# VHDL ASSIGNMENT 4

## 1- Names

Loïc Duchesne - 261052490
Yassine Mimet - 260980175

## 2- Executive summary

For this lab, we wrote VHDL code for a 7-Segment LED Decoder ( transform a BCD digit ), and then we tested it on an Altera DE1-SoC board. We also used Quartus's timing analysis to find the circuit's critical path. We found it by using Quartus violating path in the compilation summary and then using the Timing Analyzer to find more information about the critical path.

## 3- Questions

*1- VHDL explanation*

BCD adder

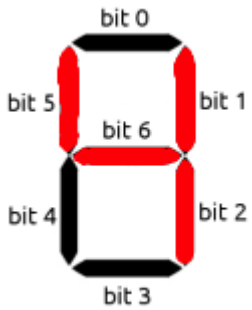
```
1  -- Structural one-digit BCD adder
2  -- Authors: Loic Duchesne & Yassine Mimet
3
4  library IEEE;
5  use IEEE.STD_LOGIC_1164.ALL;
6  use IEEE.NUMERIC_STD.ALL;
7
8  library work;
9
10 entity bcd_adder_structural is
11     port (A: in std_logic_vector(3 downto 0);
12           B: in std_logic_vector(3 downto 0);
13           S: out std_logic_vector(3 downto 0);
14           C: out std_logic);
15 end bcd_adder_structural;
16
17 architecture struct of bcd_adder_structural is
18     signal rca0_output, rca1_output : std_logic_vector(4 downto 0);
19     signal rca1_A_input, rca1_B_input : std_logic_vector(3 downto 0);
20     signal output_carry : std_logic;
21 begin
22     rca0: entity work.rca_structural port map(A, B, rca0_output);
23
24     rca1_A_input <= rca0_output(3) & rca0_output(2) & rca0_output(1) & rca0_output(0);
25
26     output_carry <= rca0_output(4) OR (rca0_output(2) AND rca0_output(3)) OR (rca0_output(1) AND rca0_output(3));
27
28     rca1_B_input <= '0' & output_carry & output_carry & '0';
29
30     rca1: entity work.rca_structural port map(rca1_A_input, rca1_B_input, rca1_output);
31
32     S <= rca1_output(3) & rca1_output(2) & rca1_output(1) & rca1_output(0);
33     C <= output_carry;
34
35 end struct;
```

We used the structural, we tried to implement it using two RCA's and a subpart that contains an OR gate and 2 AND gates. The subpart will take the fifth digit of the output (rc0_output(5)) of the first RCA (rca0), and will take it as an input for the OR gate. The two other inputs are just the outputs of the two AND gates. The first AND gate will take as input the second and the fourth first digits of the first RCA output, and the second AND gate will take as input the third and the fourth digits of the first RCA output. Then, the output of this subpart will be the output carry of the BCD adder. Finally, we get the second output S from the 2nd RCA. It will be only the 4 first digits of that RCA. That RCA takes two inputs, the first one being the first 4 digits of the 1st RCA's output. For the second input, we basically concatenated '0' & 'output_carry' & 'output_carry' & '0', so it will be either '0000' when the output carry is 0 or it will be '0110' if the output carry is 1.

7-Segment Decoder

```
1  -- Seven segment decoder
2
3  library IEEE;
4  use IEEE.STD_LOGIC_1164.ALL;
5  use IEEE.NUMERIC_STD.ALL;
6
7  library work;
8
9  entity seven_segment_decoder is
10    port (code: in std_logic_vector(3 downto 0);
11          segments_out: out std_logic_vector(6 downto 0));
12 end seven_segment_decoder;
13
14 architecture decoder of seven_segment_decoder is
15 begin
16  with code select
17     segments_out <=
18     "1000000" when "0000", -- 0
19     "1111001" when "0001", -- 1
20     "0100100" when "0010", -- 2
21     "0110000" when "0011", -- 3
22     "0011001" when "0100", -- 4
23     "0010010" when "0101", -- 5
24     "0000010" when "0110", -- 6
25     "1111000" when "0111", -- 7
26     "0000000" when "1000", -- 8
27     "0010000" when "1001", -- 9
28     "1111111" when others;
29 end decoder;
```

We used the code in the lab manual. This behavioral implementation of the 7-segment decoder transforms a BCD digit into a 7-bit number, each bit describing a segment. You can see that our code only cares about numbers from 0 to 9 (BCD digits). So basically, we use the select method to assign the value of the output. The 1 in the output means that the segment is off. Let's take '4' as an example; the BCD digit of 4 is 0011001, so we only turn off bit 0, bit 3, and bit 4.



Wrapper Design

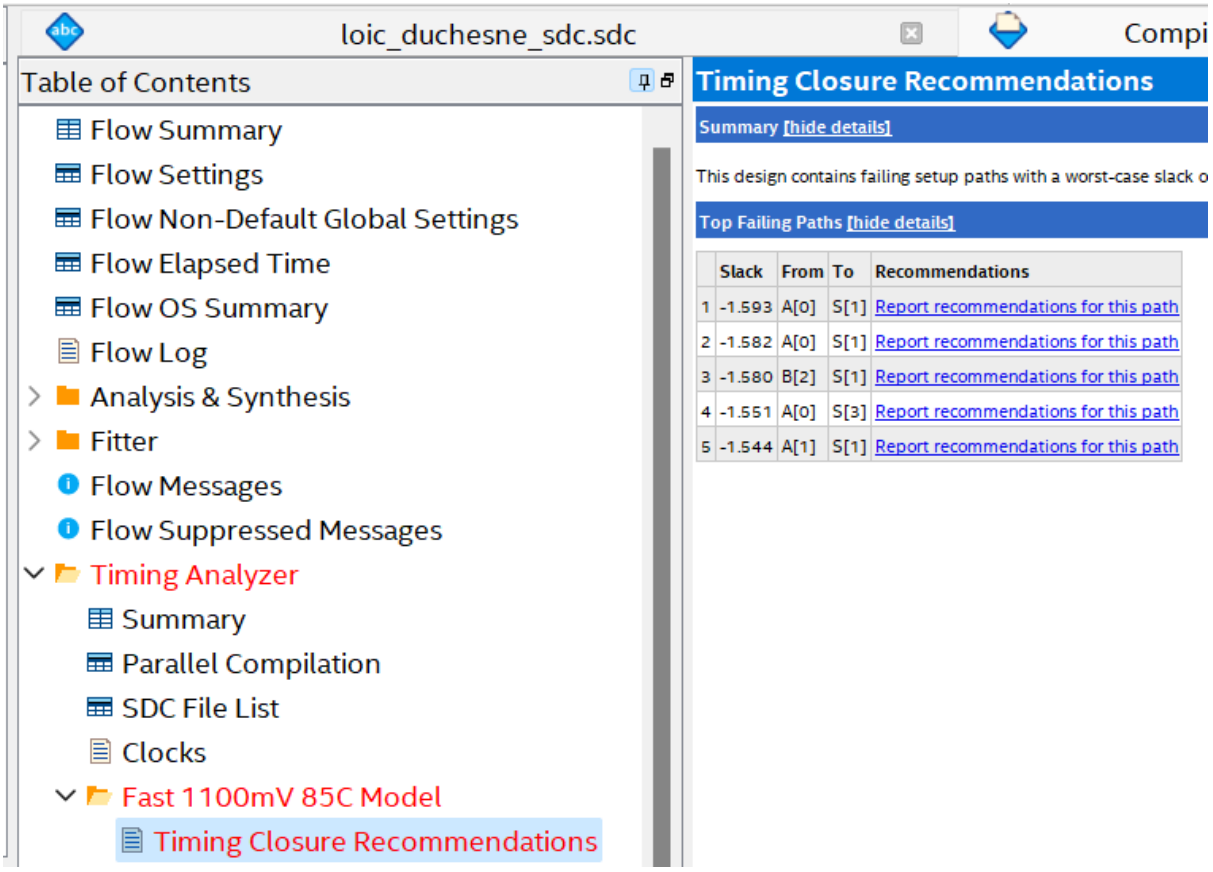
loic_duchesne_wrapper.vhd

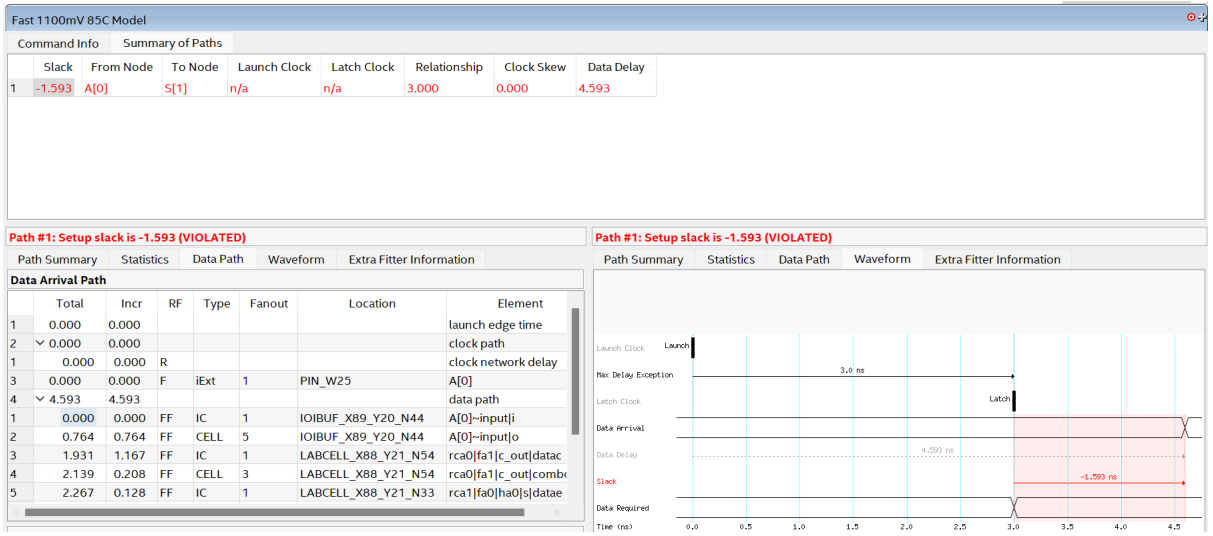
```vhdl
-- Wrapper circuit for a seven segment decoder
-- Authors: Loic Duchesne & Yassine Mimet

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

library work;

entity loic_duchesne_wrapper is
    port (A, B: in std_logic_vector(3 downto 0);
          decoded_A: out std_logic_vector(6 downto 0);
          decoded_B: out std_logic_vector(6 downto 0);
          decoded_AplusB: out std_logic_vector(13 downto 0));
end loic_duchesne_wrapper;

architecture struct of loic_duchesne_wrapper is
    signal A_in, B_in : std_logic_vector(3 downto 0);
    signal bcd_s_out : std_logic_vector(3 downto 0);
    signal c_out : std_logic;
    signal bcd_c : std_logic_vector(3 downto 0);
    signal bcd_decoded_s, bcd_decoded_c : std_logic_vector(6 downto 0);

begin
    -- A & B seven segment display
    -- *Note: A & B decimal value is capped at 9.

    seven_segment_decoder0: entity work.seven_segment_decoder port map (A_in, decoded_A);
    seven_segment_decoder1: entity work.seven_segment_decoder port map (B_in, decoded_B);

    -- AplusB seven segment display
    bcd_adder_main : entity work.bcd_adder_structural port map (A, B, bcd_s_out, c_out);

    bcd_c <= '0' & '0' & '0' & c_out;

    seven_segment_decoder2: entity work.seven_segment_decoder port map (bcd_s_out, bcd_decoded_s);
    seven_segment_decoder3: entity work.seven_segment_decoder port map (bcd_c, bcd_decoded_c);

    decoded_AplusB <= bcd_decoded_c & bcd_decoded_s;

end struct;
```

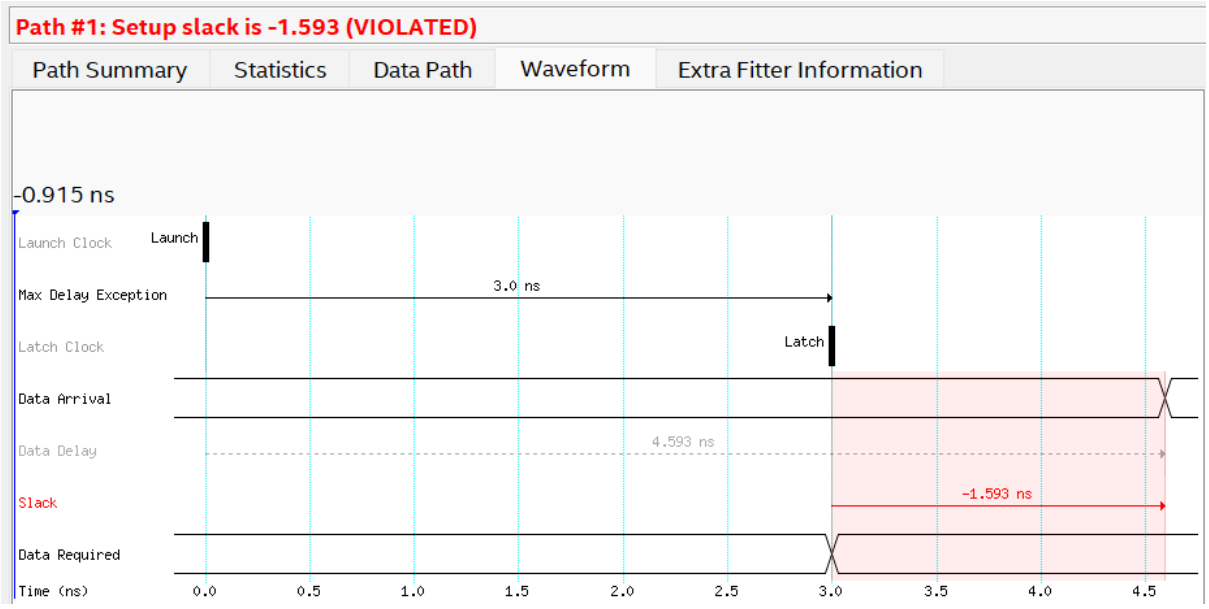
We first defined the entity as shown in the lab report. In the component part, we defined four signals (one 4-bit, one 1-bit,  and two 7-bit) that we will use. We first used the 7-segment decoder to get the decoded value of A and B. To get the decoded value of A+B, we had first to perform a BCD addition to add A and B, which will give us a sum and a carry. We used the 7-segment decoder again to get the decoded value of the sum that will be the seven less significant digits in our 7-digit output for the other seven digits; it depended on the carry value. If the carry is 1, then we need to perform the transformation on 0001; if the carry is 0, then we need to perform the transformation on 0000; that's why we had to concatenate three zeros and c with C being out LSB, and then perform the transformation. In the end, we had to concatenate both 7-bit signals to have an output of 14 bits.

*2- Timing analysis*

```
1  set_max_delay -from [get_ports A[*]] -to [get_ports S[*]] 3
2  set_max_delay -from [get_ports B[*]] -to [get_ports S[*]] 3
3  set_max_delay -from [get_ports A[*]] -to [get_ports C[*]] 3
4  set_max_delay -from [get_ports B[*]] -to [get_ports C[*]] 3
5
6  set_operating_conditions -model fast -temperature 85 -voltage 1100
```



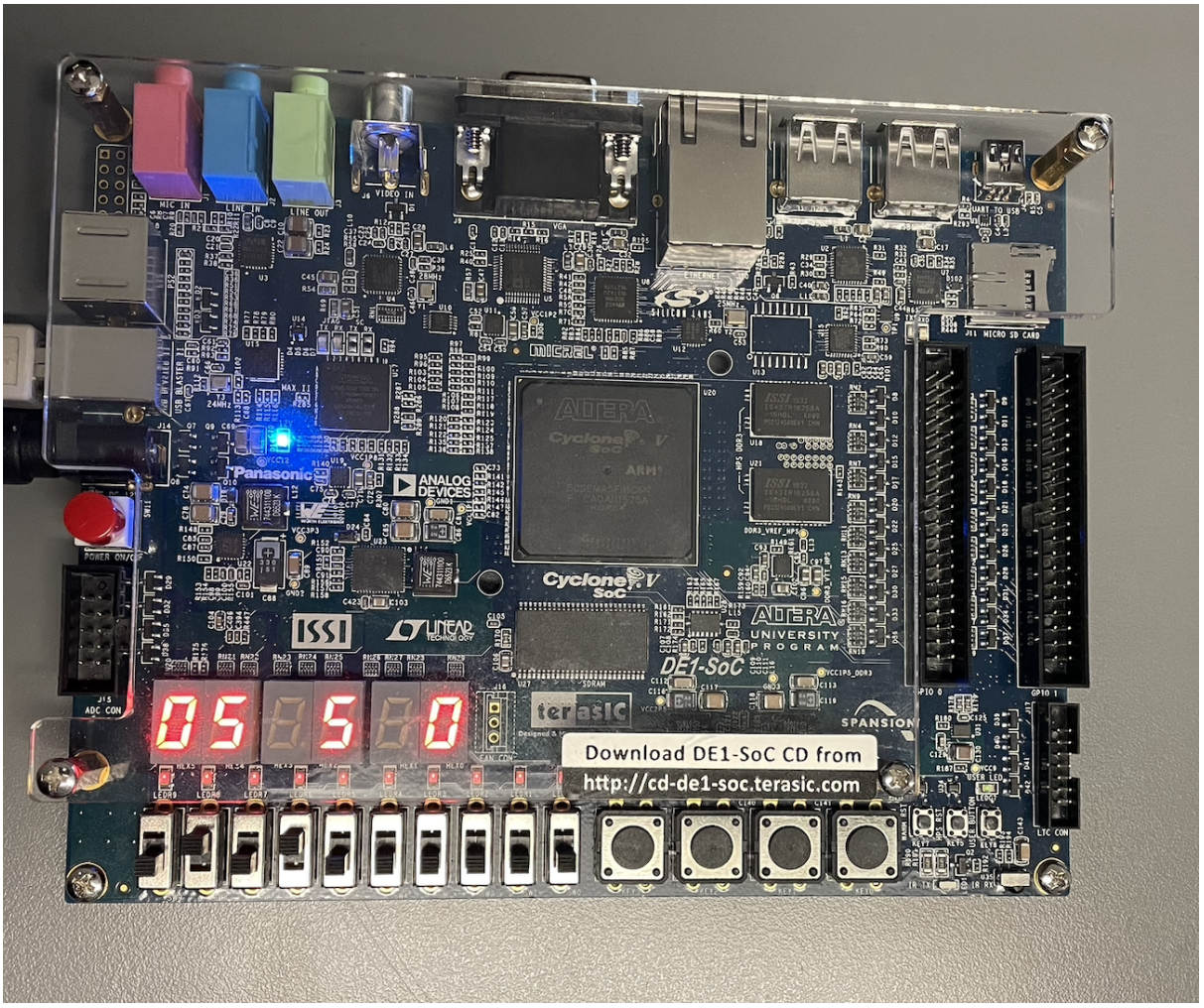



*3- VHDL code for the wrapper circuit*

loic_duchesne_wrapper.vhd

```vhdl
-- Wrapper circuit for a seven segment decoder
-- Authors: Loic Duchesne & Yassine Mimet

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

library work;

entity loic_duchesne_wrapper is
    port (A, B: in std_logic_vector(3 downto 0);
          decoded_A: out std_logic_vector(6 downto 0);
          decoded_B: out std_logic_vector(6 downto 0);
          decoded_AplusB: out std_logic_vector(13 downto 0));
end loic_duchesne_wrapper;

architecture struct of loic_duchesne_wrapper is
    signal A_in, B_in : std_logic_vector(3 downto 0);
    signal bcd_s_out : std_logic_vector(3 downto 0);
    signal c_out : std_logic;
    signal bcd_c : std_logic_vector(3 downto 0);
    signal bcd_decoded_s, bcd_decoded_c : std_logic_vector(6 downto 0);

begin
    -- A & B seven segment display
    -- *Note: A & B decimal value is capped at 9.

    seven_segment_decoder0: entity work.seven_segment_decoder port map (A_in, decoded_A);
    seven_segment_decoder1: entity work.seven_segment_decoder port map (B_in, decoded_B);

    -- AplusB seven segment display
    bcd_adder_main : entity work.bcd_adder_structural port map (A, B, bcd_s_out, c_out);

    bcd_c <= '0' & '0' & '0' & c_out;

    seven_segment_decoder2: entity work.seven_segment_decoder port map (bcd_s_out, bcd_decoded_s);
    seven_segment_decoder3: entity work.seven_segment_decoder port map (bcd_c, bcd_decoded_c);

    decoded_AplusB <= bcd_decoded_c & bcd_decoded_s;

end struct;
```

*4- Altera Board photo*

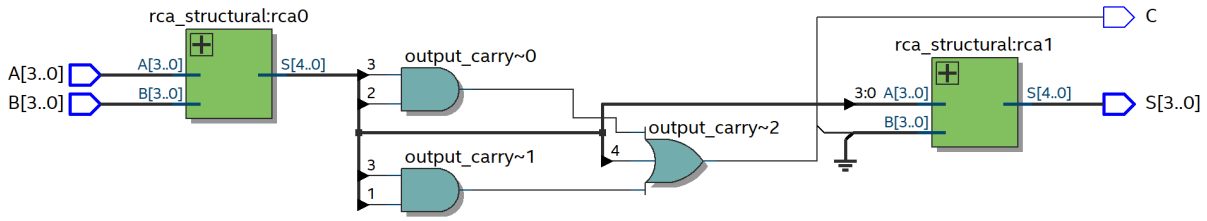



*5- Number of pins & logic modules*

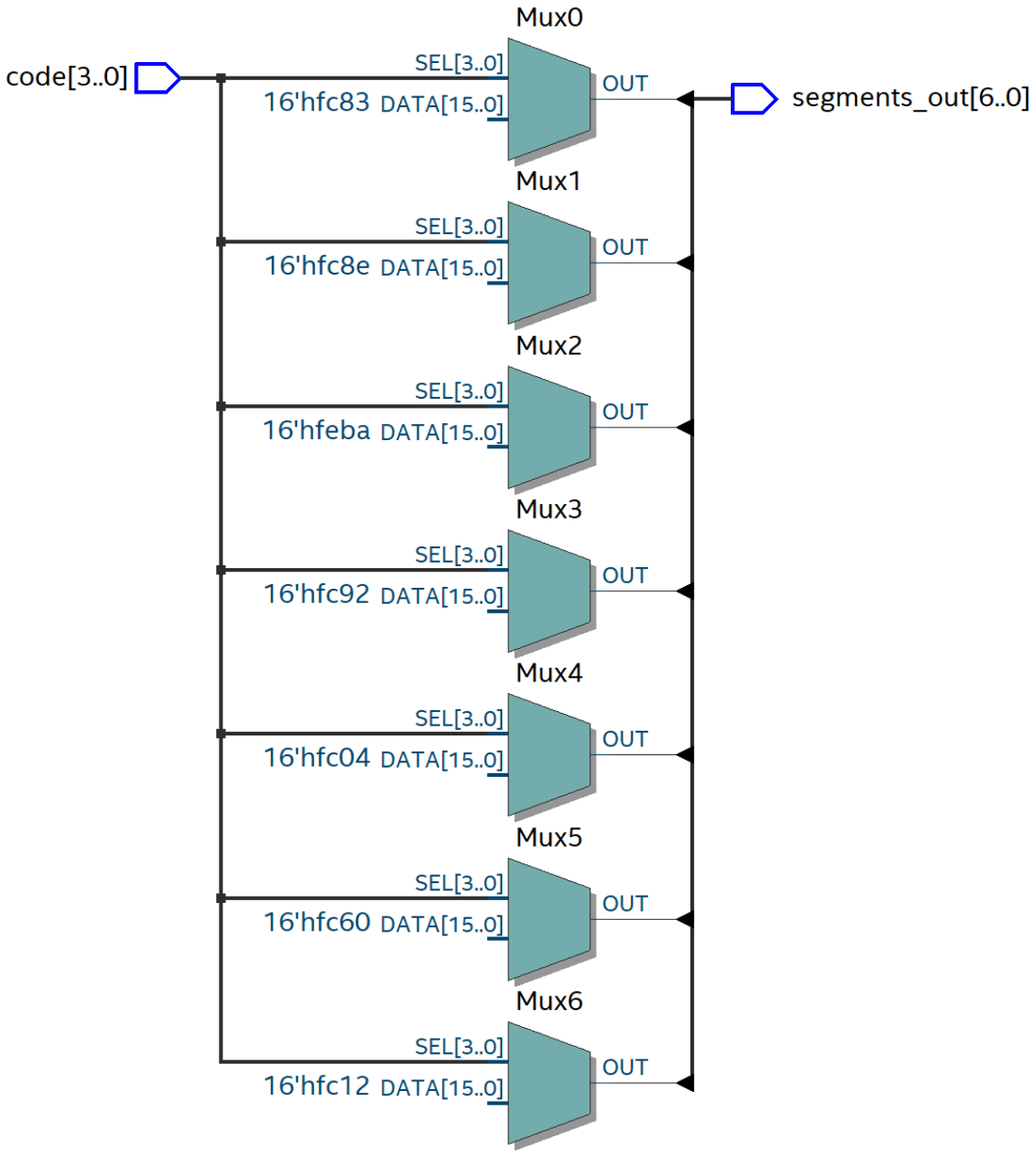
| | |
|---|---|
| Logic utilization (in ALMs) | 10 / 32,070 ( < 1 % ) |
| Total registers | 0 |
| Total pins | 36 / 457 ( 8 % ) |

**4- Additional Pictures**

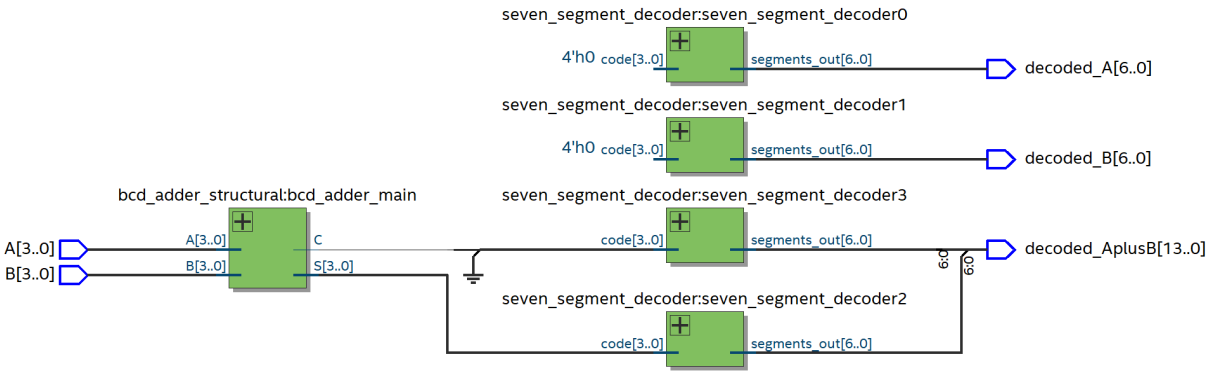
***BCD Adder structural RTL view:***




***7-Segment RTL view:***



***Wrapper RTL view:***




## 5- Explanation of the results

For the timing analysis, we had to run a timing analysis on the BCD Adder we wrote last lab. To do that, we wrote an SDC file that established the different timing constraints. In that file, we also had to specify the model on which the timing analysis would display the timing closure recommendations. Afterwards, when we compile and get those recommendations, we get the results of the top 5 paths that have the worst timing for our circuit (the first timing analysis picture), with the top path being our critical path. We then opened that path in the Timing analyzer to get more information and its waveform as shown in the above pictures.

For the wrapper circuit, we had to first assign the outputs to the correct pin using the pin assignment tool provided in Quartus. This was done by consulting the FPGA's manual to find the individual pin codes. In the pictures, you can see that the switch assigned for B[2] and B[0] are on, giving B a value of 5. For A, all the switches are off which gives A a value of 0. This will output for AplusB 5 which is correctly shown in those pictures.

## 6- Conclusion

In conclusion, this lab allowed us to get an idea of how the timing analysis works that we'll be using in the following labs; during our timing analysis, we encountered a problem, the timing closure was only in the slow model, so we had to write that last line in the sdc to force the timing closure to appear in the fast model. We also got an idea of using the Altera DE1-SoC board and simulating our code in the board, which was working as a testbench in this case. We also had to use different switches because one switch wasn't working correctly, so we started from the left instead of the right.