Lab3: BCD Adder Subtractor Unit

Objective: To develop a hardware circuit which adds and subtracts values as Binary Coded Decimals (BCD). A 7-Segment decoder will be designed and used to display results. This lab will provide students with an exercise in developing hierarchical hardware designs using VHDL, and how to setup and run **timing** simulations in Modelsim.

1) Overview & Background Knowledge:

a) Binary Coded Decimal (BCD) vs decimal

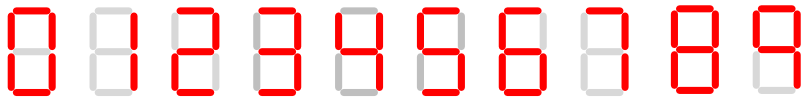
Binary Coded Decimals (BCD) is a class of binary encodings. Each BCD digit is **four** bits and represents the decimal values 0 through 9 in binary. Therefore, a BCD digit may only take on the binary values 0000, 0001, …. through to 1001.

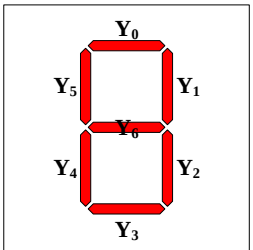
A good way to understand BCD is to consider a digital clock or old VCR display. As you've noticed, only the values 0-9 are displayed. To represent the decimal value 12 for instance, the device would require two BCD digit placings, the first digit displaying a '1' (0001) and the second '2' (0010) ; we have learned that 12 in binary however is represented as 1100. If we are to create a BCD adder subtractor unit, we would therefore need to design and integrate circuitry that can i) calculate the binary result and ii) translate the binary value (1100) to its BCD equivalent (0001 0010). We will be developing the logic in this lab.

b) 7-Segment Display (aka Hex Display)

A 7-segment display is an electronic device that is used to display decimal numbers. They are widely used in simple electronic devices such as digital clocks, calculators, and so on. An example of one 7-segment display is provided in the figure below, illustrating how the values 0 through 9 are displayed.

When we design hardware, it is common to use a 7-segment to "display" the corresponding value for the user to observe. Accordingly, we must implement conversion logic, referred to as a 7-segment converter, that **translates** the 4bit BCD input to a 7bit string of 1s and 0s. This 7bit string will light up individual LEDs on the 7segment display, outputting the corresponding BCD in a human readable decimal number format. A legend of all displayable characters considering a BCD encoding is provided below. For example, if we wanted to display BCD 1001 on a 7-segment, we would need to light up all LEDs Y0-Y2, Y5 and Y6. Please refer to your lecture which provides specific details of implementation.





7-Segment Display

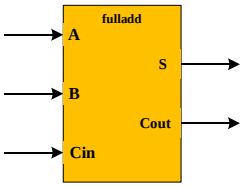
Note that the 7-segment converter you are implementing is an **active-low circuit**, meaning that a '0' will light up a corresponding LED, and '1' will turn the LED off. Although we traditionally assume active-high circuits, where '1' implies on and '0' off, please make note of this **active-low** circuitry during your lab.

c) Steps in Hierarchical design

In this lab, we will be designing the following components using VHDL to realize a BCD adder subtractor unit, with displayable output:

- **1-bit full adder –** will add A+B with a carry in, and produce a value for the sum and carry out
- **4-bit full adder** – we will instantiate four 1bit full-adders to realize this circuit, and make the necessary connections between components to realize a 4bit adder
- **BCD adder** – using and instantiating the 4-bit adder above, we will integrate additional circuitry that interprets the result of the **binary** 4bit adder as a **BCD** value ranging from 0-9. A number which surpasses this range will be represented with a carry bit, implying the most significant digit is '1', with the remaining result(sum) representing the least significant digit. In this case we will implement a two-digit unit.
- **9s compliment generator** – as the unit also performs subtraction, we must consider that 2s complement will be used to subtract binary values. Considering that we require BCD format, a 9s compliment method must be used instead. The generator will implement this circuitry.
- **1 digit BCD adder subtractor unit** – we will integrate the BCD adder and 9s compliment generator component to realize our design goal for a BCD add/sub unit.
- **7-segment converter** – the output of the BCD adder subtractor unit will be interpreted by this unit to display the result to the user in a readable format on a 7-segment display.

The next section will provide you with specifics on implementation.



2) Hardware Components & Integration:

a) 1bit full adder

Use Quartus to make a new project called **bcdaddsub_xxxx**, where xxxx is the last four digits of your SFU ID. Ensure you specify the FPGA as **Cyclone IV E,** device **EP4CE115F29C7**

- File > New > VHDL file, named **fulladd.vhd**. Set this file as your **Top-Level ENTITY**
- Declare an **ENTITY** named **fulladd**, with the ports named as shown in the diagram above.
- Write the **ARCHITECTURE** using AND, OR and XOR **operators**. Implementation is provided in the lectures. Only use **Simple Signal Assignments** to assign outputs sum and carry out.

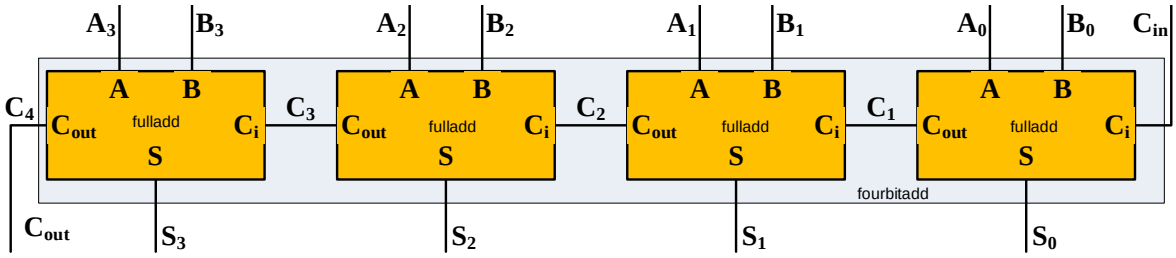
- **Compile to ensure no errors. Fix any errors accordingly**
- **Check** that the circuit is correct using the **RTL viewer.** Using **Tools > Netlist Viewers > RTL Viewer**, you may view the block diagram of your circuit to check if all gates and logic are connected and implemented correctly.
- **Verify** that the circuit is correct using **ModelSim** – create a **testbench** for the full adder circuit. Consider all possibilities for thorough testing of your **fulladder**.
- Prepare the full adder's simulation results for submission by following the steps provided in our previous lab:
    - Prepare the waveform results for submission: Provide a title, annotating the image and providing a brief discussion about simulation results.

b) 4bit full adder

- Within your **fourbitadd** project, create a new VHDL file, named ***"fourbitadd.vhd"***

- Set this ***fourbitadd.vhd*** as **the top-level ENTITY** of your project**.**

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY fourbitadd IS
    PORT( A, B   : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
          Cin    : IN STD_LOGIC;
          S      : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
         Cout    : OUT STD_LOGIC);
END fourbitadd;

ARCHITECTURE Structure of fourbitadd is
    --You fill in the rest

END Structure;
```



- **Use the code above** as a template for entering the basic information to design your circuit. Do not change the port names or datawidth in any way.
- The system will consist of instantiations of the fulladd as seen in the structural diagram above, interconnected to form a **4bit full adder circuit**.
- Re-draw the structural diagram of the 4bit full adder, **labelling internal signal nodes**, **instances** and **ports. Label** the **instances** consistently and any signals required.

- Design the system ARCHITECTURE Structure in VHDL based on the structural diagram

- Run an error check to catch any errors and fix them accordingly.
- **Compile. Quickly Check** that the circuit is correct using the RTL viewer.
- Create a **testbench** that will generate a **waveform** which sufficiently tests corner cases (i.e. extreme cases 1111+1111, 0000+0000) and average cases to verify all functionality of a 4-bit adder. Refer to your lecture notes for these cases.
- **Verify** design: Run a **functional simulation** to verify correct functionality of your circuit in Modelsim.
- Prepare simulation results for submission by following the steps provided in the previous lab. Provide a title, annotating the image and providing a brief discussion about simulation results.

c) Modelsim and Timing Simulations

- **Timing effects**: Once you have verified the design using a **functional** simulation, a timing simulation will be completed to observe the worst-case propagation delay of your adder's ripple carry chain.
- You may use the same testbench vhdl file as the functional simulations, however 1) note that the inputs used, and transitions between inputs makes a difference in propagation delay, as discussed

in Lecture 07, and 2) you must perform different steps to generate a timing simulation in Modelsim as outlined in the "**Modelsim Timing Tutorial**"

- Follow the tutorial provided: "**Modelsim Timing Tutorial**". This outlines the steps necessary to generate timing simulations with Modelsim, preparing the necessary files with Quartus tools.
- Ensure you have included the worst-case generate-propagate scenarios for your 4bit ripple carry adder in your testbench. Refer to your lecture notes for hints.

- **To measure delays**: scroll the delay cursor at the top left corner (yellow vertical line) along the waveform. The delay commences at the time you change the X,Y and or Cin value, until the desired output settles in value. i.e. scroll the cursor to view the start and finishing times, and do this for each of your test cases. Clearly outline the worst case, annotate the waveform with a discussion on how the delay was derived, answering the questions below, and ensure you include this in your lab report.

What is the worst-case propagation delay for all your test cases? What signal takes longest to settle? Discuss. Include this discussion in your waveforms submitted with this lab. The student who achieves the longest delay, with proof and justification in the submitted waveforms will receive a bonus mark. Use our lecture 07 content to help guide you through this step.

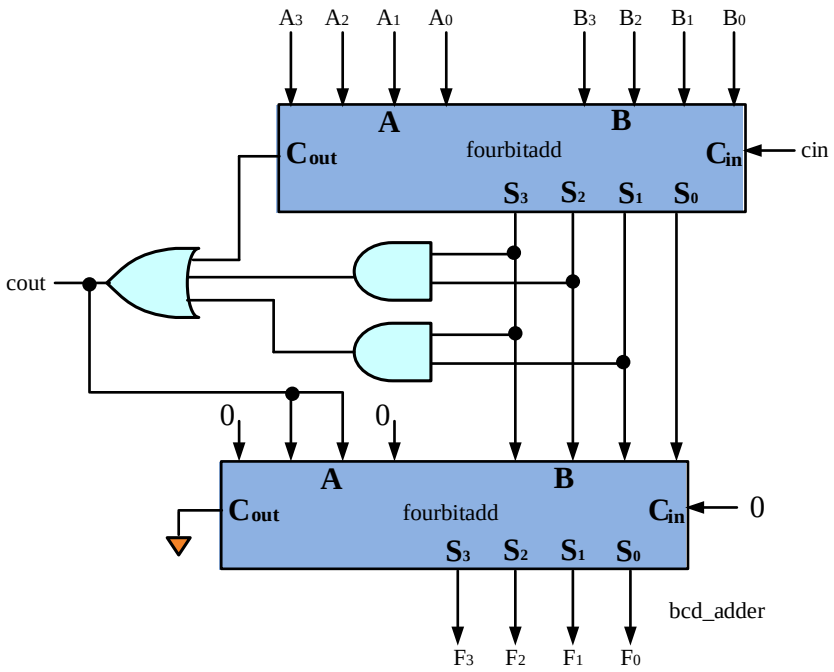
d) BCD adder

The 4bit binary adder designed in the previous section will be used to design the BCD adder in this section. A structural diagram for the BCD adder is shown on the next page, which integrates two 4bit binary adders and error correcting circuitry.

The error correcting circuitry is used to ensure that the binary value output by the 4bit adder is in BCD format. To do so, the binary 6 (0110) value is added to the result if the result is either i) over the binary value 9, or ii) a carry out is generated. As an exercise, derive the logic that proves the circuit below restricts the output values range from 0 to 9. Ensure you add a page to your Visio lab report and add this derivation.

Within your **bcdaddsub_xxxx** project, create a new VHDL file, named ***"bcd_adder.vhd".*** Set this file as **the top-level ENTITY** of your project**.**

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY bcd_adder IS
    PORT( A, B   : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
          cin    : IN STD_LOGIC;
          F      : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
         cout      : OUT STD_LOGIC);
END bcd_adder;

ARCHITECTURE Structure of bcd_adder is
    --You fill in the rest

END Structure;
```



- **Use the code above** as a template for entering the basic information to design your circuit. Do not change the port names or datawidth in any way.
- The system will consist of instantiations of the **fourbitadd** as seen in the structural diagram above, interconnected with 2 AND gates and an OR gate to form a **bcd adder circuit**.
- Re-draw the structural diagram of the 4bit full adder, **labelling internal signal nodes**, **instances,** and **ports. Label** the **instances** consistently and any signals required.

- Design the system ARCHITECTURE Structure in VHDL based on the structural diagram

- Run an error check to catch any errors and fix them accordingly.
- **Compile. Quickly Check** that the circuit is correct using the RTL viewer.
- Create a **testbench** that will generate a **waveform** which exhaustively verifies the functionality of the bcd_adder.
- **Verify** design: Run a **functional simulation** to verify correct functionality of your circuit in Modelsim.
- Prepare simulation results for submission by following the steps provided in the previous lab. Provide a title, annotating the image and providing a brief discussion about simulation results.

e) 9s complement generator

Since our final BCD unit must support both addition and subtraction, we must consider these two modes of operation when assembling our final circuit. We are aware that 2s complement is required for binary subtraction, however this does not represent our answer in the BCD format we require. Accordingly, we must use a 9s compliment method.

The logic for the 9s complement logic/truth table is presented below, consisting of **two inputs**: 1bit *mode*, 4bit *X*, and **one output**: 4bit *Y*. When mode is set to '0', this signifies that the circuit will **add**. In this case,

the value of X is passed directly to Y. When mode is '1', subtraction is required of the circuit, meaning the input must be transformed to nine's complement. Mode '1' may be represented as Y = 9 – X.

Mode	X3	X2	X1	X0	Y3	Y2	Y1	Y0
0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	0	1	0
0	0	0	1	1	0	0	1	1
0	0	1	0	0	0	1	0	0
0	0	1	0	1	0	1	0	1
0	0	1	1	0	0	1	1	0
0	0	1	1	1	0	1	1	1
0	1	0	0	0	1	0	0	0
0	1	0	0	1	1	0	0	1
1	0	0	0	0	1	0	0	1
1	0	0	0	1	1	0	0	0
1	0	0	1	0	0	1	1	1
1	0	0	1	1	0	1	1	0
1	0	1	0	0	0	1	0	1
1	0	1	0	1	0	1	0	0
1	0	1	1	0	0	0	1	1
1	0	1	1	1	0	0	1	0
1	1	0	0	0	0	0	0	1
1	1	0	0	1	0	0	0	0

- Within your **bcdaddsub_xxxx** project, create a new VHDL file, named ***"nineComp.vhd".*** Set this file as **the top-level ENTITY** of your project**.**

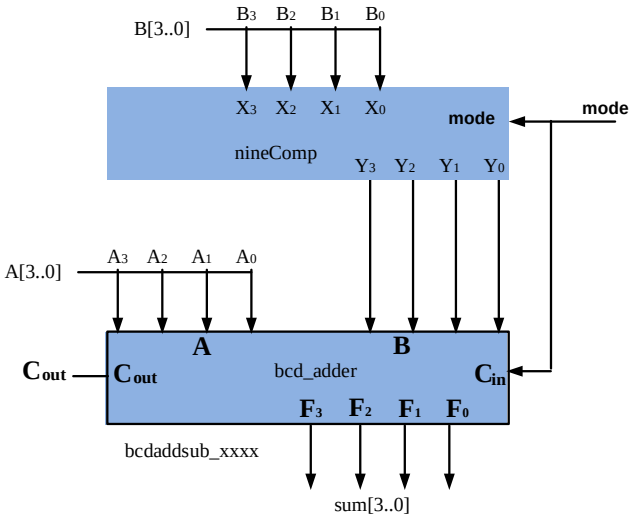
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY nineComp IS
    PORT( X      : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
          mode   : IN STD_LOGIC;
          Y      : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END nineComp;

ARCHITECTURE Behaviour of nineComp is
    --You fill in the rest

END Behaviour;
```

- Use the above VHDL code as a guide to implement the 9s complement generator
- Using the truth table above, **derive the logic necessary** for the ARCHITECTURE of this circuit. Ensure to include your derivation in your report submission.

- Run an error check to catch any errors and fix them accordingly. **Compile.**
- Create a **testbench** that will generate a **waveform** which sufficiently verifies the functionality of the bcd_adder.
- **Verify** design: Run a **functional simulation** to verify correct functionality of your circuit in Modelsim. In this case, we would like to exhaustively test the circuit for correctness.
- Prepare simulation results for submission by following the steps provided in the previous lab. Provide a title, annotating the image and providing a brief discussion about simulation results.

f) BCD adder subtractor Unit

We have now designed all components necessary to assemble the full BCD adder subtractor circuit. The final step is now to connect the 9s complement generator with the bcd adder, which will form the BCD adder subtractor unit bcdaddsub_xxxx. The structural diagram is provided below, containing **inputs:** 4bit A and B, 1bit mode, and **outputs:** 4bit sum, and 1bit cout.



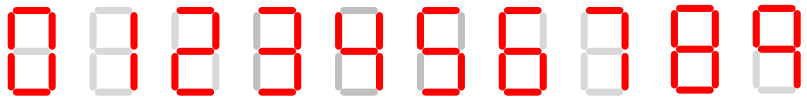
- Within your **bcdaddsub_xxxx** project, create a new VHDL file, named ***"bcdaddsub_xxxx",*** where xxxx are the last 4 digits of your SFU ID**.** Set this file as **the top-level ENTITY** of your project**.**
- Using the structural diagram above, instantiate and connect the components necessary for the ARCHITECTURE of this circuit. Do NOT amend the ports and names in any way.
- Run an error check to catch any errors and fix them accordingly. **Compile.**
- Create a **testbench** that will generate a **waveform** which sufficiently verifies the functionality of the bcd adder subtractor unit. Consider all combinations of addition and subtraction, however noting that BCD encoding is limited to the number 0-9 as discussed previously. Consider worst-case possibilities entered by user.
- **Verify** design: Run a **functional simulation** to verify correct functionality of your circuit in Modelsim.
- Prepare simulation results for submission by following the steps provided in the previous lab. Provide a title, annotating the image and providing a brief discussion about simulation results.

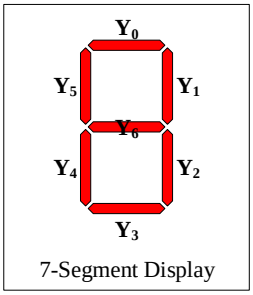
Now, to ensure that the bcd adder subtractor unit's result may be interpreted in a human readable format, we will now proceed to design a 7-segment converter to display the values to the user.

3) 7-Segment Converter

Written work/Derivation of logic:

- The input of a seven segment converter is a 4bit unsigned binary number. The output of the converter is a representation of the input, such that it may be displayed on a seven segment display.
- **Write a truth table** so that the 16 rows produce the patterns specified below for inputs 0-9. If the number entered is not between 0 -9, the circuit should shut all LEDs off in the 7seg display.
- Note that the seven segment decoder you are implementing is an **active-low circuit**.





7-Segment Display

Design

- Create a new project in Quartus called **SevenSeg**
- **Code the circuit:** Enter the design of your combinational circuit using **VHDL**.
- ***File -> New -> VHDL File***. Save as "***SevenSeg.vhd***" - verify that the file is stored in the current project directory. Ensure this is set as your top-level ENTITY. By default it should be as Quartus senses that the project and VHDL file are named identically.
- Declare an ENTITY called SevenSeg, and the PORTs as follows:

```
Port ( D : in  std_logic_vector(3 downto 0);
       Y : out std_logic_vector(6 downto 0));
```

- Write the **ARCHITECTURE** for this **ENTITY**.
  Hint: Use a case construct, or IF conditions. Ensure a PROCESS statement is used for the applicable constructs. If user enters number not between 0-9, all LEDs should be off. Further hints are provided in lecture 06 and 07
- **Synthesize** this final circuit and ensure no errors are present.

Verification:

- An incomplete testbench template, tb_sevenSeg has been provided to you to test your 7-segment display, in addition to a file called **hex_display.vho,** whose COMPONENT has been integrated within the testbench, but contents remain a black box to you.

- The **hex_disp** black box will interpret the 7bit output of your seven segment component, and display the number that would illuminate on an actual 7-segment (HEX) display. If the value 10101010 is output by hex_disp, this signifies that either 1) the value entered was not between 0-9, or 2) if you believe you have "entered" a value from 0-9, then you have made an error in your SevenSeg.vhd design. Revisit the specifications in this case, your derivation, and fix your errors.

- You will need to connect a signal from your SevenSeg's output **Y** to **hex_display**'s input **bcd_in**. Ensure your bits are mapped in the correct order i.e. (6 DOWNTO 0) as discussed in the lecture.

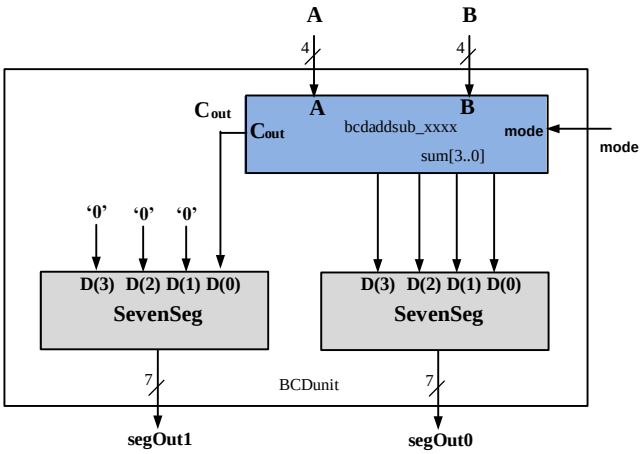
- Connect any other necessary signals to the inputs and outputs of your seven segment display, and verify that the circuit works correctly using a **functional simulation**. In particular, **test for all values, 0-9** and verify that the correct number displays to **hex_display's "number"** output signal in your waveforms.

- When importing your design into Modelsim, ensure you compile **tb_sevenSeg.vhd, SevenSeg.vhd, and hex_display.vho** (the "black box")**.**
- When adding signals to the Modelsim waveform: highlight your tb_SevenSeg (a level above the **DUT**) and add all signals in design to the waveform as per the usual. **If bcd_in and number are not**

included: Highlight the **hex_display** circuit, in the right panel which lists multiple signals, only add the signals **bcd_in** and **number** to the waveform. The rest of the signals are unnecessary and do not need to be imported to the waveform.

- **Verify the** design: Run a functional simulation to verify correct functionality
- Prepare **waveform simulation** results for submission by following these steps above. Provide a title, annotating the image and providing a brief discussion about simulation results.

4) Final Design



- Use Quartus to make a new project called **BCDunit**. Ensure you specify the FPGA as **Cyclone IV E,** device **EP4CE115F29C7**
- Create a new VHDL file, named **BCDunit.vhd**. Set this file as your **Top-Level ENTITY**

- Copy and paste the all the vhdl files from your bcdaddsub project and add them to your project folder. Add the files to your project: Project > Add/Remove Files in Project…
- Next, copy and paste the SevenSeg.vhdl and add them to your project folder. Add the file to your project: Project > Add/Remove Files in Project…

- In BCDunit.vhd, declare an **ENTITY** named **BCDunit**, with the exact port names and datawidths as shown in the diagram above.
- **Use the structural diagram shown here** as a template for entering the basic information to design your circuit ARCHITECTURE and STURCTURE
- The system will consist of instantiations of the bcdaddsub_xxxx and SevenSeg.
- Re-draw the structural diagram, **labelling internal signal nodes**, **instances** and **ports. Label** the **instances** consistently according to your code.
- Compile and fix any errors. **Quickly Check** that the circuit is correct using the RTL viewer.
- Verify your circuit's functionality using Modelsim. Create a **testbench** and generate a waveform that **exhaustively** tests the system. In this case, feel free to use your bcdaddsub_xxxx testbench stimulus with the following adjustments below to include the 7-segment display test circuit (vho).

- **In the testbench**, instantiate the **BCDunit** as the DUT, and the hex_display component, in which **the BCDunit's SevenSeg outputs are connected to hex_display component**. Ensure you include

hex_display.vho, as completed in the previous section. Connect the **segOut** port to hex_display's **bcd_in**. Ensure the proper 7segment **integer** numbers are shown in the waveform.

- **Verify** that the circuit is correct using **ModelSim** and the functional waveform, comparing expected values to the actual values obtained in the waveform. Ensure correct functionality.
- Prepare simulation results for submission by following the steps provided in the previous parts. Provide a title, annotating the image and providing a brief discussion about simulation results etc.

5) Submission Details

Source code

- **Copy/Paste** each VHDL source-code you have created in this lab **with syntax highlighting and courier new font** (via notepad++) into the visio file, as you have in your previous labs. Testbenches do not need to be included in the report, but must ALL be present in your Quartus projects.
- Within your VHDL file in Quartus, add your own comments. Your comments can be anywhere in the files and are for YOUR benefit. Add a textbox to **Title each** source code.

Waveforms

•For all testbenches created in this lab, a functional waveform was produced, and in the case of the 4bit adder, a timing waveform. For these waveforms, screenshot or print the waveforms as a pdf, and include it in the Visio file. Like your previous labs, you can also prepare an image of the waveforms using the snipping tool available on the windows start menu under accessories. For each waveform:

- Add a textbox in Visio to **Title** the simulation results. Add a textbox in Visio – enter a brief **discussion** about the simulation results.
- Using lines and textboxes, annotate the image to highlight **important features** of your simulations. Submit functional waveforms for all your designs.
- For the **timing waveform**, clearly annotating your test cases and worst-case delay.

Quartus Projects

Include your i) BCDunit Quartus project which includes all files necessary to replicate the results you present for this lab, ii) the SevenSeg project, and iii) BCDUnit. Ensure that the testbenches (vhdl files) are included in the project. Again, you do not need to include the testbenches in your report.

Prepare Full Document for Submission:

•Complete the SFU **cover page**. Use the template provided on Canvas, and change any necessary field for this lab assignment. Save the pdf.

•Check that the document contains the items specified for submission, as indicated throughout this lab.

•Save the visio file as a **PDF** file. Merge the cover page with this pdf. Ensure the filename is **LA03-252-1207-xxxx.pdf**, (using the last 4 digits of your student number as usual). Upload your .pdf file to Canvas.

•You will demo your work to your TA **during your next scheduled lab period on BBCollab**. Please see the **semester schedule for the due date** -- sign up for a demo time. Your TA will provide you with further details and time slots available.