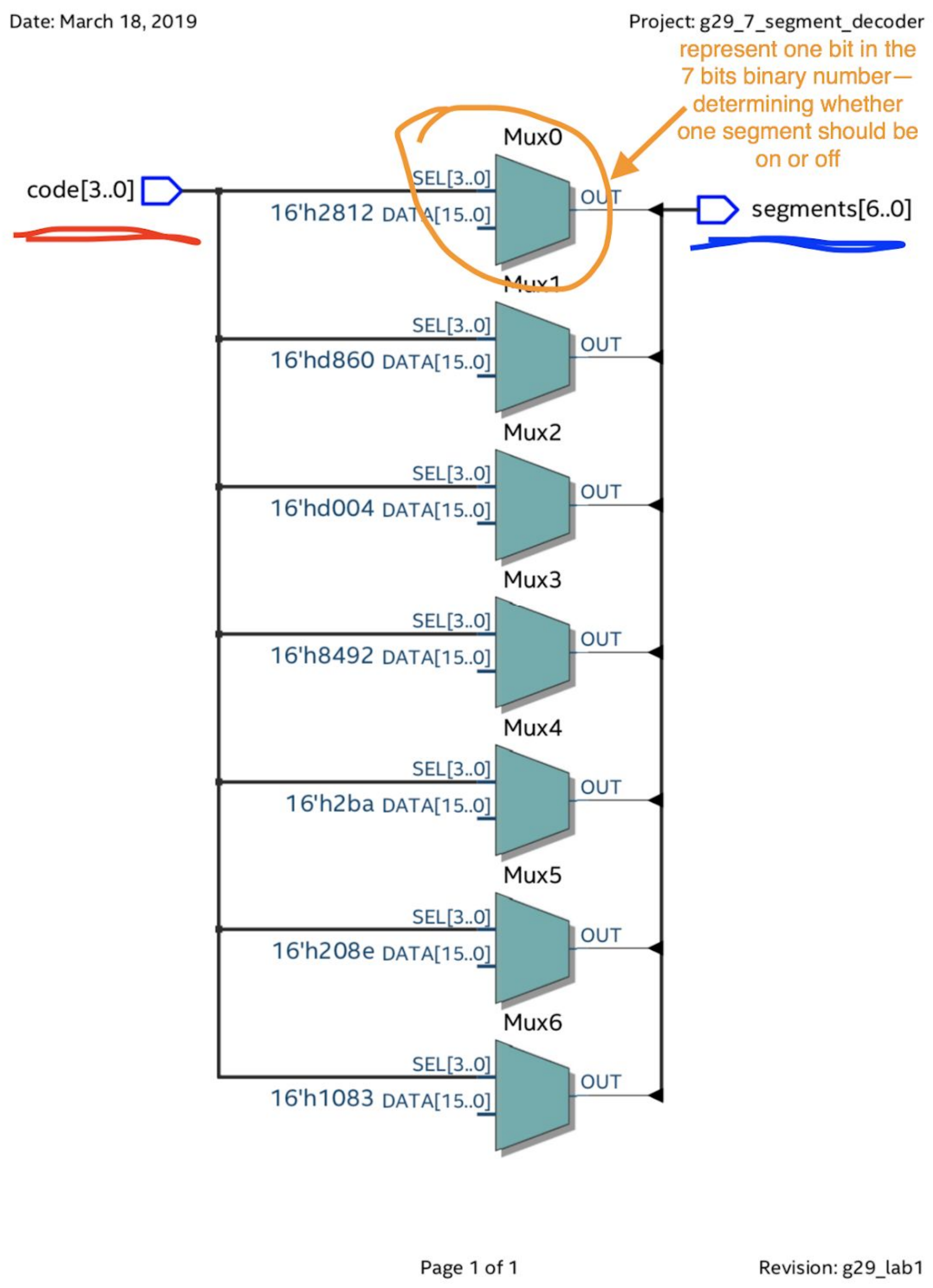
4 bits in binary needed to represent a hexadecimal number

7 bits binary number

1 bit (e.g.)

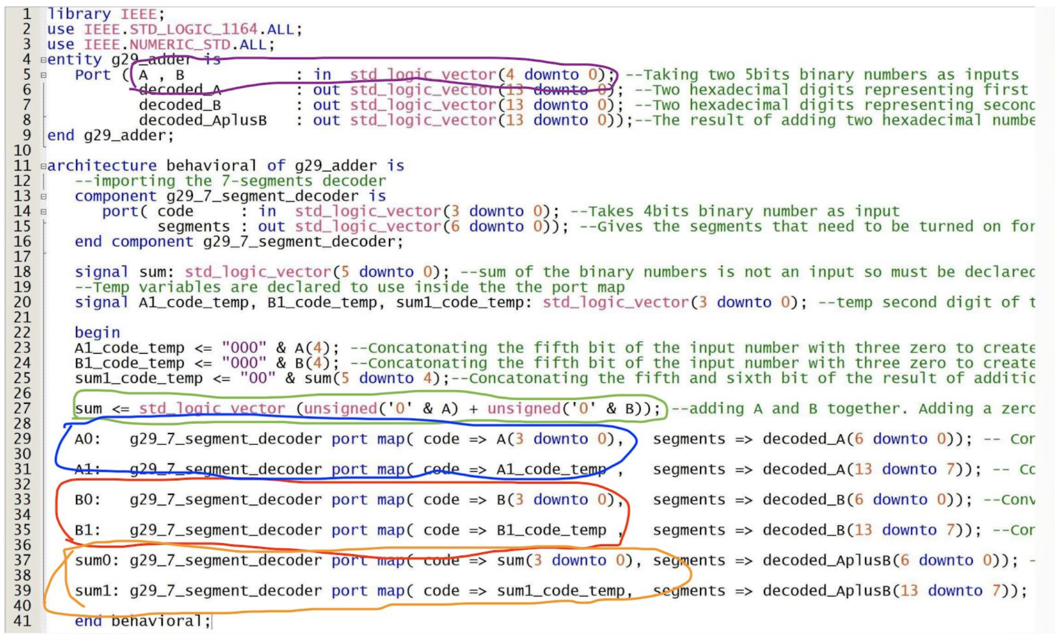
This 16 cases make up one 7-segment decoder, which is able to present 0-F in hexadecimal

To "F"



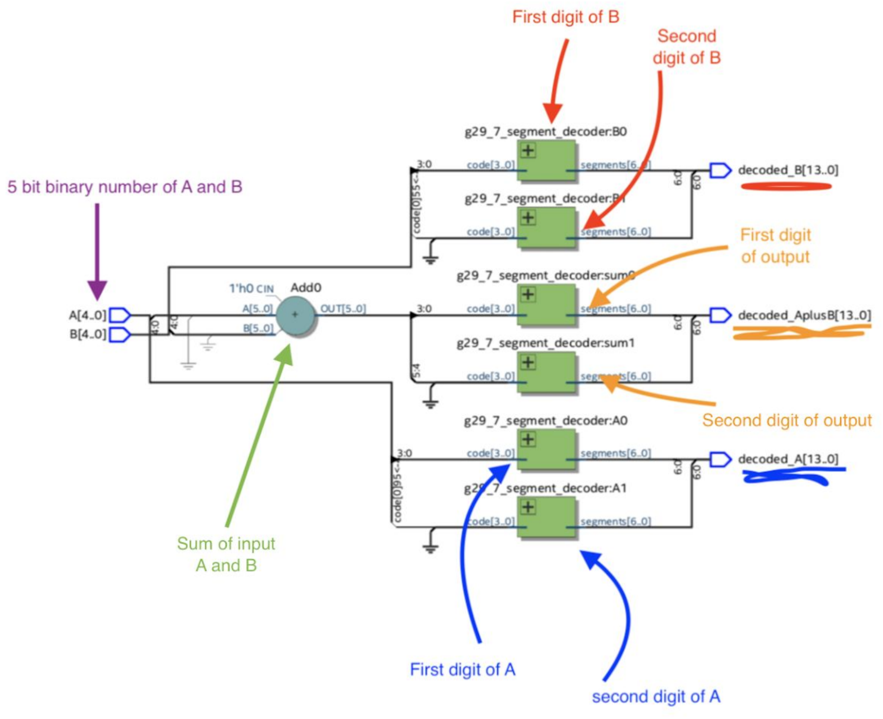
<u>Fig.4.3 These figures show the RTL schematic diagram and the corresponding code of the 7-segment decoder circuit</u>

Same colours indicate corresponding parts in the code and the schematic diagram.



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity g29_adder is
   Port ( A , B              : in  std_logic_vector(4 downto 0); --Taking two 5bits binary numbers as inputs
          decoded_A          : out std_logic_vector(13 downto 0); --Two hexadecimal digits representing first
          decoded_B          : out std_logic_vector(13 downto 0); --Two hexadecimal digits representing seconc
          decoded_AplusB     : out std_logic_vector(13 downto 0));--The result of adding two hexadecimal numbe
end g29_adder;

architecture behavioral of g29_adder is
   --importing the 7-segments decoder
   component g29_7_segment_decoder is
      port( code      : in  std_logic_vector(3 downto 0); --Takes 4bits binary number as input
            segments : out std_logic_vector(6 downto 0)); --Gives the segments that need to be turned on for
   end component g29_7_segment_decoder;

   signal sum: std_logic_vector(5 downto 0); --sum of the binary numbers is not an input so must be declarec
   --Temp variables are declared to use inside the the port map
   signal A1_code_temp, B1_code_temp, sum1_code_temp: std_logic_vector(3 downto 0); --temp second digit of t

   begin
   A1_code_temp <= "000" & A(4); --Concatonating the fifth bit of the input number with three zero to create
   B1_code_temp <= "000" & B(4); --Concatonating the fifth bit of the input number with three zero to create
   sum1_code_temp <= "00" & sum(5 downto 4);--Concatonating the fifth and sixth bit of the result of additic

   sum <= std_logic_vector (unsigned('0' & A) + unsigned('0' & B)); --adding A and B together. Adding a zerc

   A0:   g29_7_segment_decoder port map( code => A(3 downto 0),    segments => decoded_A(6 downto 0)); -- Cor

   A1:   g29_7_segment_decoder port map( code => A1_code_temp ,    segments => decoded_A(13 downto 7)); -- Cc

   B0:   g29_7_segment_decoder port map( code => B(3 downto 0),    segments => decoded_B(6 downto 0)); --Conv

   B1:   g29_7_segment_decoder port map( code => B1_code_temp ,    segments => decoded_B(13 downto 7)); --Cor

   sum0: g29_7_segment_decoder port map( code => sum(3 downto 0), segments => decoded_AplusB(6 downto 0)); -

   sum1: g29_7_segment_decoder port map( code => sum1_code_temp,  segments => decoded_AplusB(13 downto 7));

   end behavioral;
```

Date: March 18, 2019 Project: g29_adder



Page 1 of 1 Revision: g29_adder

Fig.4.4 These figures shows the RTL schematic diagram and the corresponding code of the adder circuit

Same colours indicate corresponding parts in the code and the schematic diagram.