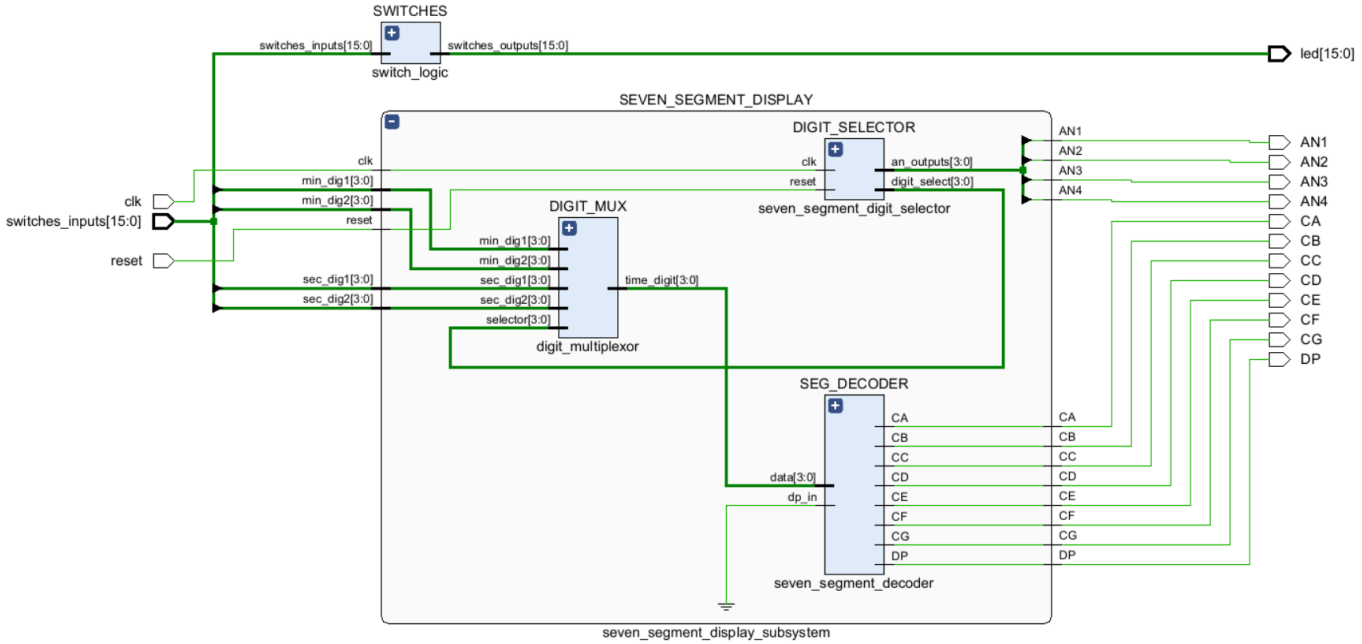
Lab 1b – Introduction to Vivado and Basys3

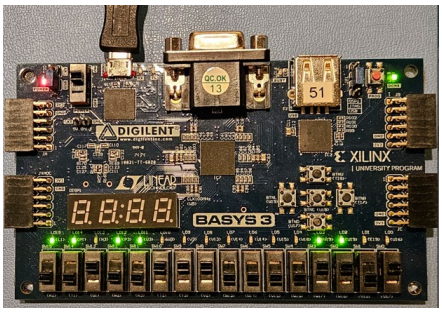
Introduction

Lab 1b will help you become more familiar with Xilinx Vivado and the Digilent Basys3 FPGA board. You will continue to learn how to synthesize a design, simulate it, and download it to the board for testing. You will also learn how to determine the maximum clock frequency for your design.

The instructions for Lab 1b are at a higher level that of Lab 1a, because it is expected that you know the basics of the design flow within Vivado and are able to use the Basys3. Lab 1b will extend your design of Lab 1a, with a fundamental building block that is a 7-segment display module. You will iteratively add this component to your design, to get more practice with Vivado and the Basys3, and to model an effective approach to engineering design. This design practice is *incremental and iterative*, so that a complex design gradually emerges from a methodical process of developing, testing, and incorporating smaller design units to the larger, more complex design.

You will end up with a project that looks similar to the RTL schematic shown below. Note that the internals of the 7-segment display module are also shown, which are packaged together as the 7-segment display subsystem.





A difference in this and future labs is that you will <u>submit your Design Record</u> to the D2L Dropbox.

Procedure

1. Prior to the lab*, watch the all the lecture videos for this week and the previous week. Also watch the tutorial video for this lab, where you will get a quick start to this lab. This video shows the main steps for this lab activity. Also read this lab document in full and be prepared to start the lab immediately. **the term "Lab" is used in this document for convenience and it translates to "Active Learning Mini-Project," in terms of the Course Outline.*
2. We will incorporate the 7-segment display subsystem, so that we can see the binary input on the slide switches, as Hexadecimal numbers on the 7-segment displays. *These instructions may not be perfect and step-by-step, so use your best judgement and adjust your steps accordingly, to obtain the desired results.*
3. Create a new project in Vivado and be sure to pick the correct FPGA part (xc7a35tcpg236-1). *Aside, we are creating a new project and bringing in copies of our files rather than working directly from the Lab 1a project, partly because we want a known good working design project to go back to, if our development results a mess. A good rule in engineering is to not modify your only working prototype.*
4. Make a copy of your Lab 1a .SV and .XDC files. Rename from *"_1a_"* to *"_1b_"*:
    - *lab_1a_top_level.sv* to *lab_1b_top_level.sv* (also edit the file contents, to rename the module to *lab_1b_top_level*)
    - *lab_1a_top_level_tb.sv* to *lab_1b_top_level_tb.sv* (also edit the file contents, to rename the module to *lab_1b_top_level_tb*, **and also** rename the *lab_1b_top_level* instantiation)
    - *Basys3_Lab_1a.xdc* to *Basys3_Lab_1b.xdc*.
    - The above edits are crucial and I anticipate many teams will be doing lots of tricky debugging if they are not careful.
5. Add following the **Design Sources**:
    - *lab_1b_top_level.sv* (be sure to set this as "top" when finished adding sources)
    - *switch_logic.sv*. (Ensure that the internal logic of this module just passes the inputs to the outputs directly (i.e. no AND, INVERTER, nor any other logic). We want the binary values of the slide switches to go directly to the LEDs, so that we can directly compare them to the Hexadecimal values on the 7-segment displays.)
    - *seven_segment_display_subsystem.sv* (note, this module instantiates the following modules, demonstrating a good functional decomposition of the 7-segment subsystem (these files must also be added)):
        i. *seven_segment_decoder.sv*
        ii. *seven_segment_digit_selector.sv*
        iii. *digit_multiplexor.sv*
6. Add the testbench in **Simulation Sources**: *lab_1b_top_level_tb.sv*. (be sure to set this as "top")
7. Add the constraints file to **Constraints**: *Basys3_Lab_1b.xdc*.
8. **Read this guidance carefully, understand it, and feel it in your bones. These are some of the most important concepts in engineering design, that you will hopefully learn to appreciate and adopt into your own practice.** First, let's ensure that you are working with a "good" design, before adding design units to it. This promotes good engineering design practice through iterative and incremental development, by building upon "known" good foundations. Further to good engineering design practice, you would ensure that the modules you are adding are also known to be good, by:
    - verifying their functionality through simulation (also known as "unit testing"), and
    - through a synthesis check to verify that they can actually be built in hardware.

    We don't have to do these steps this time, because you are being provided with known, good design units.

9. Without attempting to incorporate the 7-segment display subsystem yet, simulate the design and ensure that works as expected. **<u>DO: paste a snip of the simulation waveforms into your Design Record. Ensure that the waveforms are setup to easily demonstrate the functionality of the module.</u>**

10. Download the configuration to the Basys3 board and ensure that the design works as expected (slide the switches to various positions and check that the LEDs follow the switch positions.
11. In RTL ANALYSIS, run the schematic: **<u>DO: copy and paste an image of the RTL Schematic into your Design Record.</u>**
12. Edit the *lab_1b_top_level.sv* module to instantiate the *seven_segment_display_subsystem* module. As part of the instantiation, you will connect the correct signals to the *seven_segment_display_subsystem* module instantiation. Some your actions will include:
    - Connecting *clk* and *reset* input signals. You'll have to add these signals to your top level module port list (and later add them to your .XDC constraints file).
    - Connecting the *AN1, AN2, AN3, AN4* and the *CA, CB, CC, CD, CE, CF, CG*, and *DP* output signals. You'll have to add these signals to your top level module port list (and later add them to your .XDC constraints file).
    - Next, you will connect the slide switch inputs from the top level module (which are already in your top level module as inputs, from your Lab 1a), to the *seven_segment_display_subsystem* module. Originally, the *seven_segment_display_subsystem* module was used for stopwatch design and its input signal names reflect this design, where the inputs for the four 7-segment digits are four 4-bit nibbles in the order of (tens of minutes; units of minutes; tens of second; and units of seconds). You need to map the 16-bits coming from the slide switches, to the 16-bits of the inputs of the *seven_segment_display_subsystem* module. So, *switches_inputs[3:0]* would be connected to *sec_dig1* and *switches_inputs[7:4]* would be connected to *sec_dig2*, and follow the pattern for the remaining two digits.
13. Edit the constraints file to match the top level new signals in *lab_1b_top_level.sv*. You will not need to change *switches_inputs*. For the remaining signals, do the following steps (with helpful snips to illustrate):
    - Uncomment the *clk* signal lines, and you do not need to edit the signal name.

```
8   ## Clock signal
9   set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports clk]
10  create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]
11
```

    - Add the *reset* signal and connect it to the center pushbutton, and uncomment that line. Note that comments are preceded by a "#" symbol, and if you make an inline comment following an existing statement (e.g. line 70), you need to also add a semi-colon before the comment that follows. Also notice the very helpful comment on line 69, about the orientation of the pushbuttons (normally 0, but 1 when pushed down).

```
68  ##Buttons
69  # Basys3 pushbuttons are normally 0, and 1 when pushed down
70  set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports reset]; #set_property
71  #set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports btnU]
72  #set_property -dict { PACKAGE_PIN W19   IOSTANDARD LVCMOS33 } [get_ports btnL]
73  #set_property -dict { PACKAGE_PIN T17   IOSTANDARD LVCMOS33 } [get_ports btnR]
74  #set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports btnD]
75
```

- Add the 7-segment display signals (*AN1, AN2, AN3, AN4* and *CA, CB, CC, CD, CE, CF, CG*, and *DP*) and uncomment those lines.

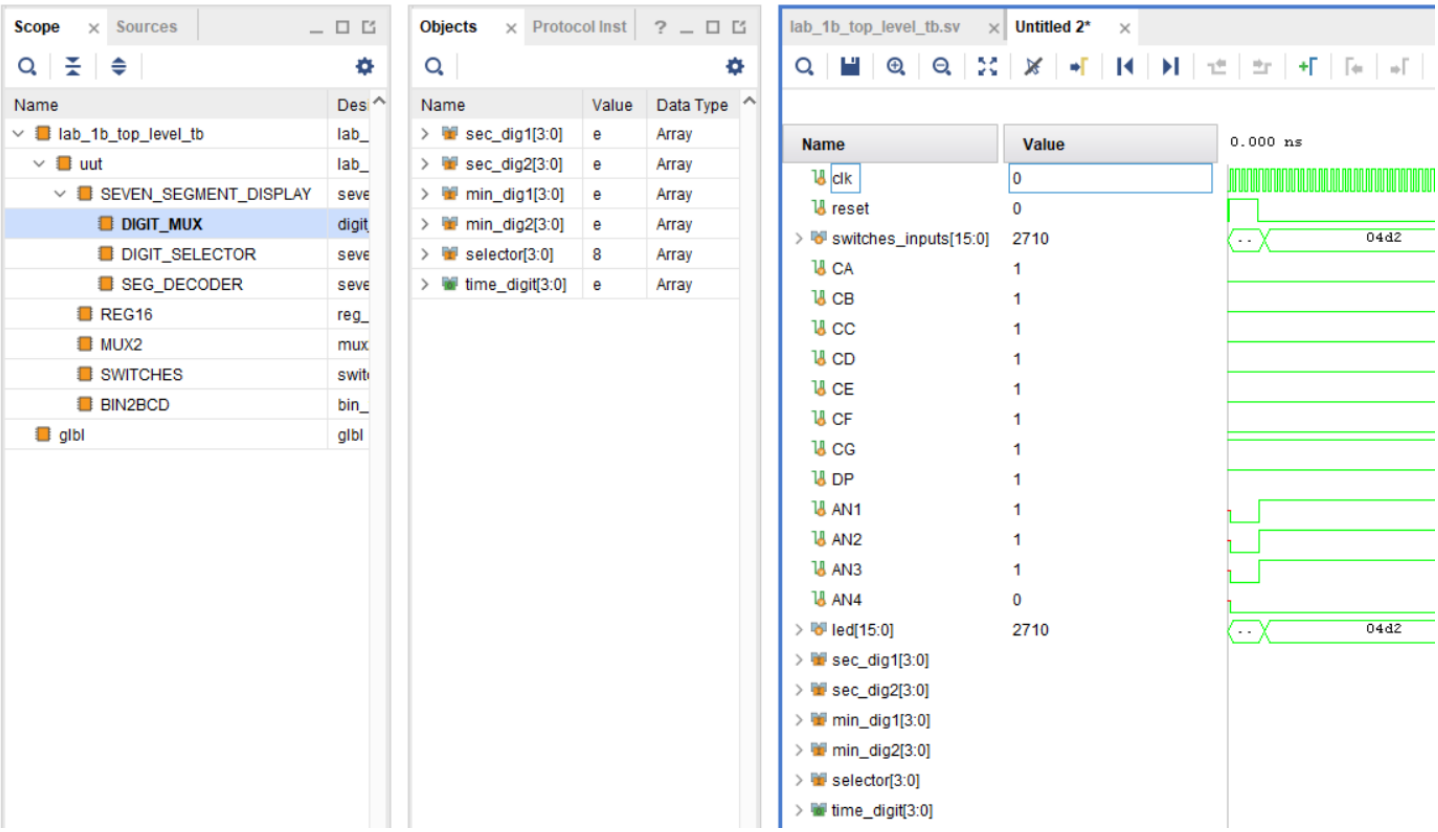
```

##7 Segment Display
set_property -dict { PACKAGE_PIN W7   IOSTANDARD LVCMOS33 } [get_ports {CA}]
set_property -dict { PACKAGE_PIN W6   IOSTANDARD LVCMOS33 } [get_ports {CB}]
set_property -dict { PACKAGE_PIN U8   IOSTANDARD LVCMOS33 } [get_ports {CC}]
set_property -dict { PACKAGE_PIN V8   IOSTANDARD LVCMOS33 } [get_ports {CD}]
set_property -dict { PACKAGE_PIN U5   IOSTANDARD LVCMOS33 } [get_ports {CE}]
set_property -dict { PACKAGE_PIN V5   IOSTANDARD LVCMOS33 } [get_ports {CF}]
set_property -dict { PACKAGE_PIN U7   IOSTANDARD LVCMOS33 } [get_ports {CG}]

set_property -dict { PACKAGE_PIN V7   IOSTANDARD LVCMOS33 } [get_ports DP]

set_property -dict { PACKAGE_PIN U2   IOSTANDARD LVCMOS33 } [get_ports {AN1}]
set_property -dict { PACKAGE_PIN U4   IOSTANDARD LVCMOS33 } [get_ports {AN2}]
set_property -dict { PACKAGE_PIN V4   IOSTANDARD LVCMOS33 } [get_ports {AN3}]
set_property -dict { PACKAGE_PIN W4   IOSTANDARD LVCMOS33 } [get_ports {AN4}]

```

14. Simulate (behavioral) your design with *lab_1b_top_level_tb.sv*. and you should not have to make too many changes to the testbench, other than:
    - the edits to add the top level port signals listed above in Step 4, to the instantiation UUT, and
    - you will need to add a clock generation `always` as shown in the image below.
    - However, you should study the modules within the *seven_segment_display_subsystem* module and realize that there will be a significant delay for the 7-segment outputs to be generated and viewed. This is because the four 7-segment display digits are time-division multiplexed, meaning that only one of the 7-segment displays gets its signals, at a time. However, the system scrolls through all four of the 7-segment displays so rapidly, that our "persistence of vision" makes the digits appear to be visible continuously. You will most likely have to greatly extend your testbench's delays, in order to see all the signals to all four 7-segment displays.

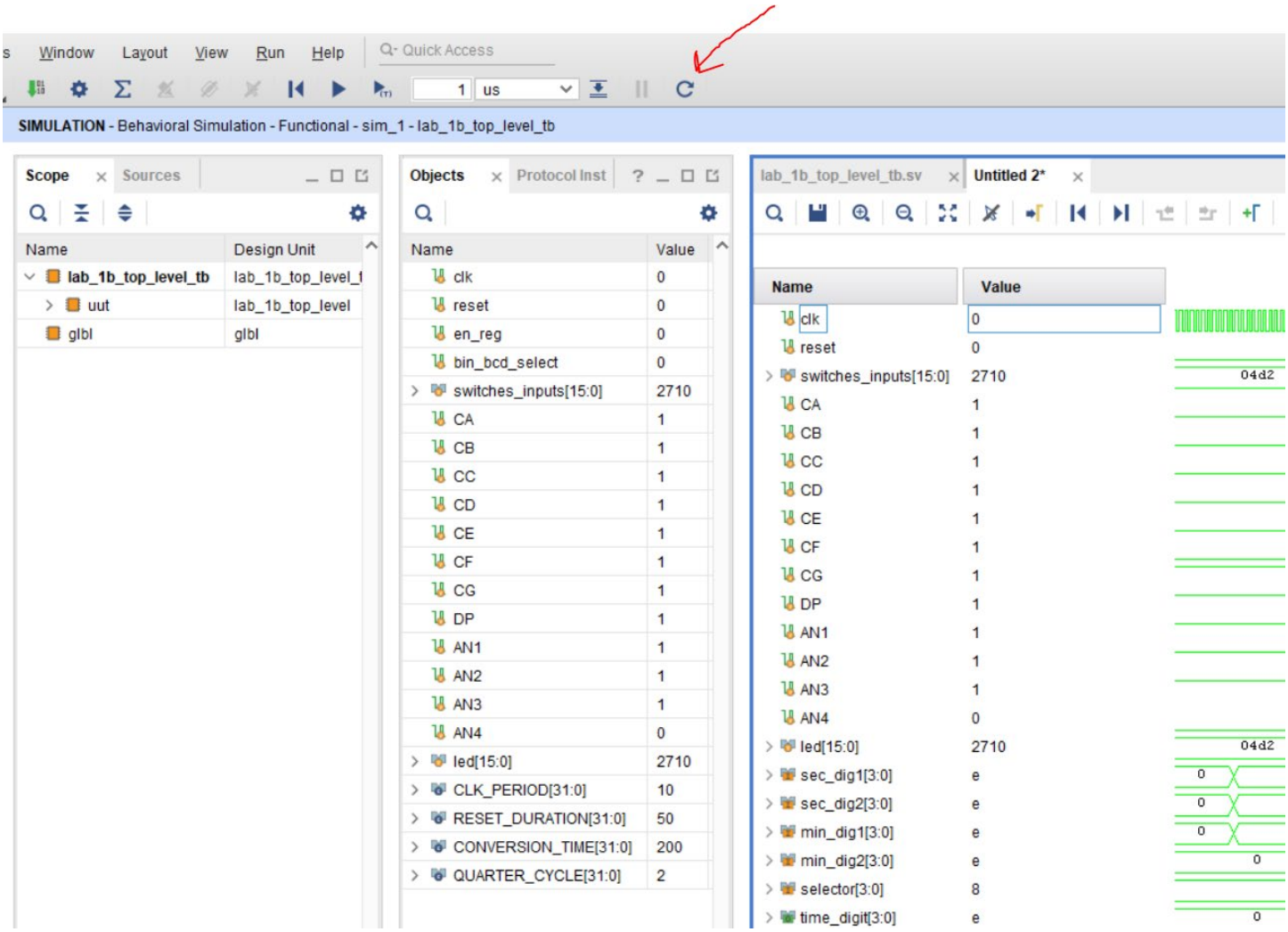
```

    // Clock generation
    always begin
        clk = 0;
        #(CLK_PERIOD/2);
        clk = 1;
        #(CLK_PERIOD/2);
    end

```

Make sure that you can see the waveforms for all the top level signals. Do the signals look correct? What about when the hexadecimal digits of A to F are supposed to be shown on the 7-segment displays, do you get the expected values? **DO: take a snip of the waveforms and paste it into your Design Record. Write an explanation of the problem you observe for the hexadecimal digits, when are they correct and when are they incorrect.**

15. Learn how to add the signals of lower-level modules, into your simulation waveforms. Within “Scope”, click the arrow for “uut” and then click the arrow for “SEVEN_SEGMENTS_DISPLAY,” to expose the lower level modules of DIGIT_MUX, DIGIT_SELECTOR, and SEG_DECODER. Left-click on DIGIT_MUX and drag it to the waveform window, to the location of the other signals names (clk, reset, etc), under **Name**. Those signals should now be in the simulation window (but without simulation waveforms). Repeat for DIGIT_SELECTOR and SEG_DECODER. (Don’t worry about the additional modules you see in the picture (REG16, MUX2, BIN2BCD), these are for later).



16. Relaunch the Behavioral Simulation by clicking the Relaunch Simulation button (pointed to by the red arrow). Now the additional waveforms should be visible. It is very useful to be able to see the lower level signals, when you are debugging and trying to figure out whether you design is working properly. Especially check the signals on the DIGIT_SELECTOR, do these signals help you understand when the new digits to the time-multiplexed 7-segment displays, are presented (i.e. for the persistence of vision)? **<u>DO: paste a snip of all your waveforms, into the Design Record and comment about the DIGIT_SELECTOR module.</u>**



17. Download the configuration to the Basys3 and verify the operation of the 7-segment displays by moving the slide switches.

18. Edit the file seven_segment_decoder.sv, to add the hexadecimal digits of A to F, to the case statement. Fill in the rows in the location pointed to by the red arrow.

```
51
52      always_comb begin
53          // Decode the input data into 7-segment display pattern
54                                      // ABCDEFG         7-segment LED pattern for
55          case (data)                 // 6543210
56              4'b0000: decoded_bits = 7'b1111110; // 0       A-6
57              4'b0001: decoded_bits = 7'b0110000; // 1   F-1     B-5
58              4'b0010: decoded_bits = 7'b1101101; // 2       G-0
59              4'b0011: decoded_bits = 7'b1111001; // 3   E-2     C-4
60              4'b0100: decoded_bits = 7'b0110011; // 4       D-3      DP
61              4'b0101: decoded_bits = 7'b1011011; // 5
62              4'b0110: decoded_bits = 7'b1011111; // 6
63              4'b0111: decoded_bits = 7'b1110000; // 7
64              4'b1000: decoded_bits = 7'b1111111; // 8
65              4'b1001: decoded_bits = 7'b1111011; // 9
66      // Students: fill in the remaining rows for this case statement,
67      // to account for the hexademcial digits A, B, C, D, E, and F
68
69              default: decoded_bits = 7'b0000000; // All LEDs off
70          endcase                     // ABCDEFG
71      end                             // 6543210
```

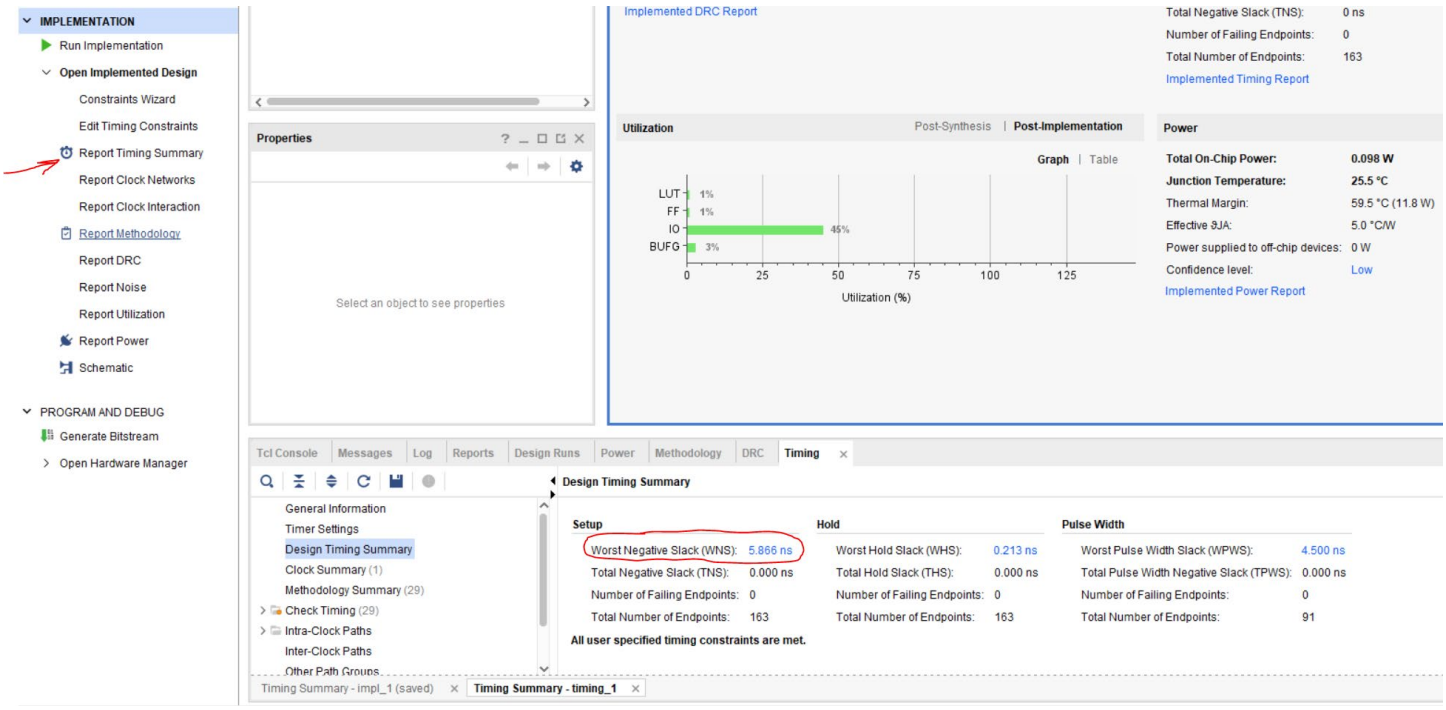
19. Relaunch your Behavioral Simulation and verify that all the hexadecimal digits are now represented in the 7-segment displays. **DO: paste an image of the simulation waveforms into your Design Record. Ensure that the required functionality is shown.**
20. Launch your Post-Implementation Timing Simulation and verify that all the hexadecimal digits are now represented in the 7-segment displays. Find examples of the following timing behaviors:
    - IBUF delay
    - IBUF_BUFG delay
    - Clock-to-Output delay (i.e. the delay for a Flip flop output to change, after the clock edge)
    - Combinational logic delay

    DO: paste an image of the simulation waveforms into your Design Record. Ensure that the required functionality is shown.

    DO: paste an image or images of the simulation waveforms that show the 4 kinds of delays listed in the bullet points.

21. Download the configuration to your Basys3 board and verify that the 7-segment displays show hexadecimal values based on the slide switch settings.

22. Calculate the maximum clock frequency your design can run, using the Worst Negative Slack (WNS). This is one of the most important features of your project, whether it can meeting timing, which in our case is a 100 MHz clock frequency. In Flow Navigator (left-hand panel in Vivado), click IMPLEMENTATION > Open Implemented Design > Report Timing Summary (see red arrow in image below). The WNS (red circle in the image) for your design will be reported in the Timing tab at the bottom of Vivado. **<u>DO: take a snip of the Timing window and paste it into your Design Record.</u>**



Calculate the maximum frequency that your project can run on the Basys3, by using the user desired clock period from the .XDC file, shown in nanoseconds in the red circle below. The user desired clock period should be 10.00, for 10 ns, and let's call this *UserClockPeriod*.

```
## Clock signal
set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports clk]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]
```

The calculation is:

$$MaxClockFrequency\ (MHz) = \frac{1}{UserClockPeriod - WNS}$$

For our example, the calculation is:

$$MaxClockFrequency\ (MHz) = \frac{1}{UserClockPeriod - WNS} = \frac{1}{10\ ns - \ 5.866\ ns} = 241.9\ MHz$$

The result of 241.9 MHz means that we meet our minimum required clock frequency of 100 MHz, so our design should run fine if our functionality is correct. If the calculated maximum clock frequency (based on the WNS and the UserClockPeriod) was below 100 MHz, say 90 MHz, then our design would probably run ok, but would be at risk of glitching or producing incorrect results. This is because the FPGA's Flip Flop timing (setup time and hold time) would probably be violated. In such a case, the WNS would have a negative value, which is bad. The designer could take a couple of different approaches:

- If the negative WNS value has a small magnitude (i.e. you are close to meeting timing), the designer can adjust the synthesis settings to direct Vivado to try harder or to take some alternative approaches to optimizing the design through synthesis. This is outside the scope of ENEL 453.
- However, most likely, the designer will have to re-think their design, examine the reports of the timing failure to get guidance on where the problem might be (and/or use their intuition and knowledge of the design), and re-design that part.
  In ENEL 453, when students fail to meet timing, it is usually because they are "thinking in software" and not "thinking in hardware." It's usually mathematical operations, and of those, it's usually a naïve application of division (i.e. simply using the divide operator ("/"), like in `assign sigA[15:0] = sigB[15:0] / sigC[15:0];`) or modulus (i.e. "%" similar to the division example). Division and modulus are difficult and take many clock cycles, so we would typically use a separate, dedicated module for those operations. Multiplication is easier, because FPGAs typically have dedicated hardware multiplier blocks (like the Xilinx Artix 7 FPGA on the Basys3 board) and the synthesizer can often create a satisfactory implementation. Also note that it would be fine to use such operations of multiplication and division in testbench code, because testbenches are non-synthesizable and are essentially "software."

DO: present your calculations (show your work) and your result for the maximum clock frequency in the Design Record, and also state whether your design meets timing. ASIDE: know how to calculate the maximum clock frequency for exams.

23. **DO: Upload your Design Record to the correct location in the D2L Dropbox. Re-enter the D2L Dropbox and verify that your Design Record can be opened within the D2L viewer.**
24. **DO: Demonstrate your design to your TA, in both Vivado and the Basys3, and show them your Design Record.**

Deliverables

By the end of the lab period **or** the beginning of the next lab period, demonstrate to the TA:

1. Your Basys3 board running with the latest iteration of the lab 1b project, be prepared to explain the design and any part of the Vivado design and programming flow.
2. Your latest simulation and be prepared to explain the signals and their meaning.
3. Your Design Record document. Ensure that the Design Record is uploaded to the D2L Dropbox, before seeing the TA.

Your TA may ask any team members at random to answer any questions. Work together to ensure that all team members fully understand the lab project and its deliverables. You may also be asked to demonstrate your ability to proceed through the entire design flow, including:

1. Create and start a new project.
2. Synthesize and download your design to the Basys3.
3. Simulate your design and setup the waveforms.

Rubric

As the term progresses, the expectations will increase as your skills develop. All team members are required to participate in the entire lab period and to present their project to the TA. Missing team members will not receive a mark for the project. The three strikes policy outlined in the Course Outline, will be in effect for all labs.

Points	Criteria
4	Fully complete and high quality, questions answered well.
3	Fully complete and high quality, some questions not answered well or had to have other team members answer.
2	Mostly complete or answers are weakly answered by team members
0	Below acceptable for credit.

There is no 1 point given. Fractional points, e.g. 2.5, are not given.