VHDL LAB #4

Rachel Ruddy - 261162987
Natasha Lawford - 261178596
Rachel Bunsick - 261159677

I. Executive Summary:

In this lab, we first created a 2 digit BCD adder and implemented it on the Altera DE1-S FPGA board. We used a 7-segment decoder .vhd file to display each BCD digit on the 7-segment display. We also used our BCD adder .vhd file to handle the addition of the two BCD digits. Finally, we wrote a wrapper .vhd file which instantiated the decoder and adder. Then, we used the pin mapper to connect our output signals to the board (the 7-segment display and the switches to control our BCD digits) which were stored in the .sof file.

In the second part of the lab, we created a BCD comparator which compares two BCD values: A and B + 1. We started by writing the .vhd file for the comparator, which determines if A > B+1, A >= B+1, A < B+1, A <= B+1, and A = B+1. If B+1 requires more than 4 bits, then we set an LED to light up indicating that overflow has occurred. We also created a bit splitter .vhd file which takes in a 4-bit number in binary and splits it into a sum and carry. This is used in our wrapper circuit for the comparator so that we are able to accurately display A and B, since they may take on values from 0-15 (may need 2 BCD spots on the FPGA). Then, we added the .sof file where we kept the mapping of the switches, but remapped A and B to both have 2 BCD digit spots and also added signal maps to the LEDs which represent the output of the comparator.

II. Questions:

1. Briefly explain your VHDL code implementation of all circuits with their respective screenshots.

Seven Segment Decoder Code:

seven_segment_decoder.vhd | rachel_bunsick_wrapper.vhd

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

The seven segment decoder code was provided to us in the assignment. The code takes in a BCD number and maps each of the possible outputs for each BCD digit (0-9) and turns on the appropriate LED on the 7-segment display so that we can see a visual representation of the number. When any other binary number (10-15) is received as input, the 7-segment display is shut off. It is represented by ‘1111111’ because the BCD display uses an active low signal so when 1 is passed, the LED is off.

BCD Adder Code:

seven_segment_decoder.vhd | rachel_bunsick_wrapper.vhd

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.NUMERIC_STD.ALL;
4
5   entity bcd_adder is
6       port ( A : in std_logic_vector (3 downto 0);
7              B : in std_logic_vector (3 downto 0);
8              S : out std_logic_vector (3 downto 0);
9              C : out std_logic);
10  end bcd_adder;
11
12  architecture behavioral of bcd_adder is
13      signal temp_sum : unsigned (4 downto 0);
14      signal adjusted_sum : unsigned (4 downto 0);
15
16  begin
17      process(A, B)
18      variable sum : unsigned(4 downto 0);
19      variable A_unsigned : unsigned(3 downto 0);
20      variable B_unsigned : unsigned(3 downto 0);
21      variable S_unsigned : unsigned(3 downto 0);
22      begin
23          A_unsigned := unsigned(A);
24          B_unsigned := unsigned(B);
25          S_unsigned := "0000";
26          sum := ('0' & A_unsigned) + ('0' & B_unsigned);
27          if (sum > 9) then
28              S_unsigned := resize((sum + "00110"),4);
29              C <= '1';
30          else
31              S_unsigned := sum(3 downto 0);
32              C <= '0';
33          end if;
34          S <= std_logic_vector(S_unsigned);
35
36      end process;
37  end behavioral;
```

The BCD adder module is used to add our 2 BCD digits: A and B. If the sum of A and B is less than 9, we set the sum to be the sum we calculated directly and set the carry BCD digit to be 0. If the sum is greater than 9, we set the sum to be our calculated sum plus 6, which properly represents the sum of A and B after carrying 1 to the 10s place. We also set the carry to be 1 in this case.

Comparator Code:

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.NUMERIC_STD.ALL;
4
5   entity comparator is
6      port(
7         A,B : in std_logic_vector (3 downto 0);
8         AgtBPlusOne,AgteBPlusOne,AltBPlusOne,AlteBPlusOne,AeqBPlusOne,overflow : out std_logic);
9   end comparator;
10
11  architecture behavioral of comparator is
12  begin
13     process(A,B)
14     variable A_unsigned : unsigned(4 downto 0);
15     variable B_unsigned : unsigned(3 downto 0);
16     variable BPlusOne_unsigned : unsigned(4 downto 0);
17     begin
18        A_unsigned := '0' & unsigned(A);
19        B_unsigned := unsigned(B);
20        BPlusOne_unsigned := "00000";
21        BPlusOne_unsigned := resize(B_unsigned, 5) + 1;
22        if (BPlusOne_unsigned(4) = '1') then
23           overflow <= '1';
24           AgtBPlusOne <= '0';
25           AgteBPlusOne <= '0';
26           AltBPlusOne <= '0';
27           AlteBPlusOne <= '0';
28           AeqBPlusOne <= '0';
29        elsif (A_unsigned = BPlusOne_unsigned) then
30           AgtBPlusOne <= '0';
31           AgteBPlusOne <= '1';
32           AltBPlusOne <= '0';
33           AlteBPlusOne <= '1';
34           AeqBPlusOne <= '1';
35           overflow <= '0';
36        elsif (A_unsigned > BPlusOne_unsigned) then
37           AgtBPlusOne <= '1';
38           AgteBPlusOne <= '1';
39           AltBPlusOne <= '0';
40           AlteBPlusOne <= '0';
41           AeqBPlusOne <= '0';
42           overflow <= '0';
43        elsif (A_unsigned < BPlusOne_unsigned) then
44           AgtBPlusOne <= '0';
45           AgteBPlusOne <= '0';
46           AltBPlusOne <= '1';
47           AlteBPlusOne <= '1';
48           AeqBPlusOne <= '0';
49           overflow <= '0';
50        end if;
51     end process;
52  end behavioral;
53
```

The comparator takes in two 4-bit numbers A and B and compares A with B+1. The comparator module has 6 different outputs which correspond to different LEDs on the FPGA. When A = B+1, AgteBPlusOne, AlteBPlusOne, and AeqBPlusOne are set to 1 and all other outputs are set to 0. When A > B+1, AgteBPlusOne and AgtBPlusOne are set to 1 and all other outputs are set to 0. When A < B+1, AlteBPlusOne and AltBPlusOne are set to 1 and all other outputs are set to 0. Finally, when B+1 overflows (i.e. requires more than 4 bits), overflow is set to 1 while all other outputs are set to 0.

Bit-splitter

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.NUMERIC_STD.ALL;
4
5   entity bit_splitter is
6      port(
7         A : in std_logic_vector(3 downto 0);
8         A1 : out std_logic_vector(6 downto 0);
9         A0 : out std_logic_vector(6 downto 0));
10  end bit_splitter;
11
12  architecture splitter of bit_splitter is
13     signal A_unsigned : unsigned(3 downto 0);
14     signal A14b : unsigned(3 downto 0);
15     signal A04b : unsigned(3 downto 0);
16
17     component seven_segment_decoder is
18        port(
19           code : in std_logic_vector(3 downto 0);
20           segments_out : out std_logic_vector(6 downto 0));
21     end component;
22
23     begin
24     A_unsigned <= unsigned(A);
25        process(A_unsigned)
26        begin
27           if (A_unsigned > 9) then
28              A14b <= "0001";
29              A04b <= A_unsigned - "1010";
30           else
31              A14b <= "0001";
32              A04b <= A_unsigned;
33           end if;
34        end process;
35
36        i1: seven_segment_decoder port map (std_logic_vector(A14b), A1);
37        i2: seven_segment_decoder port map (std_logic_vector(A04b), A0);
38  end splitter;
```

The bit splitter takes a 4-bit BCD input A and separates it into two 7-segment displays. Inside the module, A is first converted to an unsigned type (A_unsigned) to enable easy comparison and arithmetic operations. A process block then checks if the value of A_unsigned exceeds 9. If it is greater than 9, then the BCD input represents a two-digit decimal number. In this case A14b is set to 0001 (representing 1 for the 10s place), and A04b is assigned A_unsigned-10, which gives the correct units value after adjusting for the overflow to the tens place. If A_unsigned is less than or equal to 9, the value is a single number so A14b is set to 0000 (no tens digit) and A04b is assigned the value of A_unsigned directly. Finally, A14b and A04b are mapped to the seven_segment_decoder components which convert them into 7-segment display signals A1 and A0 to show the tens units digit of A.

2. Provide the VHDL code that you wrote for the wrapper circuit.

seven_segment_decoder.vhd | rachel_bunsick_wrapper.vhd

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  entity rachel_bunsick_wrapper is
6  Port   (A, B : in std_logic_vector (3 downto 0);
7          decoded_A: out std_logic_vector(6 downto 0);
8          decoded_B: out std_logic_vector(6 downto 0);
9          decoded_AplusB: out std_logic_vector(13 downto 0));
10 end rachel_bunsick_wrapper;
11
12 architecture wrapper of rachel_bunsick_wrapper is
13
14 signal S, bcd_C: std_logic_vector (3 downto 0);
15 signal C: std_logic;
16 signal decoded_S, decoded_C: std_logic_vector(6 downto 0);
17
18 component seven_segment_decoder is
19    Port ( code: in std_logic_vector (3 downto 0);
20           segments_out: out std_logic_vector(6 downto 0));
21 end component;
22
23 component bcd_adder is
24    Port (  A : in std_logic_vector (3 downto 0);
25            B : in std_logic_vector (3 downto 0);
26            S : out std_logic_vector (3 downto 0);
27            C : out std_logic);
28 end component;
29
30 begin
31
32    i1: bcd_adder port map (A,B,S,C);
33    i2: seven_segment_decoder port map (A, decoded_A);
34    i3: seven_segment_decoder port map (B, decoded_B);
35    i4: seven_segment_decoder port map (S, decoded_S);
36    bcd_C <= "000" & C;
37    i5: seven_segment_decoder port map (bcd_C, decoded_C);
38
39    decoded_AplusB <= decoded_C & decoded_S;
40
41
42 end wrapper;
```

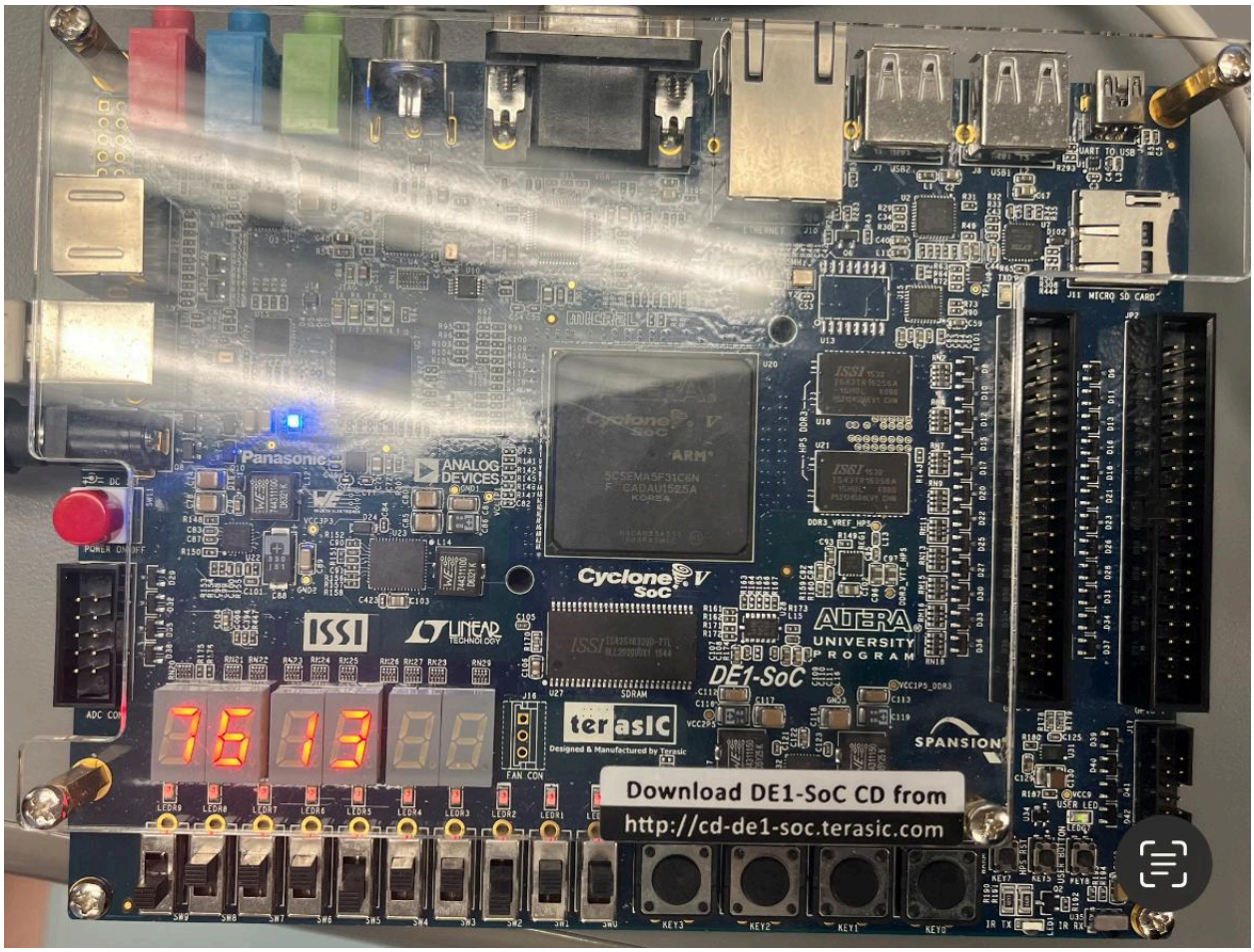
The wrapper module takes two 4-bit BCD inputs, A and B, adds them using a BCD adder, and then decodes each digit (including carry) for display on the 7-segment displays. Inputs A and B are added to produce S (sum) and C (carry). The module then decodes these values and outputs the seven-segment representation for A, B, S and C through decoded_A, decoded_B, decoded_S, and decoded_C, respectively. Additionally, it concatenates decoded_C and decoded_S into a decoded_AplusB to show the full addition result.

Wrapper for comparator:

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

entity wrapper is
    port(
        A,B : in std_logic_vector(3 downto 0);
        A1, A0, B1, B0 : out std_logic_vector(6 downto 0);
        AgtBPlusOne,AgteBPlusOne,AltBPlusOne,AlteBPlusOne,AeqBPlusOne,overflow : out std_logic);
end wrapper;

architecture wrap of wrapper is

component bit_splitter is
    port(
        A : in std_logic_vector(3 downto 0);
        A1 : out std_logic_vector(6 downto 0);
        A0 : out std_logic_vector(6 downto 0));
end component;

component comparator is
    port(
        A,B : in std_logic_vector (3 downto 0);
        AgtBPlusOne,AgteBPlusOne,AltBPlusOne,AlteBPlusOne,AeqBPlusOne,overflow : out std_logic);
end component;

begin
    i1: bit_splitter port map (A, A1, A0);
    i2: bit_splitter port map (B, B1, B0);
    i3: comparator port map (A,B, AgtBPlusOne,AgteBPlusOne,AltBPlusOne,AlteBPlusOne,AeqBPlusOne,overflow);

end wrap;
```

The wrapper for the comparator instantiates the bit splitter and comparator in order to display the results we would like to see. The instantiation of the bit splitter will split the binary representations of A and B into proper 2-digit BCD values which are displayed on the board. The instantiation of the comparator then uses the binary representations of A and B to run the comparison.

3. Show a representative photo of the board displaying the result of the addition of A and B, where A is the last (rightmost) digit of the McGill ID number of one of the students in your group and B is the last (rightmost) digit of the McGill ID number of the other student in the group (if there are three members in your group, choose two IDs and state which were chosen). Clearly indicate which 7-segment LEDs/sliding switches are assigned to which inputs/outputs of the circuit on the photo.



We set A = 7 and B = 6, where A is the last digit of both Rachel Ruddy and Rachel Bunsick's ID, and B is the last digit of Natasha Lawford's ID. The last two digits, "13" represent the sum of A and B, 13.

4. Report the number of pins and logic modules used to fit your designs on the FPGA board.

	2-digit BCD adder	Comparator
Logic Utilization (in ALMs)	16/32,070	26 / 32,070
Total pins	36/457	42 / 457

Testbench:

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

entity comparator_tb is
end comparator_tb;

architecture behavior of comparator_tb is


    component comparator
        port(
            A : in std_logic_vector (3 downto 0);
            B : in std_logic_vector (3 downto 0);
            AgtBPlusOne : out std_logic;
            AgteBPlusOne : out std_logic;
            AltBPlusOne : out std_logic;
            AlteBPlusOne : out std_logic;
            AeqBPlusOne : out std_logic;
            overflow : out std_logic
        );
    end component;

    -- Signal declarations to connect to the comparator
    signal A : std_logic_vector(3 downto 0) := "0101"; -- A = 5 in binary
    signal B : std_logic_vector(3 downto 0);
    signal AgtBPlusOne : std_logic;
    signal AgteBPlusOne : std_logic;
    signal AltBPlusOne : std_logic;
    signal AlteBPlusOne : std_logic;
    signal AeqBPlusOne : std_logic;
    signal overflow : std_logic;

begin

    |
```

```
37      uut: comparator
38         port map (
39            A => A,
40            B => B,
41            AgtBPlusOne => AgtBPlusOne,
42            AgteBPlusOne => AgteBPlusOne,
43            AltBPlusOne => AltBPlusOne,
44            AlteBPlusOne => AlteBPlusOne,
45            AeqBPlusOne => AeqBPlusOne,
46            overflow => overflow
47         );
48
49      -- Test process
50      process
51      begin
52         -- Iterate through all values of B from 0 to 15
53         for i in 0 to 15 loop
54            B <= std_logic_vector(to_unsigned(i, 4));
55            wait for 10 ns;
56         end loop;
57
58         wait; -- Stop the simulation
59      end process;
60
61   end behavior;
62
```

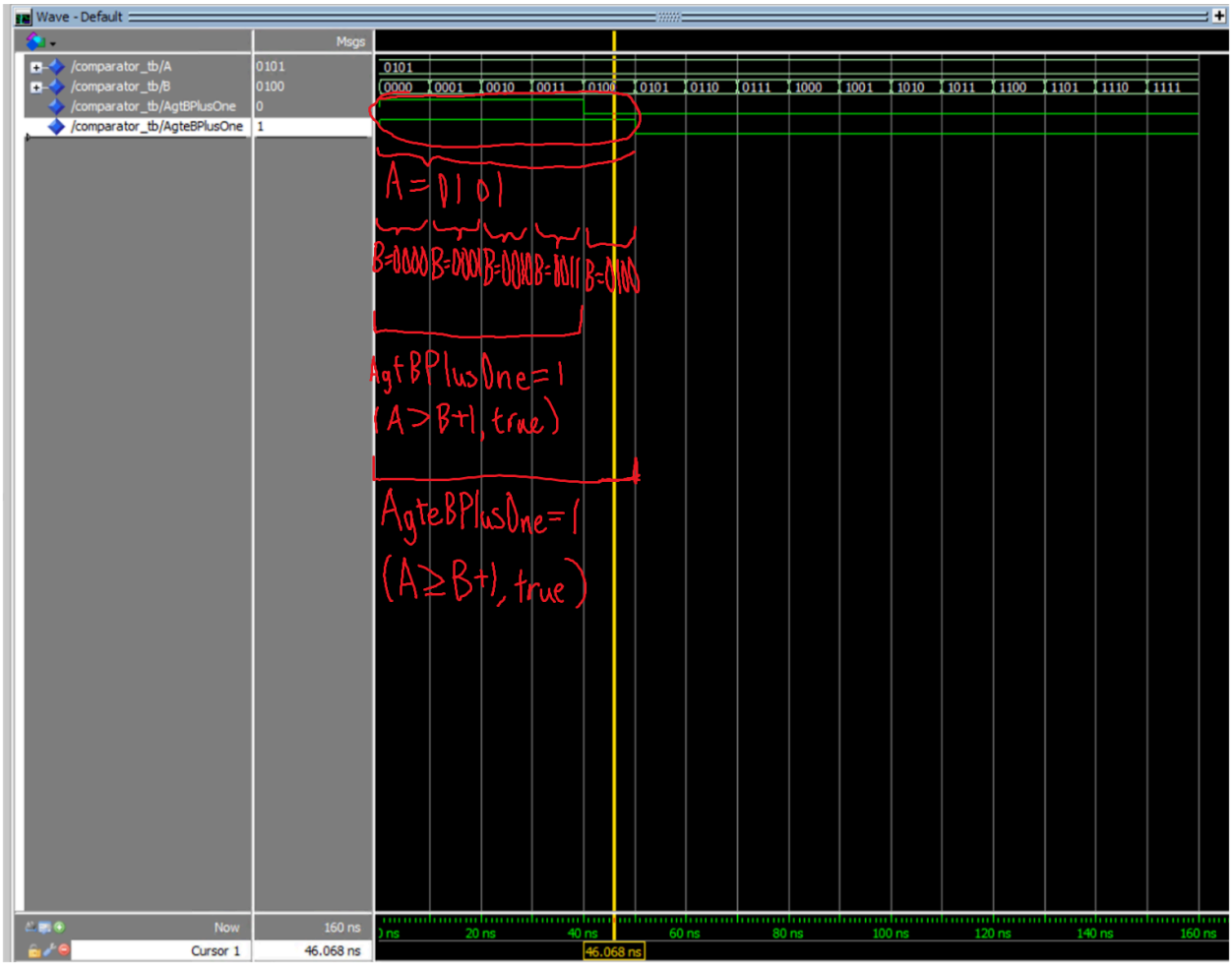
The testbench for the comparator iterates through all the possible cases for the 4-bit value of B. A is set to be a constant signal of 5 or “0101” in binary. We map A and B to the comparator and from this testbench are able to analyze all possible cases to ensure that it functions properly.

III. Schematics and simulation results (and explanations/labels of figures/important points on the simulation plots): we need to label the pictures!

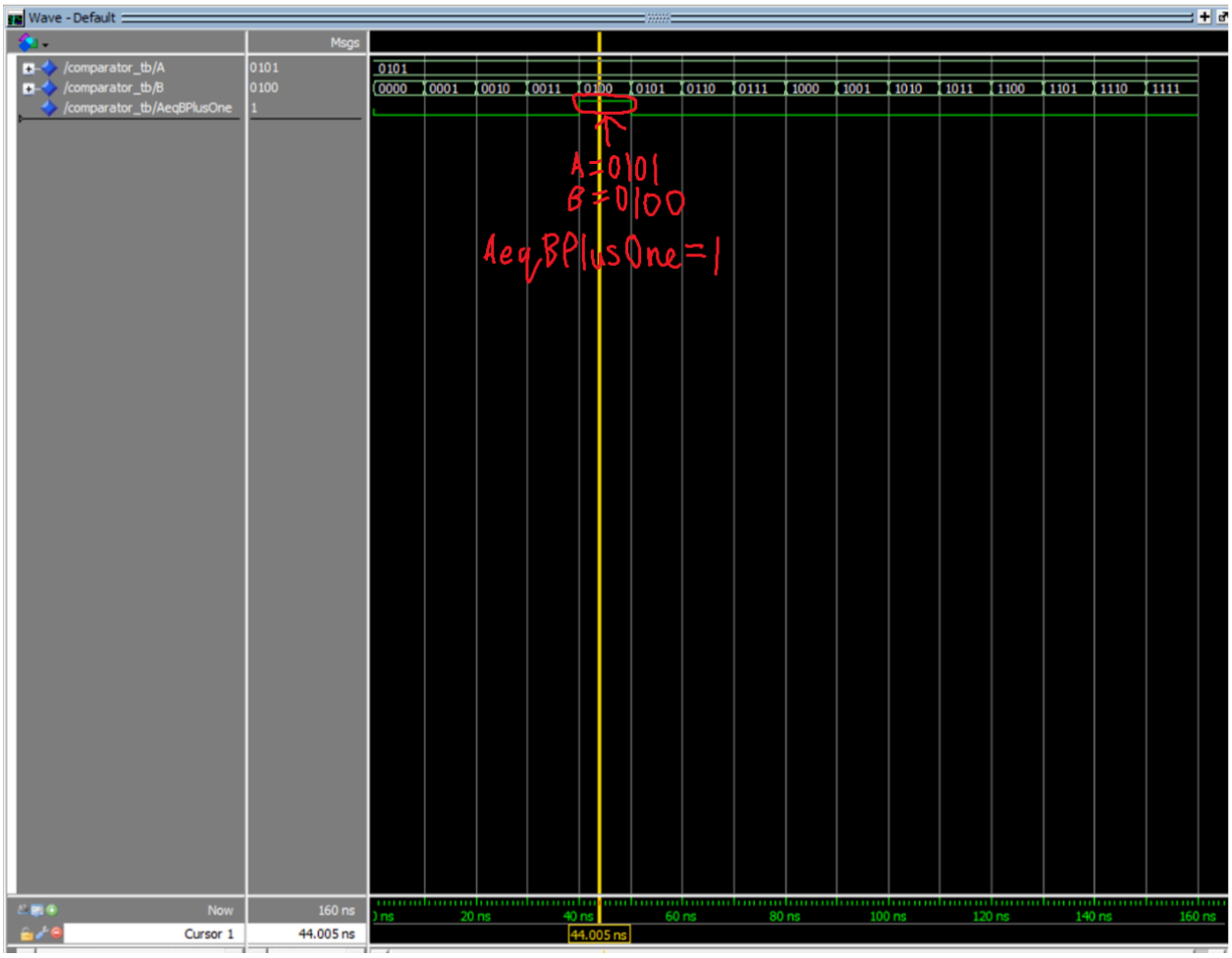
5. Given that A= 5_{10}, provide a separate simulation plot that demonstrates all possible cases for the 4-bit comparator, including a separate plot for the case where overflow occurs. A total of four plots should be included. Explain each plot and mark all inputs and outputs clearly.

Overflow: occurs only when B="1111", since B+1 is 5 bits:

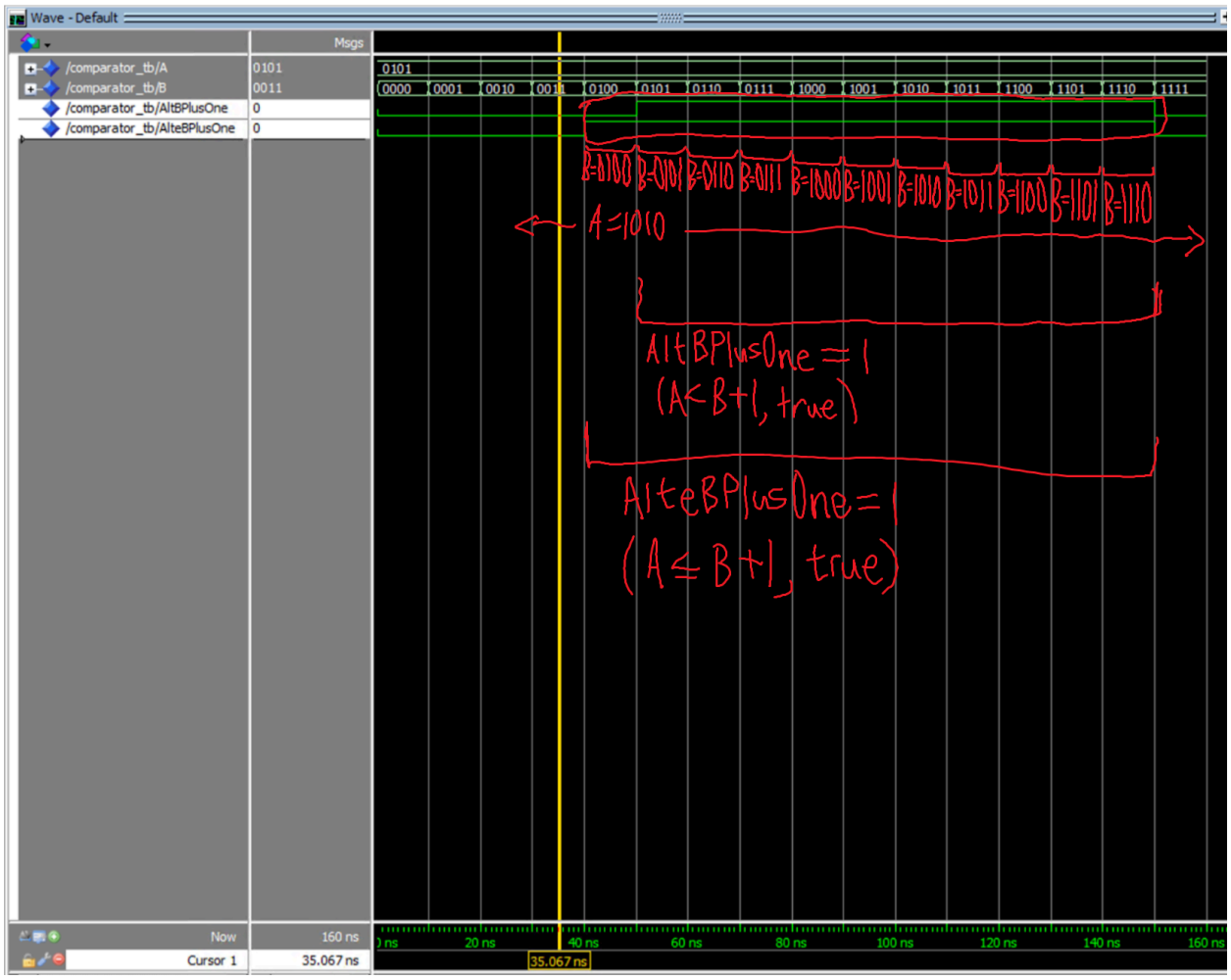
<u>A greater than/greater than or equal to B+1</u>: true only when B is 3 (for greater than, since 5>(3+1)) or when B is 4 for the case of greater than or equal to (since 5 <= (4+1)).



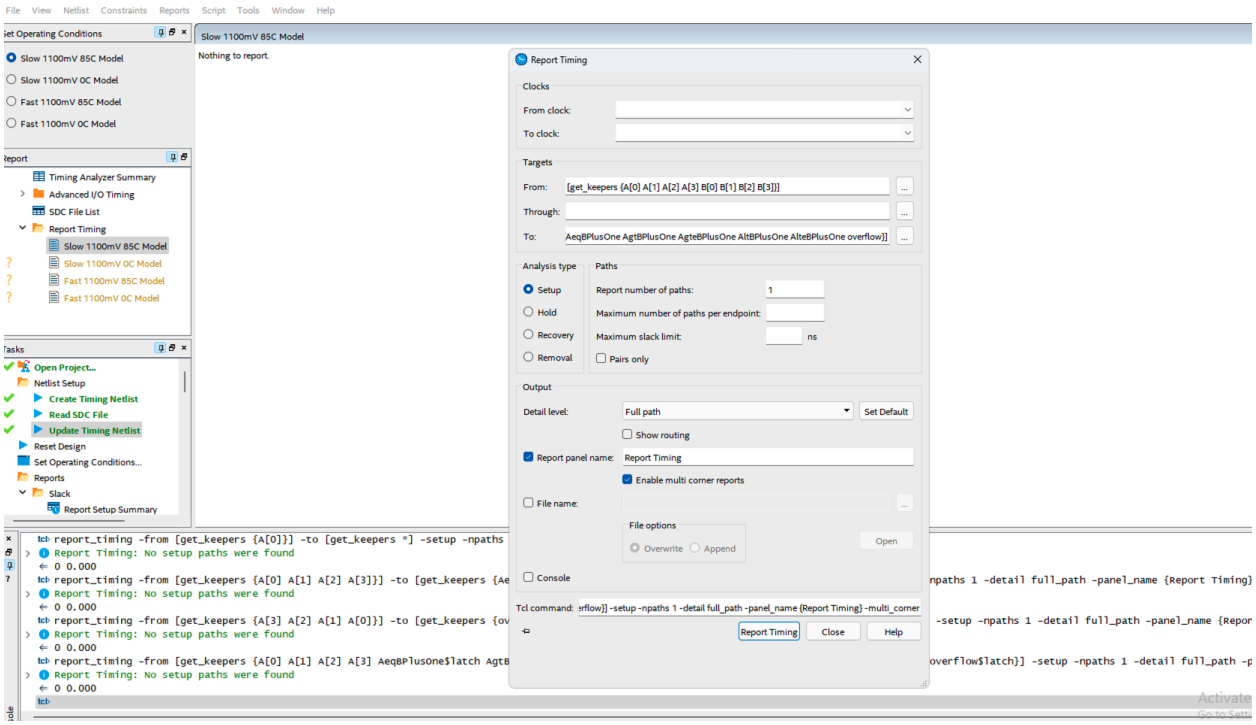
A equals B+1: true only when B=4, since 5 = (4+1)

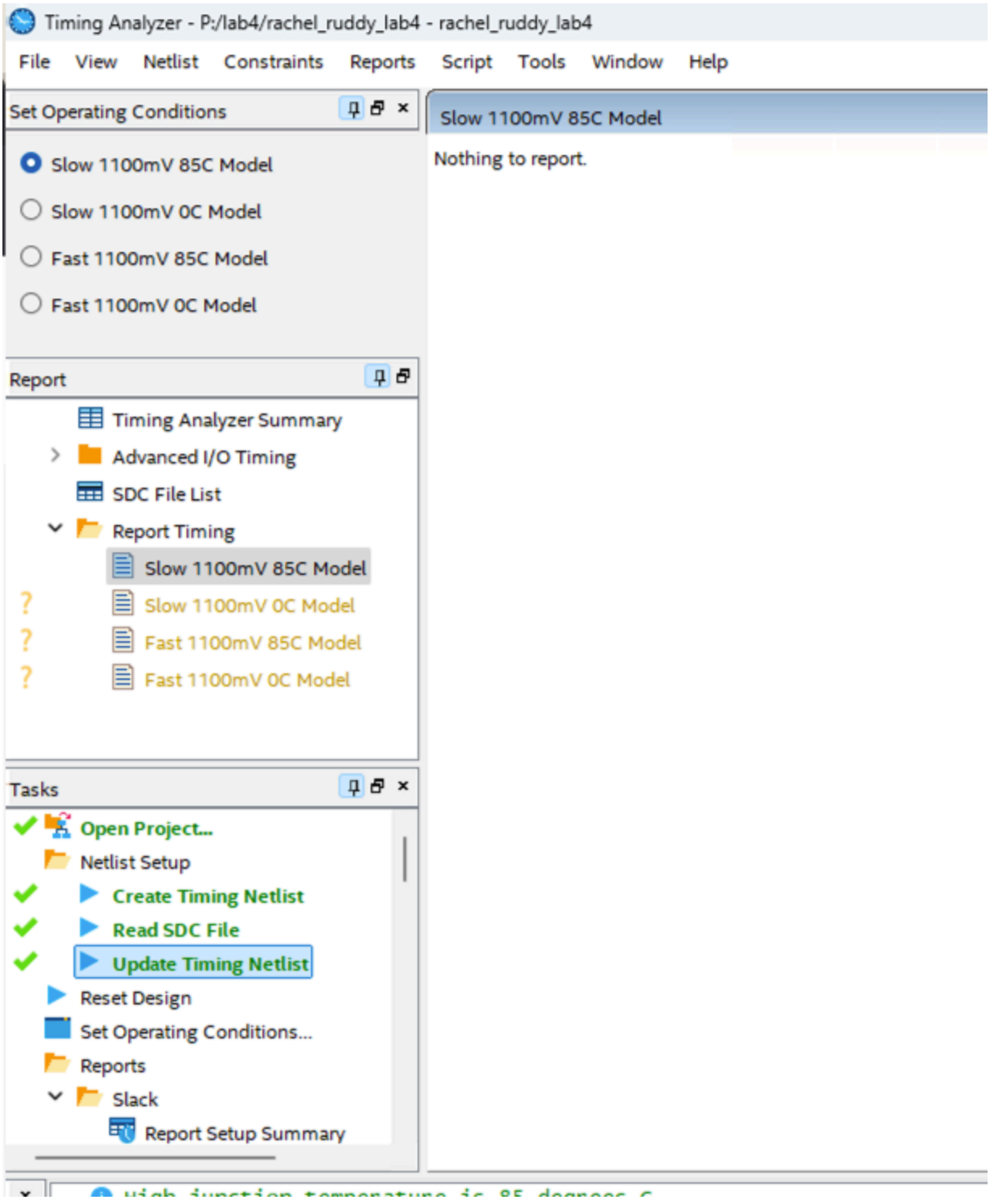


A less than/less than or equal to B+1: true only when B =4 (“0100”) (for less than or equal to, since 5=(4+1)), or for B=5 (for less than, since 5<(5+1)).



6. Perform timing analysis (slow 1,100 mV model) of the 4-bit comparator and find the critical path(s) of the circuit. What is the delay of the critical path(s)?



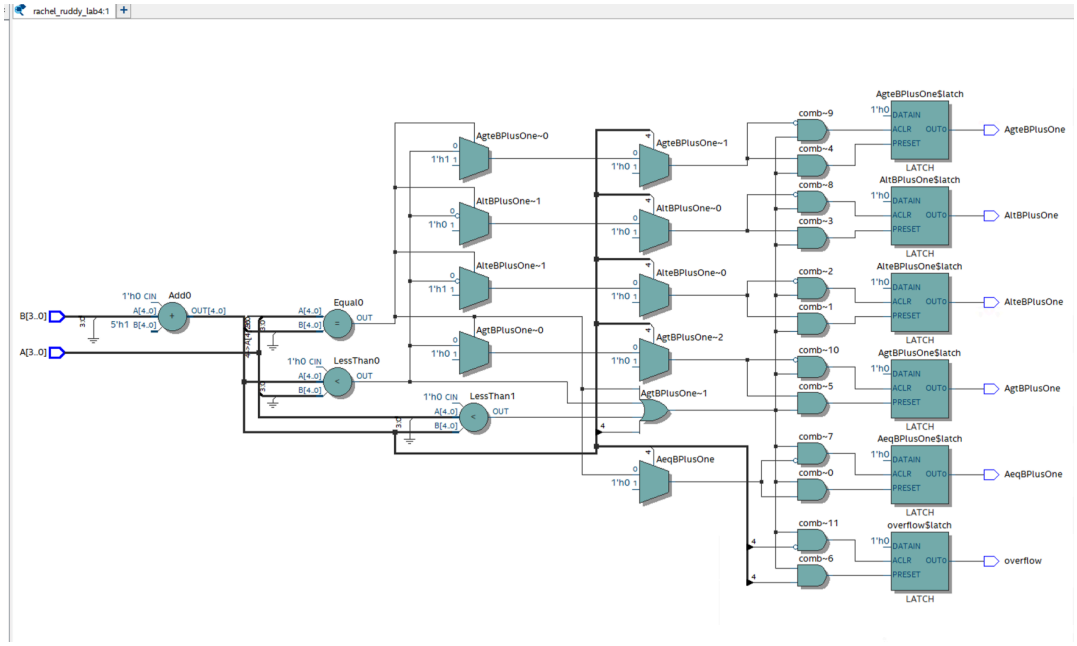

Timing Analyzer - P:/lab4/rachel_ruddy_lab4 - rachel_ruddy_lab4
File View Netlist Constraints Reports Script Tools Window Help
Set Operating Conditions
Slow 1100mV 85C Model
Slow 1100mV 0C Model
Fast 1100mV 85C Model
Fast 1100mV 0C Model
Report
Timing Analyzer Summary
Advanced I/O Timing
SDC File List
Report Timing
Slow 1100mV 85C Model
Slow 1100mV 0C Model
Fast 1100mV 85C Model
Fast 1100mV 0C Model
Tasks
Open Project...
Netlist Setup
Create Timing Netlist
Read SDC File
Update Timing Netlist
Reset Design
Set Operating Conditions...
Reports
Slack
Report Setup Summary
Slow 1100mV 85C Model
Nothing to report.

Compilation Report - rachel_ruddy_lab4 | rachel_ruddy_sdc.sdc

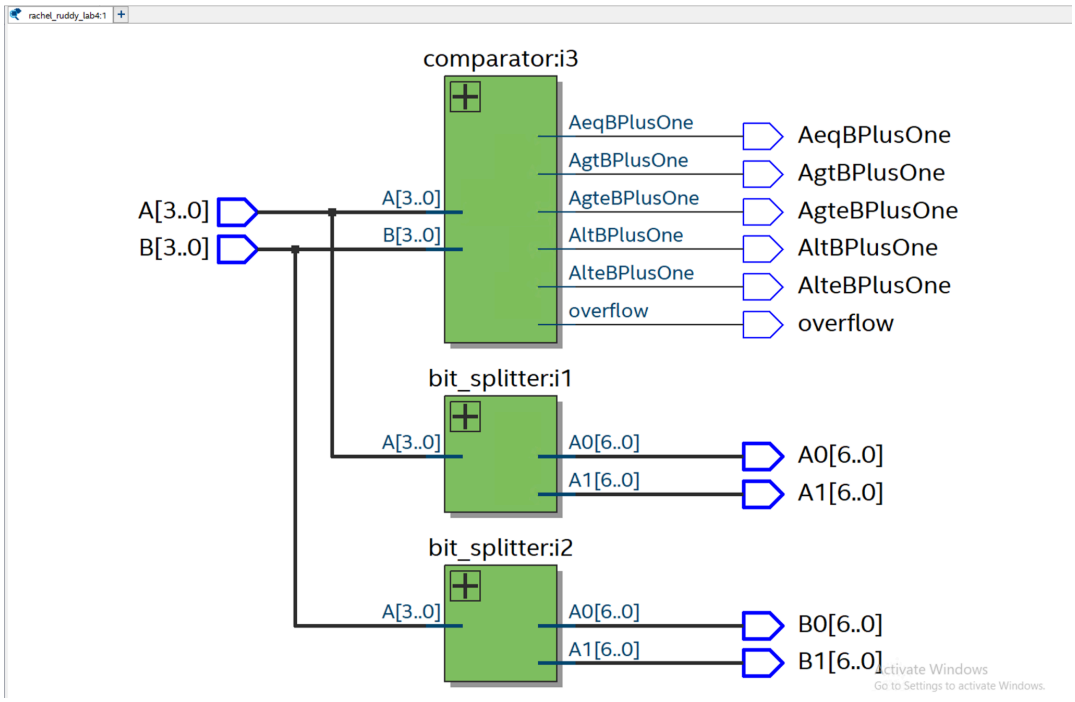
```
1  set_max_delay -from [get_ports A[*]] -to [get_ports AgteBPlusOne] 12
2  set_max_delay -from [get_ports B[*]] -to [get_ports AgteBPlusOne] 12
3
4  set_max_delay -from [get_ports A[*]] -to [get_ports AgtBPlusOne] 12
5  set_max_delay -from [get_ports B[*]] -to [get_ports AgtBPlusOne] 12
6
7  set_max_delay -from [get_ports A[*]] -to [get_ports AeqBPlusOne] 12
8  set_max_delay -from [get_ports B[*]] -to [get_ports AeqBPlusOne] 12
9
10 set_max_delay -from [get_ports A[*]] -to [get_ports AlteBPlusOne] 12
11 set_max_delay -from [get_ports B[*]] -to [get_ports AlteBPlusOne] 12
12
13 set_max_delay -from [get_ports A[*]] -to [get_ports AltBPlusOne] 12
14 set_max_delay -from [get_ports B[*]] -to [get_ports AltBPlusOne] 12
15
16 set_max_delay -from [get_ports A[*]] -to [get_ports overflow] 12
17 set_max_delay -from [get_ports B[*]] -to [get_ports overflow] 12
18
```

We followed the instructions from lab 3 to perform a timing analysis and find the delay of the critical path. Above is our sdc file, and we provide screenshots to show how we selected our inputs and outputs for the report. However, after performing the analysis under the slow model, we did not receive any information except the message “Nothing to report”. We sourced everything we could, including the previous instructions, the internet, but could not find a solution to this issue.

Comparator Schematic:



Comparator Wrapper Schematic:



IV. Conclusions:

Our final implementation of the 2-digit BCD adder and the 4-bit comparator circuits function as intended. It was especially helpful to be able to see our representation of both A and B on the FPGA board as it allowed us to test for edge cases and see how we can apply circuit design to a practical real world use.