Project 2 (8%)

ECGR 2181 – Fall 2018

Due: Wednesday, December 5, 2018

For both parts of this project, choose 2 partners of your own choice (teams of 3 only). You are expected to put forth an equal amount of effort to complete this assignment. You will get a chance to review yourself and your teammate using the form available on Canvas. Each team is expected to submit one PDF report for all 3 parts of this project and three (one per team member) peer evaluations. Peer evaluations should be submitted individually as they are to remain anonymous. For all 3 parts of this project, you should include a write-up of your approach (the basic design process, anything you had trouble with, an explanation of anything doesn't work, and any results), similar to Project 1.

Part 1 (2.5%):

Assignment Overview

In this assignment you will use Vivado 2018.2 Webpack to write, simulate and program an FPGA. You'll use the task3.vhd created in Project 1 Part 2 and connect it up to switches and the seven segment displays on the FPGA board. You'll also need to turn in a single PDF to Canvas (instructions at the end of the file).

Background

You need to declare your task3.vhd as a component (it looks exactly like the component declaration in the testbenches shown in Project 1), then instantiate it 4 times, and connect all of the signals (we are going to use it using Structural VHDL). You might find it useful to tie the switches to the LEDs to make it easier to read what the seven segment display should read. Rename task3.vhd to encoder.vhd for the purposes of this Project to avoid confusion.

Structural VHDL is not very difficult. You have to declare a component before you instantiate it. Once you instantiate it, you simply connect signals to it, sort of like working with a schematic but with text (or blackboxes being interconnected). The VHDL slides posted on Canvas offer information on structural VHDL syntax.

Drivers for the FPGA Board

The FPGA board requires some drivers for it to be recognized by Vivado. You may need to download and install Digilent's Adept System software if your system does not recognize the board, which can be located at http://store.digilentinc.com/digilent-adept-2-download-only/ .

Getting started

1. Create a directory to store all of the assignment's files.
2. Open Vivado 2018.2 and create a new project called **computer_assignment_3**.
3. It will be an RTL project
4. Add your encoder.vhd file from Project 1 and the supplied ssd_muxer.vhd file from Canvas.

5. No existing IP.
6. Add the constraints/xdc file from Canvas (Basys3_Master.xdc).
7. The Basis3 uses an **XC7A35T-CPG236C-1** FPGA
    - Family: Artix-7
    - Package: CPG236
    - Speed: -1
    - Choose the part with 41600 flip flops

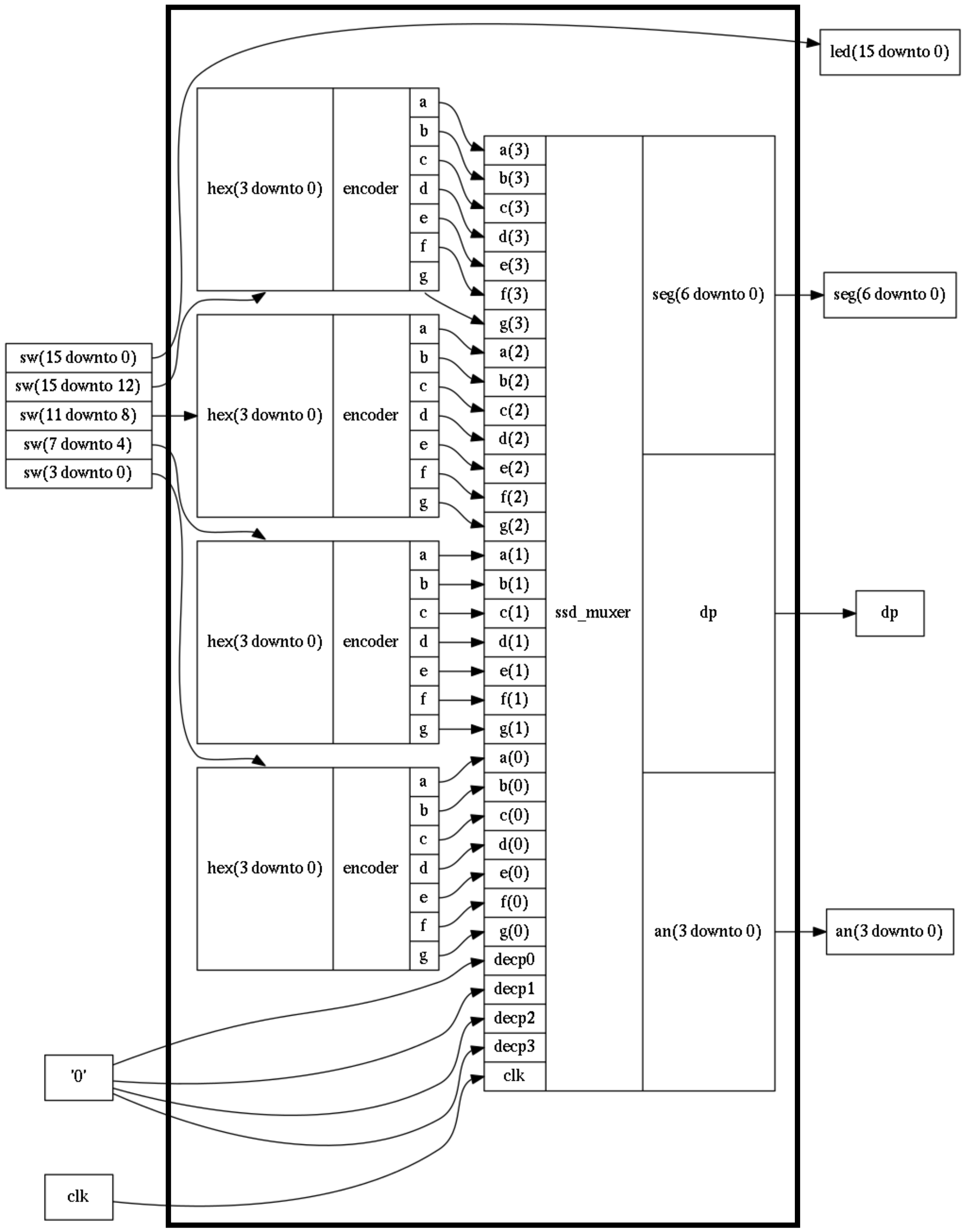

Figure 1: Overview of Task 1.

The thick-bordered rectangle around the design above represents the portion of the design that will be described in the `architecture` portion of your design. The rest of the inputs and outputs will be the port connections to the `entity` portion of your design.

Task

This task is to implement an encoder whose input is a ONE 4-bit hex signal and the output is SEVEN 1-bit signals for each display segment (A-G) – this was also done as part of Task 2 in Project 1.

1. Edit the constraints file and uncomment the lines corresponding to (some of these may already be uncommented):
    - sw
    - clk
    - seg
    - dp
    - an
    - led

2. Create the top.vhd file:
    a) Under "Flow Navigator" click "Add sources."
    b) Select "Create or add design sources."
    c) Create top.vhd and then click "Finish."
    d) Create inputs:
        - A std_logic_vector **sw (15 downto 0)**
        - A std_logic **clk**
    e) Create outputs:
        - A std_logic_vector's **seg (6 downto 0)**
        - A std_logic **dp**
        - A std_logic_vector **an (3 downto 0)**
        - A std_logic_vector **led (15 downto 0)**

3. Once you have your VHDL source file (top.vhd), edit it to implement 4 encoder entities and connect the switches to the LEDs.
    - Since you already have the encoder.vhd, you'll just be using **structural VHDL** to connect multiple encoder entities to the sdd_muxer entity.
    - If you're feeling adventurous you can use a *for generator* statement or you can manually place four encoder entities, but this is more advanced and not expected for this Project.

4. Make sure it's syntax error free and can be compiled.

5. Add the top_tb.vhd file:
    a) Under "Flow Navigator" click "Add sources."
    b) Select "Create or add simulation sources."

c) Create top_tb.vhd.
d) Make sure simulation set is “sim_1.”
e) Click “Finish”.
f) Write code to run the inputs through all possible combinations in order to simulate the design.

6. Edit ssd_muxer.vhd:
   a) Go to line 35-36, there will be two counter_max constants.
   b) Make sure the one that says `fpga` is commented out and the one that says `sim` is uncommented for the purposes of simulation.

7. Select the right simulation set:
   a) Under “Flow Navigator” click “Simulation Settings.”
   b) Under “Simulation set” input “sim_1.”
   c) Under “Simulation top module name” input “top_tb.”
   d) Click “Ok.”

8. Run the simulation and check your results. The expected output is on the last page of the assignment. If your results are shown as “U”s, that means that the code entered in step 5.f) above is not complete and needs to be re-edited.

9. Once you’ve simulated the circuit correctly, you need to edit ssd_muxer.vhd again.
   a) Go to line 35-36, there will be two counter_max constants
   b) Make sure the one that says `fpga` is uncommented out and the one that says `sim` is commented.

10. Create the file you need to load onto the FPGA (the bitstream). Click “Generate Bitstream” under the “Flow Navigator.”

11. Program the FPGA board. Make sure the board is connected via a micro-USB cable.
    a) Click “Hardware Manager” in the Flow Navigator.
    b) “Open Target” and then “Auto Connect.”
    c) “Program Device” and then choose “xc7a35t_0.”
    d) The defaults of the pop up are OK, just press Program.

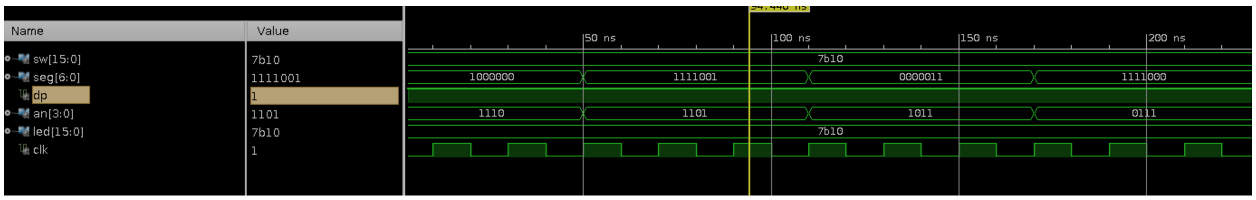

Figure 2: Expected Waveform of Simulation (Not Required)

Turn in
Include a write-up with: the basic design process, anything you had trouble with, an explanation of anything that doesn't work, etc. It isn't meant to be extensive, but a brief overview.

Grading

Printout and correctness of code (top.vhd, encoder.vhd, and testbench)	/11 Points
Demonstration	/14 Points
	/Total of 25 Points

Demonstration
Demonstrate the VHDL running on the FPGA to my grader. All that is needed is to go through some cycles of each set of 4 switches and show that the correct output is displayed on the seven segment displays.

Demonstration Grading:

Each seven segment display works correctly	/3 points x 4
The switches are tied to the LEDs.	/2 points
	/Total of 14 points

Task 2 (3.5 %):
Assignment Overview

In this assignment you will use Vivado 2018.2 Webpack to write, simulate and program an FPGA. You'll use the encoder.vhd created in Project 1. The goal of this Task is to connect the lower 8 switches to form an 8-bit two's complement signed number and display it as a decimal result on the FOUR 7-segment displays. For the seven segment displays, you need to display a negative sign (if the number is negative, nothing/all segments off otherwise) and the absolute value on the binary number. For example if you have FE_{16} as the converted binary value from the 8 switches, then on the display it needs to show "-002". If the input was positive, such as $7E_{16}$, then the display would show as " 126" (notice the "space" in the leftmost 7-segment display to show a positive number).

Background

You need to declare the `encoder`, `ssd_muxer` and `debounce` components, then instantiate them like in the previous Task. In this Task you'll only use the least-significant 8 switches. The IEEE `numeric` package will be useful in this Task. Simulation for this Task isn't required since you have to demo it, it's only to help you debug the code.

Drivers for the FPGA Board

The FPGA board requires some drivers for it to be recognized by Vivado. You may need to download and install Digilent's Adept System software if your system does not recognize the board, which can be located at http://store.digilentinc.com/digilent-adept-2-download-only/ .

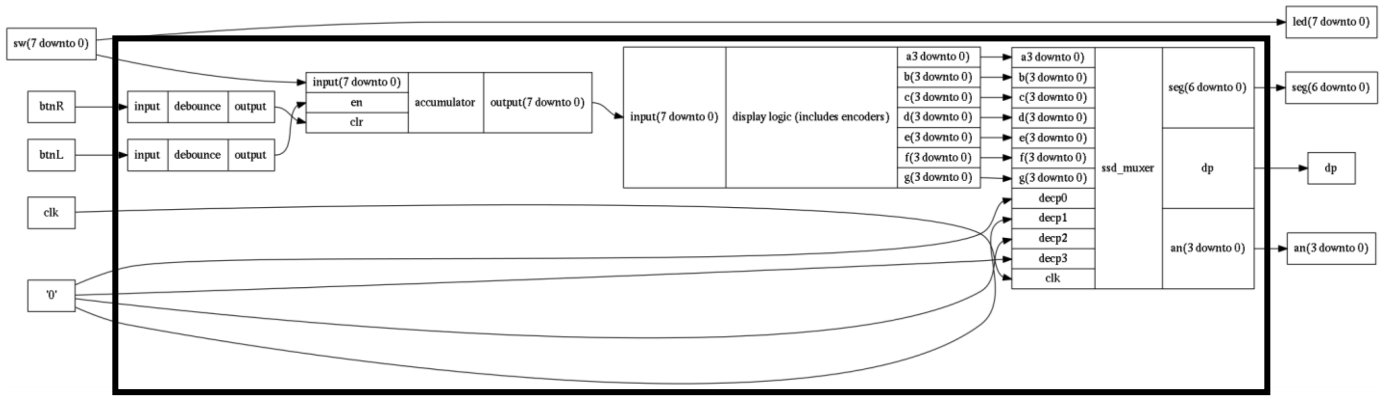

Figure 1: Overview of Task 2 (Note: accumulator and display logic aren't necessarily components, but just parts of the logic needed for this Task, they should be a `process`).

Getting started

1. Create a directory to store all of the assignment's files.
2. Open Vivado 2018.2 and create a new project called **computer_assignment_4**.
3. It will be an RTL project
4. Add your encoder.vhd file from Project 1 and the supplied ssd_muxer.vhd file from Canvas.
5. No existing IP.
6. Add the .xdc file from Canvas (Basys3_Master.xdc).
7. The Basis3 uses an **XC7A35TCPG236C-1** FPGA
    - Family: Artix-7
    - Package: CPG236
    - Speed: -1
    - Choose the part with 41600 flip flops

Task

This task is to implement a base conversion system whose input is an 8-bit signed binary number and whose output is FOUR 7-segment outputs of 1-bit each (A-G).

1. Edit the constraints file and uncomment the lines corresponding with (some of these may already be uncommented):
    - sw (7 downto 0)
    - btnR
    - btnL
    - clk
    - seg
    - dp
    - an
    - led (7 downto 0)
2. Create the top.vhd file:

a) Under “Flow Navigator” click “Add sources.”
b) Select “Create or add design sources.”
c) Create top.vhd and then click “Finish.”
d) Create inputs:
    - A standard_logic_vector **sw (7 downto 0)**
    - A standard_logic **btnR**
    - A standard_logic **btnL**
    - A standard_logic **clk**
e) Create outputs:
    - A standard_logic_vector **seg (6 downto 0)**
    - A standard_logic **dp**
    - A standard_logic_vector **an (3 downto 0)**
    - A standard_logic_vector **led (7 downto 0)**

3. Once you have your VHDL source file (top.vhd), edit it to implement the encoder, ssd_muxer and debounce instantiations.
    - Since you already have the encoder.vhd, you’ll just be using **structural VHDL** to use logic inside of it.
    - Likewise with ssd_muxer and debounce, you will use structural VHDL to use these components.
        - The debounce will map as: CLK to clk, sw to one of the buttons (btnL or btnR), sig to “open”, and sglPulse to the output signal (en or clr). First instance of debounce will allow clearing of the value stored in the accumulator. The second instance will allow loading a value from the input switches (sw 7 downto 0). The purpose of using the debounce is to make sure that one button press is recorded, not multiple presses.
        - The ssd_muxer will be wired exactly the same as in Task 1.
    - You will also need to add additional logic to be able to do the conversion from the 8-bit 2’s complement number into FOUR 7-segment digits to be passed onto the FOUR instances of your encoder.
        - Create a process dependent on the outputs of both debounce entities. Create an 8-bit standard_logic_vector signal called `accumulator` to be able to either store all zeros or store the value coming from the switches, depending on the output of the debounce entities.
        - Create a process dependent on accumulator. This process will perform the signed binary to decimal conversion and isolation of each digit. You will need to first convert the signed binary into decimal integer (hint: you may benefit from taking the absolute value of the decimal number before trying to isolate each digit. You can use the `abs(`*`int_value`*`)` function to accomplish this.) Once you have an absolute value of the integer value, extract each digit by using modulo math. In the final output, it is fine to output additional leading zeros, such as in “ 085”, this makes the logic simpler to perform.
        - The final step is obtaining the most significant 7-segment display value. The entire 7-segment will be off for a positive number. Segment ‘g’ will be on if the number is

negative. To be able to identify the sign, check the original binary input stored in the accumulator: if the MSB is ‘1’, the number is negative; if the MSB is ‘0’, the number is positive. In order to show the sign on the 7-segment display, you will need to modify the encoder.vhd file. Modify any of the values that map to A-F to accommodate these 2 additional cases since A-F are not part of our decimal integer combinations. It is suggested that you modify A to map to a positive integer (all segments off) and modify B to map to a negative integer (segment ‘g’ on only).

4. Make sure the code is syntax error free and can be simulated.
5. Add the top_tb.vhd file:
    a) Under “Flow Navigator” click “Add sources.”
    b) Select “Create or add simulation sources.”
    c) Create top_tb.vhd.
    d) Make sure simulation set is “sim_1.”
    e) Click “Finish”.
    f) Write code to run the inputs through a few combinations in order to check your design.
6. Edit ssd_muxer.vhd:
    c) Go to line 35-36, there will be two counter_max constants.
    d) Make sure the one that says fpga is commented and the one that says sim is uncommented.
7. Comment the debounces and connect the btrR and btrL signals to their corresponding signals (temporary signals for simulation only – make sure to turn these ON or OFF based on simulation cases.)
8. Select the right simulation set.
    a) Under “Flow Navigator” click “Simulation Settings.”
    b) Under “Simulation set” input “sim_1.”
    c) Under “Simulation top module name” input “top_tb.”
    d) Click “Ok.”
9. Run the simulation by and check your results. The expected output is on the last page of the assignment.
10. Once you’ve simulated the circuit correctly, you need to edit ssd_muxer.vhd again..
    c) Go to line 35-36, there will be two counter_max constants
    d) Make sure the one that says fpga is uncommented and the one that says sim is commented.
11. Edit top.vhd and add/uncomment your debouncers back in.
12. Create the file you need to load onto the FPGA (the bitstream). Click “Generate Bitstream” under the “Flow Navigator.”
13. Program the FPGA board
    e) “Hardware Manager” in the Flow Navigator.
    f) “Open Target” and then “Auto Connect.”
    g) “Program Device” and then “xc7a35t_0.”
    h) The defaults of the pop up are ok, just press program.

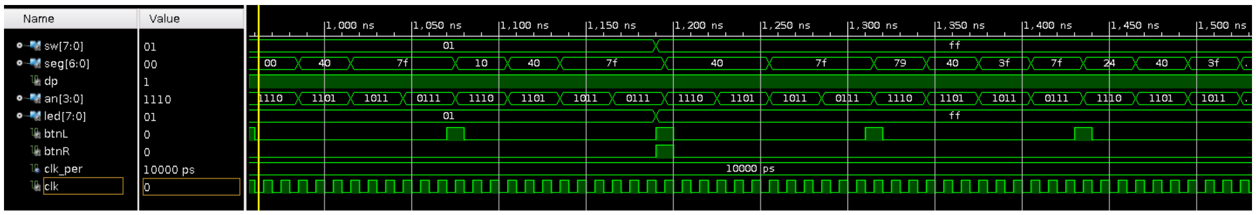

Figure 2: Expected Waveform of Simulation for Task 2.

Turn in

Include things like: the basic design process, anything you had trouble with, an explanation of anything that doesn't work, etc. It isn't meant to be extensive, but a brief overview.
Include top.vhd, encoder.vhd, or any code you wrote in the PDF.

Grading

Printout and correctness of code	/11 Points
Demonstration	/24 Points
	Total of 35 Points

Demonstration

Demo the code to my grader. Fill out the demonstration sheet, print it and fill out the self-review before the grader demo. You need to be able to accumulate positive and negative numbers. For the seven segment displays, you need to display a negative sign and the absolute value on the display. For example if your accumulated value is X"FE", then on the display it needs to show "-02".

Demonstration Grading:

Load positive and negative numbers and clear accumulator	/9 points
Convert binary to decimal correctly	/9 points
Seven segment display shows number and sign correctly	/4 points
LEDs are connected to the SWs	/2 point
	/Total of 24 point

Part 3 (2 %):

Assignment Overview

In this assignment you will use Vivado 2018.2 Webpack to write HDL, simulate (optional) and program an FPGA. In this lab, you will be creating a guessing game. One lab partner will input a number using eight of the switches, store it with btnR, then the other lab partner will guess the number and test to see if it's true with btnL. When btnL is pressed and the values match, <u>only LED15 lights up</u> and if the values don't match, all LEDs are turned off. The LED stays high/low until you test it again. You'll need use the provided debounce.vhd for the buttons (Task 2).

Background

You need to declare the debounce component, then instantiate it like in the previous Task. In this lab you'll only use the least significant 8 switches. **Simulation for this lab isn't required** since you have to demo it, it's only to help you debug the code.

Drivers for the FPGA Board

The FPGA board requires some drivers for it to be recognized by Vivado. You may need to download and install Digilent's Adept System software if your system does not recognize the board, which can be located at http://store.digilentinc.com/digilent-adept-2-download-only/ .

Getting started

1. Create a directory to store all of the assignment's files.
2. Open Vivado 2018.2 and create a new project called **computer_assignment_5**.
3. It will be an RTL project
4. Add your encoder.vhd file from Project 1 and the supplied ssd_muxer.vhd file from Canvas.
5. No existing IP.
6. Add the constraints/xdc file from Canvas.
7. The Basis3 uses an **XC7A35TCPG236C-1** FPGA
    - Family: Artix-7
    - Package: CPG236
    - Speed: -1
    - Choose the part with 41600 flip flops

Task

This task is to write and simple game where one person sets a number/value and the other person tries to guess it.

1. Edit the constraints file and uncomment the lines corresponding with:
    - sw (7 down to 0)
    - btnR
    - btnL
    - clk
    - led (15 downto 0)
2. Create the top.vhd file:
    a) Under "Flow Navigator" click "Add sources."
    b) Select "Create or add design sources."
    c) Create top.vhd and then click "Finish."
    d) Create inputs:
        - A std_logic_vector **sw (7 downto 0)**
        - A std_logic **btnR**
        - A std_logic **btnL**
        - A std_logic **clk**
    e) Create outputs:
        - A std_logic_vector **led (15 downto 0)**

3. Once you have your vhdl source file:
    - Add two debouncers, one for each button, same as in Task 2.
    - Connect the sw to led(7 downto 0).
    - Create the logic needed to store your sw value when you set it using btnR (same as Task 2).
    - Create the logic needed to test the stored and sw values and set the led(15) accordingly using btnL.
4. Make sure the code syntax error free and can be simulated.
5. Add the top_tb.vhd file:
    a) Under "Flow Navigator" click "Add sources."
    b) Select "Create or add simulation sources."
    c) Create top_tb.vhd.
    d) Make sure simulation set is "sim_1."
    e) Click "Finish".
6. Comment out the debounces and connect the btrR and btrL signals to their corresponding temporary signals used for testing.
7. Select the right simulation set.
    a) Under "Flow Navigator" click "Simulation Settings."
    b) Under "Simulation set" input "sim_1."
    c) Under "Simulation top module name" input "top_tb."
    d) Click "Ok."
8. Run the simulation and check your results. The expected output is on the last page of the assignment.
9. Edit top.vhd and uncomment your debounces back in.
10. Create the file you need to load onto the FPGA (the bitstream). Click "Generate Bitstream" under the "Flow Navigator."
11. Program the FPGA board
    i) "Hardware Manager" in the Flow Navigator.
    j) "Open Target" and then "Auto Connect."
    k) "Program Device" and then "xc7a35t_0."
    l) The defaults of the pop up are ok, just press program.

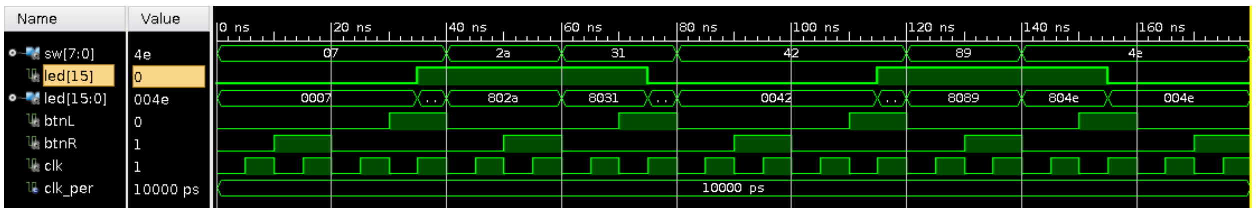

Figure 1: Expected Waveform of Simulation

Turn in

The write-up should include things like: the basic design process, anything you had trouble with, an explanation of anything that doesn't work, etc. It isn't meant to be extensive, but a brief overview. Include top.vhd, or any code you wrote in the Canvas PDF submission.

Grading

Printout and correctness of code	/8 Points
Demonstration of FPGA	/12 Points
	Total of 20 Points

Demonstration

You need to demonstrate the VHDL running on the FPGA to my grader. Fill out the demonstration sheet, print it, and fill out the self-review before the grader demo. You need to be able to set and test numbers with the switches correctly and led(15) lights up when a match is found (after testing it). **<u>ONLY LED(15) NEEDS TO BE ON, NOT ALL OTHER LED'S.</u>**

Demonstration Grading:

Set the number correctly	/5 points
Test the number correctly	/5 points
All LEDs are connected to the correct signals	/2 points
	/Total of 12 point
Write-up (all 3 parts: 4pts+4pts+2pts)	/10 points
Self- and Peer-evaluations	/10 points